UDC 004.492.2

SYSTEM FOR PROTECTING EXECUTABLE FILES OF THE JAVA PROGRAMMING LANGUAGE

VIOLETTA TROPNIKOVA, STEPAN YEHILEVSKY
Polotsk State University, Belarus

The paper describes a way to protect Java applications from unauthorized access to bytecode using symmetric encryption algorithm RC5. Great attention is paid to the implementation of javaagent and maven-plugin.

Currently, most attacks occur at the application level. Therefore, software protection has the highest priority. Difficulties often arise with software developed to order, as information protection and vulnerabilities in the system are given minimal attention, as very often modern software products are developed in a short time and with limited budgets.

One of the most effective methods for protecting software is to encrypt files with executable code. Encryption of executable files allows you to secure the software product, prevent most problems, such as unauthorized access, the introduction of bookmarks, the activity of network worms and viruses, the use and modification of the software product.

The development of a software product that provides protection for executable files of the Java programming language to prevent unauthorized access to the software product is a very urgent task.

The Java source file is a text file with a .java extension that contains one or more class definitions. The Java compiler generates a bytecode from each .java file. The bytecode obtained during the compilation process, for each class is written in a separate executable file, whose name coincides with the class name and has the extension .class. When the program starts, the necessary .class files are loaded by the JVM (Java virtual machine) and bytecode instructions can be executed [1].

The main idea of the development is that the .class files will be stored in an encrypted form, and they will be decrypted when they are downloaded to the JVM. The files can be encrypted by any program after compilation, then it will be shown how to automate this process. And the decrypt files will be javaagent - a program that can intercept the bytecode of another program when it is loaded into the JVM [2].

One of the main tasks that need to be solved at the design stage is the choice of a reliable encryption algorithm. As such an algorithm RC5 (Ron's Code 5 or Rivest's Cipher 5) was chosen - a block cipher developed by Ron Rivest from RSA Security Inc. in the mid-90's, with a variable number of rounds, the length of the block and the length of the key [3].

There are several different variants of the algorithm, in which the transformations in the "half-rounds" of the classic RC5 are slightly modified. In the classical algorithm, three primitive operations and their inversions are used:

– Addition modulo 2^W;
– XOR;
– the operation of cyclic shifting to a variable number of bits (x <<< y).

The main innovation is the use of a shift operation on a variable number of bits not used in earlier encryption algorithms. These operations are performed equally quickly on most processors, but at the same time significantly complicate the differential and linear cryptanalysis of the algorithm.

Encryption using the RC5 algorithm consists of two steps:
– procedure for expanding the key;
– encryption.

To decrypt, the key expansion procedure is performed first, and then follow the operations that are the reverse of the encryption procedure. All operations of addition and subtraction are performed modulo 2^w [3].

Now everything is ready to demonstrate the features of the implementation of javaagent as a decrypting program.

The decryption program is divided into two subsystems: decryption and downloading of bytecode. The decryption subsystem includes the RC5 class, which implements the decryption of the byte array. This class provides a decryption function, in which parameters the array of encrypted data bytes, the byte array of the key, and the original size of the .class file are transferred to the encryption. The decryption function returns an array of decoded data bytes of the original size, i.e. After decryption, the array is reduced to the size of an unencrypted .class file.

The byte-code download subsystem includes the ClassTransformer class. It implements the ClassFileTransformer interface, redefining the transform method, thereby allowing through the Java Instrument API to access the bytecode that is loaded by a third-party application of classes [4]. The transform method ensures that only those classes that have been encrypted are decrypted, the bytecode of the remaining classes remains unchanged. Also in class ClassTransformer functions are implemented to load the key from the file and load the display "class name - the initial size of the class file".

The entry point of the decrypting program is the premain function of the PreMain class. The premain function has access to the javaagent parameter passed to the function parameters, which must contain the name of the key file and the file name with the class names and their initial sizes, separated by the ";" symbol. Another parameter to the premain function is the Instrumentation class object passed to the JVM. This object will allow the JVM to pass the implementation of the ClassFileTransformer interface, i.e. class ClassTransformer. After this, the JVM loads the third-party program classes via the overridden transform method.

Earlier it was said about the automation of encryption of executable files. This problem can be solved with the help of Maven-plugin, which encodes .class-files after compilation. Apache Maven is a popular tool for building Java projects [5]. It has a variety of different plug-ins that can be used during various assembly phases. In addition, there is an opportunity to develop your own plug-in, which will be discussed later.

The encryption program (Maven-plugin) is divided into three subsystems: encryption, interaction with Apache Maven, key generation. The main functionality of the encryption subsystem and the key generation subsystem is implemented in three classes: RC5, Encryptor, EncryptionInfoSaver.

The RC5 class implements the same algorithm with a 64-bit encryption unit, key length 255 bytes and 150 rounds of encryption, which ensures that differential cryptanalysis can not be cracked. This class contains a key function called encrypt, which takes three parameters into the input: the incoming byte stream to be encrypted; the outgoing byte stream, in which, after exiting the function, the encrypted content of the incoming stream will be The key by which the incoming byte stream will be encrypted. This function returns the number of bytes read from the incoming stream. This number will need to be known during decryption, because the encrypted file can be more decrypted (or original). The difference in the sizes of the decrypted and encrypted file is explained by the fact that the RC5 encryption block has a length of 64 bits, which means that the size of the encrypted file must be a multiple of 64 bits. Therefore, when encrypting the size of the original array of bytes increases to a value that is a multiple of sixteen (bytes), and when decryption, the original size is restored.

The Encryptor class contains functions for encrypting the folder in which the compiled .class files are located. First, you determine the path to the folder you want to encrypt, the path to the folder to which to encrypt the files and the list of file filters. A file filter is a string template, to exclude those files that do not require encryption. The file filter pattern is built using the same rules as in the Windows command line and in the Unix console, for example, the "?" And "*" characters represent one or more characters respectively, and the "* Controller?. Java" filter will include such files, as "SystemControllerA.java", "FlurryController1.java", etc. After defining the paths to the folders and filters, a key for encryption is generated. The key is stored as an array of bytes. Then there is a recursive encryption of the folder. Encrypt only those files that have the extension .class and do not fall under the conditions of the user file filters, the rest of the files are copied to the destination folder. During the encryption process, the display shows "class name - the size of the original .class file". This mapping is used when decrypting.

The EncryptionInfoSaver class contains functions that store information about encryption, namely the key and the display "class name is the size of the original .class file". The information stored by this class is not encrypted. Therefore, when a key file is transmitted over an open channel, it must be previously encrypted with third-party software.

The subsystem of interaction with Apache Maven is implemented in the class CryptoMojo. This class extends the AbstractMojo class and overrides the execute method [6]. The CryptoMojo class uses the Maven Plugin API to make its functionality available to the Maven collector. Overriding the execute method is one of the requirements of the Maven Plugin API, when executing the plug-in by the Maven collector, this method is performed. The CryptoMojo class has three attributes: buildDir - the folder with the source compiled files .class; targetDir - the folder in which I will save the encrypted files; excludings - a set of filters to exclude certain files that do not require encryption. All attributes are marked with an @Parameter annotation. Therefore, the values of these attributes are loaded from the. Pom-file of the project's Maven configuration. The excluding parameter is optional and can be omitted. The CryptoMojo class itself is marked with an @Mojo annotation with the encrypt parameter and therefore, when configuring the plugin, it is necessary to set the goal "encrypt". After loading all the attributes the execute method is called, in this method the encryption methods are called, the encryption information is saved, and if Maven is started in debug mode, output information about the actions of the plugin in the general log.

The results of this study can be used by software developers and users to prevent unauthorized impacts on the software product.

REFERENCES

1. Java SE Technologies [Electronic resource]. – Mode of access: http://www.oracle.com/technetwork/java/javase/tech/index.html. – Date of access: 02.16.2018.
2. Java Agent Development Framework [Electronic resource]. – Mode of access: http://jade.tilab.com/. – Date of access: 02.16.2018.
3. Wikipedia RC5 [Electronic resource]. – Mode of access: https://en.wikipedia.org/wiki/RC5. – Date of access: 02.16.2018.
4. Package java.lang.instrument Documentation [Electronic resource]. – Mode of access: https://docs.oracle.com/javase/7/docs/api/java/lang/instrument/package-summary.html. – Date of access: 02.16.2018.
5. Apache Maven Project [Electronic resource]. – Mode of access: http://maven.apache.org/. – Date of access: 02.16.2018.
6. Plugin Developers Centre [Electronic resource]. – Mode of access: http://maven.apache.org/plugin-developers/index.html. – Date of access: 02.16.2018.