purl needs to identify proprietary software

OWASP SBOM Forum, October 2024

Executive Summary

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

One reason for this is that the database that has provided the free vulnerability lookup services that a huge number of organizations rely on, the National Vulnerability Database (NVD), is currently experiencing serious problems. These have greatly inhibited the NVD's ability to link a newly identified vulnerability (i.e., a new CVE Record) with one or more products that are affected by the vulnerability.

The NVD has also had another serious problem for a long time: The "CPE" (Common Platform Enumeration) software identifier utilized by the NVD for close to two decades is deeply flawed; truly automated lookup of software vulnerabilities is difficult with CPE. Some of these problems were discussed (on pages 4-6) in the OWASP SBOM Forum's 2022 white paper, "A Proposal to Operationalize Component Identification for Vulnerability Management". These problems have been multiplied, due to the NVD's recent problems.

Meanwhile, the "package URL" – or "purl" – software identifier is used in almost every major vulnerability database worldwide, other than the NVD and the group of databases that draw from the NVD. In less than one decade, purl has grown from just a concept to being literally ubiquitous in the open source world. The differences between CPE and purl are discussed at length in the 2022 white paper just referenced.

The solution to the NVD's problems with CPE was stated in our 2022 paper: We need to put purl identifiers on an equal footing with CPE in the NVD and other databases that draw from it. When that happens, purls can be used to identify vulnerable products in CVE Records (although CPEs will continue to be used alongside purl as long as users want to have them). At the same time, end user vulnerability management tools will be able to avoid the serious problems with CPE by using purl to search for vulnerabilities in the NVD.

The OWASP SBOM Forum believes that now is the perfect time to start the long process of moving from exclusive reliance on the CPE identifier to utilizing both CPE and purl in CVE records and in the NVD. The choice of which identifier to use can be left to the organization (usually a software developer) that reports a new vulnerability.

However, purl suffers from one important limitation, which must be addressed before these two goals can be achieved: CPE identifies both open source and proprietary software products, while purl currently identifies almost exclusively open source products. Since most private and public sector organizations today utilize proprietary software to accomplish their mission, they are

naturally most concerned about vulnerabilities in that software. How can purl be expanded to identify proprietary software as well as open source software?

In this document, the OWASP SBOM Forum describes two methods by which proprietary software products might be identified using purl, as well as the changes in current formats and procedures that may be required to implement these two methods. In the latter part of this paper, we describe how we intend to recruit a purl Expansion Working Group to design these two new methods, and most importantly to test them in "tabletop exercises" and proofs of concept.

We welcome anyone who wishes to join and/or donate funds to this effort. If you would like to discuss this, please email tom@tomalrich.com and Tony Turner at tony@locussecurity.com.

The automated vulnerability management process

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

1. Every private- and public-sector organization needs to build and continually update an inventory of software products 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 inventory available, since truly effective vulnerability management is impossible without one.
2. Each organization needs to discover, for each software product in its inventory, one or more 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 purl, 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 another reason why it is important to automate this process.
4. The result of the above should be a regularly updated and machine-readable pairing of a) products currently in the organization's inventory and b) vulnerabilities (CVEs) that have been identified in one or more of those products since the last time the organization searched the vulnerability database; 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 CVE Records and 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 is to link vulnerabilities with the products that are affected by them. Most vulnerabilities listed in vulnerability databases have a CVE identifier, although not all do. Every valid CVE is described in a CVE Record, which is originally prepared (and may later be amended) by a CVE Numbering Authority (CNA).

[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, as well as taking measures on their own to mitigate unpatched vulnerabilities, including isolating an affected device on its own network segment. 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.

CVE identifiers are created and distributed through the CVE Program. The program is administered by the MITRE Corporation and overseen by the Board of the nonprofit CVE.org; that Board includes government and private industry representatives. CVE.org is part of the US Department of Homeland Security (DHS) and is funded by the Cybersecurity and Infrastructure Security Agency (CISA), which is also part of DHS.

The mission of the CVE Program is described on the CVE website as:

> ...to identify, define, and catalog publicly disclosed cybersecurity vulnerabilities. There is one CVE Record for each vulnerability in the catalog. The vulnerabilities are discovered then assigned and published by organizations from around the world that have partnered with the CVE Program. Partners publish CVE Records to communicate consistent descriptions of vulnerabilities. Information technology and cybersecurity professionals use CVE Records to ensure they are discussing the same issue, and to coordinate their efforts to prioritize and address the vulnerabilities.

A vulnerability database is essentially a set of statements of the form, "Product XYZ Version 1.2.3 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 is to identify vulnerabilities that affect the software products and intelligent devices that an organization 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 developer of the software or firmware. However, if the vulnerability is deemed serious and a patch is not available, the user organization should act 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 supplier's common name for the product. This is because both proprietary and open-source software products are usually known by multiple names in different contexts; this situation is referred to as "the naming problem". For example, different employees or different divisions of a single large software developer will often refer to a single product with different names.

One of the most important reasons for not being able to assign a unique name to a software product is that large software developers have sometimes absorbed hundreds or even thousands of smaller developers over time. Each of those acquisitions yielded software products that were usually renamed, perhaps multiple times.

Of course, there are many other reasons, besides acquisitions, why a software product may be referred to differently by different groups within and outside of the company. For example, individual divisions or country groups may identify themselves as separate organizations, even though they are all legally part of one larger organization. They may assign their own names to similar or identical software products.

This is why a vulnerability database needs to utilize a machine-readable software identifier for every version of every product included in the database. Ideally, when a user has this identifier, it will *always* match the identifier that was used when the product was identified as vulnerable in a CVE Record. Without such an identifier, it is almost impossible to identify a software product with certainty.

The NVD's special role

The NVD has always occupied a special role in the vulnerability management ecosystem, primarily because it is owned and operated by the US government[2]. Access to the NVD 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 majority of them that have a CVE number - are identified and reported:

1. Software vulnerabilities are usually identified by the developer of the software. Sometimes, vulnerabilities are identified by independent vulnerability researchers.
2. Many of the developers and manufacturers are CVE Numbering Authorities (CNAs), of which there are currently over 400. CNAs are authorized by CVE.org and the Cybersecurity and Information Security Agency (CISA). CVE.org manages the CVE Program and maintains the database of CVEs.
3. Each CNA has a "scope" assigned to it by CVE.org; this delineates the general category or categories of software products or devices for which the CNA has reporting responsibility. This will almost always include products created by the CNA organization itself, but it will often include products created by other organizations, including those from a particular country or industry, etc.
4. When a CNA decides to report a new vulnerability in a software product, 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, the name utilized in the textual description should never be treated as an exact identifier, since even within a single developer organization, one product may be referred to with multiple names
    c. Until early 2024, NVD contractors and/or staff members almost always created a CPE name for each product/version identified in the text of the report; then, they added it to the report (see below).
    d. The most recent CVE specification allows the CNA to include a purl identifier for the affected product when they report a new CVE. However, since there is no provision

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

for utilizing purl in either the CVE.org database or in the NVD itself, it is not likely that any CNAs are doing this today.

After the CNA submits their CVE report and it is accepted by CVE.org, it is entered into the CVE.org database as a new CVE Record. From there, it is transferred to the NVD, which incorporates the new CVE number and most of the textual information from the record. In addition to this, until early 2024, a NIST/NVD staff member almost always created a machine-readable CPE name for each product listed in the text of the CVE Record and added it to the record. Doing that linked the CPE name to the CVE number in the NVD, so that it could be found using an automated search.[3] This process is called "enrichment".

For example, let's assume that the text of the CVE Record for CVE-2024-123435 includes the sentence, "Product XYZ Version 1.2.3, sold by Joe's Software, is affected by CVE-2024-12345". A CPE name that refers to that product might be "cpe:2.3:a:joes_software:Product_XYZ:1.2.3:*:*:*:*:*:*:* ".[4]

If that CPE name has not been added to the CVE Record for CVE-2024-12345, a search on the CPE name will never indicate it is vulnerable to that CVE. This means the only way a user of Product XYZ Version 1.2.3 will learn that their product is affected by that CVE is if they read the text included in the CVE Record and check one-by-one to determine if any of the software products their organization uses is described in that text.

Until 2024, almost all CVE reports in the NVD included a CPE name for each product described in the text of the report; thus, there was little need for text searches. This means that, before 2024, a search using the above CPE name would have, if successful[5], provided a list of almost every CVE that affected that product.

However, in early February 2024 the NVD experienced a problem that caused it to drastically reduce the number of CPE names it added to CVE Records. Because of this, there are currently (as of early October 2024) over 18,000 CVE Records that are in the NVD (meaning that a search on the CVE number will reveal the textual contents of the report), yet do not have a CPE name associated with them[6]. This means the only way that a user of Product XYZ Version 1.2.3 would learn that this

---

[3] Until early this year, a NIST/NVD contractor or staff member also usually added a CVSS score and one or more CWEs (Common Weakness Enumeration) to the CVE report; however, that stopped when NVD contractors stopped regularly adding CPEs. When CISA started their "Vulnrichment" program, they started adding those along with CPE. However, as of October 2024, the CNAs are being encouraged to create these two types of information themselves.

[4] This may not be a well-specified CPE name.

[5] Unfortunately, an inordinately high percentage of NVD searches using a CPE name have always been unsuccessful, due to flaws in the CPE identifier itself. Some of these are discussed below.

[6] This blog post from July 2024 quantified the backlog of "unenriched" CVE Records (i.e., CVE Records 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. Even though NVD contractors are now adding CPE names to some of the new CVE reports the NVD receives, the overall backlog continues to grow. CISA, as part of their "Vulnrichment"

product/version was affected by CVE-2024-12345 would be to do a text search through the 18,000 CVE Records in the backlog; of course, it is doubtful that many organizations would be willing to do that.[7]

Because of this, as of late September 2024, an automated search on a particular CPE name in the NVD is unlikely to reveal most of the CVEs that have been reported since February 2024, even if the product is affected by one or more of them. This is why we say that truly automated end user vulnerability management is almost impossible today, at least if it uses CPE software identifiers.

The CPE problem

As already mentioned, in 2022 the SBOM Forum (now the OWASP SBOM Forum) described, in our proposal for fixing naming problems in the NVD (pages 4-6), six serious problems with CPE identifiers (although there are other problems as well).

The biggest problem with CPE is that there is 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 mandatory fields in CPE is "vendor". Even tiny differences in spelling are significant in CPE. For example, a CPE name that includes the vendor name "Microsoft" is different from the same CPE name that includes "Microsoft, Inc", which in turn differs from the same 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.

Regular users of the NVD, such as software suppliers, have devised many ways to at least partially ameliorate these problems using fuzzy logic, machine learning, etc. In fact, it's safe to say that almost every such organization has their own bag of workarounds that make the experience less painful, although far from being automated.

However, these techniques can also lead to erroneous results, in that one vulnerability management tool may use a defined list of aliases to reference identifiers, while another uses fuzzy logic matching. This can produce inconsistent results across different tools that are ultimately all relying on the NVD.

---

program, has added CPEs to about 2,000 CVE reports that didn't have them previously. While that has helped, it has not come close to eliminating the backlog.

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

[7] Some privately-run vulnerability databases, that are based on NVD data, may be performing this service for their customers.

Moreover, 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 and repeatable 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 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 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 SBOM Forum paper describes in detail why purl is 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 in the vulnerability database (this will usually be because it 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 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 of the compound. 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 are used in a particular CPE name (especially vendor name, product name and version string) can't be predicted with any accuracy, for reasons mentioned earlier.

Therefore, when a software user is searching the NVD to learn about vulnerabilities that affect the software products they use, they may have to search many possible values for the name

of the product and the vendor name. Having to do this to find most CPE names in the NVD obviously makes truly automated searches impossible[8].

By contrast, purl is an intrinsic identifier. Today, purl primarily identifies open-source software packages that are available in package managers like Maven Central and PyPI. 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 most 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 the"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 within one package manager, for example PyPI.[9]
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).

For example, the purl for version 1.11.1 of the package named "django" in the PyPI package manager is "pkg:pypi/django@1.11.1".

Note that, absent somebody making a mistake, only one purl matches any package in a package manager and vice versa: any package in the package manager can only have one purl. When the supplier of the package (probably the open-source community that maintains django, in the above example) 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.[10]

[8] Frequent NVD users often 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 ever created by the NVD), to try to guess the CPE name used to reference a particular product.

[9] More specifically, the *version string* of the package will never change in the package manager. A later version of the product might be given a different common name and will therefore have a different purl. However, the purl for an existing version of the product should never change.

[10] 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 normally have purls – although workarounds are available in some cases.

How will purl accommodate proprietary software?

We have already pointed out that currently the purl identifier is primarily used to identify open-source software that is available in package managers; it is almost 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 mandatory CPE "vendor" field, since a free product has no vendor. The purl spec has a vendor field, but it is optional, not required).

However, the NVD (and any other vulnerability database that utilizes CPE identifiers for software) also needs a reliable identifier for proprietary ("closed source") software. CPE is simply not a reliable identifier for proprietary software, any more than it is for open-source software.

This is especially important because most public and private sector organizations worldwide run their operations primarily using proprietary software. Not only are they OK with having to pay for their software, but they also consider the fact that they must pay to be an advantage. This is because they know they can normally expect a response (although not always prompt) when they contact the supplier with a problem. They can't usually count on that with open-source software.[11]

There are some open-source software products (such as the Chrome browser) that are widely used in many private and public sector organizations. There are also many open-source products like Java libraries that are included in commercial products. However, the fact that there are so few open-source software products in heavy use by private sector organizations (except as components of commercial products) shows they are the exceptions that prove the rule.

Because of this, few organizations will be able to run a comprehensive vulnerability management program, until there is a reliable identifier they can use to learn about vulnerabilities found in the proprietary software they operate every day. Since CPE is not a reliable identifier (especially given the NVD's problems in 2024, which will undoubtedly continue in 2025) and since purl is already widely used for open-source software, it makes sense to determine now how purl can accommodate proprietary software.

To answer the above question, we first need to determine what is essential to make purl work. An important feature of purl today 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.

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 be counted on to manage the names within their namespace by removing duplicates, making sure each name is unique within the namespace, etc. purl provides a "distributed namespace" – a set of

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

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 by some responsible party.

On the other hand, there is only one namespace for CPE names: the set of CPE names found in the NVD[12]. These have (mostly) been created by NIST/NVD employees and contractors. Since the NVD has never published 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 finding a CPE name in the NVD requires so much guesswork.

Truly automated vulnerability management will never be possible if the software security community continues to rely on CPE names to search vulnerability databases. Yet, CPE is currently the only identifier available for proprietary software. How can purl accommodate proprietary software, so that most organizations will someday be able to enjoy truly automated vulnerability management?

As just discussed, it seems 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 those found in 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.

Two options for purls for proprietary software

As in the case of purls for open source software, there are probably multiple ways by which 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) and by Tony Turner, Principal Consultant of Locus Security.

1. *purls for proprietary software not distributed through online stores*

   On pages 11 and 12 of the SBOM Forum's 2022 white paper, we described, at a very high level, a methodology for making purls for legacy software; this was suggested by Steve Springett. This methodology is based on SWID tags. In principle, it could be applied to any open-source or proprietary software product: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

[12] This was true a year ago, but during the 8-9 months that the NVD has been experiencing problems, several other vulnerability databases that are based on the NVD have created their own CPE names for at least some of the CVE Records in the NVD's backlog. Each of these can be considered its own CPE namespace; these databases are currently the best option for identifying current vulnerabilities applicable to software products. However, it is unlikely that, when and if the NVD recovers from their problems, they will automatically incorporate CPE names created by these other organizations, especially because there will likely be multiple competing names for a single product.

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.

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. According to the NVD website, SWID tags 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 originally intended by NIST to be the replacement for CPE in the NVD, 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 as software identifiers; 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 the developer makes the SWID tag 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 should match the purl used by the developer to report any vulnerabilities to CVE.org, since the developer will normally create the SWID tag. This is an example of a controlled namespace, because the developer of the product determines the contents of the SWID tag for the product. Only if a developer has not created a SWID tag for their product should a third party create the tag.[13]

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 that specification needs to be tested in a tabletop exercise – and will be, sometime after the OWASP purl Working Group starts meeting. 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 produce the corresponding purl, which can then be used in a vulnerability search.

[13] Of course, there will need to be some way to identify cases where a developer and a third party have prepared conflicting SWID tags for the same product. This is just one of the many issues that need to be worked out by the purl Expansion Working Group we are proposing.

The 2022 SBOM Forum white paper did not address an important problem: how a user can locate the SWID tag for a software product they use, especially if it is not a current product. If it is a current product, the tag should be available with the binaries.

For legacy products that did not originally include a SWID tag, the developer – or their successor company, if the original developer has been purchased by or merged with another company – will need to create one[14]. But how will an end user, or more specifically the user's vulnerability management tool, discover the appropriate SWID tag? In order not to "break" the automated vulnerability management process, the tool will need to discover and read the tag without human intervention.

There are multiple conceivable mechanisms by which the user's tool could discover the correct SWID tag, without requiring human intervention. Two of them are:

A. The supplier[15] of the product could maintain on their website a page with a well-known name, such as "purlswid.txt". The user's vulnerability identification tool, such as Dependency Track, could do a DNS search for the supplier, then go to the supplier's home page. There, they could locate the purlswid.txt file and search the file for the SWID tag or tags that best match the product name and version information entered by the user (probably obtained from the "About…" tab on the home page of the user's instance of the software product).
B. The Transparency Exchange API, under development by the CycloneDX project, will be able when complete to discover a wide variety of software supply chain artifacts, including SBOMs, VEX and VDR documents, and SWID tags. Full release of the API, as an ECMA standard, is planned for the end of 2025.

When it starts meeting, the OWASP purl Working Group will identify what they think is the best automated mechanism for locating the SWID tag for a software product. However, whatever choice they make will need to be validated in one or more tabletop exercises or proofs of concept.

Whatever mechanism is chosen, it cannot require a central database of SWID tags. Building and maintaining such a database would be quite expensive, and it is unlikely that funding for it would ever be available.

---

[14] 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 can be accessed, it might be utilized to "jump start" the effort to identify legacy purls.

[15] If a software product that was originally developed by one supplier is now owned by a different supplier, there will need to be a way for the user to find the name and website URL of the current supplier. Once it starts meeting, the OWASP purl Working Group will discuss different options for doing this, and identify their preferred option.

Steve Springett has started an OWASP project called "Common Lifecycle Enumeration" that is intended to address problems like finding the current owner of a legacy product. However, that project is still in its early stages. Steve encourages anyone interested in participating in this project to contact him at steve.springett@owasp.org.

The SWID tag 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. If the supplier of a product reported a vulnerability for a product/version to CVE.org and included in the CVE Record the purl created using the SWID tag created by the supplier, that should always match the purl that a user (or a tool acting on their behalf), who is searching years later for vulnerabilities in that product/version, will create based on that same tag.

However, suppose the supplier made a mistake when they originally created the purl based on the SWID tag? For example, suppose the product name on the SWID tag was "XYZ productivity tool", but the supplier mistakenly entered "XYZ productively tool" as the product name in the purl when the supplier (or a CNA acting on their behalf, if they are not themselves a CNA) reported the vulnerability to CVE.org.

Further, suppose a future user of the product, who wants to learn about recently reported vulnerabilities, faithfully downloads the correct SWID tag from the supplier's website and uses that to create the purl to search in a vulnerability database. In this case, the purl used for the search won't match the purl in the CVE record, so the user will never learn that their product is affected by this vulnerability. What can be done to avoid this problem?

One way to avoid this problem might be require the CNA to submit a copy of the SWID tag used to create the purl in the CVE Record, when they originally report the vulnerability. The system that processes the new CVE Record can easily confirm that the product name and version string shown in the CVE Record match the product name and version string shown on the SWID tag. If they don't match, the CNA would be required to change the purl in the CVE Record.

2. Implementing the SWID tag option

Implementing the purl SWID tag option for proprietary software will require multiple steps. These steps[16] could include:

A. Conduct a "tabletop exercise", in which multiple software suppliers test use of the current SWID type specification to confirm that purls created using that specification can successfully identify proprietary software products in practice.
B. After the tabletop exercise is finished, publish the final suggested specification for the SWID type in purl. Submit a pull request to change the specification.
C. Develop proof of concept code to parse a SWID tag and create a purl based on the specification; in principle, this is what Steve Springett's "Package URL SWID Generator" app does, based on the current SWID type spec in purl. Developers of tools that search vulnerability databases will be able to incorporate this code into their tools. Also, a tool used by a CNA to create a purl based on a SWID tag provided by a software supplier will utilize the same code.
D. Train software suppliers to use Steve Springett's app (or a similar one) to generate a SWID tag "from scratch" for a new (or legacy) product/version.

[16] Some of the steps listed here may not be appropriate for the initial planning project, but rather for the rollout project to follow.

E. Test use of the Transparency Exchange API (see above) to discover and retrieve SWID tags made available by suppliers.
F. Work with CVE.org to develop training for the CNAs on creating purls and adding them to CVE records. Include discussion of the SWID purl type.
G. Work with operators of vulnerability databases that already utilize purl, such as OSS Index and OSV, to accommodate the SWID purl type.
H. Work with the NVD to accommodate purl in general and the SWID purl type in particular.
I. "Evangelize" proprietary software developers about the importance of creating SWID tags for both current and legacy product versions.
J. Explain to both software users and security tool vendors the importance of purl in general, and of the new types that accommodate proprietary software.

3. purls for software in app stores

Steve Springett and Tony Turner have both pointed out that, in the world of proprietary software products, the closest thing to a package manager is an app store like Google Play™, the Apple Store™, or the Microsoft Store™ (of course, there are other appstores, although these are probably the three largest. For example, Google Play contains over 3 million Android apps).

Like a package manager, an app store provides a single download location[17] for a variety of software from multiple suppliers (which are open source projects, in the case of package managers). Also like a package manager, an app store provides a controlled namespace; the store can be counted on to make sure each product in the store has a unique name and that product versions are consistently identified. Each app store can be treated by purl almost as if it were a package manager, meaning it will have its own purl type.

While most software products are not sold in app stores, millions of products are available in them (this is especially true for software used in mobile devices). It is impressive to think that all the products in an app store would instantly acquire a purl identifier, as soon as a new type for the store is added to the purl specification.

Because of their strong resemblance between app stores and package managers, it is likely that not a lot of work will be required to integrate the stores into the purl ecosystem. The work will mostly consist of creating the type specification for each app store and creating the pull requests for the new purl types.

Version ranges

A software vulnerability seldom appears 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. This means that vulnerability management is likely to work much better if it can be based on version ranges, not individual versions.

[17] In some cases, one store may have multiple download locations. For example, there are five different Apple App Stores, including for iOS, VisionOS, tvOS, watchOS and iPadOS. Each of these might be assigned its own purl Type, or they might all be combined into one Type.

Today, CVE Records often state that a vulnerability applies to a range of versions in a software product, not just to one version or multiple separate versions. However, that assertion is strictly in text; while the CPE specification supports version ranges, that capability is seldom used – and more importantly, it's unlikely there are any user tools that support it. 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 each CVE report.

Ideally, a software identifier should be able to specify a version range in a machine-readable format that states, for example, "Versions 3.4 through 5.6 of product XYZ are affected by CVE-2024-12345. Other versions are not affected." A user vulnerability management tool that parses that identifier will locate 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 annotated as "affected" in their organization's vulnerability management 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 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/equals operators, will work fine with vers.

Implementing vers in purl will require a big effort, mainly because of the large number of end user tools that will need to be updated by their developers or maintainers; a large education effort will also be required (currently, the CycloneDX SBOM, VEX and VDR formats are the most prominent end user tools that incorporate vers). On the other hand, the payoff will ultimately be huge, once this goal is realized.

The way forward

One year ago, there was no serious discussion in what might be called the "CVE ecosystem" (i.e., the set of vulnerability databases based on CVE identifiers, including the NVD) of moving away from

CPE as a product identifier. However, that has changed today, since the NVD's problems have led to the realization that CPE is no longer a reliable source of vulnerability data.

In fact, searching for a particular product/version in the NVD now is likely to turn up only about one quarter of the recent vulnerabilities (those reported in 2024) that apply to that product/version. This is because two thirds of the CVEs identified since early February have not been "enriched" with a CPE name; therefore, those CVEs are invisible to an automated search. It is as if your doctor had stopped reading about new diseases before the coronavirus pandemic started in 2020. Therefore, they did not diagnose Covid when you described your symptoms of Covid to them.

However, the NVD's current slowdown isn't the primary issue. The primary issue is the need to introduce purl as an equal software identifier with CPE in the CVE ecosystem, including the NVD. That is not possible today, since purl does not currently support proprietary software.

Since the great majority of medium-to-large organizations utilize mostly proprietary software to run their operations, they need to be able to locate vulnerabilities found in those products. That is why purl support for identifying proprietary software is essential. The OWASP SBOM Forum has identified two primary methods for expanding purl coverage to proprietary software: identifiers based on purl types for commercial "software stores" and identifiers based on SWID tags, using the SWID purl type.

However, both of these methods need to be tested, and the processes and procedures around their use need to be discussed and documented. Tony and Tom do not wish to do this on our own. Instead, we propose to create an OWASP purl Working Group as a subgroup of the OWASP SBOM Forum. This group will be open to all parties with an interest in expanding purl use to proprietary software. We propose that the group meet virtually either once a week or once every two weeks, at a time that is as convenient as possible for participants in the United States, Europe and Israel.

Once meetings start, the working group will:

1. Develop preliminary specifications and deployment plans for both methods of expanding purl coverage to proprietary software: via software stores and via SWID tags.
2. Document suggested policies and practices that will facilitate implementation and use of the new purl types. These will apply to software developers and distributors, vulnerability database operators and software end users. These policies and practices will not be part of the purl specification itself.
3. Conduct tabletop exercises for both methods.
4. Conduct proof-of-concept exercises to test policies and practices required to implement and use both methods.
5. Hold discussions with representatives of CVE.org about proposed changes to policies and practices of CVE Numbering Authorities (CNAs) and CVE.org staff members.
6. Document suggested new or revised purl types required to facilitate the two methods. The working group will create a type definition for each new type and submit it in a pull request to the purl project. Each software store will require its own type, just as each package manager requires its own purl type.

The above list does not include any general "rollout" of the two methods for expanding purl to identify proprietary software, other than the rollout that will naturally occur as part of the proofs of concept. The project described above is one for "planning and testing". The point of this project is to answer all the important questions that need to be answered before rollout can begin. We don't want to start rollout if significant questions about procedures, formats, etc. remain unanswered.

We hope this effort will be seen as a watershed event in software security. This is because it will not only enable truly automated software vulnerability management, but also implement it based on a much more effective identifier: purl.

Of course, this will not be a short effort; we estimate it will require at least 2-3 years for all the changes described in this document to be implemented and widely used in practice. However, nothing described here requires any technological breakthrough before it can be implemented. We'll never finish if we don't start.

The OWASP SBOM Forum can't do this on our own. We want to cooperate with other organizations – both public and private – as well as with interested individuals. We seek volunteers (both individuals and organizations) to participate in the project described above. We also seek donations to support the work.

Organizations and individuals can donate $1,000 or more to OWASP and direct that the donation be "restricted" to the SBOM Forum. OWASP is a 501(c)(3) nonprofit organization, meaning that in many cases the donation will be tax-deductible.

Donations can be made both online and through direct interaction. If you are interested in donating to and/or joining this effort, please email Tom Alrich at tom@tomalrich.com and Tony Turner at tony@locussecurity.com.