Response to Software Bill of Materials Elements and Considerations, Concerns and Questions

Alexander M. Hoole
Fortify Software Security Research
Micro Focus International PLC
Santa Clara, USA
alexander.hoole@microfocus.com

Michael F. Angelo, CISSP CRISC CDPSE
CyberRes Chief Security Architect
Micro Focus International PLC
Houston, USA
michael.angelo@microfocus.com

Luther Martin
Voltage
Micro Focus International PLC
Santa Clara, USA
luther.martin@microfocus.com

***Abstract*— Within the bounds of the request for comments several key concerns and questions must be raised. Specifically, issues around enhanced, or increased, exposure beyond legitimate users; potential reductions to the barriers of entry for attackers conducting reconnaissance and weaponization; and impact upon intellectual property and licensing. Furthermore, understanding the inherent risks associated with the potential for weaponization via the analysis of nested components embedded within the transitive closure of dependencies contained within, or derived from, a Software Bill of Materials (SBOM). Finally, it is not only the presence of a single SBOM, for a single piece of software, that is at risk; rather the analysis of many SBOMs, in concert, provides the opportunity to identify shared nested dependencies among the software used by many targets. Thus, enabling attackers to find common components that can be researched for new 0Day vulnerabilities yielding a larger field of targets. Unlike disclosed vulnerabilities which can be tracked in association to SBOMs, 0Day vulnerabilities remain undisclosed and fully weaponized.**

I. Introduction

Thank you for the opportunity to respond to the proposal for the Software Build of Materials (SBOM). We are concerned that the claimed values will not achieve the desired effect. We believe the claims of enhanced security through public knowledge of lower-level vulnerabilities and compliance with licensing, while on the surface sound great, will in reality not be met. While the RFC does not explicitly specify that SBOMs could/should be made public, the debate of the potential advantages and disadvantages to the security of software systems is extremely important. In this paper, we will first present a contextual use case that will be illustrative as we describe both positive and negative side effects of SBOM implementations for enterprise/mission critical software (such as increased transparency and reduced effort towards weaponization, respectively).

II. Contextual Example

To begin, we would contend that the SBOM, is not a security item itself. It is a ledger depicting the dependencies (perhaps transitive) contained within a software product. The questions then become ' what is the ledger used for today?', and 'what could it be used for in the future?'.
Consider the following Ledger depicting a product component dependency scenario (Figure 1).

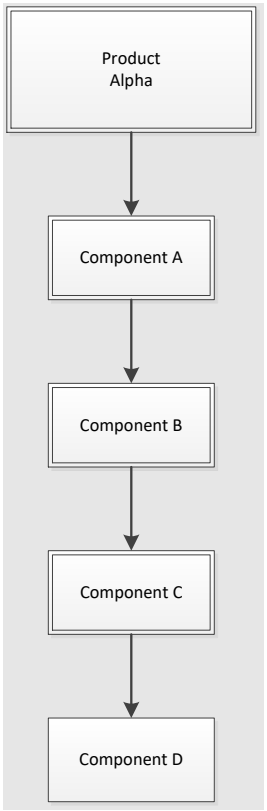

Figure 1: Dependency Tree – Transitive closure example.

Figure 1, shows a product with simple dependencies in a component chain where we have,

- product Alpha using component A
- component A using component B
- component B using component C
- component C using component D.

Note for the purposes of this response A, B, C and D are all external components (open source or commercial).

Consider the following transitive ledger walk through shown in Figure 2:

- Component D detects a vulnerability and releases an update. The owner of Component D follows

responsible disclosure practices and files a CVE with MITRE.

- Component C notices component D's CVE.
- Component C evaluates the vulnerability (figure 3) addressed in Component D. If component C determines the issue is not relevant nothing externally is done. If component C is vulnerable, the developers will update Component C and file a CVE update (saying the component was susceptible to attack as described in Component D).
- Components B, A, and Product Alpha would follow a similar process as each of the dependent subcomponents published their vulnerability. If a subcomponent was not vulnerable, then the customer would never be aware of the potential vulnerability and would not need to do anything. If the overall end state is reached, it implies that there are no known vulnerabilities present in any of the components within the product.

It is important to remember that the time to handle the vulnerability, and the delay in the time to start handling the vulnerability, will increase for each element in the chain as the complexity of the consuming element increases. For the following scenario, each component analysis is performed as demonstrated in Figure 3. If the scenario assumes a transitive ledger SBOM, as shown in Figure 2, the following timing could apply[1]:

- [D] announces a vulnerability.
- [C] responds to the vulnerability (analyze, incorporate the mitigation provided by [D], test and package the update) which takes 4 days due to the size and nature of component [C] and the amount of time it takes to become aware of a CVE disclosure from [D][2]. For example, it may take one day to become aware of the CVE and three days to provide a working patch.
- [B] would detect the vulnerability in [D]. [B] would have to wait for [C]'s mitigation and announcement of patch. Due to the complexity of [B], it may take 7 days, after becoming aware (in addition to the four days required by [C]), and to provide an update which integrates [C]'s patch into [B].
- [A] would also detect the vulnerability in [D]; But would have to wait for [B]'s integration and announcement of [C]'s patch (which is blocked by having to wait for [C]'s mitigation). Due to the complexity of [A], it may take 14 days to provide an update.
- Finally [Alpha] would also detect the vulnerability in [D]; But would have to wait for all components to provide a mitigation [A-D]. Due to the complexity of [Alpha], it may take 28 days to analyze, incorporate the mitigation provided by [A], test and package the update.

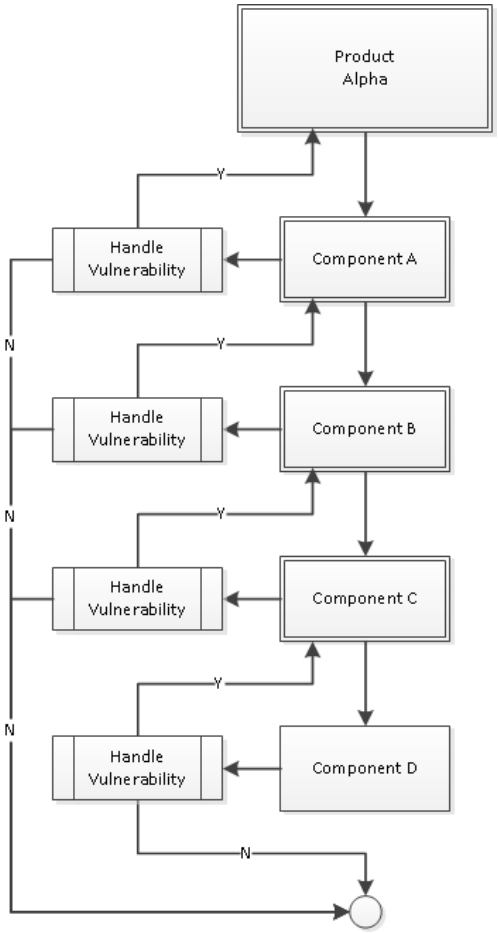

Figure 2: Vulnerability Mitigation Flow - depicting a vulnerability and mitigation flow back up to the product.

In this case, a transitive SBOM would create an exposure time for Alpha customers of 53 days assuming no issues were found in the component adoption and the customer deployed the update as soon as it was released[3]. This also assumes the customer could deploy immediately and they would not need to perform any infrastructure analysis or internal testing of their own.

[1] These times are qualitative estimates intended to be representative of what is observed in industry.

[2] A CVE disclosure from [D] may take several days before it is made available to consumers.

[3] As an example, consider the statistics for software vendors to update an android application with known vulnerabilities. https://dl.acm.org/doi/abs/10.1145/3372297.3423346. In this case, android made an update available witch took 24 days for the manufacturers and suppliers to deploy. It then took an additional 11 days for users to accept the update. Unfortunately, these statistics do not address individual component updates times which lead to the update roll-out. Therefore, in our scenarios we are ignoring the supplier / manufacturer timeframe and are assuming it is based on complexity of code. However, if we followed these time calculations, it would take an additional 11 days for each customer to deploy the product.

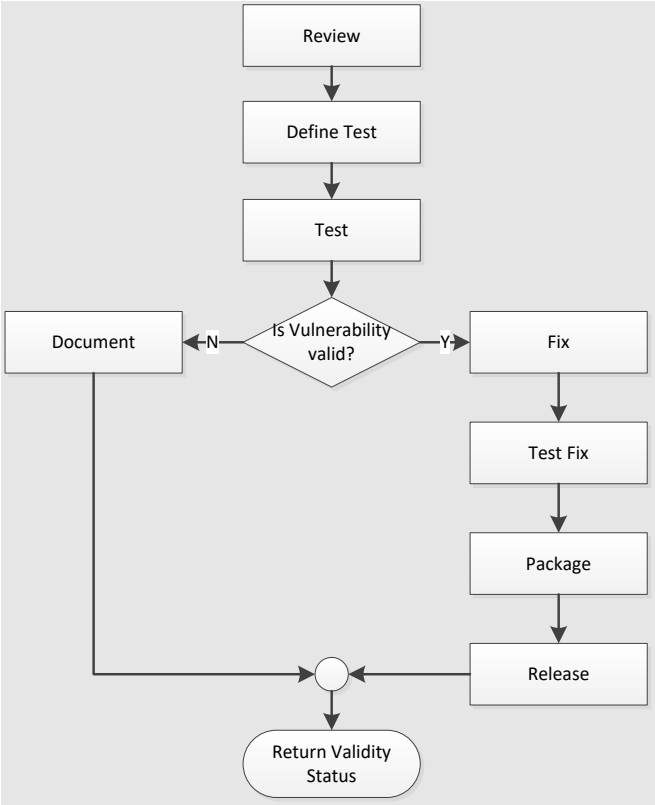

Figure 3:Handling Vulnerability - depicting a typical vulnerability handling process.

During this time hackers could attempt to target [Alpha] users and create an exploit. This gets worse when we consider that element [A] may be used by more than just [Alpha].

In Figure 3, the Documentation path could be a shortcut in terms of steps to be completed to mitigate risk. However, given the nature of customers and the potential for issues related to incomplete risk assessment of a disclosed CVE or the amount of work to complete an extensive risk assessment, the shortened path for documenting a non-impacting response of Figure 3 will be unlikely. When there are many reported risks for a piece of software, mechanisms used to evaluate the reachability or exploitability of a suspected risk could be used to prioritize work. It is not recommended to use evidence of such tools as documented proof that a risk is not reachable. It is often recommended to "simply" apply the fix.

III. Concerns Regarding SBOM

(i.e., Product Alpha only reports Component A)

The SBOM for a software artifact produced within the developing organization improves visibility and transparency in order to accomplish goals such as understanding architecture, security risks, and licensing. Are there also positive and negative side effects of having an SBOM?

A. *Side effects of a disclosed SBOM*

There are many possible side effects of the proposed SBOM. Some positive, such as improved understanding and transparency of the composition of software. Some negative, such lowering the barrier to weaponizing an attack against specific targets. First, we discuss several ***negative side effects***.

1) ***Weaponization***

Data on time to weaponize vulnerabilities is scarce, some entities claim as little as 72 hours[4], while others claim 27% of all vulnerabilities are weaponized within one month[5].

Analysis and subsequent documentation, as represented in Figures 1-3, could reduce the impact, exposure, and consequences of vulnerabilities in a dependency tree[6]. Thus, all products would have to adopt the vulnerability mitigation. Failure to do so would result in a deluge of impact requests and justifications.

In theory when component [A], from Figure 1, filed a vulnerability disclosure this would provide an attacker 28 days to create an exploit against product [Alpha]. This having been said, if everyone filed an SBOM, a transitive closure dependency tree would be trivial to generate (provided the attacker has access to the targeted software) and subsequent CVE analysis would give the attacker multiple 0Day opportunities with potentially lengthy attack windows, 53 days in the case of [Alpha], to create and deploy an exploit prior to a product update being released. Again, this is assuming that the customer applied the update as soon as it was released. Even if we factor in time, to weaponize, the attacker would have a significant period to carry out attacks which would leave the consumer exposed.

Furthermove, consider a situation where all software had published SBOMs, then malicious reconnissence could be conducted to identify a set of targets to be attacked based upon software they all use. Mining of the SBOMs of software that are common to the targets for shared dependencies would expose specific software to research for novel vulnerabilities. Once the dependencies are identified, any 0day vulnerabilities found in those shared dependencies would have an "open window" for attack until someone detects a breach or an independent non-malicious actor responsibly discloses the vulnerability.

We could not find any statistics for the scenario in which a deployed mitigation introduced secondary issues, we believe this will be a concern as entities rush to create and deploy mitigations.

---

[4] https://www.theregister.com/2021/04/06/sap_patch_attacks

[5] https://www.fireeye.com/blog/threat-research/2020/04/time-between-disclosure-patch-release-and-vulnerability-exploitation.html

[6] This covers primary and secondary dependencies as well as the transitive closure dependency trees.

2) ***Reduced barrier to weaponization***

Reducing the competency scale required for reconnaissance by attackers to determine components that are in scope for attacking a particular target. Consider the following attacker capability scale:

[Low Capability]

Kids
Students
College Students
Experts
Criminals
Nation State actors

[High Capability]

A published SBOM would move the bar towards the low capability on the above scale for weaponization of disclosed, and undisclosed vulnerabilities.

3) ***Remediation gaps***

For consumers, the ultimate remediation concern that needs to be addressed for an SBOM is that the transitive closure of depencies is free of exploitable known vulnerabilities. A software provider defines a remediation process for timely mitigation of risks, where a transitive dependency tree (n levels deep) contains a vulnerability at position n-1 and the consuming component (n-2) is a third-party component. There is a potential continuity gap in delivering remediation when componet (n-2) cannot deliver a software update in a timely fashion. Thus, the mitigation of the complete dependency tree has a temporal gap until other compoent providers complete their independent remediation steps.

4) ***Intellectual Property Exposure***

The SBOM for a proprietary product, coupled with the actual product, would enable the re-creation of that product by an individual knowledgable in the arts. This recreation would create challenges for companies in protecting their IP from theft and unauthorized derrivative products..

While there are multiple negative side effects related to weaponization of flaws found in dependencies, discussing some of the noted ***beneficial side effects*** is also warranted.

5) ***Licensing***

Another element that has been attributed to the SBOM is an enhancement to license enforcement. This may not be the case, as Product [Alpha] would not be aware of licensing agreements other than with a direct component (i.e. [A]). Explicitly product [Alpha] would not have any knowledge of the legal agreements that various parties hold within the ledger. Hence each party within the ledger would have to disclose all license agreements. These agreements are highly sensitive and would adversely affect an entity's ability to negotiate a contract.

6) ***Creation of temporary mitigations***

Customers can attempt to validate exposure to disclosed vulnerabilities contained within the transitive dependency tree and create temporary mitigations. Furthermore, customers can attempt to create a temporary mitigation plan for when vulnerabilities are disclosed against known dependencies within their transitive dependency tree (i.e. mitigations to be in play "until" and update is available).

7) ***Creation of internal database of shared components***

Customers can create a database of their applications which share common components across the set of all transitive dependency trees within their portfolio of applications. This can then be queried to identify which applications are potentially at risk to a newly disclosed vulnerability.

If we look at the value proposition to a defender, having a database containing the transitive closure of depedencies for each software artifact, it creates a situation in which they get to watch the attack develop, but cannot act on the vulnerable software (i.e. great visibility, but no empowerment to handle). Specifically, if the software flaw which is realized in the vulnerability resides in software that the defender does not have ownership/write permissions, they must wait until an update becomes available.

8) ***SBOM Access***

The question of who would or could receive the SBOM needs to also be considered.

If SBOMs are made available outside of the developer environment, we must also consider the imputed risk. Several levels of dissemination are conceivable, including the following:

- ***Origin only:***
  The SBOM materials are protected and utilized within the software company only. The SBOM is used to identify, evaluate and mitigate transitive issues within the product line.
- ***Selective:***
  The SBOM is disseminated based on a perceived need-to-know. This information could be used to provide procedural mitigations for exposures prior to vendor provided mitigations. While on the surface this practice sounds reasonable, it creates three issues.
  - *Selective SBOM disclosure in today's market creates issues for entities that receive the information and those that do not. Those that do not receive the SBOM are left at a disadvantage to those that do receive it.*
  - *Selective SBOM creates issues for companies deciding who can have access and who can't. If the SBOM was provided to a US government entity, then all non-US government entities could claim entitlement.*

The penalty for not giving access would be a closed market. This was discussed in the early1990's as part of the SkipJack and Clipper proposals.

- *Selective SBOM disclosure does not gawrantee they will not be made public or shared amongst other organizations.*

- ***Global:***
  The SBOM is published, and anyone can gain access to the materials. This bares the issues we have already discussed earlier in this document.

B. Activities enabled by SBOM

While there are concerns that need to be debated, we would be amiss to not raise some of the activities that are also enabled by the presence of an SBOM for software.

1) Consumer risk mitigation

- Procurement
- Lifetime maintenance
  - If risk – how long to get fixed?
- Customers can request flaw remediation and estimated response time (tolerance) for disclosed vulnerability.

CONCLUSIONS

1. SBOM tracking for internally developed software has obvious advantages
2. SBOM tracking of open-source software, particularly components, has obvious advantages
3. SBOM public disclosure of systems that are not publicly available has questionable advantages due to the stated risks, such as reduced cost of reconnaissance, associated with transparency of information to those who do not satisfy a need-to-know in accordance with the principal of least-privilege.
4. SBOM is not a security control, in the sense of enforcing or enabling confidentiality/integrity/availability. SBOM is a ledger system. Regarding security through/by obscurity, we cannot lose sight of why authentication and access control mechanisms were created in the beginning. Not everyone should have immediate access to sensitive information. This is not due to a lack of transparency, but rather an intentional security control to reduce risk. For example, elongating the time to successfully attack a new risk which exists *N*-levels deep in the transitive closure of dependencies for a given piece of software.
5. Sharing a SBOM with a specific external party based upon need-to-know can suffer the same challenges regarding information leakage as with enforcing Mandatory Access Control (MAC) and Discretionary Access Control (DAC). Specifically, an individual who has access to the information can leak it to additional parties via copy/paste, screen capture, or simply taking a photograph. The question now becomes, who should have access to which level of detail.
6. Sharing an SBOM with select external parties may create issues with users. This practice would leave some users exposed while enabling others to be protected. We have seen similar issues with security issue pre-announcement to selected users (I.e., not all at the same time).

MOVING FORWARD

While not all SBOM concerns are addressed in this RFC, we have attempted to make sure that the landscape has been well described. With this in mind, we believe that, perhaps an SBOM should go beyond a ledger system and consider the following:

- **Overall security of infrastructures:**
  While we are looking at products, to define secure software, we are not focusing on securing the system. Not only is the system critical, but so are the deployment and usage paradigms.
- **SBOM handling and usage requirements:**
  Exposure of SBOMs (to any entity) can create a targeting system for all bad actors. Perhaps a complement would be to look at certifying companies processes for review and handling of vulnerabilities in third-party components. This would be best facilitated by SBOMs usage internally. This raises the question of what should be shared:
  - Applications using SBOMs would not need to be disclosed.
  - Components or elements which are not able to be executed need complete SBOMs

Overall, SBOMs can provide many benefits to manufactures of software and through them their consumers. The points raised in this paper, particularly around weaponization and time-to-attack should hopefully encourage further debate.

ACKNOWLEDGMENT

We would like to thank NTIA for the opportunity to provide feedback on the Software Bill of Materials Elements and Considerations.