UDC 004

Extended Vulnerability Feature Extraction Based on Public Resources

Y. Y. Tatarinova[1,2,a], O. I. Sinelnikova[2]

[1]*Kharkiv National University of Radio Electronics*
[2]*Samsung Electronics Ukraine Company LLC*

Abstract

The focus of this research is to define a framework that automatically analyses Common Vulnerabilities and Exposures (CVE) from public and disclosed resources and makes mapping to the target computer system. The current framework calculates risk assessment and estimates vulnerabilities impact according to the features of the target platform. In this paper, we describe the main vulnerability feature set, provide approaches for automatic extraction from databases and open resources. We evaluated and improved each obtaining approaches on the recent set of security vulnerabilities (2018 year database). Comparison obtained results with results of manual expert analysis is proved.

Keywords: information and feature extraction, CVE, vulnerability analysis, security risk assessment

1. Introduction

An increasing number of public resources with disclosed vulnerabilities information, security threats and cyber attacks provides a great opportunity for both defenders and attackers. Detailed information from various public sources (National Vulnerability Database ([1]), hacker's blogs and forums, Web-accessible repositories, OSINT systems (like [2]), etc.) allows attacker easy to find and exploit the target. Successful exploitation of a minor component can lead to a compromise of the entire system . This occurs due to high complexity and connectivity of components on the target system. Security subdivisions make risk assessment of the entire environment to prevent possible cascade threats and its impact. It is necessary to provide vulnerability analysis in context of software components interaction on the target computer system.

Previously, we defined the automatic vulnerability impact assessment for the target system as a function (F), which takes an end-point product (P), extracts vulnerability characteristics set (D_i) and automatically estimate vulnerability impact W_i as $F(D_i, P)$, considering vulnerability and target computer system properties [3]. End-point product could be a personal computer, internet of things (IoT) device, wearable, etc.

Our research describes a new methodology and a tool that analyses vulnerability according to the related Web-based information from public sources and open vulnerability databases, makes analysis and feature extracting from target computer system, maps all obtained vulnerability characteristics and estimate risk assessment. The main idea of our approach is to build a detailed attack tree scenario, compute impact values and increase or decrease appropriate parts of Common Vulnerability Scoring System (CVSS) vector by obtained estimation results. Design overview and model description of this methodology were presented in our previous paper [3]. Our system architecture consists of two parts: vulnerability features extraction and impact evaluation. In this paper, we define vulnerability characteristics set (D), feature extraction approaches and describe its influence on mapping process to the target computer system (P). We make results analysis and compare them with expert evaluation, than highlight significant features based on the obtained results.

[a]yullia.tatarinova@mail.com

2. Related Work

Our approach consists from two aspects of the problem: extracting security related information from open source (blogs, security bulletins, exploits) and linking and mapping obtained information and vulnerability features to the target computer system, define significance and influence degree of each feature. In this section, we investigate the most valuable approaches and ways, which were designed to extract different kinds of vulnerability characteristics.

2.1. Vulnerability information extraction from open Web resources and vulnerability databases

There has been a large amount of work that expand the process of extracting cyber security related information about vulnerabilities from open sources. The most known public resource of security information is National Vulnerability Database (NVD) [1]. It combines information about security issues from different resources and represents each vulnerability in structured format, which includes common names, patches, links, details and short description. Each vulnerability in this database represented as Common Vulnerability and Exposure name (CVE). All CVEs are categorized by severity, vulnerability type and attack vector.

Unfortunately, significant amount of key information remain in unstructured text view. This text often includes information about which products and versions are affected, which attack can occurs, which low level functions contains security issue and even what values must contain local variables for attack.

Various techniques for knowledge extraction have been proposed. Mulwald et at [4] describes a framework that identifies threats and attacks in Web text to detect security exploit. In Ravendar [5] describes a system, which automatically extracts cybersecurity terms from blogs and bulletins using Natural Language Processing (NLP).

A number of approaches have been proposed to interconnect different data and resources on the Web by defining relations between schemas, Web links, published data to the other existing resources [6]. All above mentioned information could be linked. Joshi et al. [7] describes the security ontology with classes for Intrusion Detection System (IDS).

The recent approaches [8], [9] contain interesting and perspective risk assessment idea, except that the authors did not take into account vulnerability information from Web resources, build attack graph based on vulnerability information without consideration of system call graph.

Combining all mentioned approaches and improving some parts of them, we could identify and extract vulnerability features from open vulnerability databases and apply it to our system.

2.2. Vulnerability freshness and relevance

There are many tools, software (like Flexera) and Web resources which are devoted to vulnerabilities patches, fix, timelines and cycles. In [10] O. Alonso makes prototype system, that visualizes and make explore timelines of news archive search using crowdsourcing. But this research is not applicable for our investigation, so we try to adopt existing approaches and analytics tools for determining CVEs freshness, relevance and the degree of security community interest. Our chosen approach for retrieving impacts of the time dimension will be described in the following section.

2.3. Extracting feature information from vulnerability plain text description

Short plain text description contains the most valuable information, which is needed to understand main idea of attack surface and attack post conditions. There are several approaches were made to automatic transform plain text description to more structured one and clearly identify all needed conditions for security analysis.

Urbanska et al. [11] describes approach of information extraction about vulnerability exploitation from plain text vulnerability description. This approach is based on designed text filters and rules. Proposed algorithm extracts necessary condition about attacker actions, but it doesn't include vulnerability root cause. Similar approach with using security text related information was proposed in [12]. These filters based on additional vulnerability information from NVD site and dictionary key words lists according to each extracted entity. Also, Urbanska described Personalized Attack Graph (PAG) which is manually constructed from obtained security issue representation.

In [13] authors used two approaches: machine learning based solution and part-of-speech tagging (POS). According to experimental results, POS approach is generally better in identifying attacker action and impact. Like previous research, [13] does not take into account vulnerable function names and affected variables.

In [14] Mukherjee extend idea of PAG and formulate PAG concepts and define approach for automatic PAG retrieval. Author proposed unsupervised heuristic-based approach to parsing security related descriptions and made inferences based on grammatical patterns and parts-of-speech for English language with help of public available tools for sentiment analysis and automatic entity labeling.

There is no single approach for extracting all sets of vulnerability characteristics. One kind of vulnerability feature set such as independent characteristics can be obtained by applying standard technologies. It could describe the main vulnerability overview. Another part of such characteristics as freshness and attack graph cannot be extracted in a trivial way, so in this part of the research, we not only define different sets of vulnerability issue, but also determine most valuable extracting approaches for each kind of vulnerability feature.

We made an evaluation of the characteristic influence on the final result. Because, presence of one set of vulnerability features could increase impact and probability of successful attack while the presence of another feature set could prove system reliability or high attack complexity.

3. Vulnerability characteristics and information retrieval

In this section, we define set of vulnerability characteristics, which will be used for automatic impact evaluation. Let's define dangerous of as a set of characteristics:

$$D_i(CVE_i) = (X_i, Y_i, Z_i), \tag{1}$$

where X_i -– is a set of static vulnerability characteristics, which independent from target computer system and could be obtained via public vulnerability databases; Y_i – features, that contains detailed vulnerability information and could be obtained from public web resources; Z_i – mapping characteristics, which includes computer system feature and analysis, evaluation of X_i and Y_i, recomputing CVSS vector score including obtained results. Let's make brief introduction about each set of vulnerability characteristics and evaluate importance and influence of each feature from the set on overall compute system.

Static product independent set of features contain each vulnerability independently of the source database. These features are identifier, description, score, list of affected products, issue type. Full definition we make in table 1.

Extraction process of these features consists of database crawling, parsing and mostly depends on what kind and format vulnerability database is used. The basis of our assessment is CVSS score vector and its components ($CVSS_{base}, CVSS_{expl}, CVSS_{impact}$). Description ($Descr$) has potential influence, which depends on the further feature extraction stage. List of products (P_{list}) could affect decisive role on the mapping stage, in case of presence of vulnerable component on the target system (P).

Vulnerability type (V_{type}) and threat category (V_{treat}) could be a part of the attack graph to improve evaluation results and help to make more accurate estimation in case of confirmation of availability security issue on P. Main process and technical details of extraction this set of features we describe in next section.

Security forums, software exploit repositories and other public web resources could contain exhaustive information about security issues. Set of such kind of features (Y_i) contains all possible entities, which could be extracted from these resources. Full list of Y_i features and their sources are introduced in table 2. These features mainly related to the attack graph components and mitigation steps. One of the most relevant feature is freshness ($Rel(t)$) which could describe not only security community interests, but relevance and feasibility of issue analysis. We consider references ($Refs$) as a set, which could contain exploits ($Expl$) and patches ($Patch$) subsets. In this paper, we provide main approaches for extracting this set of features and make analysis to determine its impact on the final result.

Security news, web content and references added, modified or deleted continuously over time. Impact of time dimension on the Web has strong influence in information technologies area. Retrieving and understanding temporal metrics is a challenging task. We try to focus on how to leverage the time dimension throughout vulnerability lifecycle and information retrieval. Unfortunately, there were no valuable approaches done to detect and extract freshness and vulnerability relevance entities.

Number of released patches and exploits changes over time, so we can detect it, and possibly even predict. To determine function of changing these values over time $Rel(t)$ it's necessary to discover and detect time any relevant information appearance (($Datetime(Refs)$) and its changes over time:

$$Rel(t) = diff(Datetime(Refs) - Datetime.now(Refs)) \quad (2)$$

In practice, definition (2) becomes a challenge. Each CVE contains creation date, but very rare public reference has creation or modification date in web response.

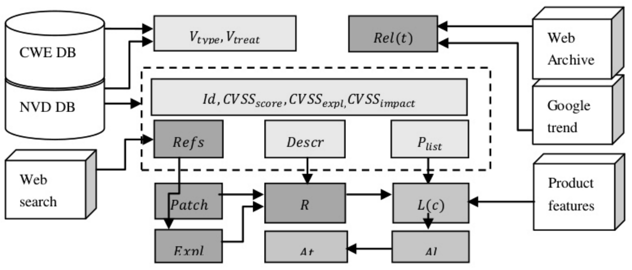

Fig. 1. Common vulnerabilities features extraction process.

Any mentions about date and time seldom occurs. This part of analysis is a tedious task and is beyond the scope of this paper. For current research, we decided to use less complicated approach.

According to table 3, we define next extraction approaches for the defined features. $L(c)$extracted with mapping attack graph, product list (P_{list}) and target system dependency graph from target system P. We get At with the help of analyzing $L(c)$, Al and product profile $Pr(P), X_i, Y_i$. Public resources features could be obtained from Google trends and archive search ($Rel(t)$). $Refs, Expl, Patch$ we extract from NVD parsing, web search, make references content analysis.

One of the most valuable resource which periodically visiting and crawling publicly available web pages are web archives. Web archives store enormous amount of information [17]. The Wayback Machine [18] is a digital archive search tool, which was launched by Internet Archive [16], a nonprofit organization. User is required to specify the URL of a web page to retrieve the number of times the URL was crawled by the Wayback Machine. It does not show how many times the page was actually updated or when it was created. We use Wayback archive to determine possible time of references changing.

The next point is to determine rate of security community interest to CVE. We include this feature in impact estimation process because it could help to compute attack probability and possible attacker's area of interests. The Google News Archive is an extension of Google News that provides a free access to scanned archives of newspapers and links to other newspaper archives on the web. Unfortunately, during experiment, this tool does not give accurate and sufficient results on common vulnerabilities keyword sets. The Google Trends that analyzes the popularity of top search queries in Google search across various regions and languages [15]. All abovementioned features and sources are shown at figure 1.

One of the most challenge part of our research is CVE description analysis. Making relation security vulnerabilities and determine how they could be exploited in a systematic manner is a task of Attack Trees (AT) and Attack Graphs (AG). Traditionally they are used in networking systems analysis. As mentioned above, in [11] Urbanska made the definition of PAG, which was used to analyze threats on a single computer system. PAG is a structure that represents interaction between existing vulnerabilities and user or attacker

Table 1. Static, product independent characteristics, X_i

Name	Description	Definition	Extraction approaches
Unique identifier	Assigned by CVE Numbering authority	Id	NVD db parsing
Vulnerability type	Potential mistakes which made in source code or during design	V_{type}	NVD description ($Descr$) and CWE id
Treat category	Treats and post-conditions, including STRIDE	V_{treat}	CWE mapping / NVD database, $Desrc$
CVSS score	Severity, composed of two: exploit ability and impact	$CVSS_{base}$	NVD db parsing
Exploitability	Reflect the ease and technical means by which the vulnerability can be exploited	$CVSS_{expl}$	NDV db parsing
Impact	Reflect the direct consequence of a successful exploit	$CVSS_{impcat}$	NVD db parsing
Description	Short summary with details about security issue	$Descr$	NVD db parsing
List of products	Products and services that are compatible with current CVE	P_{list}	NVD db parsing

Table 2. Characteristics from public resources and news, Y_i

Name	Description	Definition	Extraction approaches
References	Public references	$Refs$	NVD db parsing, web search
Exploit	References with exploit information, exploitation steps	$Expl$	References analysis, content analysis, NVD parsing
Patch	References with patch information	$Patch$	References analysis, content analysis
Freshness, relevance	Temporal characteristic, defined by querying in security community	$Rel(t)$	Google trends, archive search, [15, 16]
Root cause	Vulnerable component name, module of function	$Descr$	Description analysis, patch analysis

Table 3. Characteristics of product mapping,, Z_i

Name	Description	Definition	Extraction approaches
Affected binaries	Includes list of possible affected binaries according to cross dependency graph	$L(c)$	Mapping attack graph, product list (P_{list}) and target system dependency graph
Applicability	Applicability and availability of current vulnerability on target computer system	Al	Analyzing $L(c)$ with product profile $Pr(P)/X_i, Y_i$
Attainability	Attainability from the entry point to root cause	At	Analyzing $Al, L(c), Pr(P)/X_i, Y_i$

actions that lead to successful exploitation. PAG allows security specialist to quickly and reliably obtain the necessary information for analysis. In our work, we will redefine PAG with some changes. Vulnerability attack graph is defined as hierarchical relationships between the following components.

Vulnerable component. Presence of vulnerable software on the target system is necessary but not enough to guarantee successful attack. This part of graph could contain several types of named nodes: software product, its components (plugins, modules) and subcomponents (functions), set of affected versions. This a target point which has communications with users.

User actions. All users divided into attacker and customer (user type). According to user type, we define user's actions as regular and malicious. Regular customer could lead to security compromise. Malicious attacker actions could trigger vulnerability. All users have access label (remote, local) which could increase or decrease attack complexity in combination with CVSS vector.

Impact and post-conditions. This node represents potential impact and side effect in case of successful attack. Nodes labeled as "issue type" or "impact type". Various infected software and post-conditions could occur on the target system. Some of attack graph components could be absence due to unstructured nature of security issue description.

Constructing such graph requires approach, which could automatically retrieve all components from plain text description. Brief example of attack graph extraction from vulnerability description shown in the figure 2. Because this part of our research is too large we describe it in the next part of research.

We defined a set of characteristics, which takes into account features of target computer system product (P) for CVE_i as Z_i. It is the most significant feature set, which evaluation could help to obtain the final results. All features from this set could be extracted with the help of target product profile ($Pr(P).Pr(P)$ includes set of features, such as: main architecture, network infrastructure, common feature set for each existing binary, critical assets, entry points, etc. Extracting approaches for Z_i and product profile are the large part of this research, so we made it out of scope for this paper and will fully describe in future papers.

4. System architecture

Overall AVIA architecture was described in our previous paper. Preprocessing phase of vulnerability impact evaluation system combines two aspects of the problem. The first is feature extraction from relevant information about CVE. The second is mapping these features on target product (P). In this paper, we describe main approach, which we use for security issue feature extraction.

4.1. Vulnerability features extraction process

We make our investigation around open-source software for which we could obtain source code, associated metadata, and explicit vulnerability information. Main part of vulnerability information data we collect from NVD data feeds [1]. This database provided by National Institute of Standards and Technology (NIST). It contains information collected in structured format. To obtain X_i features for CVE_i we apply $NVD - Data_Proc(CVE_i)$ parsing:

$$X_i = NVD - Data - Proc(CVE_i) = (Id, CVSS_{base}, V_{treat}, V_{type}, Desrc). \quad (3)$$

We extend CVSS score with base CVSS score ($CVSS_{base}$), exploitability score ($CVSS_{expl}$) and impact ($CVSS_{impact}$). $CVSS_{base}$ is fundamental characteristic of a vulnerability that are constant over time and user environments, numerical score composed of two: exploitability and impact.$CVSS_{expl}$ shows how easily a vulnerability can be exploited. $CVSS_{impact}$ shows the result of a successful exploitation of a vulnerability referred to as "impacted component" [19]. According to CVSS defini- tion:

$$CVSS_{base} = CVSS_{expl} + CVSS_{impact}. \quad (4)$$

During mapping stage, we will increase or decrease $CVSS_{impact}$ and $CVSS_{expl}$ value with context of target system features (such as control and data flow map, attack graph).

Investigating [20], it is easy to notice that there is no any division or strong classification between vulnerability type and attack type. In our research, we distinguish vulnerability type according to CWE [20] classification and vulnerability treat category according to STRIDE [21]: (V_{type} and V_{treat}). This delimitation required for building attack graph and future impact estimation.

Y_i is as a vector of features that could be obtained from public resources and news as:

$$Y_i(CVE_i) = PublicFeatureExtract(CVE_i) = (Refs, Expl, Patch, R, Src, V, Rel(t)). \quad (5)$$

Full description and grounding were provided in previous section. In this section we describe extracting algorithms of such components as security patches, root cause and data relevance. Other parameters were obtained via NVD CVE entry parsing.

External references ($Refs$) for CVE_i could be obtained from NIST CVE site [22], Google Search or NVD database. In our work we use the last approach. So, external references are parsed from $NVD - Data - Proc()$ algorithm.

To identify security patches and fixes ($Patch$) in [[23] authors focused on the most popular version control systems for open-source software such as Git [24]. To find Git repositories and collect security fixes, they matched URLs with such substrings as "git", "svn", "cvs", "hg" or "mercurial". Also, in [23] URL paths were investigated using popular Git web interfaces (cgit, GitWeb, github and gitlab) and crawled with substring matching. This approach collects some commits, which

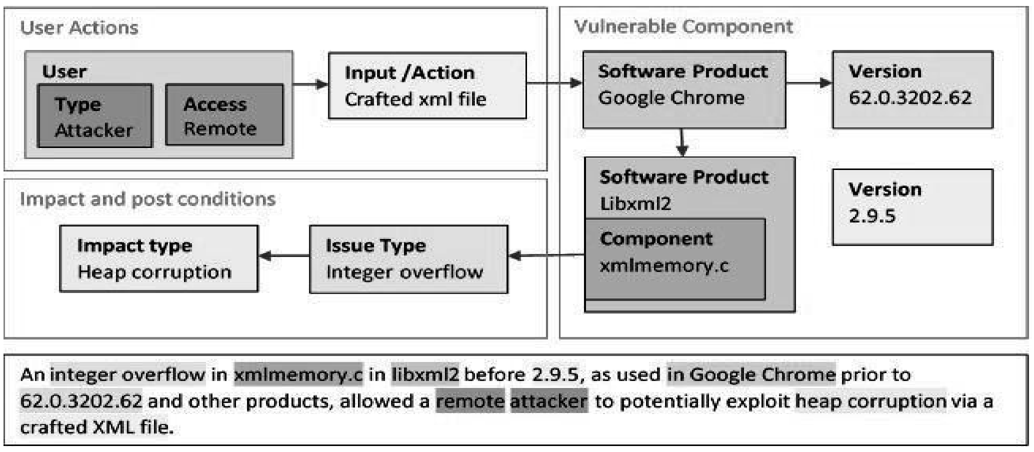

Fig. 2. CVE-2017-5130 attack graph obtained from description.

are not security fixes or just describe vulnerability, or may contain proof-of-concept exploits.

In this research we proposed our own method to get security patches more explicitly. Many of CVE's external references contains commit hash (md5 or sha1). So we make pattern search for hash appearance in URL, but it doesn't provide clear results. In [20] there is a problem, when some commit doesn't contain fixes. To detect patch presence and vulnerability root cause we analyze URL path content from Git web interface and detect diff commit patterns. Most of open source projects with Git web interfaces display changes via combined diff format with showing a merge [[25]. Applying patterns which match combined diff format to CVE's external references we get list of patches with source code and root cause. Sometimes references contain other references to patch code. In this case we make recursive pattern search for security fix. In our research recursion level equals 1.

Algorithm 1 Feature extraction from public resources

```
Require: NVD CVE entry, $X_i$
Ensure: $Y_k = \{Refs, Expl, Patch, Src, V, Rel(t)\}$
 1: $V, Refs = NVD - Data - Proc(CVE)$
 2: for $ref \leftarrow 1$ in $Refs$ do
 3:     if $ref$ is $Patch()$ then
 4:         $patches.append(ref \rightarrow content)$
 5:     end if
        $wayback_{res} = WaybackSearch(ref)$
 6:     if $wayback_{res}$ is not empty then
 7:         $rel.append(wayback_{res})$
 8:     end if
        $trend_{data} = GoogleTrend(Id)$
 9:     if $trend_{data}$ is not empty then
10:         $rel.append(trend_{data})$
11:     end if
12: end for
```

The next step is to determine relevance of found data ($Rel(t)$). Most popular approaches how to get temporal features and process them were described in previous section. In our research we use web archive search to get temporal metrics. Unfortunately, web archive search systems have some drawbacks such as absence results on huge number of URLs. For our research we use Wayback Machine Internet Archive [16].

So, we need to apply complex approach for this challenge. For each CVE we analyze set of obtained URL references ($Refs$) with Wayback Machine [18]. The next step is to use Google Trend Analysis application for each CVE id entry and merge obtained results with previous step. Unfortunately, Google Trend service has no any public official API, so during implementation, we need to use proxy for a huge number of requests. Full algorithm for extracting Y_i feature set see 1:

Root cause (R) could be obtained only during description analysis and building attack tree. This part of our research is out of scope of current paper.

4.2. Binary processing on target product

In this section we fully describe approach, that was used to analyze binary files. The scope of our research is Linux-based operation systems for AMD and Intel processors. On target (P), we investigate executable and linkable format files (ELF files) (C). For current research, we include executable and shared object type of ELF. Kernel object files require future studying. Every binary file has cross references. They divided into code cross-references and data cross-references. In this research we use code cross-references because it's better to understand all possible control flows and this could be done in static analysis way. Data cross-references should be used during dynamic analysis as an extra feature. Ida Pro it is the most powerful and accurate way for computation code cross references. For our work, we need to compute all possible code references in target P. All cross-references are made from one address to another address. The "from" and "to" addresses may be either code or data addresses. Addresses defined as nodes and cross references as the edges of the graph. We defined own approach to resolve this task.

Set of binary ELF file (C) contains set of executable ELF C_{exe} and shared object ELF C_{shared} described as:

$$C = C_{exe} \cup C_{shared} = \{c_1, c_2, ..., c_n\}. \quad (6)$$

I it is intermediate representation set of each ELF, which could be obtained during $Pre{-}Process(C)$. This algorithm returns the representation each binary c_i as a set of functions F, dependencies $Deps$ and cross-references graph :

$$I = Pre - Process(C) = \bigcup_{i=1}^{n}(F, Deps, CFG). \quad (7)$$

Algorithm 2 Preprocessing ELF file, Pre-Process(C)

```
Require: List of ELFs, $C$
Ensure: $c_i = \{F, Deps, CFG\}$
1: for $c_i$ in $C$ do
2:     $Deps = read - dynamic - section(c_i)$
3:     $F = decompile(c_i)$
4:     for $f_i$ in $F$ do
5:         $f_i \rightarrow Type = get - symbol - type(f_i \rightarrow Name)$
6:         $CFG.add(f_i \rightarrow callers)$
7:         $CFG.add(f_i \rightarrow callees)$
8:     end for
9: end for
```

We generate two types of cross references: to symbol($f_i \rightarrow callers$) and cross references from symbol ($f_i \rightarrow callees$). Algorithm 2 we designed and implemented using python ELF package and Capstone framework.$f_i \rightarrow callers$ and $f_i \rightarrow callees$ we detect by examine each asm instruction and check if it jump or branch instruction. When we meet branch or jump instruction – get the address parameter and check its function. Functions, whose code is not presented within dynamic linked library or executable – checked in dynamic section and marked as exported.

Statically linked binaries contain all of the code for the libraries that have been linked to the program. As a result, building call graph for such type of ELF is different challenge and become out of scope current paper. Dependency list $Deps$ for each executable c_i was retrieved by dissecting and examining dynamic section with tag DT_NEEDED, which holds string table offset to the name of needed library. In order to distinguish among overloaded functions, compilers generate unique names with additional characters, which is used to encode additional information about function. This decoration called name mangling. All functions must be demangled before any usage or searching inside other binaries.

After building call graph and getting dependency set for each binary, the next step is to build global control flow graph (G):

$$G = GlobalGraph(I) = \bigcup_{i=1}^{n}(Deps_i, CFG_i). \quad (8)$$

Graph construction algorithm is to merge each binary $c_i \rightarrow CFG$ with taking into account dependencies ($Deps_i$). Obtained global graph G is mapped to the attack tree and attack graph for future impact evaluation.

5. Experiments and system evaluation

We used NVD dataset of 2018 and 2016 years for out experiment. We took dataset of 2016 to prove influence of information temporal changes and detect possible area of interest in certain vulnerabilities of security community and how it changes over time.

Major source (NVD database) represented in .json file format. Other input resources were in Web page textual form. All collected features has textual, numerical or numerical set representation, so we use database storage and move there all collected data for future processing.

Table 4 is dedicated to patch extraction results because this is one of the most significant feature in out set and it was not mentioned in related work. Patch existence significantly reduces vulnerability impact. NVD tagging approach lays in patch tags, which could be extracted during json/xml NVD database parsing. Our own content analysis approach was fully described in previous section. Another our approach for patch extraction – it is URL analysis (hash method) is present in table 4. During result analysis, we noticed, that combined approach (NVD tagging and content analysis) could provide better results for our future parts of research.

We decided not to include URL analysis in combined approach because of high false positive rate. In figure 3, common distribution of vulnerabilities displayed in appliance with CVSS score system.

During experiment and results calculation, it was noticed that CVE's references could contain several URLs with commit and patch information, which devoted to fixing security issue. We investigated this finding more in depth. During manual verification it was determined, that not all detected commits could really contain security patch. We need to consider only content analysis approach to extract root cause (R). At the implementation phase of $NVD - Data - Proc()$ we noticed, that security issue could have more than one CWE item, sometimes from different security domains, undefined subdomain or be empty. Only 69% of CVEs contains CWE mark during parsing NVD database.

We should mention, that number of vulnerabilities, patches and exploits is proportional to NVD vulnerability CVSS scoring distribution. According to this, we could define the most significant CVSS score interval in accordance with histogram peaks. With the advent of the publication of the vulnerability, there is an increased amount of data on how to exploit vulnerability. The amount of such information is much higher than public information about updates, patches or other security advisories, but situation changes over time. Figure 3c and 3d prove that: amount of security advisories and patches exceed number of found exploits and PoCs, so we need to consider tem- poral metric as most relevant during automatic impact evaluation during runtime.

Comparison of approaches for catching temporal metrics from vulnerability information are summarized in table 5. Both tables (4 and 5) show how many times declared features met in the chosen NVD database set per year. Table 5 shows, that temporal features

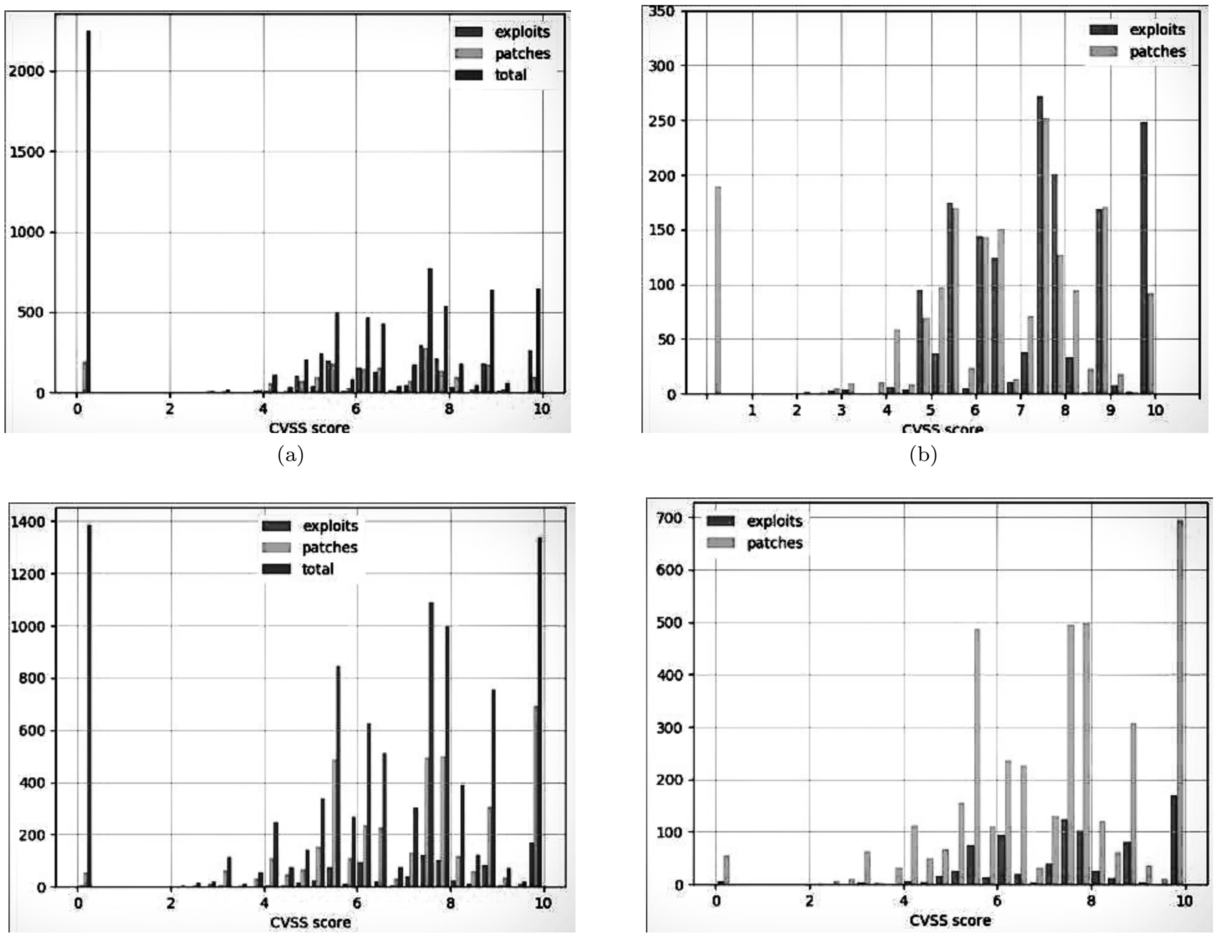

Fig. 3. Patch and exploit statistics by year: 3a - patch and exploit statistics via total number of references by 2018; 3b - comparison of unique CVE's patches and exploit by October 2018; : 3c - patch and exploit statistics via total number of references by 2016; 3d - comparison of unique CVE's patches and exploit for 2016.

Table 4. Retrieving patch information results.

Year	Items	NVD tagging		Content analysis, %		URL analysis, %		Combined approach, %		Total
2018	CVEs	1623	21.7	536	7.18	672	9	1855	24.86	7460
	References	1816	9	672	3.51	792	4.14	2123	11.1	19105
2016	CVEs	3749	38	1119	11.38	1274	12.9	4005	40.74	9837
	References	5205	12.54	1514	3.64	1626	3.9	5672	13.66	41548

Table 5. Retrieving patch information results.

Item	Google trends		Wayback machine		Total
CVEs	508	6.8%	3413	45.73%	7460
References	N/A		4640	24.29%	19105

occurs very rare according to features from public resources and product independent sets. Please, note, that Google Trend takes as input CVE while Wayback machine takes URL as input, so to compare results, we made mapping to CVE surface. During result analysis of Google Trend search, we need to consider, that this approach shows how world security community interested in certain vulnerabilities. We made ranking obtained CVEs by CVSS score and received the distribution that proportionally to overall CVSS distribution over the time. Wayback machine results could be used to show and analyze changes of security notes. As we determined, internet archive could give us more detailed information to retrieve and analyze temporal metrics. Unfortunately, most urls, which were crawled by Wayback Machine refer to vendor's notes or security advisories.

6. Conclusions and ongoing work

Recently we proposed concept of automatic vulnerability impact evaluation framework and proved its performance on the set of libxml CVEs as an example. In this research, we define a model with a basic set of vulnerability characteristics, its extraction approaches for automatic vulnerability impact evaluation framework. In this paper for the first time, it was proposed to take into consideration the security community interests and information temporary changes for the risk assessment. We implemented feature extraction module which automatically extracts proposed vulnerability feature set. Obtained results were compared with manual expert results.

Declared feature set provides full-scale base to make result complete and reliable for attack tree and rule based automation impact evaluation. Additional features, such as root cause and attack tree will be obtained with the help of natural language processing techniques (named entity recognition) and will be described in the next research article. The weight of temporal metrics should be recomputed according to obtained experiment results. In this study we concluded, that declared temporal metric has the most significant influence during impact evaluation and should be analyzed with the help of time series analysis algorithms.

References

[1] N. I. of Standards and T. (NIST)., "National vulnerability database." https://nvd.nist.gov.

[2] J. Matherly, "Shodan." https://www.shodan.io/.

[3] Y. Tatarinova, "Avia: Automatic vulnerability impact assessment on the target system," pp. 364–368, 08 2018.

[4] V. Mulwad, W. Li, A. Joshi, T. Finin, and K. Viswanathan, "Extracting information about security vulnerabilities from web text," in *Proceedings of the 2011 IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology-Volume 03*, pp. 257–260, IEEE Computer Society, 2011.

[5] R. Lal, "Information Extraction of Security related entities and concepts from unstructured text.," Master's thesis, May 2013.

[6] C. Bizer, T. Heath, and T. Berners-Lee, "Linked data: The story so far," in *Semantic services, interoperability and web applications: emerging concepts*, pp. 205–227, IGI Global, 2011.

[7] A. P. Joshi, "Linked data for software security concepts and vulnerability descriptions," tech. rep., Maryland Univ Baltimore Country Baltimore Md dept of Computer Science And . . . , 2013.

[8] M. U. Aksu, K. Bicakci, M. H. Dilek, A. M. Ozbayoglu, *et al.*, "Automated generation of attack graphs using nvd," in *Proceedings of the Eighth ACM Conference on Data and Application Security and Privacy*, pp. 135–142, ACM, 2018.

[9] M. U. Aksu, M. H. Dilek, E. İ. Tatlı, K. Bicakci, H. İ. Dirik, M. U. Demirezen, and T. Aykır, "A quantitative cvss-based cyber security risk assessment methodology for it systems," in *2017 International Carnahan Conference on Security Technology (ICCST)*, pp. 1–8, IEEE, 2017.

[10] O. Alonso, "Time-based exploration of news archives,"

[11] M. Urbanska, I. Ray, A. E. Howe, and M. Roberts, "Structuring a vulnerability description for comprehensive single system security analysis,"

[12] A. Joshi, R. Lal, T. Finin, and A. Joshi, "Extracting cybersecurity related linked data from text," in *2013 IEEE Seventh International Conference on Semantic Computing*, pp. 252–259, IEEE, 2013.

[13] S. Weerawardhana, S. Mukherjee, I. Ray, and A. Howe, "Automated extraction of vulnerability information for home computer security," in *International Symposium on Foundations and Practice of Security*, pp. 356–366, Springer, 2014.

[14] S. Mukherjee, *A Heuristic-Based Approach to Automatically Extract Personalized Attack Graph Related Concepts from Vulnerability Descriptions.* PhD thesis, Colorado State University, 2017.

[15] "Google trends." https://trends.google.com/trends/?geo=US.

[16] "Internet archive search." https://web.archive.org/.

[17] N. Kanhabua, R. Blanco, K. Nørvåg, *et al.*, "Temporal information retrieval," *Foundations and Trends® in Information Retrieval*, vol. 9, no. 2, pp. 91–208, 2015.

[18] "Wayback machine." https://en.wikipedia.org/wiki/Wayback_Machine.

[19] "Cvss explained." https://www.beyondsecurity.com/vulnerability_assessment_requirements_cvss_explained.html.

[20] "Common weakness enumeration." https://cwe.mitre.org/.

[21] "Application threat modeling.." https://www.owasp.org/index.php/Application_Threat_Modeling.

[22] MITRE, "Common vulnerabilities and exposures." https://cve.mitre.org/.

[23] F. Li and V. Paxson, "A large-scale empirical study of security patches," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pp. 2201–2215, ACM, 2017.

[24] "Git." https://git-scm.com/.

[25] "Git documentation." https://git-scm.com/docs/git-diff.