A study of the correctness of the execution of a class file with an embedded digital watermark in different environments

P I Sharikov, A V Krasov and V N Volkogonov
St. Petersburg State University of Telecommunications of the prof. M. A. Bonch-Bruyevich, pr. Bolshevikov 22, build.1, St. Petersburg, 193232, Russia

E-mail: sharikov.pavel@ro.ru

Abstract. In this work, the authors set a goal of carrying out a complex study. Preparation of different types of environments and operating systems of different bit version is made. After that, preparation, creation, and embedding of a digital watermark in a class file of the program module written in Java take place. Creation and embedding of a digital watermark are carried out according to the method, previously developed by the authors and associated with the use of undocumented features of Java virtual machine. After all the steps, we execute the file on various platforms with different presets, and then we check if a digital watermark in a class file is nor overwritten nor destroyed and if the class file itself functions in the same way on all platforms. Among other things, the checks are performed to ensure that the digital watermark meets the previously defined criteria.

1. Introduction
Technologies are developing rapidly. The most popular programming language as of March 2020 according to TIOBE index is Java [1].

Java systems are written in different languages for different systems and processors; the main rule is a consistent performance. We decided to conduct this study due to the differences in the operation of Java virtual machines with different bit versions in different hardware environments.

The novelty of this study, as mentioned above, is that at the moment there is almost no publicly available scientific papers devoted to such a narrowly focused issue, that is the possibility of running a Java file with a hidden embedded digital watermark on different operating systems, processor architectures and equipment configuration [2][3]. By the time of writing this article, the authors had not found in open sources scientific papers that met the condition of relevance on this topic.

It is necessary to make sure that an attached message or a digital watermark will not change, and its operating codes in the Java file will not be overwritten with other operating codes of the Java byte code due to differences in the configuration of the devices on which an application will be running. Also, the class file involved in the study should work without errors in both cases.

To do this, we need to deploy a virtual environment with different configurations. In the course of work, a KVM hypervisor will be installed, on which we will create several virtual machines with different operating systems. On one of them, the emulated processor architecture will differ from the one that was used to embed a digital watermark in the class file. To accomplish this task we need to configure and install all the components necessary for virtualization, install guest virtual machines, and configure a network for virtual machines.
2. Relevant works

No identical works in which the issue of this article is investigated have been found in public access. A scientific work associated with verifying the operativeness of an executable Java file with embedded digital watermark on different hardware environments, under different conditions, and on different configurations is a dynamic study.

However, there are several works associated with testing JVM performance on different platforms or with different OS. According to the authors of this article, the main disadvantage of the following studies is their irrelevance for the moment, because even if compare to the version of 2015, Java virtual machine underwent changes in operation mechanism and security analyzer. Also, the gap between Java versions at the moment is too wide to use the data of those studies.

The first such work "Comparative evaluation of programs execution time on various platforms", [4] authored by Khashkovsky V.V., Lutai V.N., and Yurchenko V.V., was carried out in 2009. Its authors compared the speed of program execution on Windows XP and Linux Open SUSE using identical hardware platforms. According to the results, the Linux environment shows slightly higher performance. The work mentioned above is similar to this one because it also included measurement of a program execution time. However, there is to consider that the purpose of this research is more than just measuring the program execution time.

The second work of Ukrainian colleagues Didukh A. I. and Tishchenko V. V. "Comparison of Java performance on the Raspberry Pi microcomputer" [5] was carried out as late as in 2015. It compared how the alterations of processor frequency affected the performance of Java programs.

That work is not suitable for comparison with the current article, as it used the same operating system and the same hardware, with only a processor frequency being changed. Therefore, it is close to this study by only one of several topics that the current study covers.

3. Differences in Java bit version

First, the disadvantage of the 64-bit architecture is that the same data structures take much more memory. As you develop or run your applications, a 64-bit Java uses up to 30-45% larger stack than a 32-bit Java.

Object headers take 12 bytes on a 64-bit virtual machine. On a 32-bit virtual machine, the distribution is different: object references take 4 or 8 bytes depending on the virtual machine, flags, and stack size. Compared to 8 bytes for headers and 4 bytes for links in a 32-bit JVM, there is a noticeable difference in resources overheads [6].

A large stack of data in Java can cause long pauses when Java virtual machine decides that it is necessary to get rid of various references to data, objects which can not be accessed, and other already unnecessary data or variables. Of course, a delay in the operation of Garbage Collector affects the application execution time, which also increases significantly. The main problems with bit versions of Java are address spaces. For example, Windows splits an address space of a process in half. The first half is the kernel space that user processes can not use. The second half is the user's space. Also, it does not matter how much RAM there is. A 32-bit process can only use 2 GB.

4. Virtual machines — setup

In advance, we prepared CentOS 6.3 operating system for installing 32-bit and 64-bit guest virtual machines, and a physical server with a 64-bit operating system on which KVM will be installed along with the virtual machines (host server).

All actions were performed under the root user on Intel Xeon Quad Core E3-1230 3.20 GHz / 16 GB / 2 x 1 TB configuration.

Preparatory stage. We need to install KVM and virtualization libraries on a physical machine, start the KVM service, check the module load and connection to it. Then you need to create a storage for virtual machines. There are several types of such storages. The authors used the following: for each virtual machine, a file in *.img format is created for a virtual hard disk. These files are placed in the /guest_images directory.
First of all, we need to select the hard drive that we want to edit. After that, we create a new partition on this disc and save the changes. We need to specify that we are creating ext4 file system in the new partition and using all the free space on the disc for this purpose. After all the steps we have taken to create and configure a partition on the hard drive, we need to create a mounting point on our hard drive for the files of virtual machines. Accordingly, we need to mount the partition, start the storage and add it to autostart.

After the storage configuration, a configuration of the host server was performed. Before that configuration, we checked if bridge-utils package was installed. Remote server connection loss without physical access to it leads to many problems. For external connection, we decided to use eth0 interface. It was configured in the following way: IP address 10.110.10.15 from /24 network, mask 255.255.255.0, gateway 10.110.10.1.

Once done configuring the host server, we should start installing CentOS on a guest virtual machine. After installing CentOS on the guest virtual machine, connect to the host server. After that, the setup and connection process is completed and you can use this virtual machine to run a class file containing a digital watermark which was embedded in it on a machine with different configuration and bit version.

5. Hidden embedding of a digital watermark in a Java class file
The study lies in the possibility of running a class file with an embedded digital watermark and checking the immutability of the embedding on processors of a different architecture and bit depth [7]. In this connection, we chose the previously proposed method of creating and hidden embedding of a digital watermark in the class file of Java application [8].

For information embedding into the class file, we chose a multifunctional program Java Bytecode Editor that allows controlling the process of replacing byte code commands with equivalent ones and notifies about possible errors at a new file start.

As a class file used to embed information it byte code, we chose one of the modules of the recreational application, which is designed to introduce Java programming language to primary school students. The class file is a separate part of the application. Thus, when checking the file on the virtual machine, attention will be paid not only to its execution and the content of the digital watermark but also to the execution time. The class file code contains quite a lot of conditional if-else jumps. In Java bytecode all conditional jumps operators have several equivalents which are used by virtual machines in different situations [9]. These opcodes perform the same operations in applications but use a different amount of memory. Therefore, if you rigidly specify in a byte code changing one opcode to an equivalent one and do not recompile a file, the file size either grows or decreases depending on your actions. With the help of this undocumented feature of how Java virtual machine works with files we will embed a digital watermark to the class file.

Figure 1 shows the unmodified byte code of this class. Let us analyze the opcodes that we can replace to obtain a digital watermark. This class file, of course, cannot claim to be a full-fledged Java program, which is why the attachment that we will embed in this file will be relatively small, compared to the attachments that can be embedded in real projects.

In the presented byte code we see 4 opcodes responsible for conditional jumps in the application code. Let us examine these opcodes in more detail.

- if_icmpgt: execute the condition if Value 1 is greater than Value 2
- if_icmpge: execute the condition if Value 1 is greater than or equal to Value 2
- if_icmple: execute the condition if Value 1 is less than or equal to Value 2
- if_icmplt: execute the condition if Value 1 is less than Value 2.
Now let us examine the codes of these operations in binary and hexadecimal formats in Java virtual machine. The codes are presented in table 1.

Table 1. Binary and hexadecimal representation of opcodes in JVM

| Operating code in JVM | Binary representation | Hexadecimal representation |
|-----------------------|-----------------------|---------------------------|
| if_icmpgt             | 1010 0011             | A3                        |
| if_icmpge             | 1010 0010             | A2                        |
| if_icmple             | 1010 0100             | A4                        |
| if_icmplt              | 1010 0001             | A1                        |

By converting these representations to the alphabetic ASCII representation (e.g. code table 866 - MS-DOS), we get “г”(163), “в”(162), “д”(164), and “б”(161) respectively. At this step, using the previously mentioned method [10], we can assume that we already have in our class file a digital watermark that we do not like. Now it is “твдд”. Taking into account that equivalent opcodes swap branches (values) of Boolean expression in some places, it is necessary to edit this point in the byte code as well [11]. Thus, if our first conditional jump is if_icmpgt containing a Boolean expression (x>y), then the replacement that will influence less on the byte code is swapping the given opcode with if_icmplt containing a Boolean expression (x<y). The replacements are made with considerations of minimal byte code editing. When editing the byte code of a given class-file, we strive to make sure that our changes are as follows:

1. Opcode if_icmpgt is replaced with if_icmplt, and the values of the Boolean expression are also replaced.
2. Opcode if_icmpge is replaced with if_icmple, and the values of the Boolean expression are also replaced.
3. Opcodes if_icmple are replaced with if_icmpge, and the values of the Boolean expressions are also replaced.
Thus, after completion of all replacements, when checking the resulting digital watermark by our method and the chosen code table, we should get an alphabetic sequence: “бдвв”.

Let us make the necessary replacements in the class file via Java ByteCode Editor to speed up the replacement and prevent memory errors.

After replacing all the necessary opcodes, we check on the local machine if the replacement is successful and if the file is working. Figure 2 shows that the replacement of the byte code is successful; if we compare code representations of the opcodes in the modified file with the selected code table, we will get the required character sequence.

Figure 2. Class file with modified byte code.

6. Checking how a file with a modified byte code works on virtual machines

From the article "Comparison of Java productivity in Windows and Red Hat Linux" [12], which compares the work of Java virtual machine on Linux and Windows operating systems, we have got information that Java runtime environment can work differently on different operating systems with the same hardware due to their differences in file systems and hardware management [13, 14]. Knowing that there is a probability of incorrect work of Java applications in Unix-like systems and that the open-source OpenJDK is used frequently instead of the standard JDK provided by Oracle, the class file chosen for this study was ported to another operating system.

Figure 3. Modified byte code in *nix environment.
As you can see in figure 3, the byte code of the class file, launched on a virtual machine with the CentOS operating system, did not change. The program performance also remained unchanged, which is confirmed by figure 4 that shows the class file run.

![Figure 4](image)

**Figure 4.** Program execution on CentOS.

As you can see in figure 4, the program runs in the same way as in a Microsoft Windows environment.

7. **Conclusion**

During this study, we found that changes in software, hardware, or operating systems do not affect program execution. Also, we found that the execution of a class file with an embedded digital watermark in other operating systems and configurations does not affect the program operation. We proved that the digital watermark does not get overwritten, which means that the internal structure of the file is fully intact and the class file code is correctly compiled into the Java byte code. Consequently, the methodology used in this study can be used on different operating systems and workstation configurations without having to think about how the program and the embedded copyright token behave.

**References:**

[1] Website https://www.tiobe.com/tiobe-index/

[2] Sharikov P I, Krasov A V and Shterenberg S I 2017 Method of creation and attachments digital watermark into an executable java file by means of substitutions opcode *T-Comm* **11**(3) 66-70

[3] Sharikov P I and Krasov A V 2017 Sterenberg S.I. Methods of creating and embedding of digital watermarks in Java executable files, based on opcode replacements *T-Comm — Telecommunications and Transport* **11**(3) 66-70

[4] Khashkovsky V V, Lutai V N and Yurchenko V V 2009 Comparative evaluation of programs execution time on various platforms *South Federal University news. Engineering science* **12** 176-81

[5] Didukh A I and Tishchenko V V 2015 Comparison of Java performance on the Raspberry Pi microcomputer *Bulletin of National Technical University of Ukraine "Kiev Polytechnic Institute". Radiotechnics Series. Radio Engineering* **60** 107-13

[6] Krasov A V and Sharikov P I 2017 A method of protecting Java program byte code from decompiling and stealing the source code by an intruder *Bulletin of Saint-Petersburg State University of Industrial Technologies and Design* **1** 47-51

[7] Khomyakov I N and Krasov A V 2014 A possibility of hidden information embedding in Java byte code *Information technologies of modeling and management* **2** (86) 185-91

[8] Sharikov P I and Krasov A V 2017 A study on the possibility of embedding a digital watermark in byte code by replacing a vulnerable Java class byte code *Proc. Conf. Information Security of the Regions of Russia, ISRR-2017* (SaintPetersburg, Russia: SPIIRAS) pp 499-500

[9] Sharikov P I 2015 A method of finding the size of the most useful container in executable file
formats *High Tech In Earth Space Research* **7**(5) 58-62

[10] Sharikov P I, Krasov A V, Gelfand A M and Kosov N A 2019 Research of the Possibility of Hidden Embedding of a Digital Watermark Using Practical Methods of Channel Steganography *Proc. Int. Symp. on Intelligent and Distributed Computing* (Berlin, Germany: Springer) pp 203-9

[11] Ivanov A V, Krasov A V and Sharikov P I 2018 A study on the technique of hidden digital watermark embedding in class files in virtualized platforms with different architecture *Bulletin of St. Petersburg University of State Fire Service of EMERCOM of Russia* **2** 1-8

[12] http://www.javaportal.ru/articles/comparison_productivity_Java_in_and_Red_Hat_Linux.html

[13] Shterenberg S I, Krasov A V and Ushakov I A 2015 Analysis of using equivalent instructions at the hidden embedding of information into the executable files *Journal of Theoretical and Applied Information Technology* **80**(1) 28

[14] Krasov A V, Arshinov A S and Ushakov I A 2018 Embedding the hidden information into Java byte code based on operands’ interchanging *ARPN Journal of Engineering and Applied Sciences* **13**(8) 2746-52