purl needs to identify commercial software

version 2.0

OWASP SBOM Forum, December 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 close to 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.

Fortunately, there is an alternative to CPE. 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 being 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.

One important component of the solution to the NVD's problems with CPE was stated in our 2022 paper: CVE Numbering Authorities (CNAs), the organizations that report new vulnerabilities identified with CVE numbers, need to include purl identifiers in the CVE records they create. This will create the required link between a CVE and one or more open source software products, identified using purl, that are affected by that CVE. However, even though the recently approved CVE 5.1 specification permits inclusion of purls in CVE records, very few CNAs are including them now, since there are currently no documented policies or procedures for doing this.

But an even bigger step is required for purl to become co-equal to CPE in the vulnerability management world: purl must be able to identify both open source and commercial software products. Currently, purl can identify almost exclusively open source products, not commercial ones. Since most private and public sector organizations today utilize commercial software to accomplish their mission, they are naturally most concerned about vulnerabilities in commercial

software. How can purl be expanded to identify commercial as well as open source software products?

In this document, the OWASP SBOM Forum describes a set of changes by which commercial software products might be identified using purl, as well as the changes in current formats and procedures that may be required to implement these changes. In the latter part of this paper, we will describe how the Purl Expansion Working Group will design these changes, and most importantly test them in "tabletop exercises" and a proof 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 Alrich at 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 most widely used vulnerability database in the world is the National Vulnerability Database (NVD); this is run by the National Institute of Standards and Technology (NIST), part of the US Department of Commerce. The only software 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 search process (often using a commercial tool) to look up every product in their inventory in the NVD and/or other vulnerability databases. If the organization utilizes at least some commercial software products, the only way to do this today, without having to search multiple vulnerability databases using both CPE and purl identifiers, is to utilize the NVD or one of the databases derived from it – despite the problems with the NVD and CPE described earlier.
4. The goal of this automated search process is for the organization to produce (ideally once a day) a 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 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 (for example, "CVE-2021-

[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.

44228") 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).

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.org 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 and mitigate vulnerabilities that affect the software products and intelligent devices that an organization uses, to prevent hackers from exploiting those vulnerabilities to compromise the organization.

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 commercial 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, as well as to the company they work for, with different names.

One important reason 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 also assign their own names to similar 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 found an identifier for a software product, it will *always* match the identifier that was used when a CNA assigned a CVE number and reported a vulnerability in the product. Unless there is such a match, it will be almost impossible to learn about vulnerabilities in a software product using an automated process.

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. 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 that have a CVE number - are identified and reported:

1. Software vulnerabilities are usually identified by the developer of the software. A small percentage of vulnerabilities are identified by independent vulnerability researchers.
2. Many software developers are CVE Numbering Authorities (CNAs), of which there are currently over 400. CNAs are authorized by CVE.org, an agency that is part of the US Department of Homeland Security (DHS). CVE.org manages the CVE Program and maintains the database of CVEs.[2]
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 reports 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).

[2] The CVE.org database used to be identified as "MITRE". The MITRE Corporation originally developed the database under contract with the US government. MITRE employees still maintain the database, but legal responsibility for it is now vested in CVE.org, a nonprofit organization whose board consists of representatives from the US government (DHS and the Department of Commerce) and private industry.

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 for utilizing purl in either the CVE.org database or in the NVD itself (the two main destinations for CVE records today), it is not likely that many CNAs are doing this now.

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-2024-12345 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 using 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 is certain to learn their product is affected by that CVE is if they read the text included in each CVE record and check one-by-one to determine if any of the software products their organization uses is described in that text. Given that at least 29,000 new CVE records have been added to the NVD this year and that this process would need to be executed daily, this is obviously impossible for most organizations.

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 the 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 December 2024) over 18,000 CVE records that are in the NVD, yet do not have a CPE name

[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 began 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 flaws are discussed below.

associated with them[6]. This means that a user of Product XYZ Version 1.2.3 that performs a straightforward search of the NVD, using a CPE for that product/version, will not normally learn that it is affected by CVE-2024-12345 unless they perform a text search through the 21,000 CVE records in the NVD's backlog (as of December 2024); of course, it is doubtful that many organizations would be willing to do that on their own.[7]

Because of this, as of late 2024, an automated search on a particular product/version in the NVD is unlikely to reveal most of the CVEs that have been reported for that product/version 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 six serious problems with CPE identifiers[8] in our proposal for fixing naming problems in the NVD. However, 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 usually no single possible 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 including one of these variants will come up with no results if the CPE name in the NVD was created using another variant.

Regular users of the NVD, such as software suppliers, have devised many ways to at least partially mitigate these problems using fuzzy logic, machine learning, etc. In fact, it is safe to say that almost

---

[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. 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, although at a slower rate. CISA, as part of their "Vulnrichment" 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.

[7] A few privately-run vulnerability databases that are based on NVD data are performing a service like this for their customers. One such organization, VulnCheck has recently announced that in 2024 they have "enriched" a much higher percentage of CVE records than the NVD itself has.

[8] These problems are described on pages 4-6 of the paper.

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 divergent results. For example, if one vulnerability management tool uses 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.

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 is not possible.

If there were no realistic alternative to CPE names for identifying software products, the above situation would be quite depressing. However, purl currently offers a far superior method for identifying open source software. We are now proposing a way to extend purl's coverage to commercial software as well.

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 already have all the information they need to create an identifier that exactly matches the identifier used in the vulnerability database. 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 formula for 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. Therefore, Social Security number is an extrinsic identifier.

CPE 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 CPE name (especially vendor name, product name and version string) can't be predicted with any accuracy, for reasons described 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

product name and the vendor name. Having to do this to find a large percentage of CPE names in the NVD obviously makes truly automated searches impossible[9].

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.[10]
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 (in the above example, probably the open-source community that maintains django) reports a vulnerability for version 1.11.1 of the package, they will use the same purl that the user will create 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[11]. An open-source product in a package manager intrinsically has its own purl, just like a

[9] Frequent NVD users often utilize information from a variety of external information sources to try to guess the CPE name used to reference a particular product. These sources include 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). However, guesswork should never be an essential part of vulnerability identification; yet that is the case today for users of the NVD and CPE.

[10] More specifically, the *version* of the package will never change in the package manager. A purl for a later version of the same product will include a different version string, and therefore will be a different purl. However, the purl for an existing version of the product should never change.

[11] This should not be interpreted to mean a software supplier who creates purl identifiers for their products and distributes them to customers is somehow "breaking the rules." Since some users will be reluctant to create their own purls, the supplier should do what they can to make vulnerability management as easy as possible for their customers.

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.[12]

How will purl accommodate commercial software?

We have already pointed out that today 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. However, purl does not currently provide a means of identifying commercial software. In fact, the only identifier that can be used today to search for commercial software in a vulnerability database is CPE – which, as we have already described, has never been a reliable identifier and is now even less so, given the NVD's problems.

Most public and private sector organizations worldwide run their operations primarily using commercial software. Therefore, they need to very regularly search vulnerability databases to learn about newly identified vulnerabilities that affect the software they use; daily searches are best. Given the huge number of software products that are in use in even an average-sized organization, this goal can only be achieved through a completely automated process.

Due to CPE's many problems (especially the current problems), complete automation will never be possible in a vulnerability database that utilizes CPE. Therefore, 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 commercial software they operate every day. Since CPE is not a reliable identifier and since purl is already widely used for open-source software, it makes sense to determine now what would be required to enable purl to identify commercial software products.

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 source: 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. In other words, the user should always learn about that vulnerability when they search a vulnerability database using that purl.

What we have just described is a "controlled namespace": The product names in the package manager are under the control of the operators of the package manager. Those 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. Unlike with CPE, there should never be more than

[12] 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. However, the purl working group, led by Philippe Ombredanne, the creator of purl, is trying to address this problem now (December 2024).

one possible name for a software product in a purl. If a user isn't sure what an open source product is named in a package manager, they just need to check the package manager.[13]

Purl provides a "distributed namespace" – a set of controlled namespaces, which are today mostly linked to package managers. It doesn't matter how many namespaces 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[14]. 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 to include in the CPE name), there is no way for a user to be certain to create a CPE name that will match the one used when a vulnerability was reported for the product in question. 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 commercial software. How can purl accommodate commercial software, so that end user 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 commercial software is not usually available in package managers, we need to identify one or more ways in which there can be controlled namespaces for commercial software.

[13] Sometimes, the "same" open source software product will be available in multiple package managers; often, the name will vary between package managers. This doesn't affect purl, since the name in the purl always reflects the name in the package manager identified in the purl type (e.g. if the type is "maven", the package manager is always Maven Central and vice versa). However, it does affect CPE, which has no standard means of representing the package manager. It is up to the NIST/NVD contractor who creates the CPE to determine how they will represent the package manager in a CPE name they are creating. As a result, CPE names for open source products are very hard to predict.

[14] This was true before the spring of 2024, when the NVD started experiencing serious problems. Since then, 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 commercial software products. However, it is unknown whether, when and if the NVD recovers from their problems, they will automatically incorporate CPE names created by these other organizations into the appropriate CVE records, especially because there will probably be multiple competing names for a single product.

Creating purls for commercial software

As in the case of purls for open source software, there are probably multiple methodologies by which a purl can be created for a commercial 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.

One of these methodologies takes advantage of the fact that app stores (which primarily distribute commercial software) bear a strong resemblance to package managers; therefore, purls for software distributed through app stores could probably follow almost identical rules to purls for software distributed through package managers. We will discuss that idea later.

However, most commercial software is only provided in a direct transaction between the supplier and the customer – for example, the supplier allows the customer to download the software from their website after a commercial transaction has been concluded. Trying to treat every one of the millions of commercial software suppliers (including a host of "garage shops") as a separate purl type would be impossible.

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 commercial software; this methodology was suggested by Steve Springett. In the SBOM Forum's methodology, the supplier controls the namespace for each product individually by filling out a "SWID tag" (SWID stands for "software identification"). The tag includes important metadata about a software product, whether open source or commercial.

SWID tags were developed by the National Institute of Standards and Technology (NIST) to be a much more comprehensive (and well-defined) identifier for software than CPE names[15]. Based on the ISO/IEC 19770-2 specification 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.

[15] NIST is part of the US Department of Commerce. It runs the NVD.

Nevertheless, SWID tags provide a well-defined format for presenting metadata about a software product[16]. If a software developer creates a SWID tag whenever they "ship" a new or revised software product or version of a software product, an automated vulnerability management tool like OWASP Dependency Track can easily extract from the SWID tag the four or five pieces of information required to create a purl with the type "SWID".[17] Using that purl, the tool will be able to search a vulnerability database to find CVEs that apply to that product/version.[18]

When the software customer utilizes the SWID tag provided by the developer to create the purl for the product, it should match the purl used by the developer (or the CNA they use) to report any vulnerabilities to CVE.org; this is because the developer will almost always create the SWID tag, so it will be identical to the tag created by the customer (this identity will be enforced by use of a free SWID-to-purl generation tool[19], which ingests a SWID tag and outputs a purl based on it. Both the developer and the customer will be able to download that tool).

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; this is part of the agenda for the Purl Expansion Working Group. Note there are non-mandatory 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 metadata about a software product and creates a SWID tag. A developer that utilizes this tool will always create an acceptable SWID tag. While a tool to create a purl from a SWID tag is not available today, we expect that, once the Proof of Concept for use of the SWID purl type to identify commercial software products (described later in this paper) concludes, that tool will be available.

In addition, it is very likely that vulnerability management tools like Dependency Track will be able to a) discover and download a SWID tag using the Transparency Exchange Identifier (see below) and

---

[16] It is important to note that the SWID format in itself does not play an essential role in the software identification methodology described in this white paper. We could easily have replaced SWID with a generic format that just includes the ten or fifteen fields found in the purl SWID type definition. However, a lot of SWID tags were created 5-10 years ago, meaning that many legacy software products have SWID tags today. That fact, plus the fact that SWID is an international standard, made the SBOM Forum choose SWID as the format for the document containing metadata on a software product that the supplier of the product will create to be the "source of truth" for the purl for the product.

[17] Even this step may not be necessary, since the suppliers could easily distribute the purl for a product/version with the SWID tag for that product/version. See the discussion below.

[18] This assumes that CNAs start including purls for both open source and commercial software products in CVE records, and that they base the purl on the contents of the SWID tag developed by the supplier of the product/version. The rollout of the SWID purl type will need to focus heavily on training and encouraging the CNAs, since their participation is key to the success of this project. Because many CNA organizations are commercial software suppliers, and because extending purl to commercial software is key to their customers being able to manage vulnerabilities in a truly automated fashion, it is likely that at least a large percentage of CNA organizations will want to do this.

[19] This has not yet been developed, but should not be difficult to develop.

b) produce a purl based on that SWID tag. That purl can then be used to automatically search a vulnerability database for vulnerabilities in the product identified by the purl.

Discovering and downloading SWID tags using TEA

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. If it is a current product, the tag should usually be available with the binaries. However, for most legacy products (and some current products, to be sure), the binaries will not be available to a user.

There are many legacy software products that have SWID tags available. For those that do not, the product's developer (or their successor company, if the original developer has been purchased by or merged with another company) will need to create one. 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 download the tag without human intervention.

Fortunately, the Transparency Exchange API (TEA), under development by the OWASP CycloneDX project, will allow software users to discover and download a 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, although it will be available for testing in the spring of 2025.

We expect to be able to test use of TEA to distribute SWID tags as part of the Purl Expansion Design and Proof of Concept project. The two leaders of the TEA project, Olle Johansson and Steve Springett, are both active members of the OWASP SBOM Forum (as well as of the purl project).

Creating and distributing purls

It is important not to focus too much attention on the idea that software users will need to create purls by utilizing information found on SWID tags. In fact, it is likely that most customers of commercial products will never need to use a SWID tag to create a purl for one of those products since the supplier will provide the purl to them along with the tag, when they purchase the product.

This is because a software supplier who is willing to create a SWID tag for a product can easily create the purl based on that SWID tag; moreover, they can distribute the purl to their customers with the SWID tag. The reason they can do this is that, if the product name and version number do not change, neither the SWID tag nor the purl based on it will change either. Even more importantly, when the supplier creates a SWID tag, they will be able to feed it into a free SWID-to-purl generation tool, which will be guaranteed to produce a correctly formed purl according to the purl SWID type definition.

Of course, there will be some cases in which the SWID tag will be needed - for example, when a user who does not own or license a particular software product wants to compare it with similar products before making their purchase decision.

In fact, if an end user organization is comparing three similar products but only receives a SWID tag and a purl from one of the suppliers, that alone will send a powerful marketing message: "We want you to be secure. That is why we're willing to send you the information you need to secure our product, even if you haven't yet purchased a license from us."

Going one step further, a commercial software supplier may send an even more powerful message by making purls and SWID tags for their products available to the general public on their website (and through the Transparency Exchange API).

Including purls in CVE records for open source and commercial products

For a software user to be able to look up vulnerabilities for a commercial product in a vulnerability database using a purl identifier, three things need to happen:

1. Every CVE record created by the CNA needs to include a purl identifier, not just a textual description of the product.
2. The user needs to be able to discover information about the product that is needed to create a purl (the most important pieces of information are the product name and the version string). For an open source product, the user can find this information in the package manager where they obtained the product. For a commercial product, the user can find the information in the SWID tag distributed by the supplier.
3. The identifier that the user discovers needs to be the same one that the CNA uses whenever they report a vulnerability for the product in a CVE report. For an open source product, the CNA will utilize the same information from the package manager that the user does. For a commercial product, the CNA will use the same SWID tag as the user.

While the second two conditions are either being met now or will be met, the first condition is not being met at all today. This is because few if any CNAs include purls (regardless of purl type) in the CVE records they create today, even though the CVE 5.1 specification, which allows purls to be included in CVE records, came into effect in the spring of 2024[20].

Below are probably the main reasons why purls are still not being included in CVE records:

1. Currently, the primary target database for CVE records – along with CVE.org's own vulnerability database – is the NVD. Neither the NVD (and other databases that are derived from it) nor the CVE.org database (formerly known as "MITRE") can accommodate purls as identifiers today, only CPE names.
2. The CVE field in which a purl can be entered is currently a text field. This means it can't be used to perform any operation on the purl, such as validating that it is correctly formed.

[20] The SBOM Forum created the pull request for including purls in CVE records in 2022.

3. Most importantly, since most CNAs are commercial software developers, they usually only report vulnerabilities for commercial products. Since purl currently does not support most commercial products, the CNAs cannot add a purl to their CVE reports.
4. There has not been any guidance for CNAs on how to create and use purls, and there is currently no group encouraging them to do that.

The first three problems can be addressed by technical fixes, but the last is a human problem and needs to be addressed through human intervention. The Purl Expansion Working Group proposes to work with CVE.org and the CNAs to:

1. Identify policies and procedures that need to be in place for purls to be regularly included in CVE reports.
2. Identify changes that need to be made in the CVE format for purl to be a software identifier on a par with CPE (note that purls and CPE names should be able to coexist in CVE records).
3. Conduct a proof of concept exercise to confirm that, if a purl for an open source product distributed in a package manager like Maven Central is included in a CVE record and a user searches a vulnerability database using that purl, the product will be shown to be affected by that CVE.[21]

Because CVE.org is not just a professional but a social organization, the leaders of the Purl Expansion Working Group project will build ties with CVE.org staff and Board members. This will include:

A. Getting on the agenda to discuss the project in an upcoming CVE.org Board meeting. Art Manion, a member of the Board, has offered to get Tom Alrich invited to a Board meeting. Tom would be pleased to have one or two members of the purl community join him.
B. Meeting with one or more of the CVE.org Working Groups to discuss the advantages of using purl in general, as well as using the purl SWID type to identify commercial software. Members of the purl community would be welcome to join us at these meetings as well.
C. Attending regular meetings of one or more CVE Working Groups. The CVE Outreach and Communications Working Group (OCWG) is the only group that is open to the general

[21] This is different from the proof of concept to be conducted by the Purl Expansion Working Group. The purpose of that PoC (described earlier in this document) is to test whether a purl for a commercial product that uses the SWID type will successfully match the same purl that is included in a CVE record, if it is searched for in a vulnerability database.

Here, we are talking about a PoC for including a purl for an open source product in a CVE record and then confirming that a user searching a vulnerability database using the same purl will discover that record. This PoC is likely to successful, since one vulnerability database, OSS Index, matches literally millions of purls with CVE records every day just on behalf of one tool: OWASP Dependency Track. For that reason, this PoC should be conducted as soon as possible after the Purl Expansion Design and PoC project starts, since its outcome is likely to be successful.

On the other hand, there are several steps (described earlier in this document) that need to be taken before the commercial software PoC (using the purl SWID type) can even begin; these will likely put the commercial PoC 4-6 months behind the open source one. This is why it is better to conduct the two PoCs separately.

public, although some other groups, such as the Quality Working Group (QWG) and the Strategic Planning Working Group (SPWG), admit "non-CVE program members" on a case-by-case basis with annual review.

D. Holding discussions with individual CNAs or groups of CNAs regarding integration of purls, including SWID-type purls when available, into CVE records. Bruce Lowenthal of Oracle and Pete Allor of Red Hat, both active CNAs and active members of the SBOM Forum, may be able to help facilitate these discussions.

It is important to start the process of getting CNAs to include purls for open source software products – and later, purls for commercial products utilizing the SWID type – into CVE reports as soon as possible. Therefore, as soon as the Purl Expansion Design and Proof of Concept project can start work, we will start building ties with CVE.org.

Purls for software in app stores

Steve Springett and Tony Turner have both pointed out that, in the world of commercial 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 app stores, 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[22] for a variety of software products from multiple suppliers. 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 the 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 request for the new type (each store will require its own type, just as each package manager requires its own type). This work will mostly need to be performed by the staff of the app store itself.[23]

Soon, it would be good for someone in the purl community to reach out to app stores to promote the idea of their supporting purl. The Purl Expansion Working Group will be willing to support that

[22] 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.

[23] Tony Turner created a proof of concept purl type definition for the Apple Store (or alternatively for each of the five stores that comprise the Apple Store). Steve Springett submitted it to Apple, and they have expressed interest in it; however, they have yet to indicate a next step.

effort by testing use of the new purl types, using the same test environment that we create to test the SWID type.

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.

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 many 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, and any versions in between them, 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".

However, 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 due to 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).

The Purl Expansion Design and Proof of Concept project

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 widespread realization that CPE is not a reliable source of vulnerability data.

In fact, searching for a particular product/version in the NVD now is likely to turn up less than one half of the vulnerabilities identified 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 backlog 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 commercial software.

Because the great majority of medium-to-large organizations utilize mostly commercial software to run their operations, they need to be able to locate vulnerabilities found in those products. That is why purl support for identifying commercial software is essential. The OWASP SBOM Forum believes the best way to expand purl support to commercial software is by implementing the methodology based on SWID tags and use of the SWID purl type, which has been described above[24]. However, we are open to suggestions for better ways to accomplish this goal.

As the first step in achieving this goal, the SBOM Forum proposes a Design and Proof of Concept project executed by a Purl Expansion Working Group. This group will be open to all parties with an interest in expanding purl use to commercial software. We propose that the group meet virtually once every two weeks; currently, meetings are scheduled for every other Tuesday at 11 AM Eastern Time, which is the most convenient for participants across North America, Europe and Israel.[25]

A preliminary plan for this project is available here. Your comments or suggested changes are welcome. Note that as much work as possible – including developing and editing documents – will

[24] We also believe that creating purl types for app stores provides another good way to expand purl to commercial software products, although we do not ourselves have the bandwidth necessary to "evangelize" the app stores about this idea.

[25] Of course, we would like to be able to accommodate participants in Asia and Australia as well. However, there is almost no overlap in working hours between the two hemispheres.

be conducted online between meetings, and meeting notes will be published. Thus, participants who are not available to attend any of the meetings will still be able to participate in the project.

Tasks for this working group will include the following. The complete project plan can be viewed here.

1. Developing and documenting procedures for using SWID tags as the authoritative data source for purls to identify commercial software products in vulnerability databases.
2. Conducting a "tabletop exercise" in which multiple software suppliers, as well as vulnerability management service providers, test the current purl SWID type specification to determine whether purls created using that specification can successfully identify commercial software products in practice. If deficiencies are found, the exercise team will draft a pull request to change the purl SWID type definition to address the deficiencies (perhaps by adding fields).
3. Working with CNAs and CVE.org to develop policies and procedures to enable use of purls in CVE records created using the CVE 5.1 format.
4. Conducting a proof of concept exercise, in which software suppliers, CNAs, vulnerability management service providers and end users will test the entire process of: a) creating SWID tags and purls for commercial software products, b) reporting vulnerabilities for products using purls, and c) utilizing purls based on SWID tags to search for commercial software products in vulnerability databases.
5. Documenting agreed-upon procedures and lessons learned from the Proof of Concept.
6. Developing a high-level plan for rollout of production and use of purls for commercial products, based on SWID tags (the rollout will take place in a subsequent project with separate funding).

We hope you will join our effort!

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 (we estimate the Purl Expansion Design and Proof of Concept project will require about $100,000, although we will just need a fraction of that amount to commence work on the project).

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.