Method and Utility for Recovering Code Algorithms of Telecommunication Devices for Vulnerability Search

Mikhail Buinevich*, Konstantin Izrailov**
*The Bonch-Bruevich Saint-Petersburg State University of Telecommunications, Russian Federation,
Russia Federation, Saint-Petersburg, 192242, 23-4 Bucharestskaya, apt.41
**The Bonch-Bruevich Saint-Petersburg State University of Telecommunications, Russian Federation,
Russia Federation, Saint-Petersburg, 192212, 17-2 Budapeshtskaya, apt.9
bmv1958@yandex.ru, konstantin.izrailov@mail.ru

Abstract— **The article describes a method for searching vulnerabilities in machine code based on the analysis of its algorithmized representation obtained with the help of an utility being a part of the method. Vulnerability search falls within the field of telecommunication devices. Phase-by-phase description of the method is discussed, as well as the software architecture of the utility and their limitations in terms of application and preliminary effectiveness estimate results. A forecast is given as to developing the method and the utility in the near future.**

Keywords— **binary codes, information security, reverse engineering and decompilation, program language extension, telecommunications**

I. INTRODUCTION

One of the important problems in the field of information security is the use of trusted telecommunications devices [1]. The existence of vulnerabilities in the software (hereinafter – SW) directly affects the ability to ensure the transmission of sensitive information. Despite the obvious relevance of search for such vulnerabilities, suitable methods are absent or limited to a small area of SW application. Successful solution to this problem allows reducing the occurrence of the relevant risks not only inside a single device, but also to the extent of a whole telecommunication system. One of the authors [2] described the interrelationship between vulnerabilities and threats based on artificial neural network.

II. METHOD

One way to solve the above problem is using a method of reverse engineering of the machine code of telecommunications devices (hereinafter – the Method), conceptually outlined in [3]. A specially designed software tool (hereinafter – the Utility) is used for code recovery. Using an algorithmic representation of the code, a specialist in security analyzes it in terms of vulnerabilities search. Herewith some suspicious fragments of code will be labeled automatically by the Utility. The scope of the Method, in principle, is any computer code, but basically it is intended for SW research of telecommunication devices.

A. Description

The main idea of the Method is to combine manual and automated methods of finding vulnerabilities in machine code; each of them has its advantages and drawbacks. Manual method allows using security expert's opinion for accurate detection of vulnerabilities and determination of their type, but it is extremely labor-consuming. Automated method, on the contrary, features a high speed of code processing, but it is less efficient in terms of the number of vulnerabilities detected. The use of a computer-aided tools to get a conception well-suited for further manual analysis by a specialist will be many times more effective than use of any individual method.

The results of the analysis of this representation can be applied by a specialist to create his/her own professional corrective data for the machine code: registers of arguments and return values of functions, names of variables, addresses of global objects, etc. In case of iterative application of the Method, such metadata in input assembly code will allow for more correct code recovery with the automated tool. As a result, effectiveness of the Method will increase.

Note that despite of separate implementation of manual and automated methods, options of their joint application (especially iterative) are virtually absent. Herewith the primary intent of such code recovery utilities is getting compiled source code rather than search for vulnerabilities therein.

B. Phases

The Method consists of three main phases having a fundamentally different purpose and implementation. A way for producing the machine code from the telecommunications devices SW is not included in the Method, being well known and addressing a separate area – getting the binary image of a device.

The scheme of applying the Method is shown in the following diagram.

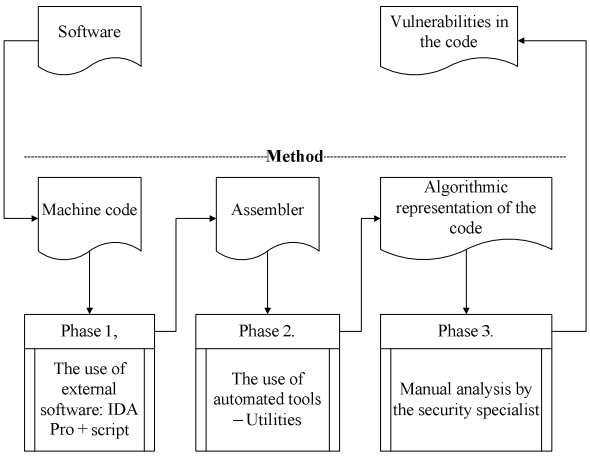

Figure 1. The scheme of applying the Method

As can be clearly seen in the diagram, the consistent application of the phase-by-phase Method converts the machine code of the software to a form suitable for analysis by a security expert who will identify its vulnerabilities.

Description of the phases is given hereinafter.

1) Phase of Method – Converting Binary Representation of the Machine Code into an Assembly Language Code: This phase is a preparatory one, because it is intended to transform the initial form of machine code to a form suitable for automated processing during the next phase.

Initially, the machine code is a linear set of binary data (though sometimes divided into sections of data implementation). Such a set is not suitable for manual processing, and the development of appropriate automated means (as existing ones are not suitable for the vulnerability search) is extremely sophisticated. Although the initial phase of any means of handling the formal input binary data is not particularly difficult (a binary code parser is simpler than a similar text parser), the further development and debugging of such applications are immeasurably more difficult due to the only fact that, in order to write test cases, assembling of text representation in binary one is very likely to be used. At the same time (due to indirect retrieval of the input data for processing and some characteristics of the selected assembler tool), the coverage of the test will certainly be limited. Accordingly, a text representation of the machine code has been chosen as the final form within this phase, being the assembler of the code under examination.

The phase may be implemented using existing tools – disassemblers [4], since the purpose of the latter coincides with the purpose of the phase. With a view of exact matching of the assembler format to the input format of the next phase, a well-known product – disassembler IDA Pro – has been selected [5]. For automation of the process of transformation of the representations, a specially designed script can be used being run in the said product, which generates the assembly code in the required format. Also, the script complements the assembler meta-information created by IDA Pro, which is used during the following phases. The implementation of the script does not depend on the type of the machine code processor, as the product supports disassembly and code processing for a large number of processors. Thus, the phase can be considered as practically independent of the target machine code.

2) Phase of Method – Processing Assembly Machine Code Representation and Its Algorithmization: This phase is the central one of the Method, since the other phases are deemed to be preparatory and final ones. It converts the assembly text of the machine code (having linear and ill-analyzed view) in an algorithmic representation well-suited for analysis by a specialist.

For this type of algorithmized views, the following requirements have been set forth. Firstly, it should be similar to the syntax of the modern high-level programming languages (such as C and C++), because the latter are popular, well apprehended by programmers and designed for efficient execution of SW. Secondly, it shoud be adapted to description of SW algorithms specific to telecommunication devices. And thirdly, it should support detection of potential vulnerabilities. An additional requirement, logically appearing from the above three requirements, should be “dedication” of the final form to description of the nature of algorithms, i.e. it may be free of some irrelevant features of real programming languages, detailed observance of their semantics, strict adherence to the programming paradigms and some features of compilers.

The phase is realized with the help of a specially developed Utility mentioned above. Since the phase consists entirely in the application of this tool, it can be considered as fully automated. Given the significance of the Utility for the Method in general, it will be described in more details in the next section.

3) Phase of Method – Analysis of Algorithmized Representation of the Machine Code and Search for Vulnerabilities: This phase is the final one and is executed by a security specialist who, based on algorithmic representation of the previous phase, manually examines the code for vulnerabilities, defining the nature of their “maliciousness”. In the course of the analysis, the expert can use the following features of the representation. First, the machine code is described therein in terms of code work algorithms and, therefore, vulnerability scanning is carried out more efficiently (faster and encompassing more classes). Secondly, notes on potential vulnerabilities made by the Utility reduce the risk of missing by the specialist of non-obvious vulnerabilities, such as architectural ones [7]. And thirdly, based on research, the expert can make updates to the code (for example, identifying previously unidentified functions, number of their parameters, or objects in memory). When restarting the algorithm phase, it helps the Utility in more correctly restoring the code work algorithms. This feature is implemented with the help of meta-information being placed in its input assembler.

III. UTILITY

As stated above, the Utility is a software tool designed for implementation of automated algorithms for machine code recovery, the optimal target whereof is SW of telecommunication devices. In terms of task and implementation, Utility is similar to a class of software called decompilers [8]. However, the latter (at least based on their name) carry out the process which is reverse with respect to compilation, and, therefore, it seeks to obtain the compilable source code. This apparently minor difference, however, strongly affects the application of the latter for solution of the current task of vulnerability search and the implementation of the Utility. After all, even though the restored source code of SW is acceptable to be studied by a specialist, it still contains much unnecessary information which hinders understanding of the algorithms; and this is without considering the fact that the correct recovery of the source code itself is an extremely difficult problem to be solved, which has many pitfalls and impose the use of erratic decisions. Thus, it is appropriate to develop other software products (such as the Utility) that meet the above requirements.

Currently, a prototype of Utility (hereinafter – the Prototype) has been developed, that supports algorithmization of code for the processor PowerPC (being common in telecommunication devices). There is an operating online version of the Prototype at the appropriate Internet site, including also its description, similar resources and additional information [9].

A. Description

The Utility is designed as a console application, input and output data of which have a textual representation; data input is carried out with the help of the operating system files. The Utility performs the following tasks through sequentially executed steps. Firstly, the parsing of assembly code is performed into an internal representation suitable for computer processing. Then, the internal representation, depending on the type of the input assembler, is translated into a platform-independent one (hereinafter it means independence with respect to processor machine code); in the future, this will allow supporting different types of processors, changing only the implementation of these steps. The resulting universal representation is transformed step-by-step (and iteratively) into a set of specialized representations that describe the code at issue from the following points of view: code operations, control and data flow, structured description of the algorithms, information about the registers modification, lifetime of objects, predicted meta-information (e.g., input and output function parameters, local and global variables, the bit capacity of objects), information about potential vulnerabilities, the final description of the code [10]. Moreover, optimization of the final presentation of the code is made that improves its readability. The final step is generating a description of the output description at the level of detail set out by the user.

B. Input data

Text input is an assembler notation IDA Pro and additional metadata. Example of a function for finding the maximum of three numbers in this format for the PowerPC processor is given in the following listing.

```
// Function of the maximum of three numbers
max3(){     // Function meta-information:
            //   Name, Arguments, Body Begin
  0x00000001: max3:  // { x(r3),y(r4),z(r5)
                     //  ,m(r6),n(r7)
  0x00000002: cmpw r3, r4  //  if(x > y)
  0x00000003: ble label_1  //  {
  0x00000004: mr r6, r3    //   m = x;
  0x00000005: b label_2    //  }
  0x00000006: label_1:     //  else {
  0x00000007: mr r6, r4    //   m = y;
  0x00000008: label_2:     //  }
  0x00000009: cmpw r6, r5  //  if(m > z)
  0x0000000A: ble label_3  //  {
  0x0000000B: mr r7, r6    //   n = m;
  0x0000000C: b label_4    //  }
  0x0000000D: label_3:     //  else {
  0x0000000E: mr r7, r5    //   n = z;
  0x0000000F: label_4:     //  }
  0x00000010: mr r3, r7    //  return n;
  0x00000011: blr          // }
}           // Function meta-information:
            //   Body End
```

This listing (of course, without comments to the source code) is ill-suited for understanding the principles of the code and much less for finding vulnerabilities.

C. Output data

Text output data code is the code having a syntax similar to the C language, but adapted for describing algorithms and free from information irrelevant to their understanding. The format is described in more detail in [11]. Output data obtained from the input data using the Utility from the above example are shown in the following listing.

```
max3(x, y, z) {
  if (x > y) {
    x = y;
  }
  if (x > z) {
    x = z;
  }
  return (x);
}
```

As you can see from this listing, the output data is a good description of the algorithm, implemented using the assembly in the input data.

D. Architecture

Software architecture of the Utility is similar to a classic compiler, although it has some fundamental differences. The work phases of the Utility are the following.

1) Phase of Utility – FrontEnd: This phase is responsible for parsing the input data and building an internal representation while converting it into a platform-independent form, fully ready for software processing. Herein, for example, the CPU instructions are given in the form of a sub-tree, a top node of which contains the type of operation and its properties, and subsidiary nodes store information about instruction operands.

2) Phase of Utility – MiddleEnd: This phase analyzes and converts the data into specialized representation referred to above in the description of the Utility. Platform independence provided by FrontEnd allows implementing the algorithms in terms of generalized concepts (such as a register of specified bit capacity, conditional branching, mathematical operations), leaving out of account the processor features. Of course, totally "disengaging" from the processor is impossible; such a task can be solved by describing the characteristics of the latter in a special metadata section of the input assembler. It is at the last steps of this phase that the prediction of data lost in the process of compiling source code is realized [12], as well as the search for vulnerabilities, because the representation of algorithms code from different perspectives has a form well-suited for software analysis. Then, to improve the readability of the Utility output, this phase optimizes its outgoing data.

3) Phase of Utility – BackEnd: The only purpose of this phase is generation of the output representation of the Utility according to user's options (such as compact form, notes on vulnerabilities, and comments on optimized operations).

E. Limitations

The Method in general and the Utility, and the current Prototype in particular, have a number of limitations as regards their application, which only emphasizes their intended use for a specific task – finding vulnerabilities in machine code. At the moment, the Prototype supports only the PowerPC processor, however, based on the platform independence, the Utility architecture can be extended to a larger number of processors. The Utility output code does not match the syntax of any existing programming languages, as it is not intended for compilation. The code may contain erroneous notes on vulnerability or not have them for existing ones, as the Utility only increases the probability of finding such vulnerabilities, but does not give a 100% guarantee. It should be noted that the concept of vulnerability is subjective, and, as a rule, it can be determined only on the basis of expert opinion by a security specialist – improvement the quality of work of the latter being the main purpose of the Utility (and that of the Method itself).

IV. EVALUATION

The feasibility of the Method was determined by comparing its performance with other methods based on the principle of restoring the original code, a typical software implementation of which belong to the class of decompilers. For a preliminary evaluation of effectiveness, mainly carried out by code recovery included therein (in this case, the Prototype of the Utility and available decompilers), a high-level estimation procedure has been applied as described in the article [13].

Following this procedure, groups of tests have been created, each of them being a separate assembly code with individual vulnerability. Next, the evaluated tools were used for testing the groups while restoring the source code. At the next stage, the code was analyzed by security specialists who detected presence of vulnerabilities and assessed the complexity of their search with the help of numeric coefficient. After creating a table containing the results of the analysis, calculation of the coefficient was carried out, giving an aggregate evaluation of effectiveness of the tool. The result of the comparison of the estimates revealed an evident advantage of Prototype over similar software, allowing the authors to conclude that the Utility (and, consequently, the Method) should be in demand for solving actual problems in the field of information security with its optional positioning in the area of vulnerability research in SW of telecommunication devices.

V. CONCLUSION

Logical development of the work should be the completion of the Utility realization. In such a case, the main criterion for a positive evaluation of the Method should be its effective use to find vulnerabilities in real-world projects. This, in particular, will make possible its use as a strategic tool for ensuring information security [14]. In the future, it is necessary to expand the number of processors supported by the Utility to the set of the most relevant ones, used in modern telecommunication devices – in particular, including MIPS and ARM architectures.

ACKNOWLEDGMENT

In conclusion we have to thank the people who laid the groundwork for, and allowed developing of, the line of investigation touched upon in this article. Firstly, Prof. Sergey Mikhailovich Dotsenko, DSc in Engineering, vice-rector of research at the St. Petersburg State University of Telecommunications (SPbSUT), for the opportunity to participate in research and his faith in its success. Secondly, Andrey Gennadyevich Vladyko, PhD, head of research training unit of SPbSUT, for his assistance in the publication of articles and objective positive evaluation of the scientific activity. And thirdly, a doctoral research student at SPbSUT, Engr. Tiamiyu A. Osuolale from Nigeria, for his assistance in editing and translation of the article into the working language of the conference.

The authors are grateful to their parents for their craving, since the childhood, for painstaking work, knowledge and scientific achievements, repeatedly demonstrated by them on a personal example.

REFERENCES

[1] Tiamiya A. Osuolale, “Analysis of IP routing protocols for trusted routing in the global data networks”, Newspaper of Engecon, 2012, № 8(59), pp. 157–159.

[2] Izrailov K.E., “Prediction model threats of telecommunication systems based on artificial neural network”, Newspaper of Engecon, 2012, № 8(59), pp. 150–153.

[3] Buinevich M.V. and Izrailov K.E., “The method of algorithmization machine code of telecommunications devices”, Telecommunications, 2012, № 12, pp. 2–6.

[4] Wikipedia: Disassembler [Online]. Available: http://wikipedia.org/wiki/Disassembler, Jun, 2013

[5] Site of the IDA Pro [Online]. Available: http://www.idapro.ru/, Jun, 2013

[6] Buinevich M.V. and Izrailov K.E., “The utility of algorithmization machine code of telecommunications devices”, Telecommunications, 2013, № 6, pp. 2–9.

[7] Buinevich M.V. and Izrailov K.E., “Architectural software vulnerabilities”, Sixth congressional research undergraduate and graduate students ”Engecon-2013“, 2013

[8] Wikipedia: Decompiler [Online]. Available: http://wikipedia.org/wiki/Decompiler, Jun, 2013

[9] Site of the utility of algorithmization machine code: Demono [Online]. Available: http://www.demono.ru, Oct, 2013

[10] Izrailov K.E., “The internal representation of a prototype utility for the algorithmization of the code”, Fundamental and applied research in the modern world of Proceedings in The II International Scientific and Practical Conference, 2013, pp. 79–90.

[11] Izrailov K.E., “C-language extension for algorithm description of telecommunication devices code”, Information technology and telecommunications, 2013, № 6, pp. 21–90.

[12] Izrailov K.E., “Representations of the program code from the perspective of metadata”, Fundamental research and innovation in national research universities in The proceedings of the II International Scientific and Practical Conference, 2012

[13] Izrailov K.E., Vasilyeva A.Y. and Romazanov A.I., “Enlarged assessment method of effectiveness of automated tools, recovering source code in search for vulnerability”, Newspaper of Engecon, 2013, № 8(67), pp. 143–146.

[14] Izrailov K.E., “Algorithmization machine code telecommunication device as a means of strategic information security”, National Security and Strategic Planning, 2013, № 2(2), pp. 28–36.

Mikhail Buinevich was born in 1958 in the USSR. He studied to be a military engineer of electronics.

He served in the Navy and government agencies, ensuring data protection; he taught at various universities. His research interests lie in the field of the methodology of information security. He has over 100 publications. Major publications are as follows:

1. Buinevich, M.V. and other, “Ensuring the safety of critical facilities of the Navy from the effects of damaging factors in crisis and emergency situations in peacetime”// Under Ed. Admiral V.S. Vysotsky. – St. Petersburg: Publishing House of ELMOR, 2008.– 300 pp.

2. Buinevich, M.V. and other, “Organizational and technical sustainability of the operation and safety of the public telecommunications network”// Under Society Ed. S.M. Dotsenko. – St. Petersburg: Publishing House of the SPbSUT, 2013.– 142 pp.

Prof. Buinevich, DSc in Engineering (Russian Scientific Degree “Doctor of Technical Sciences”) is currently a Professor of Department of Information Security of Telecommunication Systems at the St. Petersburg State University of Telecommunications (SPbSUT).

Konstantin Izrailov was born in 1979 in St. Petersburg (Russia). In 1996, he graduated from the of Physics and Mechanics Department of the St. Petersburg State Polytechnic University (SPbSTU).

Currently, he is a postgraduate student at the Chair of the Department of Information Security of Telecommunications System of St. Petersburg State University of Telecommunications (SPbSUT). He has published about 10 articles, participated in the execution of 2 scientific researches and holds a patent on a software tool. His research interests are the information security, reverse-engineering (decompilation) and telecommunication devices.

Mr. Izrailov was ranked the best postgraduate student of the year 2012 of SPbSUT and was a Presidential Scholar in 2013.