Открытые системы СУБД
26.05.2011

Open Systems. No. 04, 2011. P.38-41.

Software Certification Without Source Codes

Alexander Barabanov, Alexey Markov, Andrei Fadin

NPO Echelon, Russia

Abstract. How to evaluate the security of programs in the absence of any source codes and what needs to be done to lower risk when using them.

Keywords*: software security, security code review, vulnerability, software testing.*

Increases in vulnerability in programming systems are directly linked to their high structural complexity as well as to the dynamism of versions and techniques, which do not allow guaranteed evaluations of software security to be obtained within an acceptable time. Evaluation of the conformity of software security requirements is determined by certifying software certification for the absence of undeclared features or, where there are no such requirements, by auditing the security of the code. In the first case, the reference point is the Regulatory Document [1] of the Russian State Technical Commission. In the second case, openly-accessible international standards can be used: PCI DSS/PA-DSS, OWASP Top Ten, CWE.

The State Technical Commission's regulatory document outlines the requirements for programming documentation, the assigned condition controls and the source code static and dynamic analyses, which constitute an essential part of the certification test process [2]. In other cases, there may be no requirement for software source code availability as, for example, is the case in Common Criteria certification texts or the auditing of banking applications for PCI DSS matching where, of course, national security is not an issue.

In practice, situations sometimes occur where software developers are not able to create source codes in full – for example, where versions have been lost or incorrectly modified, where external codes with licence restrictions have been used or for reasons of confidentiality. In addition, software source codes may not be present in an explicit form, either for the purposes of allowing direct code generation to be used, or where the complex binary code transformations are not directly linked with the program source codes.

Until recently, the black box test method (i. e. stress testing in abnormal software mode) and the reverse engineering method (including binary code disassembly with an attempt to simulate the logic of the original program application) were the two basic approaches used to test software security in the absence of source codes [3–5].Both approaches are extremely time-consuming and are not specified in regulatory documentation. However, alternatives do exist.

The following measures increase software assurance:

- evaluation of the options for high-quality software decompilation for the carrying out of certification tests, taking into account the existing regulatory structure
- checking the core-image codes to allow security levels to be accurately evaluated and decisions on possible risk reduction linked with software use to be made
- evaluating options for applying additional information protection measures, linked with the use of software not having source codes.

In any case, in the first instance, the risks and vulnerabilities of the resources of the computer-based system must be evaluated, together with the corresponding risks (including legal risks) linked with software component use where source codes are not present. Evaluating the risks allows alternative options to be selected (reduction, transfer, acceptance, refusal) and, in the case of their reduction, allows measures for increasing software security to be adopted.

Vulnerability carries risks for data security only in cases where it can be exploited. This needs to be established during testing. Where source codes are absent, however, risk evaluation becomes much more difficult and an approach based on external software links is then adopted.

Evaluating Decompilation Options

Platforms allowing high-quality decompilation and able to restore program source codes whilst retaining full functional and information object hierarchy, its links and control structure are now widely available. This is appropriate to programming languages with intermediate representation in the form of byte-codes and virtual machines for their execution – for example, the Oracle JVM (Java Virtual Machine), familiar as a tool for implementing languages such as Java, NetRexx, Ruby (JRuby), JavaScript (Rhino), Python (Jython), Groovy, PHP (Quercus), Clojure, Scala, etc.

In some cases, high-quality decompilation can be implemented for platforms. In Net, where CLR (Common Language Runtime) is used, it is possible to enter codes written in the ASP.Net, C#, Visual Basic. Net, C++/CLI, F#, J#, JScript. Net, Windows PowerShell (and other) programming languages. The features of this platform and others like it – for example, ActionScript Virtual Machine and Microsoft P-CODE Virtual Machine – allow source codes to be compiled in intermediate high-level binary representation (and not in microprocessor commands), which is already converted into processor instructions at the execution stage.

In order to confirm usage options when certifying the program texts obtained during decompilation, the Echelon Development and Production Centre conducts software experimentation and undertakes research and certification testing in its accredited test laboratories. The JVM (Java Virtual Machine), the Java programming language and the CLR and C# environments were chosen as the programming platforms. Test programs containing current (2010) vulnerabilities were also created. Testing was undertaken on programs with source codes and decompilations. An accredited “AK-VS” source code security analyser able to support international CWE vulnerability classification was used to detect vulnerability.

For the JVM (Java) and Net (C#) platforms, the results revealed a high correlation between the source codes and the codes obtained as a result of decompilation (Table 1). Among other things, the ability to detect code vulnerability was proven during the course of the experimentation. Basic checks (including report compilation) for the absence of undeclared second level control features were also carried out based on statistical analysis.

Table 1. Verification of decompilation options for running certification tests and code audits

Options	**Platform**	
	JVM (Java)	**. NET (C#)**
Decompilation option	+	+
Decompilation quality: semantic agreement syntactic agreement	 100% 70%	 100% 40%
Search for vulnerability in decompiled text	100%	100%
Recompilation from source codes obtained	100%	90%
Creation of functional object list	100%	100%
Creation of information object list	100%	100%
Construction of information matrix	86%	100%
Construction of control matrix	100%	100%
Creation of activation routes	80%	80%

The use of high-quality compilation allows 100% detection of source code vulnerability and also guarantees the provision of high-level computer-based certification reports. This raises a paradox concerning the shortcomings of the formal requirements of regulatory documents – for example, a lack of success (in some instances) in creating matrices and activation routes, despite successful detection of all the vulnerability affecting the security system. The diagram shows an example of semantic agreement and syntactic disagreement in a potentially dangerous software fragment.

```
//Фрагмент исходного текста. Уязвимость CWE-89: SQL-инъекция
public static void SQL_Injection(string connString, string userName, string ItemName) {
SqlConnection conn = new SqlConnection(connString);
string query = «SELECT * FROM items WHERE owner = ‘» + userName + «’ AND itemname = ‘» +
ItemName + «’»;
SqlDataAdapter sda = new SqlDataAdapter(query, conn);
DataTable dt = new DataTable();
sda.Fill(dt);
}
//Фрагмент декомпилированного текста. Уязвимость CWE-89: SQL-инъекция
public static void SQL_Injection(string connString, string userName, string ItemName) {
SqlConnection conn = new SqlConnection(connString);
SqlDataAdapter sda = new SqlDataAdapter(«SELECT * FROM items WHERE owner = ‘» +
userName + «’ AND itemname = ‘» + ItemName + «’», conn);
DataTable dt = new DataTable();
sda.Fill(dt);
```

It must be pointed out, however, that in practice, high-quality compilation can have various limitations, linked with code obfuscation (protection of code meshing routes), partial data loss, etc.

Additional Run Code Checks

Where high-quality decompilation has been unsuccessful, it is helpful to record all the source code imperfections and perform a test linked with the given risk evaluation. The checks can be used to pursue the following aims:

- determination of the integrity of the software and its source, including a check on licence validity
- identification of external interests
- control of the integrity of the software environment during installation, de-installation and upgrading
- analysis of any known risks linked with internal and external components by using open-access sources (security bulletins) and vulnerability, and exploitation data bases.

Basic checks:

- licence control – a check of manufacturer and supplier details, their presence in open-access registers and in software and software component information repositories
- checks on external dependence, including the correspondence of lists of actually-used external objects with declared versions – for objects (for example) such as: virtual machines and application code readers; software-linking sub-programs; cache servers, databases, web servers; dynamic link libraries; RPC target components and interfaces (remote procedure activation)
- security bulletin search for information relating to vulnerability inherent in the given objects
- installation/de-installation process checks linked with actually-used and declared software component list correspondence checks
- upgrade process checks linked with actually-replaced and declared upgrade component list correspondence checks

- checks on components and their internal dependence linked with actual and declared software component list correspondence checks and checks on absent or surplus software components (including integration elements)
- antivirus checks and vulnerability scans linked with checks on distribution and the state of the software on installation.

The results of these checks can be produced as a final technical report, which includes specific configuration recommendations when using the software.

Certainly, from the security standpoint, such an approach is as good as testing the software for the absence of undeclared features, in accordance with level four checks in which only the legal risks are determined. Consequently, it is possible to outline the certification requirements for software not having source codes, for example, in second and third class personal data information systems.

Installing Additional Protection Measures

In order to reduce the remaining risk to acceptable levels (determined by specialists when analysing and controlling risk), additional equipment and measures may be used to safeguard against risks linked with software not having source codes.

In addition to traditional protection mechanisms (back up, firewalls and monitoring), it is possible to employ a range of special measures for securing data at program application level (Table 2).

Table 2. Examples of measures to protect data at application level

Class of data protection measure	Function in brief	Examples
Firewalls at program application level	Filtration of application layer protocol requests sent to potentially risky software	OWASP Web Application Firewall, AppGuard
Proxy servers	Intermediate link application between potentially risky software and other software able to evaluate security policy regulations	Myosotis, Pgpool
File system monitors and registers, network traffic interceptors and analysers; process monitors; API activation monitors	Investigates anomalies in application behaviour	Process Monitor, PortMon, DiskMon, “Scanner VC”, Wireshark; API Monitor

Using filtration measures on given application layer protocols prevents intruders of any kind from exploiting software vulnerability, for example, SQL injections or cross-site scenario implementation. The application of firewalls, including intellectual proxy servers, allows the volume of network protocol data between potentially risky software and other programs to be controlled. If, for example, program operation does not envisage any network protocol interaction, it is recommended that potentially risky software be prohibited from sending and receiving network protocol data. This reduces the likelihood of any vulnerability being exploited.

Surveys undertaken on software not having source codes must include basic function risk and use restrictions. Preliminary information can be obtained from a range of open-access sources and instruction manuals. At the testing and trial stage, software operation should be monitored by specialists and the results compared with those recorded for standard software operation. This allows any anomalies in software behaviour to be detected and any corresponding security risks to be evaluated.

***

The measures put forward allow decisions to be made on the options for installing and using programming systems not having a full complement of source codes. These are interesting results, which testify to the possibility of carrying out certification tests on the non-source code software developed in modern programming systems. It is essential that the decompilation faults revealed are not allowed to affect the capacity to detect potentially dangerous fragments and zones of risk.

It is recommended that execution code checks be organised. Their effectiveness in respect of security requirements should be commensurate with the level four tests designed to check software for the absence of undeclared features.

In order to remove any remaining risk linked with the absence of source codes, the introduction of further data protection measures at programming application level is recommended.

Literature

1. Regulatory Document. Protection from Unauthorised Access to Data. Part 1. Computer Software as a Means of Protecting Information. Control Level Classification for the Absence of Undeclared Features. – Russian State Technical Commission, 1999.
2. Markov A. S., Mironov S. V., Tsirlov V. L. Detecting Vulnerability in Source Codes // Open Systems, No. 12, 2005.
3. Eilan E. Reversing: Secrets of Reverse Engineering. – Wiley City: Wiley Publishing, 2005. – 595 p.
4. Kalinovsky A. Covert Java: Techniques for Decompiling, Patching, and Reverse Engineering – Indianapolis: Sams, 2004. – 288 p.
5. Sutton M., Greene A., Amini P. Fuzzing: Brute Force Vulnerability Discovery. – Addison-Wesley Professional, 2007. – 576 p.

Alexander Barabanov, *Alexey Markov*, *Andrei Fadin – NPO Echelon, Moscow*, *Russia.*

Article URL: http://www.osp.ru/os/2011/04/13008784/
A. Barabanov, A. Markov, A. Fadin. Software Certification Without Source Codes. Open Systems. 2011. №4. P.38-41.