Semantic model of an exploit’s source code for data protection in automation systems

E V Fedorchenko¹³ and A V Fedorchenko²

¹ Senior Researcher, Laboratory of Computer Security Problems, SPC RAS, SPIIRAS, St. Petersburg, Russia
² Junior Researcher, Laboratory of Computer Security Problems, SPC RAS, SPIIRAS, St. Petersburg, Russia
³E-mail: doynikova@comsec.spb.ru

Abstract. Data protection is especially relevant in automation systems nowadays. The paper proposes a semantic model of exploit’s source code as a basis for unknown attacks detection, forecasting and assessment and therefore for automated data protection from cyber attacks. The proposed model combines a control flow graph with a function call dependencies graph. The paper describes model generation technique that is based on the analysis of exploit’s source code and incorporates code compilation, its decompilation, building the functional semantic models of the exploits, and, finally, comparing these models and generating the standard semantic model of exploits’ source code. Application of the technique for generation of exploit’s model is demonstrated on a case study.

1. Introduction
The challenge of detection, forecasting and assessment of cyber attacks that exploit earlier unknown vulnerabilities (so-called zero day vulnerabilities) is highly relevant for reliability and data protection in automation systems nowadays as soon as they can lead to serious damage [1]. Currently, various methods are used to detect such attacks, including signature-based methods, rule-based methods and heuristic methods. But an efficient solution allowing detection of multiple previously unknown vulnerabilities and their exploitation in cyber attacks still doesn’t exist. This research makes a step towards the novel method of detection, forecasting and assessment of such cyber attacks.

The idea underlying the method under development is that the most new exploits use the parts of already known exploits but in other combinations. Determination of these parts and features of their exploitation, their interconnections, as well as their connections with different types of software and system weaknesses will allow detecting, forecasting and assessing cyber attacks timely. To specify such parts and their interconnections we propose a standard semantic model of an exploit’s source code in this research. This model is a variation of semantic graph, and combines a control flow graph with a function call dependencies graph. The semantic graph incorporates the concepts and relationships between them. In our case the concepts are the parts of exploit’s source codes. The control flow graph is a directed graph where nodes represent basic blocks of source code and edges connect nodes that can be executed consequentially [2, 3]. The function call dependencies graph is a directed graph where nodes represent functions and edges represent function calls [4].
The proposed semantic model is the basis for further calculation of probabilities of transition from one source code part to another, for mapping of the features code running detected while software/system operation, mapping them to the semantic model an forecasting of the further cyber attack steps.

2. Semantic model of an exploit’s source code

In this section we analyze related works in the area, describe the developed semantic graph, and provide a case study.

2.1. Related research

Currently, there are signature-based and rule-based methods for cyber attacks detection, forecasting and assessment. These methods are focused on known vulnerabilities and exploits and do not allow detecting of exploitation of zero day vulnerabilities.

Traditionally penetration testing, reverse engineering and fuzzing of the software and hardware are used to detect unknown vulnerabilities. But efficiency of these methods directly depends on expert qualifications and they are time consuming.

For the last years analysis of a source code using heuristic methods is being evolved. In this case potentially harmful files are executed in the isolated environment ("sandbox") to generate program execution traces and to compare them with normal behavior [5, 6]. But in this case the generalized models of normal behavior and subjective features are used that decrease accuracy of the results and access to the potentially harmful files is required. The developed semantic model of exploits will allow generating of objective external features of vulnerabilities (including unknown vulnerabilities) exploitation from the model itself (i.e. function calls, related vulnerabilities and weaknesses, probabilities of transition between the function calls) and from the external sources analysed while exploits’ code execution (namely, event logs and network traffic).

As soon as the proposed semantic model is based on the exploit’s source code analysis we review existing models for source code representation below, including an abstract syntax tree, an abstract semantic graph, a control flow graph, a program dependence graph, and a code property graph.

The abstract syntax tree represents the process of language sentences construction using the language grammar from which some elements are removed [7]. The abstract semantic graph differs from the abstract syntax tree by the fact that while its generation semantic information is considered [7, 8]. Within the control flow graph each node corresponds to a basic block, while edges connect nodes that can be executed consequentially [3]. The program dependence graph represents the data and control dependencies within a program [2, 9]. The code property graph is a combination of abstract syntax tree, control flow graph and program dependence graph [10]. The considered models do not satisfy our requirements to the exploit’s source code model, namely need for a formalized description of the code functionality suitable for further features extraction. Thus, we developed our own model based on the control flow graph and the function call dependencies graph.

2.2. Model generation technique

To generate the standard semantic model of exploits’ source code we:

- compile the source code of separate exploits first;
- decompile it;
- build the functional semantic models of the exploits;
- compare these models and generate the standard semantic model of exploits’ source code that is the basis for feature extraction.

We implement the first two steps, namely, compilation and decompilation of the exploit source code to exclude exploits with syntax errors in the source code.
We propose the exploit’s source code semantic model where the nodes are represented with the “names” of imported modules and their functions and the edges specify the sequence of function calls for these modules. We generate the proposed model on the basis of control flow graph where nodes are replaced with functions and classes of objects of imported modules (conditionally global names). To construct the control flow graph we search conditional and unconditional, relative and absolute jumps, and exception handler instructions in the decompiled code.

Finally, to generate the standard semantic model of exploits’ source code we compare the models of different exploits and search for linked name pairs to join them.

2.3. Case study

To implement and analyze the proposed model we used the exploit database “Exploit DataBase” (EDB) [11]. EDB contains the following types of exploits [2]:

- 4292 webapps;
- 1829 dos;
- 1078 local;
- 1030 remote, and
- 287 shellcode.

The statistics of exploits in EDB considering target platform is as follows [2]:

- 531 exploits for the hardware platform;
- 3610 exploits for the operational system;
- 132 exploits for the software platform;
- 3432 exploits for the WEB platform, and
- 1646 undefined.

The most used language for the exploits provided in EDB is Python, thus we use the exploit of this type in the case study below.

We used the exploit with identifier “30688” from EDB for the case study. We generated the control flow graph first (figure 1). We simplified it to the single route from Entry Point (EP in figure 1) to the Implicit Return (IR in figure 1). In figure 1 the nodes represent the code blocks, while the numbers in the nodes represent an offset of the first instruction of each block. The regular program execution is highlighted with bold in figure 1.

Figure 1. Control flow graph for exploit with identifier “30688” from EDB.
To generate the semantic functional model of the exploit’s source code we replaced the nodes with the functions and classes of objects of imported modules (figure 2). The resulting model reflects an order and call and functional dependencies of calls of imported names. There are names of two functions (urlencode() and urlopen()) and of one class constructor (Request) within the model in figure 2.

The proposed model can be further applied to extract objective features for detection, forecasting and assessment of cyber attacks exploiting unknown vulnerabilities and therefore for automated data protection from cyber attacks.

![Figure 2. Exploit’s source code model.](image)

3. Conclusion
In the paper the standard semantic exploit’s source code model was introduced. The proposed model differs from the existing models by the fact that it provides formalized description of the code functionality suitable for further features extraction. The technique for its generation was provided. The application of the technique to generate the exploit’s source code model was demonstrated on the example. The developed model can be used to extract objective features for detection, forecasting and assessment of cyber attacks exploiting unknown vulnerabilities that is the direction of future work. The component for attack detection, forecasting and assessment constructed using the proposed model can be integrated with the security monitoring systems to increase their efficiency and therefore for automated data protection from cyber attacks.

References
[1] ARC Advisory Group and Kasersky 2020 The state of industrial cybersecurity in the era of digitalization Retrieved 1 November 2020 https://ics-cert.kaspersky.com/media/Kaspersky_ARC_ICS-2020-Trend-Report.pdf
[2] Fedorchenko A, Doynikova E and Kottenko I 2019 Determination of System Weaknesses based on the Analysis of Vulnerability Indexes and the Source Code of Exploits Journal of Universal Computer Science 25(9) 1043-65
[3] Patterson E, Baldini I, Mojsilovic’ A and Varshney K R 2018 Representation and Analysis of Software Proc. of the Twenty-Seventh International Joint Conference on Artificial Intelligence 5847-9
[4] Code2graph: Automatic Generation of Static Call Graphs for Python Source Code 2018 Proc. of the 33rd IEEE/ACM International Conference on Automated Software Engineering 880-3
[5] Firdausi I, Erwin A and Nugroho A S 2010 Analysis of machine learning techniques used in behavior-based malware detection Proc. of the second international conference on advances in computing, control, and telecommunication technologies 201-3
[6] Singh J and Singh J 2020 Detection of malicious software by analyzing the behavioral artifacts using machine learning algorithms Information and Software Technology 121
[7] Duffy E 2011 The Design & Implementation of an Abstract Semantic Graph for Statement-Level Dynamic Analysis of C++ Applications All Dissertations
[8] Neamtiu I, Foster J S and Hicks M 2005 Understanding source code evolution using abstract syntax tree matching ACM SIGSOFT Software Engineering Notes 30(4) 1-5
[9] Horwitz S and Reps Th 1992 The use of program dependence graphs in software engineering Proc. of the 14th international conference on Software engineering 392-411
[10] Yamaguchi F, Golde N, Arp D and Rieck K 2014 Modeling and discovering vulnerabilities with code property graphs Proc. of the IEEE Symposium on Security and Privacy 590-604
[11] Exploit Database Retrieved 7 September 2018 https://www.exploit-db.com/