Analysis of complexity metrics of a software code for obfuscating transformations of an executable code

Analysis of complexity metrics of a software code for obfuscating transformations of an executable code*

M A Kuznetsov, V O Surkov
Reshetnev Siberian State Aerospace University
31 “KrasnoyarskiyRabochiy” prospect, Krasnoyarsk, 660014, Russia.

E-mail: programist1024@yandex.ru

Abstract. The complexity metrics of the software code applicable for the analysis of binary files, as well as the methods used at overlaying the obfuscating transformations of these files are considered in the article. The results of the research of the metrics dependence on the performed by special software products obfuscating transformations of a vulnerable binary file are presented. The research confirms the applicability of the technique of the automated vulnerabilities search in binary files, whose analysis is difficult due to obfuscating transformations overlay.

Modern software tends to more effective problem solving in all spheres of human activity, including the aim of increasing the demand for the product. Together with this desire the product internal design is changing, as a rule towards more complex mechanisms and logic. Such changes lead to an increase in the number of lines of the program source code which in its turn produces more errors in the program. According to the research conducted by Coverity Company with increasing the code base the average number of errors / vulnerabilities grows in 741 products with an open source code (Table 1). The only exceptions are projects with a number of lines of more than 1 million, which may be due both to a statistical error and increasing the code quality requirements for large projects (more stringent requirements for code coverage, use of static and dynamic tools for source code analysis). [1]

Table 1. The statistics of a number of errors/ vulnerabilities for programs in C/C++.

Size of the code base	Average number of errors/ vulnerabilities per 1000 lines of the code
Less than 100 000	0,35
From 100 000 to 500 000	0,5
From 500 000 to 1 000 000	0,7
More than 1 000 000	0,65
Average in all projects (252 010 232 lines of the code)	0,59

The presence of errors in a software product allows causing significant damage by such actions as the unauthorized copying of programs, their illegal distribution and use, thereby reducing its quality, until the complete cessation of its operation, which ultimately causes

* This work supports by the president's grant for young Russian scientists. Contract No. 14.Z56.15.6012.

significant material damage to the manufacturers of software. This fact gives rise to the need to use different methods of analysis and evaluation of the program code in order to detect errors in the source code.

To date, one of the most popular tools used to assess the program code are metrics. They are a set of software assessments that enable developers and project managers to get an idea of a developing or existing product: to understand which methods need re-working or additional testing; to see the potential risks; to understand the current state of the project, to track the progress of work in the software development process. [2]

A typical scenario of using metrics in the development and modification of the software is as follows:

1. A set of measures that characterize the metrics complexity, the laboriousness of its design and analysis, style, reliability, etc. in accordance with the standards that are adopted in a company or an organization is calculated for an existing program in accordance with metrics accepted for its evaluation. It is assumed that the existing code corresponds to the reference values of the used set of measures.

2. After making changes to a program the evaluation of its metrics performance is re-executed. New values of the measures are compared with both the reference values and the characteristics of the program before its modification. Thus, not only checking of a new code for compliance with the developer standards is performed, but also the effects of changes on the previously achieved performance (complexity, reliability, clarity, etc.) of a software product are evaluated.

3. The decision on the possibility and feasibility of introducing a new code into the project is made. The effort and personal contribution of the developers to making changes in the program are evaluated.

During the development of programs and evaluation of developers' efforts some metrics (especially quantitative ones) may serve rather as testimonials as all sorts of tricks on the part of not quite bona fide developers are possible with the aim of both to minimize and maximize some used measures (for example, one and the same code structure can be written in different number of lines and steps). At the same time, little quantitative evaluation can indicate not a lack of effort, but the complexity of the performed work – both the development or modification and the analysis of programs. For example, searching for an error in the code snippet may take a long time, and correction of the detected errors may be reduced to a single line of code. The latter problem can be largely solved by the use of the quantitative metrics instead of complexity metrics to assess the complexity of the problem to be solved by the developer. [4]

Application analysis task in the absence of source codes sets slightly different requirements to the choice of code metrics than the development and modification of applications as a number of quantitative metrics mainly those used for the evaluation of project labor costs is based on the characteristics of the source code.

In [4-12] a number of metrics used to evaluate the complexity of the binary code is considered.

Quantitative metrics:

- File size [4];
- the number of lines of a source code [4];
- the average number of lines for functions [4];
- total number of functions [4];

- Jilb metrics [5];
- cyclomatic complexity [6];
- Halstead metrics [7];
- ABC-metrics [4];

Metrics of complexity of the control flow:

- McCabe metrics [5];
- Pivovarsky metrics [5];
- Harrison and Magel metrics [8];
- boundary values metrics [5].

Metrics of data stream complexity evaluation:

- span metrics [9];
- Henry and Kafura metrics [9];
- Card and Glass metrics [10];
- Oviedo metrics [11];
- Chapin metrics [12];
- metrics of appeal to global variables [4].

Hybrid metrics:

- Cocol metrics [5];

It should be noted that the evaluation complexity metrics are not limited to the above mentioned ones. There is a large set of object-oriented metrics, the metrics that take into account previous changes or observations in the code that are not applicable in the framework of a binary code analysis. [4-12]

Besides the absence of the source codes, the software application analysis can be complicated by special obfuscating transformations the use of which is due to the protection of intellectual property of developers, who are usually not interested in publishing or transmitting their designs such as the authoring algorithm and the key data used in the program to third parties. This fact contributes to the appearance of random errors and vulnerabilities, as well as software bugs deliberately included in the program code. Such state of affairs explains the urgency of the problem of finding vulnerabilities in binary files

For reasons stipulated by the influence of various types of protection on the efficiency of code execution, these protective mechanisms are used heterogeneously. Critical in terms of security areas whose implementation time is relatively small are protected with the most powerful means; fragments that are important, but critical in terms of counting time are protected by less costly transformations. A considerable part of the application which does not contain an important or unique code and is of no interest for analysis may be generally free from any protection. It seems expedient to choose a set of metrics applicable to both software application units (functions and modules) and arbitrary application fragments of different lengths (sequences of listing instructions, performance routes and subroutes of selected algorithms). These metrics should quite clearly characterize the code fragments according to the presence of obfuscation and other security mechanisms in them, and accurately locate the differently protected areas within the analyzed fragments. The applicability of metrics to sufficiently short code or route fragments should allow building profiles of complexity and security of the analyzed code, allowing only localizing protective equipment in the application code. [4]

Such techniques as data obfuscation and control obfuscation are used for binary files obfuscation.

Work [4] examines these techniques in terms of their impact on the previously listed code metrics. We will briefly mention the effect of different methods of binary files obfuscation on the software code metrics.

Data obfuscation:

– conversion of data placement and encoding (Chapin metric, span metric, Kafura metrics, Cocol metrics, McCabe metrics);
– conversion of data aggregation (data structure complexity metrics);
– conversion of data sorting (control complexity metrics).
– Control flow obfuscation:
– aggregation conversion (all complexity measures of the control flow based on the assessment of cyclomatic complexity, ABC metrics, Jilb metrics);
– calculations conversion (control flow metrics);
– opaque predicates (McCabe metrics, Myers metrics, Hansen metrics, Harrison and Magel metrics, Pivovarsky metrics, Jilb metrics);
– tabular interpretation (control flow complexity metrics).

For automatic analysis and searching for vulnerabilities in binary files which contain confusing transformations it is necessary to study the behavior of code metrics during the above mentioned transformations. To solve this problem we conducted the experiment in which with the help of software package (idametrics plugin for disassembler IDA Pro) [13-15] metrics have been received for a sample of 10 binary files containing a vulnerability (an open database of vulnerable applications exploit-db [16] was used as a data source) prior to and after obfuscating transformations.

To perform transformations, the analysis of existing means of binary file obfuscating was carried out in which the following software tools have been chosen (as they are free software or have a trial period of use): VMProtectUltimate 3.0.6 Demo, TheEnigmaProtector 4.40, yodaProtector 1.03.

The presentation of the results for each of the metrics is not possible in a single article. However, the behavior of 18 of the 20 metrics is similar; therefore we restrict ourselves to only some of the results of metrics. We consider the behavior of the metrics: the file size, Halstead B and ABC (Tables 2-4 and Figures 1-3).

The data in Table 2 and Figure 1 provides insights about the original size and the size received after the transformation of the binary files sample. This result does not allow drawing conclusions about the behavior of given metrics.

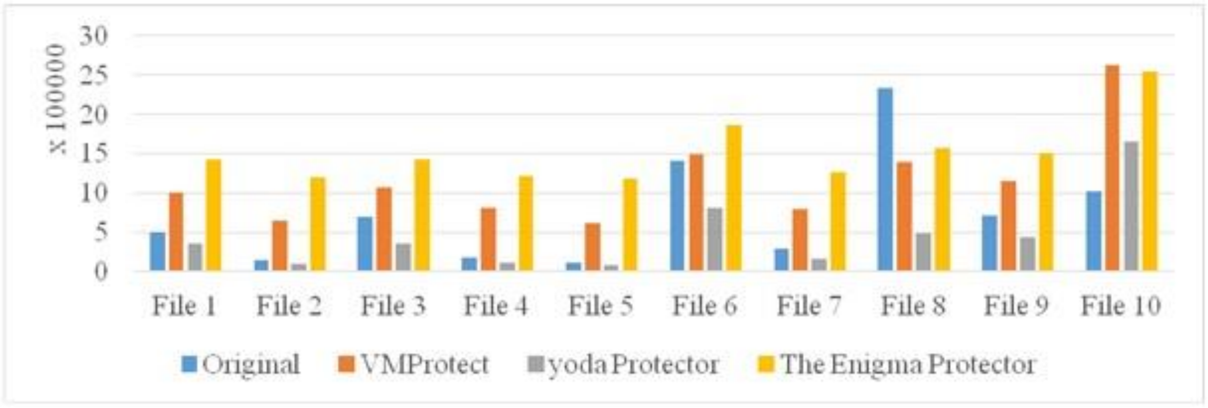

Figure 1. Metrics: Size of a file

Table 2. Metrics: Size of a file.

File Name	Original	VMProtect	yodaProtector	TheEnigmaProtector
File 1	507904	999424	356352	1421312
File 2	139264	638976	98304	1200128
File 3	688128	1069056	352256	1421312
File 4	176128	802816	118784	1216512
File 5	110592	614400	77824	1179648
File 6	1406280	1495040	815104	1856840
File 7	282624	790528	163840	1265664
File 8	2334720	1384448	483328	1568768
File 9	704512	1142784	438272	1503232
File 10	1024000	2625536	1646592	2547712

The data in Table 3 and Figure 2 provides insights about the source ABC-metrics value and the received one after the binary files sample transformations. This result leads to the conclusion about the behavior of the metrics during obfuscating transformations of a binary file: the value of the metrics is reduced by orders of magnitude.

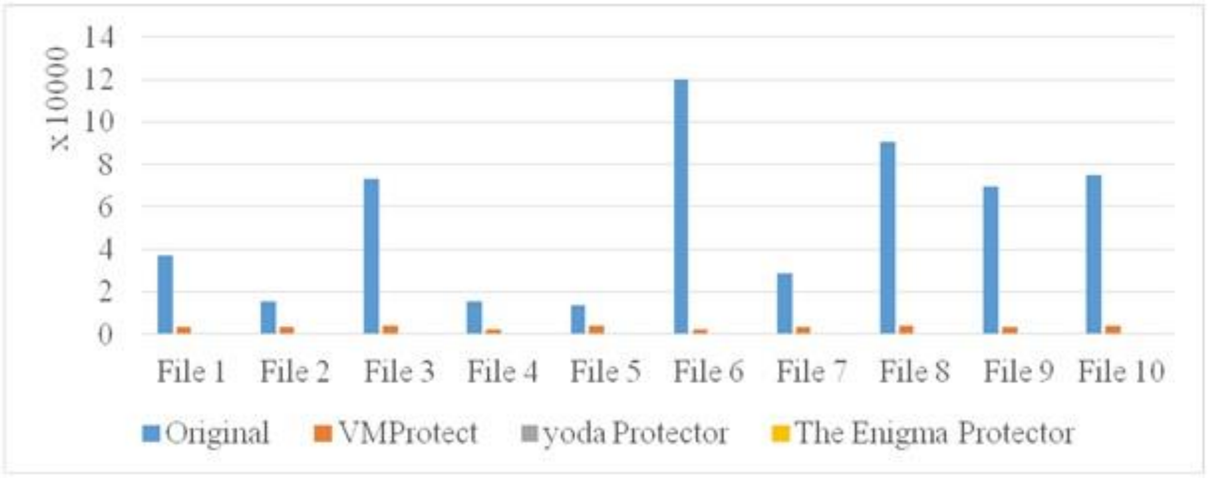

Figure 2. Metrics: ABC-Metrics.

Table 3. Metrics: ABC-metrics.

File name	Original	VMProtect	yodaProtector	TheEnigmaProtector
File 1	36758,57	3639,49	28,72	45,88
File 2	15115,50	3278,71	28,72	26,79
File 3	73331,94	4123,88	28,72	77,93
File 4	15187,43	2340,86	28,72	88,26
File 5	13818,99	4039,84	28,72	16,41
File 6	119812,77	2469,51	34,72	106,31
File 7	28792,80	3159,90	28,72	85,56
File 8	90420,16	3968,47	28,72	38,09
File 9	69389,09	3639,13	28,72	14,87

In addition, let us consider Halstead metrics (data in Table 4 and Figure 3). This metrics has a behavior similar to the results obtained for the ABC-metrics.

As a result of the pairwise comparison of metrics values for a binary file and metrics for the binary file with the presence of obfuscating changes, the following conclusion was drawn: 18 of 20 metrics of a binary file with the presence of obfuscating transformations have a clear decrease in value. As for the other two metrics: the file size, the average number of lines of the function code – the behavior is not clear.

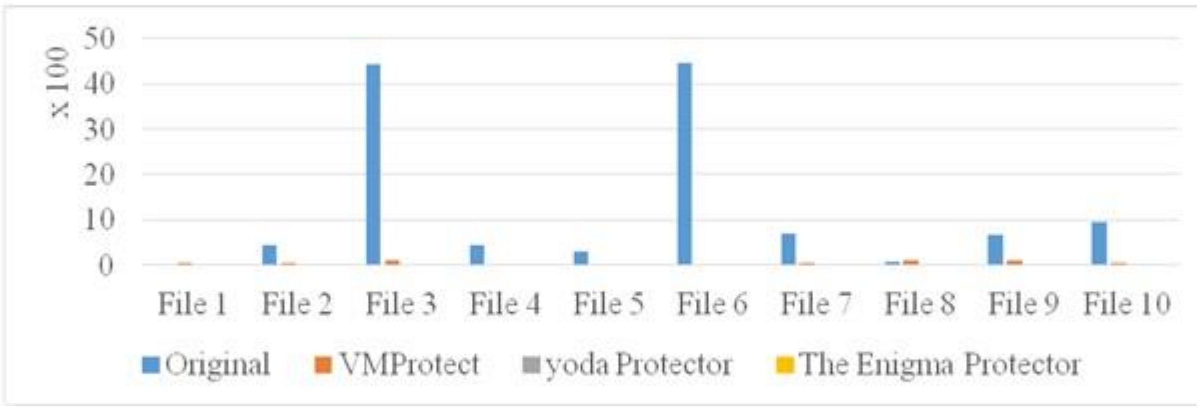

Figure 3. Metrics: Halstead metrics

Table 4. Metrics: Halstead metrics.

File name	Original	VMProtect	yodaProtector	TheEnigmaProtector
File 1	10,22	55,66	0,18	0,40
File 2	434,28	37,35	0,18	0,18
File 3	4425,78	93,88	0,18	0,67
File 4	454,93	7,43	0,18	0,75
File 5	294,46	6,62	0,18	0,08
File 6	4466,61	6,02	0,24	0,20
File 7	704,62	59,43	0,18	0,61
File 8	79,53	97,59	0,18	0,23
File 9	672,28	96,23	0,18	0,06
File 10	945,90	42,80	0,18	0,23

To obtain more information on binary metrics changes in the process of obfuscating transformations let us conduct another study in which we compare the values of code metrics for each function of the binary file before and after transformations.

Let us consider the study results illustrated by the example of the vulnerable function of a binary file (Table 5 as well as Figure 4 show the change of metrics after file transformation).

On the basis of research results data (Table 5) it can be seen that the change in the metrics values: H.B, ABC, H.V, Assign, Cocol, LOC, H.D, R is within 13% (in particular H.B metrics – 5.82% and ABC – 12.53%).

In [13] code metrics are used in order to prioritize the search for vulnerabilities in one or another function of a binary file. The effectiveness of each metrics in the analysis of the file is determined by examining the effectiveness of each metric in searching for vulnerabilities. The results of the study are shown in Figure 6.

Table 5. Analysis of changes in metrics of a vulnerable function of a binary file at overlaying the obfuscating transformations.

Metrics name	Obfuscating transformations		
	Before	After	% of change
H.B	0,287049	0,270337	-5,82%
ABC	35,49648	31,04835	-12,53%
H.N*	240,7106	315,5177	31,08%
H.V	935,9502	962,3421	2,82%
Assign	34	30	-11,76%
Cocol	77,28705	74,27034	-3,90%
LOC	68	69	1,47%
Harr	401	140	-65,09%
H.D.	27	24	-11,11%
H&C	45	6	-86,67%
Ovideo	40	17	-57,50%
Span	25	9	-64,00%
Bound	130	15	-88,46%
Pi	131	15	-88,55%
BBLS	18	10	-44,44%
CC	9	5	-44,44%
Chapin	21	12	-42,86%
C&S	4	0	-100,00%
R	1	1	0,00%
Jilb	0,176471	0,115942	-34,30%

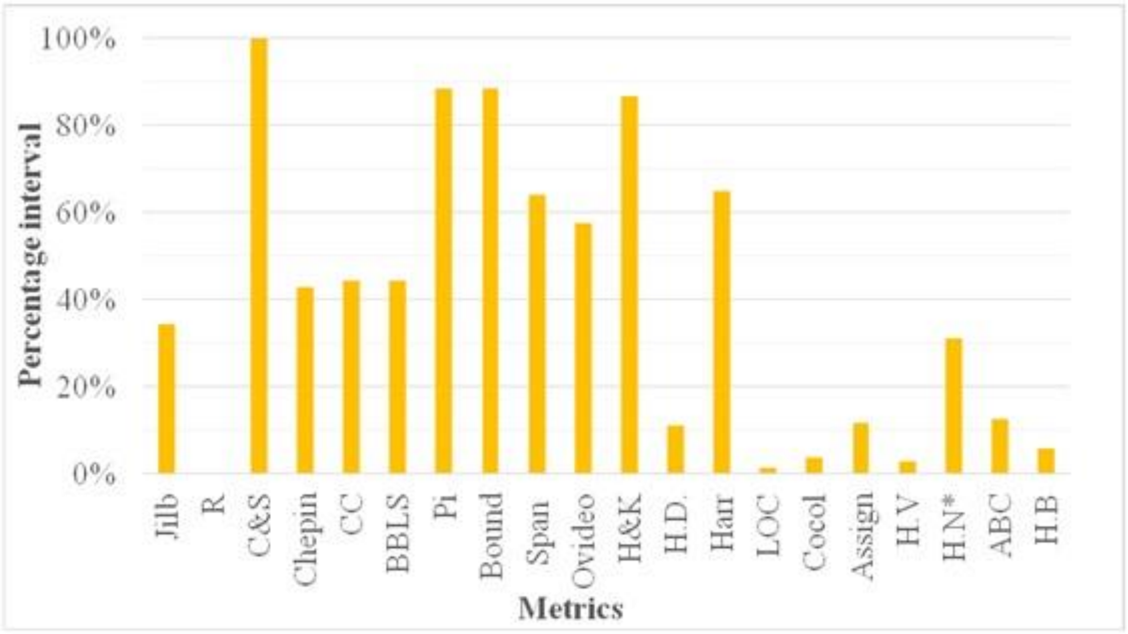

Figure 4. Graph of absolute metrics values change when applying obfuscating transformations.

Table 5 and Figure 4 use the following symbols for the metrics: H.B, H.D, H.V, H.N* – Halstead metrics B, D, V and N; ABC – ABC metrics; Assign – metrics of the number of assignments of values to variables; Cocol – Cocol metrics; LOC – metrics of the number of code lines; Harr – Harrison metrics; H & C – Henry and Kafura metrics; Oviedo – Oviedo metrics; Span – Span metrics; Bound – metrics of boundary values; Pi – Pivovarsky metrics; BBLS – metrics of the number of basic blocks; CC – cyclomatic complexity; Chapin – Chapin metrics; C & S – Card and Glass metrics; R – metrics of ratio of the graph arcs number to the number of base blocks; Jilb – Jilb metrics.

Metrics marked according to the results of the analysis in Table 5 (H.B, ABC, H.V, Assign, Cocol, LOC, H.D., except for R metrics) are included in the 10 most effective ones (on the basis of the data in Table 5 and Figure 5).

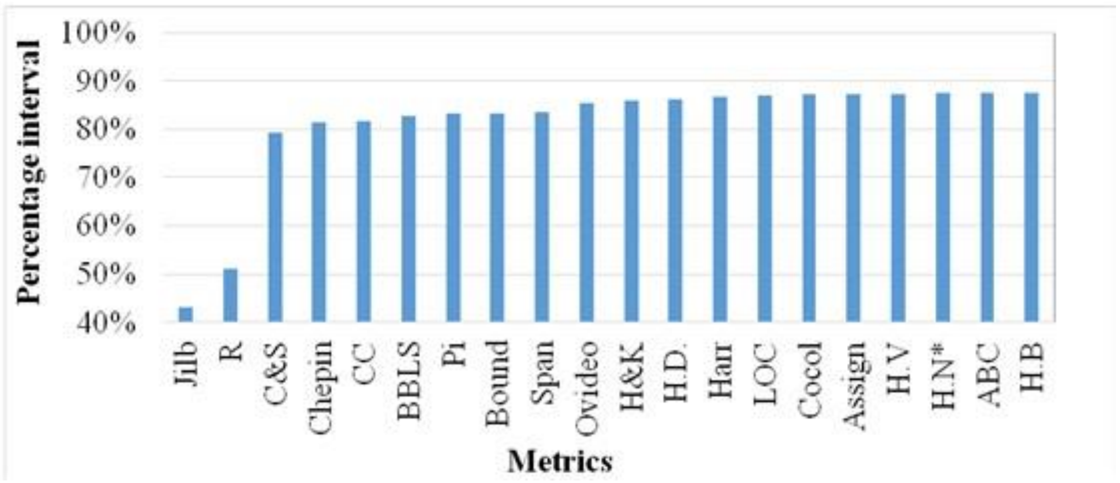

Figure 5. Effectiveness of code metrics in searching for a vulnerable function of a binary file.

It is worth noting that in order to increase the reliability of the study results it is required to conduct research on a bigger experimental sample. However, the received results allow us to draw a preliminary conclusion about insignificant dependence of complexity metrics of a software code of a vulnerable function: changes in the values of the metrics most effective in searching for vulnerabilities (H.B, ABC, H.V, Assign, Cocol, LOC, H.D.) are within 13%.

Such a result (with some adjustment for the change of metrics – 13%) allows considering the possibility of using the method of automated searching for vulnerabilities in the executable code, proposed in [13], for automated searching for vulnerabilities in binary files, which are difficult to analyze due to overlay of obfuscating transformations.

References

[1] CoverityScan: 2013 OpenSourceReport. http://softwareintegrity.coverity.com.
[2] Code Metrics Values https://msdn.microsoft.com/en-us/library/
bb385914.aspx.
[3] Software code metrics http://www.viva64.com/en/a/0045.
[4] Software complexity metrics. Technical reporthttp://www.ispras.ru/preprints/docs/prep_25_2013.pdf.
[5] Abran A 2010 Software Metrics and Software Metrology Hoboken, NJ: Wiley-IEEE Computer Society

[6] McCabe T J 1976 A complexity measure. IEEE Transactions on Software Engineering, (4), 308-320
[7] Halstead M H 1977 Elements of Software Science. Amsterdam: Elsevier North-Holland Inc 127 pp
[8] Harrison W A, Magel K I 1981 A complexity measure based on nesting level. ACM Sigplan Notices, 16(3), 63-74
[9] Henry S, Kafura D 1981 Software structure metrics based on information flow. IEEE Transactions on Software Engineering, (5), 510-518
[10] Card D, Glass R 1990 Measuring Software Design Quality. Prentice Hall
[11] Oviedo E I 1993 Control flow, data flow and program complexity. In Software engineering metrics I (pp. 52-65). McGraw-Hill, Inc
[12] Chapin N 1989 An entropy metric for software maintainability. In System Sciences. Vol. II: Software Track, Proceedings of the Twenty-Second Annual Hawaii International Conference on (Vol. 2, pp. 522-523). IEEE
[13] Shudrak M, Zolotarev V 2016 Improving fuzzing using software complexity metrics *Lecture Notes in Computer Science* Volume **9558** pp. 246-261
[14] Shudrak M, Zolotarev V 2012 The new technique of decompilation and its application in information security *UKSim 6th European Symposium on Computer Modeling and Simulation*. Valletta, Malta. pp. 115 – 120
[15] Shudrak M, Zolotarev V 2013 The technique of dynamic binary analysis and its application in the information security sphere *In Proceedings of IEEE EuroCon*, Zagreb, Croatia. pp. 112-118
[16] Vulnerable applications and exploits database: http://www.exploit-db.com/.