TELKOMNIKA Indonesian Journal of Electrical Engineering
Vol.12, No.6, June 2014, pp. 4330 ~ 4336
DOI: 10.11591/telkomnika.v12i6.4610

Software Vulnerability Analysis Method Based on Adaptive-K Sequence Clustering

Di Wu[1,2], Jiadong Ren[1]*
[1]College of Information Science and Engineering, Yanshan University, Qinhuangdao 066004, China
[2]Department of Information and Electronic Engineering, Hebei University of Engineering, HanDan 056038, China
*Corresponding author, e-mail: bestmoogoo@163.com

Abstract

Software vulnerability analysis has become a hot topic recently. However, the traditional methods for analyzing software vulnerability have higher false positive rate. In this paper, adaptive K function is defined, and SVAAKSC (Software vulnerability analysis method based on adaptive-K sequence clustering) is presented. The collected objects in software vulnerability sequence database SVSD are pretreated to equal length vectors. Moreover, according to adaptive-K based sequence clustering algorithm, all software vulnerabilities in SVSD are clustered into K clustering. Afterwards, by matching the similarities between detected vulnerability from software and each clustering center, whether the detected vulnerability is a real software vulnerability can be judged. Finally, the corresponding analysis report is obtained. The experimental results and analysis show that SVAAKSC has lower false positive rate and better analysis time.

***Keywords**: software vulnerability analysis, sequence clustering, adaptive-K, false positive rate*

1. Introduction

Information security technology has been accepted and widely applied in a lot of fields recently [1]. However, the safety of network and information system faces great challenge because of the incursion of hackers. Software vulnerability is one of the root causes of information security problems. It is the defect in behavioral logic, data access and other aspects [2]. As to software vulnerability has passive and static characteristics, computer software can easily be exploited by malicious attackers without authorization. Thus it can do great damage to human society. Software vulnerability analysis has become a popular and important topic of information security theory research and practical work [3].

In order to prevent the attack and intrusion of software vulnerabilities, the effective discovery and analysis of the software vulnerabilities is essential. Therefore, how to analyze the software vulnerabilities fast and effectively which can be seen as a key role for improving software security performance [4].

2. Related Work

A lot of research work of software vulnerability analysis has been carried out in recent years. Edge-weighted call graphs mining algorithm for software bugs localization was presented by Eichinger [5]. A novel reduction technique for call graphs which introduces edge weights was discussed. On the basis of graph mining and traditional feature selection, an analysis technique for weighted call graphs was also introduced. However, when the scale of graph is very large, the analysis efficiency of software bugs is decreased. In order to reduce the amount of sliced codes, and facilitate the subsequent calculation of code coverage information, He et al. proposed a software defects analysis method based on program structure reversing data dependency [6]. Reverse data dependence analysis focuses on the same program execution path to analyze data, and the data dependencies are extracted and stored on this path. Meanwhile, the stored and reversed data dependence is traversed based on a particular variable. Then code statements related to the specific variable are found.

Received October 2, 2013; Revised December 28, 2013; Accepted January 21, 2014

In order further improve the analysis time, many scholars used the clustering method to analyze software vulnerabilities. Mahaweerawat et al. presented a two-level clustering method to predict software fault [7]. SOM method was employed to classify historical data into clusters. Software faults that occurred in cluster components are predicted by RBFN. However, as the number of SOM units grows, the time cost is large. On the basis of dynamic information flow, software security failures are analyzed in literature [8]. The tool of DynFlow is used to record information flow profiles of executions. In the light of automatic cluster analysis, the executions are selected. The efficiency of information flow anomaly analysis depends on the types of programs, and the existed types of program defects. Wang et al. discussed DSVRDC (Detecting Software Vulnerabilities Method based on Rapid Density Clustering) [9]. New definition of *rd*-entropy and *s*-order are presented in this method. According to utilizing *rd*-entropy-based density clustering, the vulnerability sequence pattern database is built. Analyzing sequences are detected by the variation of *s*-order.

The above algorithms have been improved the time cost of software vulnerability analysis. However, good sequence similarity measurement for software vulnerability is still not well addressed. DVCMA (Detecting Vulnerabilities basing on Clustering and Model Analyzing) was proposed [10]. In this approach, the identification distance is to filtrate initially before calculating the edit distance of sequences. The vulnerabilities hiding in the software can be mined under a novel edit-distance-based similarity function. Although DVCMA can effectively detect software vulnerabilities, and with lower false positive and false negative rates, but the similarity measurement between software vulnerabilities sequences is based on edit-distance, it still has a high computational complexity.

In this article, in order to improve the analysis time of software vulnerability, on the basis of the number of all sequence elements, and the number of common sequence elements between two vulnerabilities, the vulnerability similarity measurement is designed. In accordance with two-phase similarity matching, whether *DV*(detected vulnerability) is a real software vulnerability or not will be determined. Afterwards, in order to reduce the impact of parameter *K* to the clustering quality, further improve the performance of the false positive rate. On the basis of a new definition of adaptive *K* function, AKSC (Adaptive-K-based Sequence Clustering) algorithm is presented. *The* object with the smallest *Adaptive*(*K*) is deemed to the optimal *K*.

The reminder of this paper is organized as follows. In section 2, we describe the related work of software vulnerability analysis. Section 3 gives problem definitions. Section 4 concludes the SVAAKSC method. Section 5 contains experimental results, and we offer our conclusions in section 6.

3. Problem Definitions

SVSD(Software vulnerability sequence database) is composed of collected software vulnerability sequences, $SVSD=\{SVS_1,SVS_2,\ldots, SVS_N\}$, wherein, *SVS*(Software vulnerability sequence) represents an orderly program operation sequence which can lead to produce vulnerability. $SVS=a_1a_2\ldots a_n$. a_i denotes the item of *SVS*, $a_i \in L(1\le i\le n)$. $L=\{a_1,a_2,\ldots,a_m\}$ is the item set, m is the number of item set. N is the number of *SVS* in *SVSD*.

Let *SVSE* be a set of the software vulnerability sequence elements in *SVSD.* $SSVSE=\{SE_1,SE_2,\ldots,SE_r,\ldots,SE_{|SSVSE|}\}$, sequence element SE_r is a pair of items a_ia_j of *SVS, where* $i<j$. The number of software vulnerability sequence elements in *SSVSE* is denoted as |*SSVSE*|. Each *SVS* can be represented as a |*SSVSE*|-dimensional vector.

Suppose that SVS_x and SVS_y are any two software vulnerability sequences in *SVSD*, $SE(SVS_x)$ and $SE(SVS_y)$ indicate the software vulnerability sequence pair sets of SVS_x and SVS_y. The similarity measurement between them is expressed as $SVSim(SVS_x,SVS_y)= |SE(SVS_x)\cap SE(SVS_y)| \,/\, |SE(SVS_x)\cup SE(SVS_y)|$.
Where $|SE(SVS_x)\cup SE(SVS_y)|$ represents the number of all sequence elements between SVS_x and SVS_y. $|SE(SVS_x)\cap SE(SVS_y)|$ denotes the number of common sequence elements between SVS_x and SVS_y.

In traditional *K*-means, how to make a rapid and accurate parameter *K* is a critical problem. The selection of *K* has a great influence on the clustering results. In this paper, to adjust the number of clusters dynamically, adaptive *K* function is proposed.

Definition 1. Suppose that *SVS* is any software vulnerability sequence in *SVSD*. *K* represents the number of clusters, the center of software vulnerability cluster SVC_P is described as $SVCC_P$. The adaptive *K* function is shown as follows.

$$\text{Adaptive}(K) = \frac{\sum_{P=1}^{K} \sum_{SVS \in SVC_P} (1 - SVSim(SVS, SVCC_P))^2}{N \cdot \underset{P,q}{Min}(1 - SVSim(SVCC_P, SVCC_q))^2} \quad (1)$$

Wherein, *N* is the number of objects in *SVSD*. *P*=1,2,…,*K*-1; *q*= *P*+1,*P*+2,…,*K*. The adaptive *K* is gained by the minimum Adaptive(*K*). In this way, the average dissimilarity between software vulnerability sequences in the same cluster should be as small as possible. And the minimum value of dissimilarity between each cluster needs to be the largest.

4. The SVAAKSC Method

Software vulnerability analysis method based on adaptive-*K* sequence clustering named SVAAKSC includes five stages. First and foremost, on the basis of the collected common software vulnerabilities, *SVSD* is established. Afterwards, the objects in *SVSD* are preprocessed to equal length vectors. Moreover, by adaptive-*K*-based sequence clustering algorithm, *K* vulnerability centers are gained. Next through two stage similarity measure, the most similar vulnerability to *DV* can be get. Finally, the analysis report of the detected vulnerability is output. The framework of SVAAKSC is shown as Figure 1.

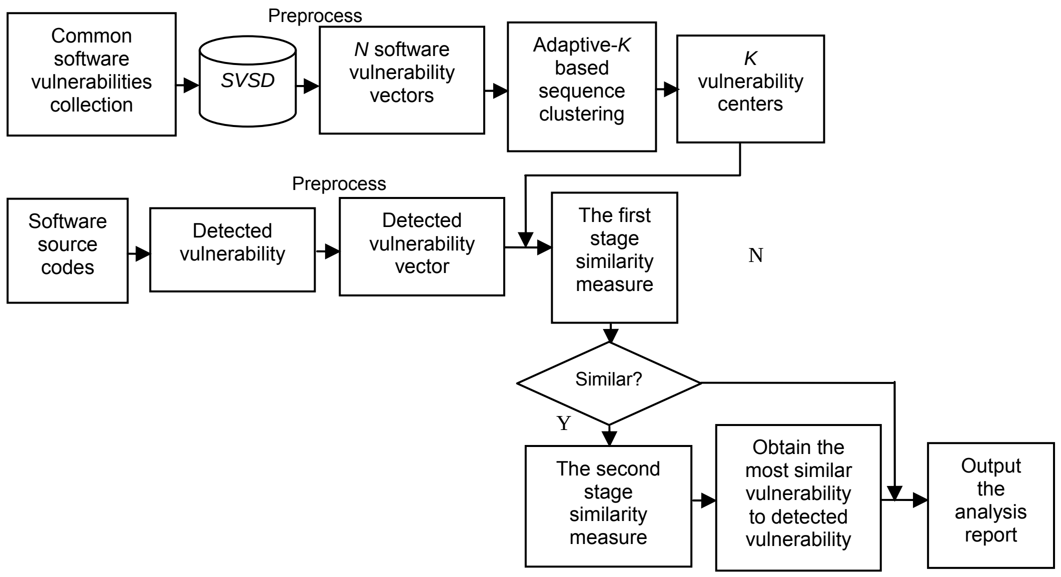
Common software vulnerabilities collection
Preprocess
SVSD
N software vulnerability vectors
Adaptive-K based sequence clustering
K vulnerability centers
Preprocess
Software source codes
Detected vulnerability
Detected vulnerability vector
The first stage similarity measure
N
Similar?
Y
The second stage similarity measure
Obtain the most similar vulnerability to detected vulnerability
Output the analysis report

Figure 1. The Framework of SVAAKSC

4.1. *SVSD* Establishing

Common software vulnerabilities are collected to establish *SVSD*. *SVSD* is composed by five-dimensional tuples, and each tuple is expressed as <*SVSN*, *SVS*, *SVSTY*, *SVSINF*, *SVSRANK*>. Wherein, *SVSN* is the number of *SVS*. *SVSTY* indicates the *SVS* type. *SVSINF* is the relevant feature information of *SVS*. SVSRANK denotes the *SVS* rank.

4.2. *SVS* Preprocessing

As to the lengths of software vulnerability sequences in *SVSD* are not the same usually. Therefore, it is necessary to preprocess them. By scanning the *SVSD* once, item set $L=\{a_1,a_2,\dots,a_m\}$ and $SSVSE=\{SE_1,SE_2,\dots,SE_r,\dots,SE_{|SSVSE|}\}$ are gained.

In accordance with the support relationships between SVS_x and SE_r in *SSVSE*, each SVS_x can be preprocessed as a |*SSVSE*|-dimensional vector. If SVS_x supports SE_r, then the value is 1, otherwise, the corresponding value is 0.

For example, $SVSD=\{SVS_1, SVS_2, SVS_3\}$. $SVS_1=aba$, $SVS_2=bcaa$, $SVS_3=cbba$. According to simple analysis, $L=\{a,b,c\}$, *SSVS*E={aa,ab,ac,ba,bb,bc,ca,cb,cc}, |*SSVSE*|=9. The pretreatment results are SVS_1={1,1,0,1,0,0,0,0,0}, SVS_2={1,0,0,1,0,1,1,0,0}, SVS_3={0,0,0,1,1, 0,1,1,0}.

4.3. Adaptive-*K*-Based Sequence Clustering

In traditional *K*-means sequence clustering algorithm, it needs to set parameter *K* ahead of time by user. In this paper, a novel AKSC (adaptive-*K*-based sequence clustering) algorithm is presented. It only needs to restrict the range of *K*. As a result, the similarities of *SVS* in the same are cluster as large as possible, in the meantime, the similarities between different clusters are as small as possible. By applying AKSC algorithm, *SVS* in *SVSD* are clustered into *K* clusters. *K* adaptive software vulnerability clusters can be gained. The specific process of AKSC is shown as follows.

```
Algorithm AKSC (SVSD, N, SVS, $K_{max}$ )
Input: SVSD: software vulnerability sequence database; N: the number of SVS in SVSD; SVS: any software vulnerability sequence of SVSD; $K_{max}$: The value of the largest K.
Output: K adaptive software vulnerability clusters
BEGIN
Step 1: For K=2 to $K_{max}$
Step 2: In SVSD, select K SVS as initial clustering centers randomly;
Step 3: Compute the similarity between SVS in SVSD and current clustering centers, each SVS is assigned to the most similar cluster;
Step 4: For each cluster, calculate the average similarities of SVS, clustering centers are updated, repeat the Step3 and Step4, until the clustering results do not change any more, jump to Step 5;
Step 5: According to formula (1), the corresponding Adaptive（K) is obtained;
Step 6: Compared with the value of the present Adaptive（K) and the former one, the corresponding K of the smallest one is saved;
Step 7: Output K adaptive software vulnerability clusters.
END
```

Wherein, in general conditions, $2 \le K \le K_{max}$. And the parameter *K* is much less than *N*, it is the number of *SVS* in *SVSD*. We set K_{max} is $\lfloor \sqrt{N} \rfloor$ or $\lfloor 2\ln N \rfloor$. Here, the lower limit of the corresponding integer value is used. The optimal clustering results can be obtained effectively. The impact of inaccurate selection of parameter *K* for the clustering quality of traditional *K*-means is reduced greatly.

4.4. Two-Phase Similarity Measuring

After *SVS* preprocessing, the *DV* (detected vulnerability) and objects in *SVSD* are all preprocessed to equal-dimensional vectors. Here, *DV* is extracted from the software source codes after static analysis, and further needed to analyze its vulnerability feature according to some certain rules.

In AKSC, the similarities between *SVS* are calculated by utilizing software vulnerability sequence elements similarity $SVSim(SVS_x,SVS_y)$. It is designed to analyze to the sequence elements that SVS_x and SVS_y contains. Thus the software vulnerabilities analysis efficiency can be enhanced greatly.

The process of similarity measurement is divided into two phases. In the first stage, the similarities of *DV* and *K* software vulnerability clustering centers are computed. If the most similar software vulnerability clustering center to *DV* is found, the second stage of similarity measure will be started. In the most similar software vulnerability cluster, by calculating similarities between *SVS* and *DV*, the most similar *SVS* to *DV* can be gained. The analysis report of *DV* is recorded. It is mainly about the relevant feature information *SVSINF* of the corresponding *SVS*. On the contrary, if *DV* is not similar with any software vulnerability clustering centers, then it can be viewed as a software operation sequence. The corresponding analysis report is outputted. In a word, whether *DV* is a real *SVS* or not will be determined in accordance with two-phase similarity matching.

5. Experimental Results and Analysis

In order to verify the performance of SVAAKSC algorithm, FTP server software wu-ftpd under linux is adopted in this section. Traditional static analysis tool ITS4 is also used during comparing the false positive rate analysis. Wherein, the effective source code lines of wu-ftpd are 13582, and the vulnerability code lines are 64. 10000 software vulnerability sequences were collected to establish *SVSD*.

Our experiments are run on the Intel Core 2 Duo 2.93GHz CPU, 2GB main memory and Microsoft XP. All algorithms are written in MyEclipse 8.5. We compare SVAAKSC with DVCMA [9] in false positive rate and analysis efficiency.

5.1. False Positive Rate Analysis

In this section, the false positive rates of three algorithms are analyzed by the formula as follows.

$$R_{FPR} = \frac{1}{m}\sum_{t=1}^{m}(1-\frac{Num_{arvt}}{Num_{v}}) * 100\% \qquad (2)$$

Wherein, Num_{arvt} denotes the number of analyzed real vulnerabilities in the *t*-th false positive rate analysis. Num_v is the number of vulnerabilities in software. We set m=10. The analysis results of false positive rates of SVAAKSC, DVCMA and ITS4 algorithms are shown as Table 1.

Table 1. The Analysis Results of False Positive Rates

Algorithm or Analysis Tool	False Positive Rate（%）
SVAAKSC	25.4%
DVCMA	30.8%
ITS4	36.9%

From Table 1, we can see that in FTP server software wu-ftpd, the average false positive rate of SVAAKSC is lower than DVCMA and static analysis tool ITS4. Thus the meaning of clustering results can be explained accurately by SVAAKSC.

For SVAAKSC, by adaptive-*K*-based sequence clustering algorithm, SVSD is clustered by adaptive-*K*-based sequence clustering algorithm AKSC. It does not set cluster number *K* in advance, but only restrict its range. By comparing with all the *Adaptive*(*K*), *the* object with the smallest *Adaptive*(*K*) is deemed to the optimal *K*. The impact of inaccurate selection of parameter *K* for the clustering quality is reduced greatly. The optimal clustering results of software vulnerabilities can be gained. The false positive rate will be reduced greatly. In this way, the final obtained *SVS* is the most similar with *DV*.

5.2. Analysis Time Test

To test the software vulnerabilities analysis time of SVAAKSC and DVCMA, *Num* denotes the number of software vulnerabilities. We set *Num*=1000, 2000, 3000, 4000, 5000. The test results of the running time of the two algorithms are shown as Figure 2.

As shown in Figure 2, the running times of two algorithms are growing linearly with increasing *Num*. In accordance with the support relationships between software vulnerability sequence *SVS* and sequence element SE_r in *SSVSE*, each *SVS* can be preprocessed as a |*SSVSE*|-dimensional vector. If *SVS* supports SE_r, then the value is 1, otherwise, the corresponding value is 0. Furthermore, on the basis of the number of all sequence elements and the number of common sequence elements between two *SVS,* the computation complexity of the similarity measurement in SVAAKSC is decreased. Finally, the software vulnerability analysis time is improved effectively.

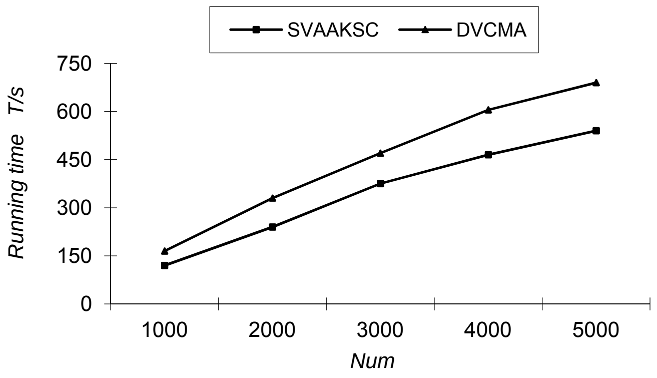

Figure 2. Running Times of SVAAKSC and DVCMA

6. Conclusion

In this work, in order to improve the performance of the false positive rate of software vulnerabilities analysis time, software vulnerability analysis method based on adaptive-*K* sequence clustering named SVAAKSC is discussed. First and foremost, common software vulnerabilities are collected to establish *SVSD*. Afterwards, according to the support relationships between software vulnerability sequences *SVS* and sequence elements, each *SVS* can be preprocessed as equal-dimensional vector. Moreover, on the basis of a new definition of adaptive *K* function, a novel adaptive-*K*-based sequence clustering algorithm AKSC is presented. It does not set cluster number *K* in advance, but only restrict its range. *The* object with the smallest *Adaptive*(*K*) is deemed to the optimal *K*. By adopting AKSC algorithm, *K* vulnerability centers of *SVSD* are gained. The impact of inaccurate selection of parameter *K* for the clustering quality is reduced greatly. The optimal clustering results of software vulnerabilities can be gained. Afterwards, in accordance with two-phase similarity matching, whether detected vulnerability *DV* is a real *SVS* or not will be determined. On the basis of the number of all sequence elements and the number of common sequence elements between two *SVS,* the computation complexity of the similarity measurement is decreased. Finally, the analysis report of the detected vulnerability is output. Our experimental results show that SVAAKSC can analyze *DV* of software source codes with lower false positive rate, and better vulnerability analysis time.

Acknowledgements

This work is supported by the National Natural Science Foundation of China (No.61170190), Youth Foundation of Hebei Educational Committee (No.Q2012070) and Science and Technology Research and Development Program of Handan (No:1321103077-3). The authors also would like to express their gratitude to the reviewers, whose valuable comments are very helpful in revising the paper.

References

[1] Li QY, Luo L. Determining the Minimal Software Reliability Test Effort by Stratified Sampling. *TELKOMNIKA Indonesian Journal of Electrical Engineering*. 2013; 11(8): 4399-4406.
[2] Ren JD, Xie YJ, Zhang AG, et al. A Closed Sequential Pattern Mining Algorithm for Discovery of the Software Bugs Feature. *Journal of Computational Information Systems*. 2011; 7(7): 2322-2329.
[3] Sun SJ, Xiao J. A Software Reliability GEP Model Based on Usage Profile. *TELKOMNIKA Indonesian Journal of Electrical Engineering*. 2012; 10(7): 1756-1764.
[4] Ning JF, Hu M. Study on Software Quality Improvement based on Rayleigh Model and PDCA Model. *TELKOMNIKA Indonesian Journal of Electrical Engineering*. 2013; 11(8): 4609-4615.
[5] Eichinger F, Böhm K, Huber M. Mining Edge-Weighted Call Graphs to Localize Software Bugs. *Lecture Notes in Computer Science*. 2008; 5211: 333-348.
[6] He H, Zhao L, Li Q, et al. *Analyze Software Defects with Program Structure Dependency*. Proceedings of the 2nd International Conference on Computer and Applications (CCA). 2013; 17: 53-57.

[7] Mahaweerawat A, Sophatsathit P, Lursinsap C. *Adaptive Self-Organizing Map Clustering for Software Fault Prediction*. Proceedings of the 4th Intl. Joint Conference on Computer Science and Software Engineering. Thailand. 2007: 35-41.

[8] Masri W, Podgurski A. Application-based Anomaly Intrusion Detection with Dynamic Information Flow Analysis. *Computers and Security*. 2008; 27(5): 176-187.

[9] Wang YY, Wang YN, Ren JD. Software Vulnerabilities Detection Using Rapid Density-based Clustering. *Journal of Computational Information System*s. 2011; 8(14): 3295-3302.

[10] Ren JD, Cai BL, He HT, et al. A Method for Detecting Software Vulnerabilities Based on Clustering and Model Analyzing. *Journal of Computational Information Systems*. 2011; 7(4): 1065-1073.