Software Provenance

Roy E. Wilson

IN A RECENT MAJOR CRIMINAL TRIAL, THE PROSecution was introducing a critical item of evidence to the court to support its case. When the evidence was introduced, the defense objected on two grounds. The first objection was that the item was "planted" by law enforcement personnel and the second was that there was a violation of chain of custody procedures for handling the evidence. The prosecution was embarrassed because it could not positively substantiate the origin of the evidence, nor could prove that rigorous chain of custody rules were followed. The judge ruled that the evidence must be thrown out and the case subsequently was dismissed. Because the prosecution did not ensure that the provenance of this critical evidence—that is, origin and chain of custody—the case was lost and the lead prosecuting attorney was dismissed from her position. More on this later.

The above story is a parable, designed to illustrate the importance of provenance for any critical item. Just as in the legal field, provenance applies to items of software. One of the major issues facing the Department of Defense (DoD) is the issue of software provenance, or specifically the lack thereof. The DoD is examining several measures to address the issue of software provenance. Measures already put in place include rules to protect against buying software that has Russian or Chinese provenance. This includes the new Federal Acquisition Regulation (FAR) Clause 52.204-23, "Prohibition on Contracting for Hardware, Software, and Services Developed or Provided by Kaspersky Lab and Other Covered Entities."

What is provenance? The word provenance (alternatively spelled provenience) comes from the Latin provenire, meaning "to come forth, originate." Thus, software provenance refers to the verifiable information regarding the

Wilson *is professor of Cybersecurity at the Defense Acquisition University and a retired U.S. Air Force (USAF) officer with more than 35 years of experience in aviation systems engineering for the USAF and the U.S. Navy.*

origin of a software configuration item; the development facility or location where it was developed, who wrote the code, what secure coding standards were applied, what static and dynamic code analysis methods for security assessment were used, as well as its chain of custody; that is, who possessed the software configuration item since its development. An equally important part of provenance is to fully control how software is managed throughout sustainment to ensure that the software's security posture and provenance is unchanged. The requirement for software provenance applies equally to new software developed specifically for a DoD purpose and to Commercial Off The Shelf (COTS) or other Non-Developmental Item (NDI) software.

Conversely, an unprovenanced software configuration item is software whose origin and chain of custody is unknown or partially unknown. The risk to the DoD is that unprovenanced software configuration items may have been counterfeited or altered to include malware or weaknesses that are designed to negatively impact the DoD mission associated with the software capability. Use of unprovenanced software for critical functions in a DoD system introduces an elevated risk level and potential consequences which may be unacceptable to accomplishment of the systems' mission.

The concept of software provenance is simple to grasp—i.e., know the origin and chain of custody and you know the software provenance. It is much more difficult to put into practice. Figure 1 provides a simple illustration on building an assurance case for software provenance.

Details for each element in the Figure 1 logic diagram follow.

The Development Facility

Commercially developed software is inherently "unprovenanced" in most cases because the development facility is protected only as far as the company desires to protect any intellectual property. Software developed by overseas firms or software purchased though 3rd party distributors carries the highest degree of risk.

Facilities used for DoD-funded software development must provide for physical and logical protection of code development platforms and networks. Software development in classified facilities is performed on systems inspected and authorized by the Defense Security Service. These development facilities are normally "air gapped" from other networks and access is restricted to cleared personnel. Unclassified software developed for the DoD is afforded facility protections as required by the contract through imposition of appropriate Defense Federal Acquisition Regulation Supplement (DFARS) security clauses and National Institute of Standards and Technology standards.

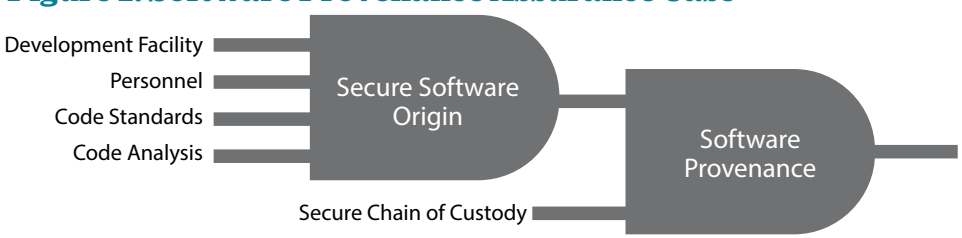
Personnel

The insider threat during software development is perhaps the most insidious and may have the highest potential for catastrophic impacts. To ensure software provenance, software development personnel must pass security background checks commensurate with the classification level and/or criticality of the software project they are working on. Personnel must be trained on cybersecurity, threats to software, secure code development, software security testing and other areas of software assurance prior to working on any code development. Cybersecurity refresher training above and beyond annual cybersecurity awareness materials are essential. The refresher training should include deep dive study into Common Weakness Enumeration (CWE), Common Vulnerabilities and Exposures (CVE), Common Attack Pattern Enumeration and Classification (CAPEC) and other available resources with direct applicability to secure code development. Personnel are encouraged to visit the DoD Joint Federated Assurance Center (JFAC) at https://jfac.navy.mil/ for further training and support in software and hardware assurance topics.

Secure Code Standards

No amount of analysis and patching can imbue software with high levels of security or other important properties. Such properties must be designed in and built in (baked in). Good choice of language, platform and coding discipline are worth orders of magnitude more than reactive efforts. Secure coding standards have been published for most common programming languages. One example of a resource where one may find secure code standards is the Carnegie Melon University Software Engineering Institute (SEI). The SEI helps coordinate the development of coding standards for commonly used programming languages such as C, C++, Java and Perl, and the Android platform. These standards are developed through a broad-based

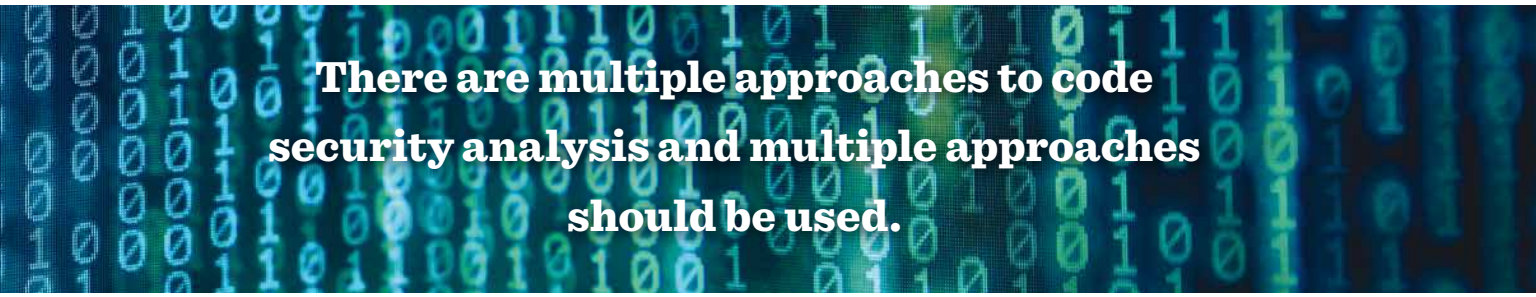
Figure 1. Software Provenance Assurance Case



Source: The author.

community effort by members of the software development and software security communities.

Use of secure code standards can reduce the likelihood of an attacker finding a vulnerability. For example, one common attack technique is to use a buffer overflow attack. From the MITRE CAPEC database, we find that most buffer attacks involve retrieving or providing more input than can be stored in the allocated buffer, resulting in the reading or overwriting of other unintended program memory. So, what does prevention of a buffer overflow "buy" you? The European Space Agency (ESA) lost a $7 billion rocket in Ariane 5's first test flight (Ariane 5 Flight 501) on June 4, 1996. The rocket self-destructed 37 seconds after launch due to a simple buffer overflow condition. With attention to the secure code practice of buffer overflow prevention, the ESA would have saved a $7 billion rocket and better kept up with the United States in the space race. That would have been a very good return on investment.



Code Security Analysis

There are multiple approaches to code security analysis and multiple approaches should be used. Up front and early, code should be inspected for CVE and CWE identified vulnerabilities and weaknesses that should be corrected. Table top reviews, peer reviews, team code walk-throughs and automated static testing tools should be used to identify security issues with code. Instead of using only one code inspection tool, it is highly recommended to use two or three tools to significantly raise the percentage of discovered weaknesses in a software configuration item. Finally, subsystem and system penetration testing conducted by Blue and Red Teams is necessary to fully assess the system. The results of the Pen Testing should be used to develop a risk matrix including the likelihood of a successful attack(s) and the associated consequence(s) given attack success.

Secure Chain of Custody

Chain of Custody is an essential element of Supply Chain Risk Management (SCRM) and is a characteristic of a Trusted System or Network as defined in DoD Instruction 5200.44. All DoD contracts should require the contractor to develop a SCRM Plan in acccordance with the Government's Program Protection Plan (PPP) and the contractor's Program Protection Implementation Plan (PPIP). The SCRM Plan should be a Contract Data Requirements List (CDRL) item subject to government approval. DFARS clauses 252.239-7017 Notice of Supply Chain Risk (November 2013) and 252.239-7018 Supply Chain Risk (October 2015) also should be included in all contracts to help control supply chain risk.

The Cyber Industry Technical Advisory Group (CITAG) has identified key practices for assuring software chain of custody in the DoD supply chain. These include the following:

- Digital Signature
- Encryption
- Out-of-Band Hash
- Keyless Signature Infrastructure
- Origin to End Platform Supply Chain Control
- Transmission Security
- Positive Handoff
- Operations Security
- Obfuscation

Conclusion

This article began with a hypothetical legal case thrown out of court because of a lack of provenance for a key piece of evidence. In the world of DoD software, the end user needs to be the judge and jury when it comes to accepting the provenance of the code that they will be using. In some cases, they are placing trust for their lives and the lives of others on the provenance of the software to be used in battle. If the acquisition and sustainment communities responsible for the development, delivery, installation and maintenance of that code cannot prove provenance of the software, the end user has every right to throw it out as unacceptable. Knowing the provenance of a software item enables the DoD to make proper risk-based decisions on its acceptability for use.

In summary, the software provenance framework is made up of origin and chain of custody. From there you can deep dive into the component parts that support this top-level framework.

The author can be contacted **roy.wilson@dau.mil**.