Untrusted Developers

Code Integrity in a Distributed Development Environment

Courtney Cavness

with Helmut Kurth and Stephan Mueller

2010-08-10

Version 1.0

atsec information security corporation
9130 Jollyville Road, Suite 260
Austin, TX 78759
Tel: +1 512 615 7300
Fax: +1 512 615 7301
www.atsec.com

Table of Contents

Audience

This paper is aimed at organizations that want to have their product evaluated using the Common Criteria standard at assurance levels EAL3 and EAL4. At lower levels, this issue is not evaluated. For higher assurance levels, the requirements expressed in this paper may not be sufficient, and more formal analysis of the semantics of third-party code and more strict controls of the development environment may be required to ensure that an attacker with a high attack potential would be unable to attack the organization's computers and manipulate the code developed on its systems.

Establishing Trust

An organization must establish trust in their developers to have confidence that the code they produce or include in the product is not malicious and does not inadvertently introduce vulnerabilities.

This confidence can be established by a variety of techniques including vetting developers via hiring procedures that include a background criminal check, reference checks, etc., which helps determine whether the individuals are trustworthy.

Additionally, an organization can test the assumption that a developer has not included malicious code by requiring some form of code review, such as developer peer reviews, to provide a second set of eyes to examine the code's functionality. This is usually combined with a tool-based code examination to ensure that defined coding standards have been observed.

Maintaining Integrity

Once the assumption of code-integrity is confirmed, the organization must work to prevent their trusted code from being tampered with by unknown and untrusted entities. And so, the organization can define procedures, enforce processes, employ various technical means to restrict logical and physical access to the code only to authorized users, and record the modifications they make.

But an important question is: how can restricted physical access to the code be ensured when developers are working remotely?

Remote workers

It is commonly accepted (though not explicitly stated) in the Common Criteria (CC), that site visits are not expected to be performed for remote developers (that is, individual developers working from home). Such site visits are neither feasible nor cost-effective. Therefore, CC evaluators must rely on an organization's policies regarding employees working remotely to gain confidence that security policies and best practices are followed by these individual workers, either when working at home or when traveling.

For example, an organization may require remote workers to use a secure access connection for access control (via required password) and confidentiality (via encryption). Additionally, viewing confidential information in a public setting, such as an airport or coffeeshop, where bystanders could potentially view information on the worker's computer screen might be discouraged. The policies may also describe situations to avoid when traveling, such as not leaving a laptop in a rental car. And data stored on the computer itself is typically required to be protected via a cryptographic mechanism — which would protect it even from an unauthorized person who gains physical access to the computer.

Another consideration is the security of the remote system itself that is used to access the organization's data assets. Policies may include requirements for remote workers to use company laptops, or else to install company-approved virus and malware software with scheduled disk scans, automatically install software updates, store data only in encrypted form, and enable a firewall. Any audits the organization performs on the remote worker's compliance to these policies can provide additional confidence that the prescribed measures are in use.

Third-Party Developers

The previous scenarios assume that the developer is trusted, both to write non-malicious code, and to follow the organization's policies as specified. However, it is becoming more common that organizations are employing third-parties to develop and/or test their source code. This may be based on a business decision to save money 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. It is also increasingly the case that organizations use third-party libraries to reduce development costs.

If a Target of Evaluation (TOE) in a CC evaluation includes code that was developed and/or tested by a third-party, that third-party is outside the control of the TOE organization's own development environment and policies. Therefore, unless bound by contractual agreement, this third-party is not required to follow the TOE organization's policies for development security or coding standards.

So, how should an evaluator handle certain aspects of the CC Development Security family (ALC_DVS) that requires the evaluator to perform a site visit to conduct interviews with developers and inspect the development environment? Some validation schemes have issued interpretations to be considered in this instance. For example, the Bundesamt für Sicherheit in der Informationstechnik, has proposed one potential solution, which is to perform a site visit to the third-party development / test site to examine their personnel policies and development security practices to determine whether they are sufficient.

But, what happens when the third-party developer is a collection of remote workers, such as an open source community of anonymous developers? Or, what if the third-party will not allow a side visit to be performed? How, then, can this code be trusted?

One possible answer is via an acceptance procedure.

Acceptance Procedure

An acceptance procedure is not defined in the CC. Therefore, several aspects should be considered when specifying what would make any acceptance procedure sufficient in terms of having confidence that the code an organization accepts from a third-party is trustworthy. Those are not much different from the procedures usually applied for accepting code developed by one of the company's employees into a product.

For example, almost every company typically performs two types of tests: 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 TOE 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 TOE code base.

Why must the review be manual?

Note that just running code through software such as FxCop and/or Coverity is not adequate as an "acceptance procedure."

Such code analysis tools are useful in verifying that coding standards have been followed and typical coding bugs are not present in the code. But, while these tools can be used to support an acceptance procedure, in and of themselves, they usually have no mechanism to interpret what the code is doing. Therefore, it is necessary that the acceptance procedure consist of the trusted employee(s) of the developer manually evaluating the code and determining whether the code does only what it is supposed to do, has no vulnerabilities, is not malicious, does not contain back doors, etc.

Which code must be reviewed?

All third-party code that is part of the TOE security functionality needs to be reviewed, regardless of whether it relates to security functionality (that is, regardless of whether it is SFR-enforcing, SFR-supporting, or SFR-non-interfering). Code that is not part of the TOE security functionality may not need to be considered if it can be shown that any, even malicious, behavior of this code will have no impact on the enforcement of the security objectives and the security claims defined in the Security Target.

For open source development, such a review might be a daunting task if the incorporated code is large. Nevertheless, the developer must somehow show that all of the necessary code has been reviewed.

For example, once the required code (as defined above) has been 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. Confidence in the assumption that the process has been bootstrapped and perhaps only patches need to be reviewed from a certain point forward might also be strengthened if the developer has employees who are part of the open source community and who monitor the mailing list and have write access to the open source repositories. This is the way Linux distributors work.

Another benefit of an acceptance procedure that meets the above criteria is that the manual review might also find bugs 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.

Documenting the Acceptance Procedure

An organization's acceptance procedure should be documented to describe the following aspects:

- the role or department that is required to review the code
- the depth and rigor of the review
- the activities to be performed (including the tools to be used)
- how the results of those activities are to be documented and how this documentation will be managed
- the life-cycle plan for how and when the third-party code is reviewed, accepted, and checked into the TOE organization's configuration management system for maintenance and inclusion into the TOE

It should also be stated how the organization defines when a third-party has finished its work.

Additional Responsibilities for the TOE Organization

If the CC evaluation includes augmentation for flaw remediation, then the TOE organization has an additional responsibility to distribute any applicable fixes to the third-party's code. In other words, any third-party code must be fully supported. That is, the TOE organization must monitor the third-party for reported flaws associated with the code. When a fix is available, the TOE organization must test the fix (including following the acceptance procedure for the modified code), and provide the fix to its customers via the TOE organization's regular, secure, distribution channels.

The onus of providing fixes to third-party code applies not only to code that is incorporated into the TOE, but also to any third-party code that is bundled and distributed with the TOE.

How and When Third-Party Code is Integrated

It must be the TOE organization who stores and manages at least a copy of the third-party source code, and decides which source code gets integrated and built to form the TOE.

It may be that the third-party developer checks the source code into the TOE organization's configuration management system before it is reviewed. However, there must be a process for the source code to be marked in some way as "ready for integration" so it can be included in a build. The key point here is that it must be the TOE organization that decides which code is ready to integrate into the product build; what happens before that point doesn't much matter. It is at this integration point that the acceptance procedures come into play and it is verified that they have been applied — not necessarily at the initial acceptance of the third-party source code.

The development life cycle usually identifies exactly who can make any changes at different points in the cycle. It is common that before a file is marked for integration very little attention is paid to changes. But, after a file has been marked for integration, changes are much more closely examined. At that point, typically very few changes are allowed by anyone, either as a process implemented by the TOE organization or as a restriction enforced by a development tool or the configuration management system. The closer it is to the release date, the fewer changes are allowed and the more scrutiny those changes undergo.

In a CC evaluation, only the code that gets integrated into the build is of concern. After integration, the code is under the control of the TOE organization and remains so, and the CC evaluator is only concerned with what happens from that point in the life cycle forward.

Integrated code versus bundled code

To be incorporated into the TOE, the third-party code must be compiled by the TOE organization (rather than delivered as object code). This model can be referred to as centralized development with de-centralized developers. Otherwise, if the third-party code is delivered as object code and the TOE organization simply puts them together to form a product, then there can be no acceptable review of the source, and so an alternative resolution must be agreed upon by the CC certification scheme (for example, as referenced above, performing site visit at the third-party developer's site).

Conclusion

Other standards for Information Technology such as the Capability Maturity Model Integration (CMMI) and the international standard ISO/IEC 12207 : 2008 include acceptance procedures that address specifying criteria or testing that must been met upon acceptance of acquisitions from contractors, third-parties, etc. The CC should be updated to reflect the growing trend of TOE organizations that receive source code development and/or test services on features, components, or even kernels that are included in the TOE boundary.

Although the concept of an acceptance procedure is not currently defined in the CC, 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.