Inherent Problems in the Information Technology Supply Chain

Author: Courtney Cavness, atsec information security corp., USA. courtney@atsec.com

Problem

It is becoming increasingly common for IT companies to incorporate components (or even entire applications or operating systems) created by 3rd-parties into their offerings. This may be done simply to bundle applications together for the convenience of the end user. Or, the impetus may come from cost advantages for the developer, such as a reduction in development effort by using existing code or products (including open source code and 3rd-party libraries) so that the developer doesn't expend efforts "reinventing the wheel."

Some companies also employ 3rd-parties to test their source code. This may be based on a business decision to save money by using more cost-efficient labor, or to make use of time zone differences so that, for example, testing can be performed on newly-developed code in a cycle that is not interrupted by typical business hours in a single country.

When viewing this process from a security standpoint, the resulting supply chain of vendors/individuals involved in the production of the IT product must follow the customer-focused definition given by Hines[1], "Supply chain strategies require a total systems view of the linkages in the chain that work together efficiently to create customer satisfaction at the end point of delivery to the consumer."

When taking this stance, the following issues become apparent:

- **Lack of Trust**: The developer's own employees are assumed to be trusted, typically supported by their hiring practices and internal peer review for coding to established best practice standards.

  How, then, can it be shown that the 3rd-party developer/tester of the incorporated code is not malicious and did not inadvertently introduce vulnerabilities. Even if the 3rd-party is known and trusted by the developer, how can the developer ensure that the 3rd-party themselves followed secure development practices (such as using secure access connections) to prevent maliciously-injected code?

  The 3rd-party code itself must be reviewed during an acceptance procedure.

  - If the incorporated code is pre-compiled, it is a virtual black box inside the IT product and can't be reviewed.
  - If the incorporated code is large, such as an open source kernel, the effort to perform code review is a daunting task. However, once reviewed, integrity of this base code can be shown via an associated hash value, and then as fixes are applied, only the deltas would require review.

- **Configuration Management**: The developer must now keep at least a copy of the incorporated code at a specific version, so that their own product can be re-created, if necessary.

- **Supply Chain Management**: Who determines when the 3rd-party code is finalized? How and when is the 3rd-party code accepted into the code base? After that point, who can make additional changes?

  What happens if the vendor goes out of business and no longer updates their code? The developer must assume responsibility for updating this code.

[1] Hines, T. 2004. Supply chain strategies: Customer driven and customer focused. Oxford: Elsevier.

- **Stability**: What happens if the vendor stays in business and updates their code to patch security issues? The developer must now be responsible for monitoring this vendor and distributing fixes as necessary.

Existing Approach in Security Standards

Some well-known security standards have a concept of formulating acceptance procedures. For example, the Capability Maturity Model Integration (CMMI) and the international standard ISO/IEC 12207 : 2008 include the concept of acceptance procedures to address the developer and contractors/3rd-parties working together to establish criteria or testing that must been met upon acceptance of acquisitions. However, the specifics of this are left up to the developer to define.

Other standards, such as the Common Criteria, don't address the concept of acceptance procedures to review code from 3rd-parties, at any assurance levels (or, depth of rigour of evaluation).

Defining an Acceptance Procedure

The specification of what would make any acceptance procedure sufficient for accepting code modified by a third-party are not much different from the procedures usually applied for accepting code developed by the company's own employees into the code base.

For example, almost every company typically performs two types of tests on their IT products: integrity and functionality. The integrity, or regression, test can be as simple as checking that the new source code successfully compiles and doesn't break the build, or it can mean running the entire test suite on the new code. The functionality, or unit, test ensures that specific features function as expected. However, the acceptance procedure suggested in this paper is meant to test that security functionality has not been modified by the third-party developer's code. So, the acceptance procedure is aimed at checking for malicious code, unintentional vulnerabilities, back doors, etc.

The acceptance procedure should be a manual review (potentially combined with some tool-based analysis) performed by one or more employees of the developer's organization that is undergoing evaluation. The review should be performed on any code that is developed or changed by subcontractors or third-parties (including open source communities) to make sure it performs the expected — and only the expected — functionality before it is accepted into the code base.

Another benefit of an acceptance procedure that meets the above criteria is that the manual review might also find flaws such as buffer overflows that occur simply as an extension of being incorporated with another developer's code. For example, one developer has a module that has no external interfaces, and therefore this code assumes that trusted interfaces won't request invalid parameters. However, the developer of another component that has an external interface may not check for valid parameters because it doesn't handle that specific function, but merely passes it on to the other component. This might expose the code not only to buffer overflows, but also to cross-site scripting and similar types of vulnerabilities caused by insufficient parameter checking.

Conclusion

Although the exact definition of an acceptance procedure for 3rd-party code in IT products is not currently specified in the mentioned IT security standards, if an organization has an acceptance procedure that meets the specifications outlined in this paper, it can be argued that the code can be trusted, even though it may have been developed or changed by an unknown, and/or untrusted, third-party before being incorporated into an IT product's code base.

The author intends to propose specific requirements to specify what constitutes a sufficient acceptance procedure for 3rd-party code for assurance levels EAL3 and EAL4, at the next International Common Criteria Conference's call for papers.