Abstract

End-users of products that contain software are using SBOM ("Software Bill of Materials") data to gain insight into the potential risks from any third-party software components that are incorporated in that product. (For the purposes of this paper a "product" can have several meanings including a physical form such as a device, a binary software form, a source code software form, or referring to a larger system of systems). Specifically, users want to know to what extent they are affected by known vulnerabilities, which includes vulnerabilities in upstream software components.

SBOMs provide this insight into the composition of the product, but they do so at a high level that does not convey the extent to which a known vulnerability can be exploited in the product. This lack of "exploitability" results in many "false positives" being represented in the SBOM data and thus obscuring the high risk exploitable vulnerabilities.

Software components may contain vulnerabilities, yet an attacker may not be able to exploit these vulnerabilities in the product due to the manner of utilization of the component or the configuration of the product.

In strict terms, for use in this discussion, a vulnerability is a set of conditions or behaviors that have a security impact, typically violating some security policy, possibly an implicit one (e.g., I do not want my software to run arbitrary code at someone else’s behest). Vulnerabilities exist in or affect components. Vulnerabilities by their nature are exploitable, however a vulnerability can, in theory, exist in a component, and the vulnerability may not be practically exposed or exploitable. The goal of "VEX" is to convey this degree of exposure or exploitability, particularly in downstream components.
Introduction

As vulnerabilities in products become more apparent through mechanisms such as Software Bills of Materials (SBOMs) and software composition analysis (SCA) tools, this paper explores when a producer desires to inform the consumer of the impact of a given vulnerability upon a product. This document is meant to give guidance on what interfaces and information elements are necessary as part of the technical solution to describing the state of potential vulnerabilities in a product.

Commented [9]: suggest change to "work" - a "work" is representative of a component, and SBOM VEX doesn't generally represent the final product, it represents the "work" of the product; and multiple items that create the 'product'.

Commented [10]: ahh well that is not always the case. Bruce convinced me there is a valid use case where the VEX is product centric. Also we should wait for the glossary to harmonize terminology.

Commented [11]: A glossary is high priority

Commented [12]: suggest change to "which"

Commented [13]: suggest "work"
Situation

Increasingly products and their associated systems are making use of a wide assortment of both open source and commercial packages. These systems may incorporate software that is being utilized for purposes the original upstream authors never considered.

When a vulnerability is discovered in a component, it is eventually announced through a well-established CERT process, using the common vulnerability & exposure (CVE®) framework.

While a CVE assigned vulnerability is present in the original implementation, that does not necessarily mean that it is exploitable or even present in final products or systems that contain the software component. This may be the case because of several different situations:

- The vulnerable software is not executed
- The vulnerable code has been excluded from the executable due to conditional compile statements
- Existing security controls make it impossible to exploit
- The vulnerable code has been fixed by a downstream developer.

If history can serve as prologue, then only a very small percentage of published vulnerabilities can be exploited. Despite such analysis, raw percentages or averages are not necessarily useful risk measurements. A small percent of a large number may still be a meaningful number. Also, any single vulnerability can have a high potential impact and pose a significant risk. Many vulnerability management approaches aim to avoid false negatives (i.e. choosing not to respond to a vulnerability that turns out to be exploitable).

Therefore, the results of a vulnerability scan of an SBOM may result in many unexploitable vulnerabilities. This could obscure the risk of exploitable vulnerabilities. The VEX is an additional source of information from the Downstream Developer to further inform the End User and increase the "signal to noise" ratio of such an automated scan to provide useful and actionable intelligence.

If a developer is aware of the vulnerability, then they can choose to fix the vulnerability, request an upstream fix, or remove the impacted component from their product. If end customers are aware of the vulnerability, they can mitigate it through upgrades, replacements, remove the product from service, or other means.

At any point in the supply chain there is the potential for someone to create a mitigation that may not improve the security posture of the product and could introduce new vulnerabilities. Accurately knowing whether a particular product or system is vulnerable is critical to that evaluation.

---

1 "CVE" is a trademark of MITRE Corporation.
2 http://www.rsaconference.com/industry-topics/presentation/how-understanding-risk-is-changing-for-open-source-components
3 https://www.sonatype.com/2020ssc
Ecosystem Participants

Art to fix all the wrong terminology I’m using.

CERT / Vulnerability Database

Our story begins when a vulnerability has been reported to a CERT somewhere around the world for some software. At that time, a CVE is created and a CVE-ID is assigned by the supplier. At some later point, the information is shared with the public along with a reference to some remedial advice.

As part of the CVE information, a package name is identified. This package name may relate to a package name found in an SBOM or otherwise determined to be in a product via SCA analysis.

*Cloud server image courtesy of Josh Twist (http://www.thejoyofcode.com/Cloud_Artwork.aspx)

Commented [47]: I’m going to need you to come in on Saturday... and Sunday... mmm... yeaahhh, ok? Good.

Commented [48]: IMO we can use this as an ‘example’ but shouldn’t ‘bind this’ to SBOM intrinsically. Yet again we delve into use case specifics for existing processes that are not globally used by all. IMO SBOM VEXes should also exist independently of centralized sources and be able to stand independently of any centralized governing body of controls and systems, but, should be able to leverage such capabilities for existing open frameworks as the developer/company decides to do so - their own systems, other countries (CNVD), or CERT/VUDB, etc. at their discretion. This is why the ‘key id’ for the VEX itself is critical; because it provides its own direct identifier and is signed / hashed to ensure authenticity (to a reasonable degree).

Commented [49]: I think the "key id" is the (supplier, product name, version, hash) we defined in the framing document.

Commented [50]: IMO we can use this as an ‘example’ but shouldn’t ‘bind this’ to SBOM intrinsically. Yet again we delve into use case specifics for existing processes that are not globally used by all. IMO SBOM VEXes should also exist independently of centralized sources and be able to stand independently of any centralized governing body of controls and systems, but, should be able to leverage such capabilities for existing open frameworks as the developer/company decides to do so - their own systems, other countries (CNVD), or CERT/VUDB, etc. at their discretion. This is why the ‘key id’ for the VEX itself is critical; because it provides its own direct identifier and is signed / hashed to ensure authenticity (to a reasonable degree).

Commented [48]: IMO we can use this as an ‘example’ but shouldn’t ‘bind this’ to SBOM intrinsically. Yet again we delve into use case specifics for existing processes that are not globally used by all. IMO SBOM VEXes should also exist independently of centralized sources and be able to stand independently of any centralized governing body of controls and systems, but, should be able to leverage such capabilities for existing open frameworks as the developer/company decides to do so - their own systems, other countries (CNVD), or CERT/VUDB, etc. at their discretion. This is why the ‘key id’ for the VEX itself is critical; because it provides its own direct identifier and is signed / hashed to ensure authenticity (to a reasonable degree).

Commented [49]: I think the "key id" is the (supplier, product name, version, hash) we defined in the framing document.

Commented [50]: IMO we can use this as an ‘example’ but shouldn’t ‘bind this’ to SBOM intrinsically. Yet again we delve into use case specifics for existing processes that are not globally used by all. IMO SBOM VEXes should also exist independently of centralized sources and be able to stand independently of any centralized governing body of controls and systems, but, should be able to leverage such capabilities for existing open frameworks as the developer/company decides to do so - their own systems, other countries (CNVD), or CERT/VUDB, etc. at their discretion. This is why the ‘key id’ for the VEX itself is critical; because it provides its own direct identifier and is signed / hashed to ensure authenticity (to a reasonable degree).
Upstream Developers and Suppliers

The upstream developer typically controls a component that is used by others. This party will generally create CVEs (although others may perform this as well) and assign CVE-IDs. This upstream developer may release software patches or new versions intending to correct the vulnerability and may update the CVE based upon that information. Upstream developers may be under contract restrictions to support downstream developers, or they may provide the software on an “as is” basis, as would typically be the case with open source components.

Downstream Developers

These developers make use of software components developed and/or maintained by others (i.e. “upstream developers” or “suppliers”), and deliver their products to end-users or additional downstream developers. Downstream developers:

- May or may not be required to make statements as to whether or not a particular version of their product contains a vulnerability.
- May want to issue patches for specific versions of their product.
- May be required by customers to make claims about the security posture of their product. Developers will want to avoid making false claims.
- May want to be able to communicate that they are not vulnerable to a particular CVE across a range of products.
- Will want to assure the attestation of any statements or claims.

Developer Tooling

Developer tooling is used to build and package products. As part of this activity, tools may be able to identify that a particular component has a vulnerability, based on a CVE related to an included package. Tightly integrated developer tooling, such as CI/CD, can produce additional information to better identify products which are or are not impacted by a CVE and which ones have been corrected.

End Users

End users are the people who use a finished product. End users do not further distribute a product to another user. Their interests are that the product runs securely in the end user’s environment, cannot maliciously be caused to malfunction, and cannot be caused to attack others. (e.g., cannot be exploited).

- End users have varying degrees of risk tolerance, based on where products are utilized, how they are used, and what harm can be caused by an exploit to them or others.
- Some environments such as exploit testing labs may even be risk-seeking, while others such as critical infrastructure providers will be highly risk-averse.
- End users may want a developer’s justification (from a list of potential justifications defined by this working group) for an exploitability posture declaration.
End users may be required by regulators to receive vulnerability information and to maintain their products, or they may desire to do so on their own. End users will develop different levels of trust in the validity of Downstream developers’ declarations.

End User Tooling
End user tooling may include network and asset management systems, firewalls, mobile device managers, cloud orchestration systems, and many others. Tooling in this context can be used to identify which vulnerabilities are present on which devices within an environment, to report and remediate the risk. For this to work, the tooling must be able to accurately state whether a vulnerability is present, exploitable, and harmful to the end user.

Value Chain

- Delivers components for use by others.
- Creates CVEs
- May create SBOMs

- Consumes and utilizes components created by others.
- May deliver a product to the end user or another downstream developer
- Creates CVEs
- Consumes upstream CVEs
- May create SBOMs
- May indicate Product vulnerability to upstream vulnerabilities
- Wants to minimize end user calls for non-affected Products

- Makes use of the Product
- Consumes CVE, SBOM, and VEX information
- Desire to understand their risk exposure
- Desire to minimize false positive vulnerabilities.
this is ambiguous - open and closed sourced, 1st and 3rd party, commercial and non-commercial is the matrix. Trimming down is probably better then stepping into this matrix at all. suggest "software components" and leaving it be.

We really need a glossary. In any case, there are open source commercial products such as MySQL and Java (yeah, I know there are parts of the cryptographic libraries that are not open source because of licensing issues but nearly all the code is open source)

suggest "source, object code, and related" - a subtle difference exists between code and software - code is the work, software is the product.

suggest "other than those originally intended" and leaving it outside the focus of OSS TPC or CS TPC.

suggest: "announced," strikeout remainder. Why, I believe not everyone uses CVE and not every country is CVE'd and this is a very NA/US centric view. IMO it's just best to leave it to the dev / co, to decide how to do their announcement and not pigeon hole them. Further, CVE may change in the future and tagging it now with a (R) leaves the door open for future work which is unnecessary. Let's close the door now.

We need a glossary. The group acknowledges there are multiple entities that assign vulnerability identifiers but that, for now, we are using CVE as a short hand for all of them and are not excluding them from SBOMs. We could replace CVE with terms like "vulnerability identifier" but some would confuse that with CWEs, etc. I think CVE is best for now in these kinds of documents but needs to be cleaned up an an actual standard.

We need a glossary (I'll stop saying this now). CVE, or things that act like CVEs is meant, not CWEs.

Software is better because it is more encompassing. A broken table or config file can result in a vulnerability even though few would call it code or object.

suggest adding ", including code which is no longer maintained by the upstream provider."

I think a different statement would be better because suppliers will not own up to this and this point does not need to be made to validate the need for VEX. We can say that many suppliers are unaware of published vulnerabilities in 3rd party components they include (e.g. because of lack of pervasive use of SBOMs)

suggest strikeout - upstream may provide vex, and ingested by downstream natively; or modified, or created by downstream.
suggest postpend "and upstream the fix if appropriate,"

suggest "or remove, reconfigure or disable"

suggest "through hotfixes, patches, workarounds, upgrades,"

suggest ", off product hardening and protection e.g. firewall filters"

suggest insert new paragraph line and title "supply chain" title of some sort