

Software Supply Chains and Provenance for Software Binaries

Position Paper for NIST meeting of 6/2-6/3/2021

Paul Anderson, VP of Engineering, GrammaTech, Inc.
paul@grammatech.com

Summary

As described in the President's executive order 14028 of May 12, 2021, sections 4(e)(ii, vi, and viii), there is a pressing need to improve the software supply chain with respect to software provenance.

Software binaries are often the only tangible artifact from which conclusions can be safely drawn about security risks. Rules and regulations to facilitate a secure software supply chain should encourage the use of tools that can scan software binaries to establish their origin and to determine if they contain known security vulnerabilities.

Introduction

Computers do not execute source code—they execute machine code instructions that are either compiled or interpreted from source code. As such, machine code binaries constitute the ground truth of the semantics of a program. Some vulnerabilities can only be identified by scanning the binary code, and in many cases the binary code is the only artifact that is available for examination. Consequently, securing the software supply chain should allow for analysis of binary code.

Unfortunately, binary analysis has been hampered not only by significant technical challenges, but also because software licenses typically forbid reverse engineering of binaries to protect the intellectual property of the authors. However, techniques are now emerging that have overcome many of the technical difficulties and that do not impinge significantly on the intellectual property rights of the producer.

N-day Vulnerability Detection on Binaries

A best practice for understanding risks in the supply chain is to use software composition analysis tools to identify the components of a software product in the form of a software bill of materials (SBOM), and to then associate those elements with known (i.e. N-day) vulnerabilities. The use of these tools is now commonplace in software development organizations because they work well on software source code and on packages that are copied verbatim from standard sources such as package managers.

However, much off-the-shelf software is delivered to customers in the form of software binaries that contain machine code intended to execute natively on the target platform. Those binaries are most often created by a process that compiles source code and links the resulting object code into executables and shared libraries (e.g., DLLs). The compilation process serves to obfuscate the identity of the original components.

For example, the open source software library *libpng*, for handling PNG images, is very widely used. It is written in C, so is prone to the multitude of risks that language entails, and thus has had many CVEs

reported against it; a search of the CVE database yields 74 entries. CVE-2015-8472 identifies several buffer overflow defects in versions 1.6.x before 1.6.20. If a software supplier compiles vulnerable version 1.6.7 from source code, and links the resulting object code into the delivered executable, the exact set of machine instructions that end up in that executable is very sensitive to the specific compiler used as well as the level of optimization chosen. Therefore, someone who has only that binary to inspect, and no knowledge of how it was created, will have difficulty identifying that it contains libpng 1.6.7 and is consequently vulnerable to CVE-2015-8472. Techniques that work well for SCA for source code are designed either to scan human-readable text (i.e, the source code), or unchanged sequences of bytes (e.g, signature matching), but compiled code is neither legible nor fixed so those techniques are ineffective at identifying such compiled-in libraries.

In an ideal world, the software supplier will have generated an SBOM at the time the library was compiled, which the user of the software can then use to look up any reported vulnerabilities. However, it may be a long time before software producers are routinely supplying SBOMs for the software they deliver, and in any case many organizations will be uncomfortable with relying on SBOMs only, because doing so requires putting a lot of trust in the supplier. Even if the supplier is acting in good faith, human error or tool shortcomings may yield imprecise SBOMs.

Recent advances in binary analysis and machine learning have made it possible to identify components compiled into executables Tools use a variety of techniques that each yield pieces of evidence derived from different properties of the executable. The most straightforward techniques derive evidence based on easy to extract properties such as the set of strings found within the file. Strings are often very idiosyncratic; for example, diagnostic messages are often unique to a component, so they can constitute strong evidence of its presence. More sophisticated techniques work by computing features of the set of machine instructions found within each function, then computing similarity measures against a database of known components. The combination of these pieces of evidence is used to compute a likelihood score, which if it exceeds a threshold, yields a conclusion that a component is present. Under the hood, tools use machine learning techniques to select the set of features that carry most signal about where the component originated.

These techniques make no attempt to determine what the code under analysis is intended to do, nor how it is either designed or implemented. They may decode instructions, but they neither disassemble nor decompile. Consequently, these techniques do not constitute reverse engineering of the code. Some of the accumulated evidence may be presented to the user, but not in the form of code.

Therefore binary analysis to identify provenance and detect the presence of known vulnerabilities should be a key part of any comprehensive supply chain security analysis.

Conclusion

Software binaries are currently a key weak link in the security of the software supply chain, so understanding their provenance and the provenance of all the code that is compiled into them is critical. Despite much needed efforts to institutionalize techniques such as SBOM production that will capture this provenance, adoption is likely to be slow and such passive approaches may not be acceptable for users of high-criticality software. Tools to analyze binaries to derive provenance are becoming available and their use should be encouraged. This can be done in a way that is considerate of the reasonable intellectual property concerns of the supplier.