Journal of Universal Computer Science, vol. 25, no. 9 (2019), 1043-1065
submitted: 16/1/19, accepted: 9/7/19, appeared: 28/9/19 © J.UCS

Determination of System Weaknesses based on the Analysis of Vulnerability Indexes and the Source Code of Exploits

Andrey Fedorchenko, Elena Doynikova, Igor Kotenko
(St. Petersburg Institute for Informatics and Automation
of the Russian Academy of Sciences, St. Petersburg, Russia
{fedorchenko, doynikova, ivkote}@comsec.spb.ru)

Abstract: Currently the problem of monitoring the security of information systems is highly relevant. One of the important security monitoring tasks is to automate the process of determination of the system weaknesses for their further elimination. The paper considers the techniques for analysis of vulnerability indexes and exploit source code, as well as their subsequent classification. The suggested approach uses open security sources and incorporates two techniques, depending on the available security data. The first technique is based on the analysis of publicly available vulnerability indexes of the Common Vulnerability Scoring System for vulnerability classification by weaknesses. The second one complements the first one in case if there are exploits but there are no associated vulnerabilities and therefore the indexes for classification are absent. It is based on the analysis of the exploit source code for the features, i.e. indexes, using graph models. The extracted indexes are further used for weakness determination using the first technique. The paper provides the experiments demonstrating an effectiveness and potential of the developed techniques. The obtained results and the methods for their enhancement are discussed.

Keywords: Vulnerability analysis, Exploit analysis, Vulnerability metrics, Abstract semantic graphs, Data mining, Data classification, Open security data sources, Security monitoring.
Categories: H.3.1, H.3.2, H.3.3, H.3.7, H.5.1

1 Introduction

One of the important security monitoring tasks is to automate the process of determination of infrastructure weaknesses based on known vulnerabilities and exploits of the analyzed system. According to the "Common Weakness Enumeration" (CWE) glossary a weakness is a "type of mistake in software", while a vulnerability is "an occurrence of a weakness (or multiple weaknesses) within software" [CWE Glossary, 2019]. An exploit is software or a sequence of commands using vulnerabilities to implement cyber attacks.

This direction is selected because in the modern information society the task of countering cyber attacks is highly relevant. Obviously, manual security data processing in an acceptable time is impossible for modern information systems and appropriate data sets. The modern tools for information security monitoring provide functionalities for detection of known vulnerabilities in the analyzed system over time. These tools include, in particular, security scanners Nessus [Nessus, 2018] and Nmap [NMap, 2018]. They allow vulnerability detection using open vulnerability databases, such as “Common Vulnerabilities and Exposures” (CVE) [CVE, 2018], “National Vulnerabilities Database” (NVD) [NVD, 2018], etc. The vulnerability databases contain both all known vulnerabilities and information on their presence in a particular software and hardware, i.e. they contain relations between the software and hardware represented in the uniform format “Common Platform Enumeration” (CPE) [CPE, 2019], and vulnerabilities represented in the uniform format CVE [CVE, 2018].

It is impossible to eliminate all vulnerabilities in a system because of the complexity and cost of this process. A more reasonable approach is to eliminate weaknesses of the system that lead to appearance (usage) of vulnerabilities. It can be done via modification of system configuration, via modification of security policy or via implementation of appropriate security tools. The source of data on the known weaknesses is weakness database CWE [CWE, 2018]. But, to the best of our knowledge, there is no database that includes information on the direct links between the software and hardware represented in the uniform format CPE and weaknesses represented in the uniform format CWE. Thus, to determine the weaknesses of information system, considering its software and hardware, we solve the task of vulnerability classification by weaknesses. It allows indirect linking of the software and hardware with their weaknesses via their vulnerabilities. The most obvious approach to compete the task is to use the NVD database to determine the links between the vulnerabilities and weaknesses of the analyzed system. Vulnerability entries in NVD have links to the weaknesses in CWE. Main challenges that prevent automated mapping of weaknesses and vulnerabilities using links between NVD and CWE consist in the absence of links to the weaknesses for some vulnerability instances in NVD, and in the high level of abstraction of the linked CWE instances. To overcome these challenges we propose to use data classification methods to link vulnerabilities and weaknesses. We tested several methods and selected the one that shown the highest accuracy. We selected vulnerability indexes specified by the vulnerability scoring system CVSS (“Common Vulnerability Scoring System”) of version 2.0 [Mell et al., 2007] and 3.0 [FIRST, 2015] as the classification features because NVD contains their values for vulnerabilities. We analyzed in details the vulnerability scoring system CVSS of version 2.0 and 3.0, and the indexes for vulnerabilities specified by this system. In the process of analysis we detected some features and inaccuracies of these systems and acceptable (but not ideal) classification accuracy. This approach and the related technique were described in details in the paper presented on the IWCC’2018 workshop held in conjunction with the ARES’2018 [Doynikova et al., 2018].

In this paper we extend the previous research proposing the second technique to compete the task. It concerns to the case when there are exploits but there are no associated vulnerabilities and therefore the indexes for classification are absent. It is based on the analysis of the exploit source code for the features, i.e. indexes, and their extraction using graph models. The extracted indexes are further used as input data for the weakness determination using the first technique. Known exploits for the software can be found in the exploit databases (for example, “Exploit DataBase”, EDB [EDB, 2018]). The EDB database also contains links from some exploit entries to the related CVE vulnerabilities. This technique can be used both (1) to determine the used weakness for the exploit, if the entry in the EDB database does not contain the link to the CVE vulnerability, and, potentially, (2) to enhance the classification accuracy of the first technique, if the entry in the EDB database contains the link to the CVE vulnerability.

The main contribution of this research is the approach to automated weaknesses determination including the technique based on known system vulnerabilities and their indexes, and the technique based on the available exploits and their features. Automation of vulnerability classification process by weaknesses can help experts to specify links to weaknesses for new detected vulnerabilities and to avoid subjectivity of manual specification. Besides, the proposed approach can help to determine weaknesses for the exploits without links to the vulnerabilities that can help to enhance attack models [Doynikova and Kotenko 2018]. We also believe that further extension of the set of features for classification using the analysis of the exploit source code will help to overcome the high level of abstraction of the linked to the vulnerabilities CWE instances and to increase the mapping accuracy.

The paper is organized as follows. *Section 2* reviews related works. *Section 3* describes the proposed approach and the developed techniques for the weaknesses determination, including the technique based on analysis of the CVSS indexes and the technique based on analysis of the exploit source code. *Section 4* outlines the data for analysis, the data preprocessing stage, and the experiments. The paper ends with discussion, conclusion and future work.

2 Related Work

In the research we use open security data sources. Currently different open data sources are used to solve various tasks of security analysis. The data on vulnerabilities, products and configurations from the CVE [CVE, 2018], NVD [NVD, 2018] and other databases are used for vulnerability assessment [Chang et al., 2011; Wu and Wang, 2011], attack generation [Aksu et al., 2018], risk management [Das et al., 2012; Houmb and Franqueira, 2009; Radack and Kuhn, 2011; Zhang et al., 2015], checking for compliance to the security standards such as FISMA [FISMA, 2019], FDCC [FDCC, 2019], PCI DSS [PCI DSS, 2016] and security testing. Besides, vulnerability sources are used by the vulnerability scanners [Nessus, 2018].

The data on attack patterns from the CAPEC database can be used to develop security policies, define security requirements, analyze security risks, and test security. In [Pauli and Engebretson, 2008] the tool for secure system development using CAPEC is proposed. The data on the weaknesses from the CWE database is used for design and development of secure systems [Son et al., 2015], and for penetration testing. In multiple research works, various security sources (such as CVE, CPE, CWE, and CAPEC) are used jointly to construct knowledge databases for risk analysis [Gamal et al., 2011; Wang and Guo, 2009].

The main data sources in the study are the vulnerability database NVD [NVD, 2018], the weaknesses database CWE [CWE, 2018], and the exploit database EDB [EDB, 2018]. NVD contains the entries of known vulnerabilities in the CVE format [CVE, 2018]. This format assumes the presence of unique CVE identifier. The NVD entries also include the brief vulnerability or security flaw description, the source of information about the vulnerability, the publishing date and the date of the last modification, the metrics of CVSS of version 3.0 and/or 2.0 that describe vulnerability, and the references to advisories, solutions and tools. Besides, in NVD the vulnerability entries have links to the vulnerable products in the CPE format and the used CWE weaknesses. The most interesting fields of the NVD entries for us are the CVSS metrics (indexes) for vulnerabilities and the linked weaknesses, as soon as the CVSS metrics are used in the research to determine weaknesses used by vulnerabilities.

The CVSS indexes characterize vulnerability considering preconditions and postconditions of its exploitation [Mell et al., 2007]. In recent years several CVSS versions were issued. Thus, CVSS of version 2.0 has some uncertainties that have been fixed in the new version 3.0 [Hanford and Heitman, 2015]. The detailed comparison of CVSS of version 2.0 and CVSS of version 3.0 is provided in [Doynikova et al., 2017]. CVSS of version 2.0 and 3.0 includes basic, temporal and contextual indexes. Basic indexes for the vulnerabilities can be found in the NVD database. Temporal indexes can be found in the open vulnerability databases, but they represent change of vulnerability in time and do not influence on vulnerability mapping to some weakness. Contextual indexes are related to the system security requirements and also do not influence on vulnerability mapping to some weakness. So we consider in this research only basic indexes.

For the CVSS indexes, possible quantitative and qualitative values are specified. These values are set for the vulnerabilities in the NVD database by the experts (from the Computer Security Divisions' Information Technology Laboratory of National Institute of Standards and Technology). We assumed that these values can be used to classify vulnerabilities by weaknesses, because the classes of weaknesses are also differ by the access type they provide and possible damage from their exploitation. The set of known weaknesses of the software and hardware in the CWE format [CWE, 2018] is provided in the CWE database. The dictionary has several views: by research concepts, by development concepts and by architectural concepts. Different views contain different number of entries. Currently we use for analysis the view by research concepts and the view by development concepts. The CWE schema [CWE Schema, 2019] incorporates three main groups of attributes: “Common Attributes”, “Structural Schema Elements”, “Administrative Information”. In terms of our research there are two interesting subgroups of the “Common Attributes” group: the subgroup “Prescriptions and Recommendations”, as soon as it describes possible security tools and means for the system, and the subgroup “Background and Contextual Information” that includes the field “Observed_Examples” containing the examples of vulnerabilities that use the weakness. This field can be used to enhance the accuracy of classification. Another valuable group is the “Supplemental Information” that contains the field “Related_Attack_Patterns”, because it allows connecting weaknesses and attack patterns from CAPEC [CAPEC, 2018].

The exploit database EDB [EDB, 2018] contains known exploits and corresponding vulnerable software, i.e. existence of an exploit supposes existence of a vulnerability and a weakness. Consequently some EDB entries have links to CVE, but some entries do not have such links.

There are a few research works devoted to classification of vulnerabilities. In [Tripathi and Singh, 2011a] the trends in vulnerability classes in NVD are analyzed, including the most popular classes and their severity levels. This paper analyses only vulnerabilities that have links to the weaknesses. In [Tripathi and Singh, 2012] the authors compare classifications used by various databases and conclude that the CWE classification has the best classification properties. In [Tripathi and Singh, 2011b] the authors prioritize vulnerability classes using the CVSS scores. But these papers do not consider the task of automated classification of weaknesses. The approach provided in [Wang and Guo, 2010] is the most closest to our approach to weaknesses determination. The authors classify vulnerabilities in NVD using the CWE classes and a naive Bayesian network. But they demonstrate the results only for one product and one vulnerability and do not analyze the accuracy of their results.

Our second technique for the weaknesses determination supposes analysis of an exploit source code and its representation as a graph for the further analysis. Currently the following models for representation of the software source code exist: an abstract syntax tree (AST), an abstract semantic graph (ASG), a control flow graph (CFG), a program dependence graph (PDG), a code property graph (CPG) [Caprile et al., 2003]. There are a lot of research works related to the software source code analysis using the listed and other models.

AST is a processed parse tree, from which some elements are removed. The parse tree is a tree that represents the process of construction of language sentences starting from the terminal symbols using the language grammar [Duffy, 2011]. ASTs are often used for the type inference, architecture recovery and call graph extraction [Caprile et al., 2003]. Thus, in [Moses and Syman, 2001] the authors introduce a dynamic syntax tree to analyze the dynamic programming languages. In [Neamtiu et al., 2005] the authors use AST matching to analyze the C source code evolution. The practical implementation of AST for the Python language is available [astdump 4.3, 2016].

ASG is generated from AST using semantic rules, i.e. in contradiction to AST it includes semantic information. In [Duffy, 2011] the process of the ASG generation is described. The author uses ASG for the static and dynamic analysis of C++ code for the software development. In [Stein, 2016] ASG is used to analyze the dynamically typed languages.

CFG is a directed graph where each node corresponds to a basic block and edges connect nodes that can be executed consequentially [Patterson et al., 2018]. CFGs are usually used for the flow analysis and the impact analysis [Caprile et al., 2003]. For example, they are used to analyze the source code in [Gold, 2010; Patterson et al., 2018]. The practical implementation of the CFG for Python is available [Coet, 2018].

PDG represents the data and control dependencies within a program. PDGs are used, among other things, for the program slicing [Agrawal et al., 1990, Caprile et al., 2003]. In [Hsieh et al., 1992] it is used for information flow control. The practical implementation of the PDG for Python is available [Blais, 2018].

Analysis of the software code is often used to detect vulnerabilities in the software [Yamaguchi et al. 2014]. In [Yamaguchi et al. 2014] the authors introduce CPG that is generated combining AST, CFG and PDG. It allows detection of buffer overflows, integer overflows, format string vulnerabilities, or memory disclosures.

Structure analysis issues for the .pyc files are considered in [Batchelder, 2008; Fireeye, 2016]. We used it to determine the structure of exploit source code. In [Code2graph, 2019; Gharibi et al., 2018] the authors introduce the tool to automatically analyze a Python-source code and to generate a static call graph and a similarity matrix of all possible execution paths in the system. A static call graph is a representation of relationships between system's functions. Each node of the graph is a function, and each edge is a function call. The authors suggest the technique, including extraction of the code structure, using the static analysis code traversal approach. The code structure is applied to extract the caller-callee relationships (relationships between each caller function initiating the call and its targets) that are represented as a 2D matrix. This matrix is used to construct a call graph. The call graph is realized to generate similarity matrix that can be applied to detect the similar execution paths and generate machine learning models.

In [Caprile et al., 2003] the issues of interoperability of the source code analysis tools based on different models are considered.

Currently there are security data sources that accumulate knowledge of information security community on the vulnerabilities and weaknesses of products and exploits for particular vulnerabilities. But such sources are large, contain uncertainties, and they are usually incomplete. Thus there is a need in techniques for security data processing that will eliminate uncertainty and incompleteness. We aim to eliminate one of such uncertainties related with an absence of the direct links between the products and appropriate weaknesses. It is required to simplify selection of security measures (that are applicable for a weakness, i.e. a group of vulnerabilities, instead of a separate vulnerability). From our point of view an approach including analysis and classification of vulnerabilities as well as analysis and classification of exploits can be realized to determine associated weaknesses. We propose to analyse the exploits structure (namely, exploits with the Python-source code) and its graph representation to classify the vulnerabilities used by these exploits, i.e. to determine used weaknesses. It involves comparison of the execution paths of exploits and further extraction of features, i.e. indexes, of the similar paths. These indexes are the basis for classification of the exploits by weaknesses. Though there are a lot of models for source code analysis, these models are not suitable for our goal because we need to extract a formalized description of the code functionality. Particularly, AST doesn't represent code functionality, ASG loses function call names in the existing implementations, and CFG can contain not functional nodes. In the section 3.2 we describe our model that is an ASG variation. It combines CFG and the function call dependencies graph.

3 Automated Determination of System Weaknesses

As it was mentioned in the introduction, the goal of this research is automation of the infrastructure weaknesses determination based on the available data about analyzed system. It is one of the steps towards automated selection of security measures. We distinguish the following steps of automated selection of security measures (Fig. 1): detection of vulnerabilities of the analyzed system and/or available exploits; determination of weaknesses of the analyzed system (that make possible the detected vulnerabilities and/or exploits); determination of cyber threats to the analyzed system (that are possible because of the determined weaknesses); selection of security measures to protect against determined cyber threats. The task being solved in this paper is highlighted in Fig. 1. Before the automated determination of system threats and appropriate security measures, the automated determination of system weaknesses should be implemented. Let us describe the Fig. 1 in details.

We propose to use as input data the system configuration including system software and hardware, vulnerability entries from the NVD database [NVD, 2018] and exploits from the EDB database [EDB, 2018]. We selected the NVD database for the following reasons: (1) it incorporates the largest number of known vulnerabilities [Kotenko et al., 2015]; (2) it is supported by the community of experts from Information Technology Laboratory of the NIST Computer Security Division; and (3) it contains the CVSS scores for the vulnerabilities and links to the weakness database CWE [CWE, 2018] required for the connection of the vulnerabilities and attacks. In its turn, EDB is the most known open database of exploits. Its entries have links to the CVE database and description of the vulnerable platform and software.

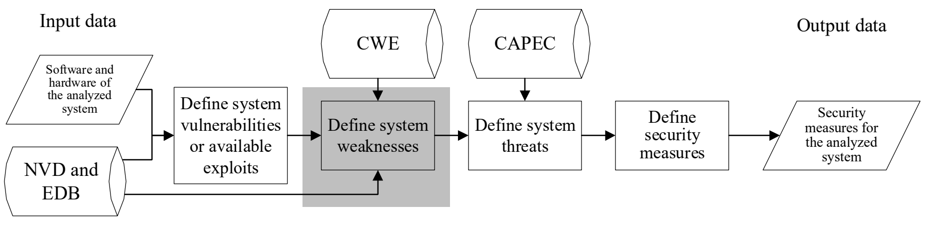
Input data
Software and hardware of the analyzed system
NVD and EDB
Define system vulnerabilities or available exploits
CWE
Define system weaknesses
CAPEC
Define system threats
Define security measures
Output data
Security measures for the analyzed system

Figure 1: The place of the task being solved

On the first step ("Define system vulnerabilities or available exploits") we determine system vulnerabilities or applicable exploits on the basis of the system software and hardware, the NVD database [NVD, 2018] and the EDB database [EDB, 2018], and vulnerability characteristics according to the scoring system CVSS of version 2.0 and 3.0 from the NVD database.

On the second step ("Define system weaknesses") we determine system weaknesses (this is the task being solved in this paper). At the end of this step CVEs, CWEs and applicable exploits of the analyzed system should be known. At the beginning of this step the following uncertainties can exist: (1) CVE and an exploit are known, CWE is unknown; (2) CVE and CWE are known, an exploit is unknown; (3) an exploit and CWE are known, CVE is unknown; (4) CVE is known, an exploit and CWE are unknown; (5) an exploit is known, CVE and CWE are unknown; (6) CWE is known, an exploit and CVE are unknown; (7) CVE is known, an exploit and CWE are unknown.

Considering the task, we do not review cases (2), (3) and (6) when CWE is known. Thus, in case (2), determined on the first step system vulnerabilities, links from the appropriate vulnerability entries in the NVD database to weaknesses in the CWE database, and entries from the CWE database are used to define the system weaknesses. But the rather large number of vulnerability entries in NVD does not have links to the used weaknesses, i.e. these vulnerabilities belong to the undefined class (Fig. 2). It complicates achievement of the goal. It corresponds to the cases (1), (4) and (7) and demonstrates the relevance of the task of vulnerability classification via the explored CWE weaknesses.

In the paper we attempt to overcome aforementioned challenge and suggest the technique for the vulnerability classification by weaknesses. It is based on the data mining using the connection between values of the vulnerability attributes determined using CVSS and appropriate weaknesses in NVD. We trained classifiers using vulnerabilities that are linked to the weaknesses from CWE. Then we determined weaknesses for the vulnerabilities that do not have such links in the NVD database.

The case (5) corresponds to the situation when there is an exploit in the EDB database, but it is not connected to the CVE. An analysis of the EDB database shown that only 462 (35%) exploits written in Python for the past 5 years have links to the CVE identifiers, i.e. there are no descriptions for the vulnerabilities exploited in 65% exploits (in Python).

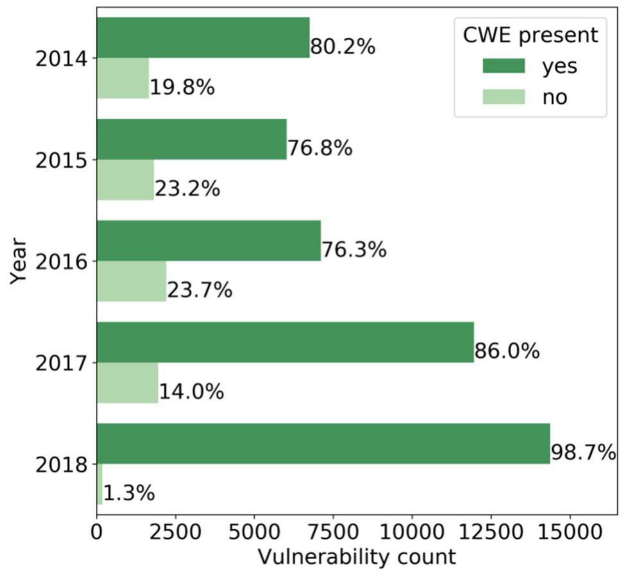

Figure 2: Number of the vulnerabilities classified using CWE by years (April,2018)

In this case the corresponding weaknesses cannot be determined using the suggested above technique. Thus we introduce the second technique that complements the first technique in the scope of the common approach. This technique allows us to determine weaknesses for the case (5) and to enhance the classification accuracy for the first technique. The second technique is based on the static analysis of the exploit source code to determine the semantically similar exploits and their features. These features specify corresponding vulnerability class, i.e. weakness. Analysis is conducted using graph models.

The common scheme of our approach to determination of system's weaknesses using two described techniques is depicted in Fig. 3. Rectangles and ovals represent the types of security information. Ovals represent known vulnerability scores and recoverable representations of exploits in the form of models and common features. Solid lines represent the semantic relations [Kotenko et al., 2018]. Dotted lines represent transitions between main steps of the techniques in scope of our approach.

The first technique of the vulnerability classification starts from application of the “CVSS score” for “Vulnerability classification” to determine corresponding

"Weakness" (Fig. 3). The second technique is used if there is known exploit, but used vulnerability is unknown. Besides, in the future work we plan to use it to enhance an accuracy of the first technique when vulnerability is known but the related weakness is unknown. The second technique starts from the "Compilation" of the "Exploit" represented with "Source code" into the "Executable code". The "Executable code" is used for the analysis. The process of analysis of exploit source code (ESC) is much more complicated than analysis of executable code processed by the interpreter. The stage of decompiling is omitted in the Fig. 3 because it is out of scope of this paper. The decompiled "Executable code" is used for "Modeling" of the semantic "Exploit code model" based on the graph of control flow and function (class) calls (use) of imported modules. As the result of generation of multiple "Exploit code models" and their consequent synthesis the "Common exploits features" are "Extracted". These "Common exploits features" are used for "Vulnerability classification" to determine the corresponding "Weaknesses". At the current stage of research, the effectiveness of exploit code modeling is "Assessed" on the basis of correlation of their features with known CVSS vulnerability indexes.

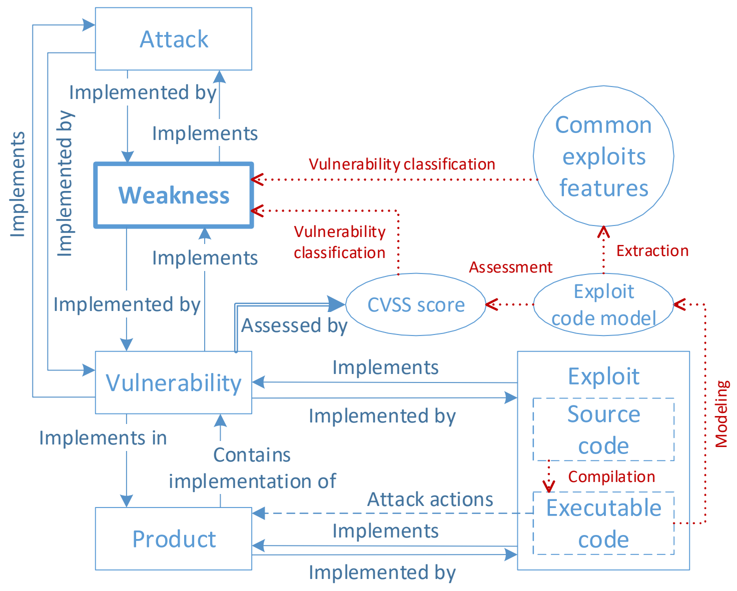

Figure 3: The common scheme of weakness determination

The Fig. 3 also represents two types of security information that are not used in the proposed techniques, namely "Attack" and "Product". They are required to connect the approach with the specific information system to select security measures (Fig. 1), i.e. security measures depend on the possible attacks that, in their turn, depend on the weaknesses and vulnerabilities of specific information system. While the weaknesses and the vulnerabilities depend on the products of this system.

3.1 The technique based on the analysis of CVSS indexes

The first technique uses the relationships between open security data sources of vulnerabilities (NVD), weaknesses (CWE), and indexes of vulnerabilities provided in NVD. It is based on the data mining using the connection between values of the vulnerability attributes determined using CVSS and appropriate weaknesses in NVD. We trained classifiers using vulnerabilities that are linked to the weaknesses from CWE to determine weaknesses for the vulnerabilities that do not have such links in the NVD database. Main stages of the technique are as follows: (1) gather and process data; (2) train a classifier using vulnerabilities that are linked to the weaknesses from CWE; (3) classify vulnerabilities that do not have links.

On the first stage we gather data on known vulnerabilities, values of their indexes, and links to the corresponding weaknesses from the NVD database. Linked weakness specifies vulnerability class. We gather data on the weaknesses from the CWE database. We use the indexes of the vulnerability scoring system CVSS of versions 2.0 and 3.0 as the source of features for the vulnerability classification. Vulnerability indexes in this system can have quantitative and qualitative (categorical) values. The target features for the vulnerability classification are upper-level classes of CWE weaknesses in two separate views: Research and Development. The data processing is described in section 4. On the second stage we train classifier using vulnerabilities from NVD that are linked to the weaknesses from CWE. On the third stage we classify vulnerabilities that do not have links using the following classification methods: decision tree, k-nearest neighbors, and random forest.

3.2 The technique based on the analysis of the exploit code

The approach to analysis of the exploit executable code underlying the proposed technique is based on the assumption that exploits for the vulnerabilities that use similar weaknesses (i.e. related to the same class) have semantically similar functional. This assumption leads from the hypothesis of compactness. But, the model of exploits as objects of statistical sampling should be specified to check it.

The developed technique of analysis of the exploit executable code incorporates the following stages (Fig. 3): (1) compiling source code; (2) decompiling source code; (3) building the functional semantic models of the exploits; (4) comparing the models; (5) extracting sustainable and valuable parts of exploit models to determine vulnerability classes.

Precompilation of the exploit source code into the machine code of the Python language interpreter (format *.pyc) on the stage 1 and its consequent decompilation (disassembling) into the sequence of interpreter instructions (opcodes) on the stage 2 allows excluding exploits with syntax errors in the source code. The errors can be related with application of the specific version of the programming language standard and/or interpreter of the Python language by the exploit's authors, and, consequently, with the used modules (specified in the interpreter environment).

On the stage 3 we use graph model of the exploit code. As it was shown in section 2 the graph models are successfully used to analyze the software source code. The nodes of our graph are specified by the “names” of the source code that are imported modules and their functions. Such modules (libraries) make up the standard runtime environment of the code interpreter, because all source codes completed successfully the compiling stage (import errors such as “No module named …” are excluded). Edges of the graph specify the sequence of function calls for such modules. The link from the source node to the destination node represents the use of the corresponding function result as an argument of the destination node function.

The proposed model is generated based on the CFG which nodes are replaced with functions and classes of objects of imported modules (conditionally global names). On the current stage of the research CFG was simplified to the single route from Entry Point (EP) to the “Implicit Return” (IR) for validation of the hypothesis about similarity of exploit source codes. Usually, authors of the exploits do not use such anti-debugging tools as obfuscation, packing, dynamic code modification, etc. to implement the proof of concept code of the vulnerability exploitation. Thus, the generation of single CFG route is sufficiently justified and should not lead to a distortion of the semantic functional model of the exploit. Additional advantage of this solution is reduction of number of names used in the source code that do not reflect its functional features.

To generate CFG, we analyze the disassembled exploit code to find conditional and unconditional, relative and absolute jumps, as well as exception handler instructions and some other ways of code branching. To generate the resulting graph of the semantic functional model of the exploit, the dependencies of names (calls) usage from the importing modules are determined for the main CFG route. Intermediate (local) names introduced by the authors in the process of exploit development are not considered. The difference of the proposed model consists, on the one hand, in the strict adherence to the main code execution route, and, on the other hand, in the reflection of only functional dependencies between imported names.

On the stage 4 we compare the models of different exploits. The generated sequences of name dependencies can significantly differ even having similar semantics. Thus, to generate common features of exploits codes on the stage 5 we suggest to use only linked name pairs. On this stage of research for preliminary confirmation of the hypothesis we compared generated sequences of name dependencies for exploits and the CVSS scores for the corresponding vulnerabilities.

4 Experiments and Discussion

4.1 Experiments for the technique based on the CVSS indexes

Data gathering and processing. Let us to review the vulnerability indexes that will be used for the classification first. The vulnerability entries in NVD have security metrics (indexes) valued using the CVSS vulnerability scoring system of version 2.0. Also, during the last years, the vulnerability entries in NVD are filled with security metrics valued using CVSS of version 3.0 (Fig. 4).

CVSS of version 2.0 includes the following basic indexes: (1) exploitability indexes (namely, Access Vector that determines how the vulnerability is exploited, Access Complexity that defines the complexity of exploitation of the vulnerability, and Authentication that reflects how many times an attacker have to authenticate to exploit the vulnerability) and (2) impact indexes (namely, Confidentiality Impact, Integrity Impact and Availability Impact that determine the damage for the confidentiality, integrity and availability, accordingly, as the result of successful exploitation). CVSS of version 3.0 includes the following basic indexes: exploitability (Attack Vector, Attack Complexity, Privileges Required, User Interaction) and impact (Confidentiality Impact, Integrity Impact, Availability Impact). Additionally CVSS of version 3.0 includes index Scope, which allows separating the affected component from a component that is damaged. We use indexes of the both versions because, as it was mentioned, CVSS of version 2.0 and 3.0 significantly differs in the set of metrics and in their values for the same metrics.

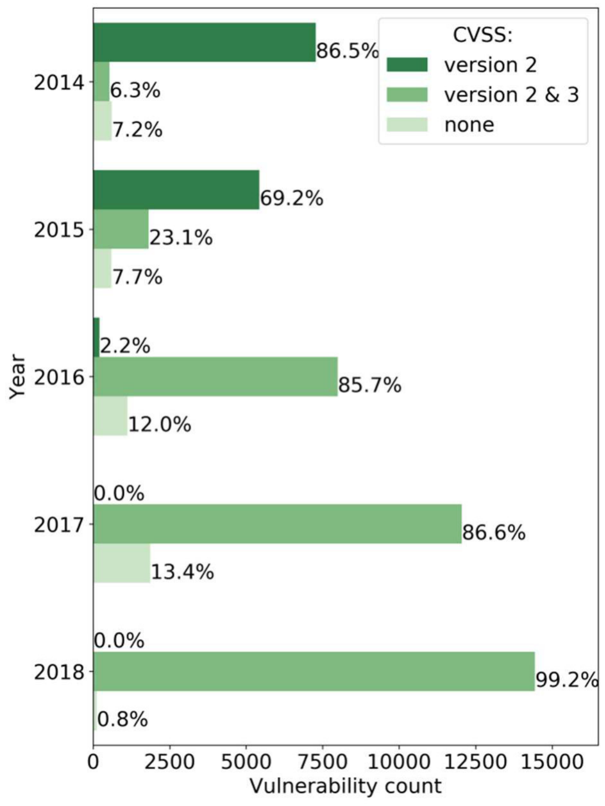

Figure 4: Number of the vulnerabilities considering the used CVSS version by years (April, 2018)

We filled the Pearson correlation coefficient matrix for the categorical features of vulnerabilities specified according to the CVSS of version 2.0 and 3.0 (the Kendall correlation coefficient and Spearman's correlation coefficient give similar results with insignificant deviation) to show it. The matrix is provided in [Doynikova et al. 2018]. The resulting Pearson correlation coefficient matrix contains Pearson correlation coefficients for the features of vulnerabilities specified according to the CVSS of version 2.0 and 3.0. This coefficient shows dependence of value of one feature (metric) from the value of another feature. To calculate these coefficients the attribute values were factorized, i.e. converted to a numerical type. This operation does not distort data because sets of values of categorical features are limited. According to the obtained Pearson correlation coefficient matrix [Doynikova et al. 2018] the logical correlation of values of the CVSS indexes of version 2.0 and 3.0 for the indexes of the Impact subgroup (Availability, Confidentiality and Integrity) is 78 - 84%. These indexes have metrics of the logical correlation of values 53 – 61% and 15 – 54% within CVSS of version 2.0 and 3.0, accordingly.

Interconnection between values of other CVSS indexes of both versions (Complexity, Access Vector, Authentication etc.) is represented with correlation coefficient up to 0.82. Some indexes have back-dependent relationship with the minimum coefficient – 0.47.

We analyzed vulnerabilities from the NVD database from 2014 year to the present (April 2018). To assess correlation between the features, the vulnerability entries without indexes of CVSS of version 2.0 and 3.0 were excluded from the original sample. For the classification of vulnerabilities the vulnerability entries without link to the CWE and vulnerabilities that implement rare CWE classes were also excluded from the test and holdout datasets. For example, most vulnerabilities (more than 95%) implement 4 classes of weaknesses from the Research view (Fig. 5).

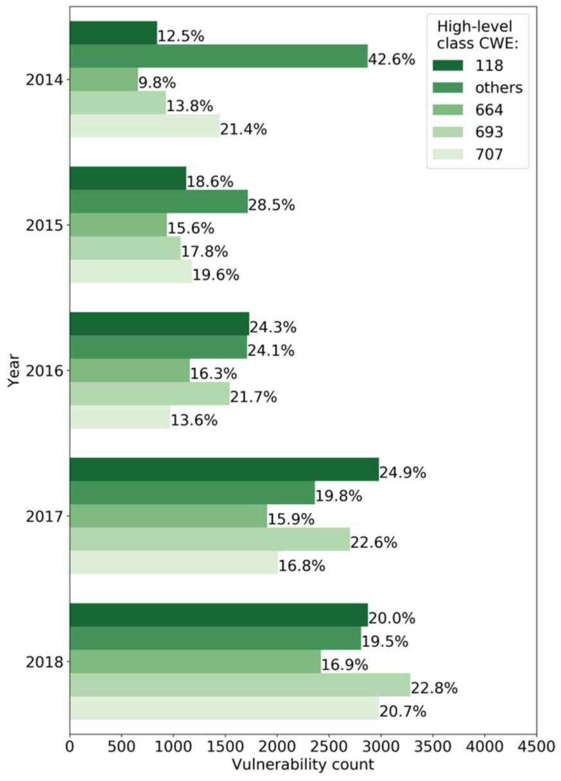

Figure 5: Number of the vulnerabilities that implement upper-level CWE classes (Research Concept) by years

Using vulnerability entries from the NVD database and the CWE classification of weaknesses we generated original datasets for classifiers training and testing. In Table 1, the characteristics of the original datasets are provided for two CWE views.

CWE view / **Dataset parameters**	**Research**	**Development**
Real object count	17668	5786
Object dataset	28000	2000
Class count	4	5
CWE classes	(118,664,693,707)	(116,119,284,345,682)

Table 1: Characteristics of the original datasets

Original data representation formats are JSON (JavaScript Object Notation) [JSON, 2019] and CSV (Comma Separated Values) [CSV, 2019] for the NVD and CWE databases accordingly. Data loading and preprocessing were performed using Python.

Training classifiers and vulnerability classification. Data analysis was performed using Python and scikit-learn module [scikit-learn, 2019], that implements selected training models. The holdout dataset for the testing of models with optimal hyper-parameters in terms of accuracy is 30% from the original dataset, and training dataset is 70% of the original dataset respectively.

In Table 2, the results of vulnerability classification for two CWE views are provided. The following classification methods were used: decision tree (DT), k-nearest neighbors (kNN), and random forest (RF). Further for each concept three characteristics of the predictive model are provided: (1) hyper-parameters of the model that give the maximum accuracy; (2) classification accuracy on cross-validation (5 blocks); (3) classification accuracy on the holdout dataset. Hyper-parameters of the predictive model for the decision tree method and random forest method are: maximum depth of the tree (max_depth) and maximum number of the used features (max_features). For the k-nearest neighbors method it is the number of neighbors. As it is shown in Table 2, the best classification results are obtained with the random forest method (RF). As soon as current classification accuracy is rather low, we plan to enhance it in the future work using the resulting indexes of the second technique and more complex algorithms including deep learning scheme.

Method	**Concept** / **Hyper-parameter**	**Research**			**Development**		
		1	2	3	1	2	3
DT	max_depth	13	70.5%	71.5%	14	70.3%	70.4%
	max_features	7			10		
kNN	n_neighbors	16	68.5%	70.4%	9	68%	67.1%
RF	max_depth	10	70.6%	71.7%	10	70.4%	70.5%
	max_features	4			8		

Table 2: Results of vulnerability classification

In Fig. 6 the metrics that show significance of features for the classification via considered CWE views using this method are provided. The figure shows difference in CWE views in terms of informativeness of the features used for classification. The most significant features are CVSSv3:Scope and CVSSv3:User Interaction from the CWE Develop Concepts view with significance index value 0.15. The indicator of features significance varies in the range [0.0, 1.0].

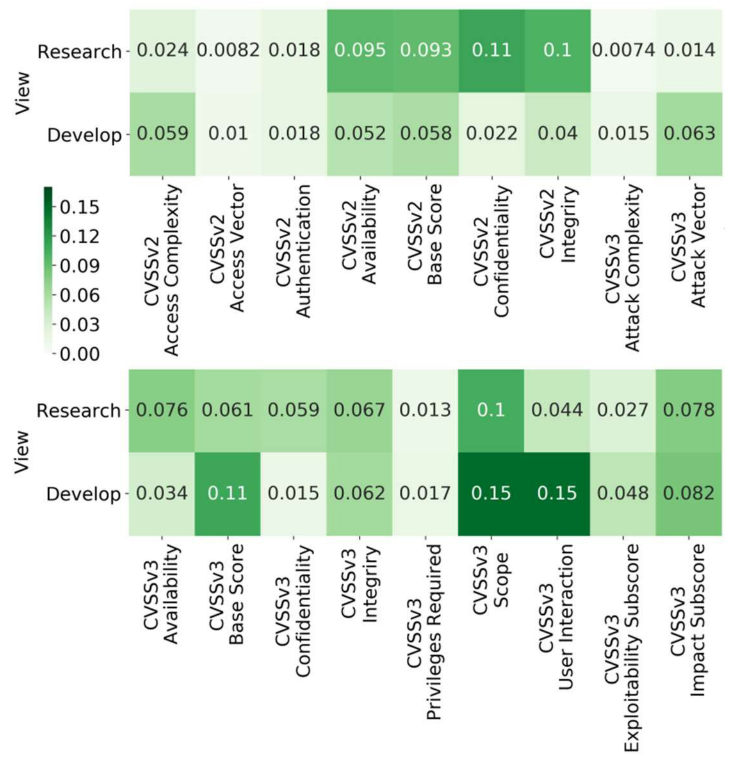

Figure 6: Indexes of the features significance for the classification using random forest method for the considered CWE views

4.2 Experiments for the technique based on the analysis of the exploit code

Data gathering and processing. Main data source for the second technique is the EDB database. Its analysis shown that the most popular language for exploit development is Python (Fig. 7). Therefore, in this paper, we analyze only exploits of this type.

The EDB database contains the following types of exploits (Fig. 8): 4292 webapps, 1829 dos, 1078 local, 1030 remote, 287 shellcode. There are 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. All exploits are represented with single source code file. It excludes application of the custom modules that go beyond the interpreter standard libraries or modules that are not described in the file (unknown modules). Based on this, all the modules used by the exploits are known and will be listed in the set of names of decompiled codes.

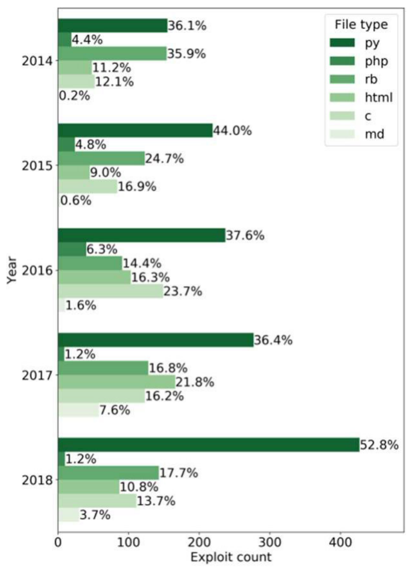

Figure 7: Statistics of exploit development languages in EDB for 2014-2018 years

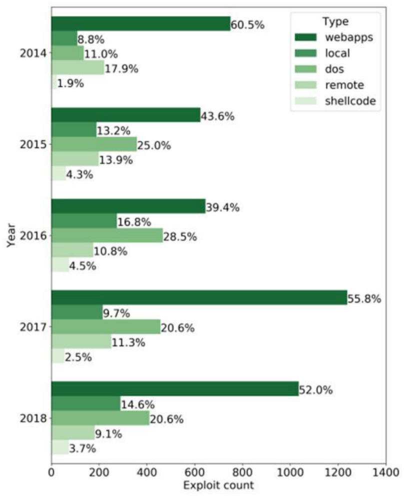

Figure 8: Statistics of exploit types in EDB for 2014-2018 year

Data analysis. In this paper we describe the experiments for the stages 1 and 3 of the proposed technique as the most interesting for us because they show the viability of the proposed technique.

Stages 1 and 2 (compiling and decompiling). There are 1315 exploits developed using the Python language for the past 5 years. Source code of 1153 exploits was compiled without errors. Therefore, no more than 13% exploits were excluded from the data set. Successfully compiled exploits were used as input data for the stage 3.

Stage 3 (generating the model). The "names" were extracted from the exploits executable code to generate the graph model. These "names" form nodes of the graph. 4690 names were extracted from 1153 successfully compiled exploits. The minimum number of names in the exploit is 1, the maximum number of names in the exploit is 132.

Analysis of the extracted names shown that the most used names are as follows: 'close' (number of uses – 582), 'sys' (487), 'len' (461), 'open' (452), 'write' (445), 'socket' (375), 'exit' (288), 'buffer' (273), 'argv' (266), 'name' (248), 'f' (242), 'os' (232), 'struct' (226), 'payload' (213), 'AF_INET' (212), 'send' (206), 'connect' (196), 'SOCK_STREAM' (195), 'doc' (190), 's' (187), 'time' (182), 'port' (180), 'shellcode' (164), 'requests' (163), 'str' (161), 'file' (156), 'host' (151), etc. The most used names clearly reflect the functional identity of the exploits. However, the data obtained is not enough to extract features (even dichatomical), because the import of the name does not mean its use in the code, especially in the main control flow of the exploit. Moreover, one exploit may use several high-level and frequently used names that have complex dependency connections and execution flow relations.

The exploit model in the form of the CFG which nodes are replaced with functions and classes of objects of imported modules (conditionally global names) was generated based on the extracted names and the names usage dependencies from the importing modules. Currently the CFG was simplified to the single CFG route. An example of the exploit code model extraction is provided in Fig. 9. It is a control flow graph (in the left) and its transformation to the exploit model (in the right) for the exploit with EDB-ID = 30688. In CFG the nodes correspond to the code blocks. The number in the node represents an offset of the first instruction of each block. The shortest path of the code execution is bold and represents regular program execution. The nodes beyond the main route represent the blocks that process exceptions and irregular situations.

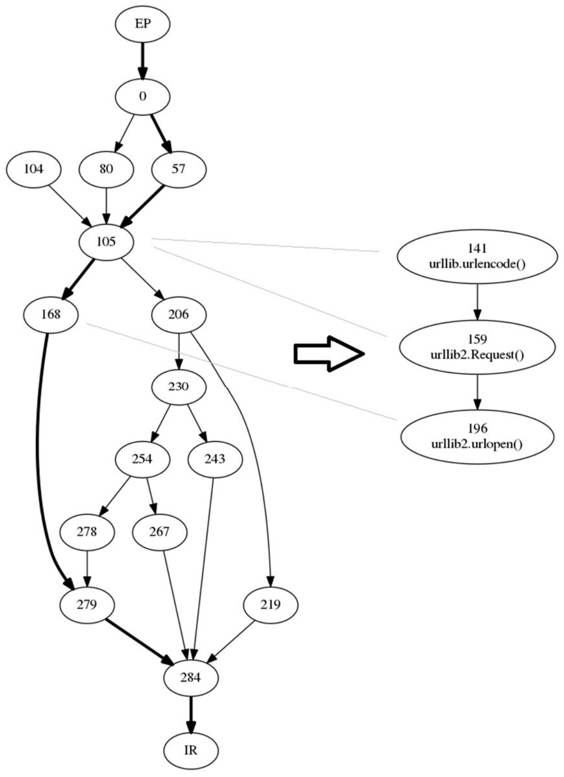

Figure 9: An example of exploit code CFG transformation

The generated semantic functional model of the exploit reflects an order and dependencies of calls of imported names (in the considered case there are names of two functions, namely urlencode() and urlopen(), and one class constructor, namely “Request”). The first two calls are made in one block. Therefore, the output of function urlencode() will be unconditionally send in the constructor of class Request (in case of absence of exceptions). This link is based only on call dependency. Function call urlopen() is made in another block. But it uses an object of class Request created earlier. This link is built both on the basis of CFG and functional dependency.

Besides, the code of this exploit contains two more imported modules. But calls of their modules do not reflect the functional features of code. It should be noticed that simplification of CFG via finding single shortest path allows excluding a quantity of names that also do not characterize the code functional.

4.3 Discussion

The challenge of uncertainty of the input data is one of the most complex tasks for security analysis and security tools selection. As a result of initiatives devoted to overcome this challenge both a lot of research and a lot of open security databases accumulating security knowledge appeared. However, the problems of accuracy, incompleteness and limitless of the existing databases for the specific security tasks still exist. Particularly, while solving the task of selection of automated security controls we faced the challenge of automated weaknesses determination for the information systems.

Usually the initial set of security measures is selected manually, so the task of automated weaknesses determination is not widely researched. Existing research works, reviewed in the related works, do not provide sufficient solution. We proposed two techniques for the automated determination of weaknesses of the analyzed system on the basis of its known vulnerabilities and its known exploits.

The *first technique* of vulnerability classification using different methods have 70% accuracy. This accuracy level follows from the source data nature. These data are CVSS indexes where the categorical values predominate over quantitative. As the result the set of possible values of the CVSS indexes used for the classification task is very limited. For example, categorical indexes in the scoring system CVSS of version 2.0 theoretically allow 729 unique combinations (the number of combinations of possible values). In the CVSS of version 3.0 they give 2592 unique combinations. Joint use of indexes of the CVSS of version 2.0 and 3.0 significantly expands vulnerability description space. But even in this case the number of unique vulnerability entries in terms of values of their indexes is a little more than 900 samples. Also uncertainty of the CWE database in terms of the structure of classes of weaknesses (one weakness can belong to upper-level several classes) influences negatively on the common predictive ability of the model. Thus, currently obtained results are not enough for the vulnerabilities classification by weaknesses and further automated determination of the required set of security controls.

To increase accuracy of the first technique, on the one hand, and to overcome situation when vulnerability is unknown, while, for example, exploit is known, we introduced the *second technique* based on the analysis of exploits. The static analysis of the source code is rather researched area for the software analysis, including vulnerability analysis, that was reviewed in details in the related works. Unfortunately, existing models and practical implementations of the graph for the Python code are not applicable for our goal, because we need the formalized description of the code functional. Particularly, AST does not represent code functional, existing implementations of ASG lose function call names, and CFG can contain not functional nodes. Thus, we introduced a new model combining CFG and DG. We preliminary confirmed the hypothesis that exploits for the vulnerabilities that use similar weaknesses, i.e. related to the same class, have semantically similar functional using the developed model. We plan to extend the technique by identifying potential dependencies between calls of imported names. To implement this we plan to analyze exploit model represented as the common dependency graph and outline the potential execution routes. We assume that the extracted in this way exploit features will be more common and informative for the construction of the predictive model of vulnerability classification by weaknesses and will complement the first technique.

The proposed techniques can be further applied to specify vulnerability metrics, detect unknown vulnerabilities, and assess the risks.

5 Conclusions

The paper proposed the approach to automation of vulnerabilities and exploits classification, i.e. determination of the system weaknesses, for their further simultaneous elimination. The developed approach is based on the open security sources and the analysis of security data provided in them. The approach incorporates two techniques that are described in the paper in details. The first one is based on the analysis of the publicly available vulnerability indexes of the Common Vulnerability Scoring System from the NVD database and it was already introduced on the IWCC workshop [Doynikova et al., 2018]. The second one extends the previous research and complements the first technique in case if there are exploits in the EDB database not related to the vulnerabilities and therefore the indexes for classification are absent. It is based on the analysis of the exploit code for the features (i.e. indexes) extraction using graph models. The extracted indexes are further used for the weakness determination using the first technique.

The paper also provided the statistics for the used open security databases that justify the relevance of the considered task and of the developed techniques. The data gathering and preprocessing process is described carefully as soon as the experiment results highly depend on it. The experiments demonstrated the effectiveness and potential of the developed techniques. Particularly, the first technique based on the CVSS indexes shown satisfying classification results, but the results should be further enhanced. The experiments with the second technique preliminary confirmed the hypothesis that exploits for the vulnerabilities that use similar weaknesses (i.e. related to the same class) have semantically similar functional. The technique and experiments should be further evolved.

6 Future Work

In the future work, we plan to evolve the developed techniques and specify the mechanism of their common application for the enhancement of vulnerability and exploits classification by weaknesses and to evolve the experiments approving an efficiency of the approach. Particularly, to evolve the technique based on the CVSS indexes, we plan to use predictive models on the second CWE level (in this work we used only the first CWE level), i.e. to use separate classifiers for each upper-level class of weaknesses, and to use more complex algorithms including deep learning scheme. Also, in the scope of the more general task, we plan to use the links from the weaknesses to the attack patterns of the attack patterns database ("Common Attack Pattern Enumeration and Classification" (CAPEC) [CAPEC, 2018]) provided by CWE, and references to the possible security measures provided for the attack patterns in CAPEC to determinate the set of security controls for analyzed infrastructures.

Acknowledgements

This research is partially supported by grants of RFBR (projects No. 16-29-09482, 18-07-01488, 19-07-01246) and by the budget (the project No. AAAA-A16-116033110102-5).

References

[Agrawal et al., 1990] Agrawal, H., Horgan, J. R.: "Dynamic Program Slicing", In Proceedings of the ACM SIGPLAN'90 Conference on Programming Language Design and Implementation. New York: White Plains (1990), 11 p.

[Aksu et al., 2018] Aksu, M. U., Bicakci, K., Dilek, M. H., Ozbayoglu, A. M., Tatli, E.: "Automated Generation of Attack Graphs Using NVD", In Proceedings of the Eighth ACM Conference on Data and Application Security and Privacy - CODASPY'18 (2018). https://doi.org/10.1145/3176258.3176339

[astdump 4.3, 2016] "astdump 4.3" (2016). Retrieved from https://pypi.org/project/astdump/

[Batchelder, 2008] Batchelder, N.: "The structure of .pyc files" (2008). Retrieved 10 January 2019, from https://nedbatchelder.com/blog/200804/the_structure_of_pyc_files.html

[Blais, 2018] Blais, M.: "snakefood: Python Dependency Graphs" (2018). Retrieved from https://bitbucket.org/blais/snakefood/overview

[Caprile et al., 2003] Caprile, B., Potrich, A., Ricca, F., Tonella, P.: "Model centered interoperability for source code analysis", In STEP 2003, Software Technology and Engineering Practice, Workshop on Software Analysis and Maintenance: Practices, Tools, Interoperability (SAM). Amsterdam, The Netherlands (2003).

[Chang et al., 2011] Chang, Y. Y., Zavarsky, P., Ruhl, R., Lindskog, D.: "Trend analysis of the CVE for software vulnerability management", In Proceedings - 2011 IEEE International Conference on Privacy, Security, Risk and Trust and IEEE International Conference on Social Computing, PASSAT/SocialCom 2011 (2011). https://doi.org/10.1109/PASSAT/SocialCom.2011.184

[Code2graph, 2019] "Code2graph: Automatic Generation of Static Call Graphs for Python Source Code", Kansas City: The UMKC Distributed Intelligent Computing group (UDIC). Retrieved 15 Junuary 2019 from https://info.umkc.edu/UDIC_Research/ index.php/code2graph/

[Coet, 2018] Coet, A.: "StatiCFG" (2018). Retrieved from https://github.com/coetaur0/staticfg

[CAPEC, 2018] "Common Attack Pattern Enumeration and Classification (CAPEC)". Retrieved 7 September 2018, from https://capec.mitre.org/

[CPE, 2019] "Common Platform Enumeration (CPE)". Retrieved 11 January 2019, from https://nvd.nist.gov/products/cpe

[CSV, 2019] "CSV, Comma Separated Values (RFC 4180)". Retrieved 15 January 2019, from https://www.ietf.org/rfc/rfc4180.txt

[CVE, 2018] "Common Vulnerabilities and Exposures (CVE)". Retrieved 7 September 2018, from https://cve.mitre.org/

[CWE, 2018] "Common Weakness Enumeration (CWE)". Retrieved 7 September 2018, from https://cwe.mitre.org/index.html

[CWE Glossary, 2019] "CWE Glossary". Retrieved 30 May 2019, from https://cwe.mitre.org/documents/glossary/index.html

[CWE Schema, 2019] "MITRE. CWE Schema Documentation". Retrieved 11 January 2019 from https://cwe.mitre.org/documents/schema/schema_d9.pdf

[Das et al., 2012] Das, R., Sarkani, S., Mazzuchi, T.A.: "Software Selection based on Quantitative Security Risk Assessment", IJCA Special Issue on "Computational Intelligence & Information Security" (2012), pp. 45–56.

[Doynikova et al., 2017] Doynikova, E., Chechulin, A., Kotenko, I.: "Analytical attack modeling and security assessment based on the common vulnerability scoring system", In Conference of Open Innovation Association, FRUCT (2017). https://doi.org/10.23919/FRUCT.2017.8071292

[Doynikova et al., 2018] Doynikova, E., Fedorchenko, A., Kotenko, I.: "Determination of security threat classes on the basis of vulnerability analysis for automated countermeasure selection", In Proceedings of the 13th International Conference on Availability, Reliability and Security (ARES 2018). Hamburg: ACM ICPS (2018), pp. 62:1-62:8. https://doi.org/https://doi.org/10.1145/3230833.3233260

[Doynikova and Kotenko 2018] Doynikova, E. V., Kotenko, I. V.: "Improvement of attack graphs for cybersecurity monitoring: Handling of inaccuracies, processing of cycles, mapping of incidents and automatic countermeasure selection", SPIIRAS Proceedings (2018). https://doi.org/10.15622/sp.57.9

[Duffy, 2011] Duffy, E.: "The Design & Implementation of an Abstract Semantic Graph for Statement-Level Dynamic Analysis of C++ Applications", Clemson University (2011). Retrieved from https://tigerprints.clemson.edu/cgi/viewcontent.cgi?referer=&httpsredir=1&article=1832&context=all_dissertations

[EDB, 2018] 'Exploit Database'. Retrieved 7 September 2018, from https://www.exploit-db.com/

[FDCC, 2019] "Federal Desktop Core Configuration (FDCC)". Retrieved 11 January 2019, from https://www.nist.gov/programs-projects/federal-desktop-core-configuration-fdcc

[Fireeye, 2016] "Fireeye: bytecode_graph", Milpitas, CA: Github (2016). Retrieved from https://github.com/fireeye/flare-bytecode_graph/blob/master/README.md

[FIRST, 2015] "FIRST: Common Vulnerability Scoring System v3.0: Specification Document", Forum of Incident Response and Security Teams (FIRST) (2015). https://doi.org/10.1109/msp.2006.145

[FISMA, 2019] "FISMA Background". Retrieved 11 January 2019, from https://csrc.nist.gov/projects/risk-management/detailed-overview

[Gamal et al., 2011] Gamal, M.M., Hasan, D., Hegazy, A.F.: "A security analysis framework powered by an expert system", International Journal of Computer Science and Security, Vol. 4, No. 6 (2011), pp. 505–526.

[Gharibi et al., 2018] Gharibi, G., Tripathi, R., Lee, Y.: "Code2graph: automatic generation of static call graphs for Python source code", In Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering - ASE 2018 (2018). https://doi.org/10.1145/3238147.3240484

[Gold, 2010] Gold, R.: "Control flow graphs and code coverage", Intern. Journal of Applied Mathematics and Computer Science (2010). https://doi.org/10.2478/v10006-010-0056-9

[Hanford and Heitman, 2015] Hanford, S., Heitman, M.: "Common Vulnerability Scoring System v3 . 0 : User Guide", FIRST-Forum of Incident Response and Security Teams (2015).

[Houmb and Franqueira, 2009] Houmb, S.H., Franqueira, V.N.L.: "Estimating ToE risk level using CVSS", In Proceedings - International Conference on Availability, Reliability and Security, ARES 2009 (2009). https://doi.org/10.1080/00206816709474435

[Hsieh et al., 1992] Hsieh, C.S., Unger, E.A., Mata-Toledo, R. A.: "Using program dependence graphs for information flow control", Journal of Systems and Software, Vol. 17, No. 3 (1992), pp. 227–232. https://doi.org/https://doi.org/10.1016/0164-1212(92)90111-V

[JSON, 2019] "Introducing JSON". Retrieved 11 January 2019 from https://javaee.github.io/tutorial/jsonp001.html

[Kotenko et al., 2015] Kotenko, I., Fedorchenko, A., Chechulin, A.: "Integrated repository of security information for network security evaluation", Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications, Vol. 6, No. 2 (2015), pp.41–57.

[Kotenko et al., 2018] Kotenko, I., Doynikova, E., Fedorchenko, A., Chechulin, A.: "An ontology-based hybrid storage of security information", Information Technology and Control, 4 (2018), pp.655-667.

[Mell et al., 2007] Mell, P., Scarfone, K., Romanosky, S.: "A Complete Guide to the Common Vulnerability Scoring System Version 2.0", FIRST Forum of Incident Response and Security Teams (2007).

[Moses and Syman, 2001] Moses, T., Syman, D.: "Static Analysis: a Dynamic Syntax Tree implementation" (2001).

[NCPR, 2018] "National Checklist Program Repository". Retrieved 11 January 2019 from https://nvd.nist.gov/ncp/repository

[Neamtiu et al., 2005] Neamtiu, I., Foster, J.S., Hicks, M.: "Understanding source code evolution using abstract syntax tree matching", ACM SIGSOFT Software Engineering Notes (2005). https://doi.org/10.1145/1082983.1083143

[Nessus, 2018] "Nessus vulnerability scanner". Retrieved 2 July 2018 from http://www.tenable.com/products/nessus-vulnerability-scanner

[NMap, 2018] "NMap reference guide". Retrieved 2 July 2018 from http://nmap.org/book/man.html

[NVD, 2018] "National Vulnerability Database". Retrieved 7 September 2018 from https://nvd.nist.gov/

[Patterson et al., 2018] Patterson, E., Baldini, I., Mojsilovic´, A., Varshney, K.R.: "Representation and Analysis of Software", In Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence (IJCAI-18) (2018), pp. 5847–5849.

[Pauli and Engebretson 2008] Pauli, J. J., Engebretson, P. H.: "Towards a specification prototype for hierarchy-driven attack patterns", In Proceedings - International Conference on Information Technology: New Generations, ITNG 2008 (2008). https://doi.org/10.1109/ITNG.2008.23

[PCI DSS, 2016] "Payment Card Industry (PCI) Data Security Standard". Requirements and Security Assessment Procedures Version 3.2 (2016).

[Radack and Kuhn, 2011] Radack, S., Kuhn, R.: "Managing security: The security content automation protocol", IT Professional (2011). https://doi.org/10.1109/MITP.2011.11

[scikit-learn, 2019] "scikit-learn". Retrieved 2 July 2018 from https://scikit-learn.org/stable/

[Son et al., 2015] Son, Y., Lee, Y., Oh, S.: "A Software Weakness Analysis Technique for Secure Software", Advanced Science and Technology Letters, Vol. 93 (2015), pp. 5–8.

[Stein, 2016] Stein, D.: 'Graph-Based Source Code Analysis of Dynamically Typed Languages' (2016).

[Tripathi and Singh, 2011a] Tripathi, A., Singh, U. K.: "Analyzing Trends in Vulnerability Classes across CVSS Metrics", International Journal of Computer Applications (0975 – 8887), Vol. 36, No. 3 (2011a), pp. 38–44.

[Tripathi and Singh, 2011b] Tripathi, A., Singh, U. K.: "On prioritization of vulnerability categories based on CVSS scores", In 2011 6th International Conference on Computer Sciences and Convergence Information Technology (ICCIT). A. Tripathi, U. K. Singh (2011b).

[Tripathi and Singh, 2012] Tripathi, A., Singh, U. K.: "Taxonomic analysis of classification schemes in vulnerability databases", In 2011 6th International Conference on Computer Sciences and Convergence Information Technology (ICCIT). Seogwipo, South Korea (2012).

[Wang and Guo, 2009] Wang, J., Guo, M.: "OVM: an ontology for vulnerability management", Proceedings of the 5th Annual Workshop on Cyber Security and Information Intelligence Research (2009). https://doi.org/10.1145/1558607.1558646

[Wang and Guo, 2010] Wang, J. A., Guo, M.: "Vulnerability Categorization Using Bayesian Networks", In Proceedings of the Sixth Annual Workshop on Cyber Security and Information Intelligence Research. Oak Ridge, Tennessee, USA (2010). https://doi.org/10.1145/1852666.1852699

[Wu and Wang, 2011] Wu, B., Wang, A.: "EVMAT: an OVAL and NVD based enterprise vulnerability modeling and assessment tool", Proceedings of the 49th Annual Southeast Regional Conference (2011). https://doi.org/10.1145/2016039.2016074

[Yamaguchi et al., 2014] Yamaguchi, F., Golde, N., Arp, D., Rieck, K.: "Modeling and discovering vulnerabilities with code property graphs", In Proceedings - IEEE Symposium on Security and Privacy (2014). https://doi.org/10.1109/SP.2014.44

[Zhang et al., 2015] Zhang, S., Ou, X., Caragea, D.: "Predicting Cyber Risks through National Vulnerability Database", Information Security Journal (2015). https://doi.org/10.1080/19393555.2015.1111961