A New Method to Construct the Software Vulnerability Model

Xiang Li [1,2], Jinfu Chen [3,*], Zhechao Lin[1,2], Lin Zhang [3], Zibin Wang[1,2], Minmin Zhou [3], Wanggen Xie [3]
[1] (National key laboratory of science and technology on information system security, Beijing, 100101, China)
[2] (Beijing Institute of System Engineering, Beijing, 100101, China)
[3] (School of Computer Science and Communication Engineering, Jiangsu University, Zhenjiang, 212013, China)
* e-mail: jinfuchen@ujs.edu.cn

Abstract—**With the development of information technology, software plays an increasingly important role in the process of social development. However, at the same time, the number of software vulnerabilities is growing, posing a threat to national security and social stability. Therefore, some scholars and research institutions are paying their attention to the study of software vulnerability. In this paper, we propose a new vulnerability model construction method by considering the vulnerability causes and characteristics. Firstly, the causes and characteristics of software vulnerability are analyzed, and a formal vulnerability model is also established. Based on the causes and characteristics of software vulnerability, we establish the vulnerability model using the extended chemical abstract machine and deduce the software vulnerability through a formal method. We verified the effectiveness and efficiency of the proposed model using software vulnerability datasets. In addition, a prototype system is also designed and implemented. Experimental results show that the proposed model is more effective than other methods in the detection of software vulnerabilities.**

Keywords-computer security, vulnerability, vulnerability analysis, vulnerability model, chemical abstract machine

I. INTRODUCTION

With the development of Internet and software technology, software plays an increasingly important role in the process of social development [1-3]. However, the number of software vulnerabilities is increasing at the same time, causing great distress and social insecurity [4-5]. The rapid growing trend of vulnerabilities does not only affect the normal work and safety of e-ecommerce but also poses a major threat to national security and social stability.

The study of vulnerability began in the 1970s, which mainly aimed at classifying vulnerabilities of the operating system (such as UNIX system) [6-9]. The classification is just limited to the causes of the vulnerability, which cannot fully reflect the nature of vulnerability [10]. With the advancement of research, the scope of the vulnerability research is becoming more extensive. Researchers began to study the classification from macro prospective [11-13], and introduce the concept of classification, which influences the causes and analysis of vulnerability. The main analytical work includes risk-based vulnerability analysis method, which divides the cause into three categories [14-17]: (1) problems in the system design process, (2) problems in the operation and the use of the system, and (3) deliberate abuse. Cohen proposed an attack-oriented vulnerability analysis method, to analyze various kinds of possible attacks [17]. He divided software vulnerability into errors, omissions, data spoofing, input spill and other types. Krsul et al. proposed an impact-oriented vulnerability analysis method, which divides the software vulnerability into four categories: access data, execute commands, execute code, and denial of service [2, 3]. Nowadays, analysis of vulnerability causes is more comprehensive. Scholars have proposed a lot of vulnerability model systems based on various factors [18, 19]. In recent years, a growing number of important information security incidents happened because of software vulnerabilities. Therefore, studying characteristics of software vulnerabilities and analyzing the causes of vulnerability are of great significance, which can effectively reduce the negative impact on social life and national information security which are as a result of direct or indirect consequent of vulnerability.

In this paper, we propose a method to construct software vulnerability model by considering the vulnerability causes and characteristics and demonstrate some practical examples to show how the model can be used in detecting vulnerabilities. Furthermore, a prototype system based on the vulnerability model is also designed and implemented. Experimental result shows that the system based on the proposed model is effective in the discovery of software vulnerabilities.

II. THE REPRESENTATION FOR THE CAUSES AND CHARACTERISTICS OF VULNERABILITY

Software vulnerabilities are always the main causes for the emergence and existence of most software security loopholes. However, it is unfortunate that there is currently no widely accepted definition of the concept of "vulnerability". According to the latest report by CVE 2017, "vulnerability is a weakness in the computational logic found in software and some hardware components that when exploited, results in a negative impact to confidentiality, integrity or availability". Here, we define software vulnerability as follows: Software vulnerability (SV) refers to a software defect that can be exploited to cause serious security problems, resulting in software loopholes.

Based on the analysis of the existing causes and characteristics of a large number of vulnerabilities, a vulnerability guidance model is established in this paper. This vulnerability model is represented by a formal combination of tree and graph. This model is unique as it has a wide coverage and high accuracy in modeling and representing vulnerability characteristics.

978-1-5386-2030-4/17/$31.00 ©2017 IEEE

A. Description of Vulnerability Characteristics

The vulnerability attribute covers the first-level type information and the second-level cause and characteristic information. Let VulAtt represent the vulnerability attribute, then VulAtt = {TypeIno, ReaFeat}, where TypeIno represents the first-level type information and ReaFeat represents the second-level cause and characteristic information.

The first-level type information mainly contains information items such as vulnerability class (VC), user group (UG), software type (ST), action scope (CS), trigger condition (TC), operation method (OM), resolve scheme (RS) , and hazard level (DL). Therefore, the first-level type information can be expressed as TypeIno = {VC, UG, ST, CS, TC, OM, RS, DL}.

The second-level cause and characteristic information mainly includes the vulnerability name (VN), number (NO), type description (TD), formation cause (FC), main characteristic (MC), vulnerability testing(VT), judge rule (JR), resolve scheme (RS), find date (FD), patch information (PI) et al. Therefore, the second-level cause and characteristic information can be expressed as ReaFeat = {VN, NO, TD, FC, MC, VT, JR, RS, FD, PI}.

Among this information, the causes for the formation of the vulnerability are the most critical factors which contains: (1) input verification error: no valid validation of the data entered by users; (2) boundary condition error: no valid validation of boundary condition; (3) buffer overflow error: no length and format validation of the input buffer data; (4) access verification error: logical errors existing in access verification; (5) competitive condition error: timing or synchronization error; (6) configuration error: parameter or policy configuration error in system or software; (7) unexpected condition error: program logic does not take account of accident and exception; (8) other errors. Identifying the causes and characteristics information of the vulnerability is conducive to further classification and analysis of the software vulnerabilities.

B. Vulnerability Model

Based on the previous analysis of the causes and characteristics of vulnerabilities, we can conclude that the main reasons for exposing the software security problem is to establish the following software vulnerability model. Software vulnerability consists of the internal factors and external factors. The internal factors are mainly caused by unsafe coding inside the software itself which may lead to its internal security defects. Whiles the external factors are environmental issues which may cause the software not to run. What's more, external factors also include the memory, process, registration information, interface parameters, external code, network, disk files and so on. The vulnerability model is shown in figure 1.

The external environment factors can further be expressed as follows: EM = {IP, M, DF, PRS, NET, REG, EC}. Through the analysis of the causes of vulnerabilities, we can summarize the common causes of different external factors. We can summarize the common causes of different external factors namely as: interface parameters (IP), memory (M), disk file system (DF), process (PRS), network (NET), registration environment information (REG), external code (EC) and so on.

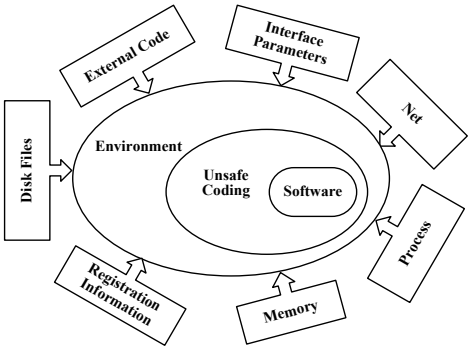

Figure 1 Vulnerability model.

According to the analysis above, the vulnerability model can be expressed as: VM = {IC, EM}, where IC represents insecure coding factor, EM represents external environment factor. We can also state the common reasons for vulnerabilities caused by software internal factors. That is to say, common insecure programming internal factors (IC) have the following 11 "misconduct": improper data validation (DO), improper type initialization (TI), improper reference of null pointers (NP), improper handling of the return values (RV), improper operation of numbers (NO), improper operation of class and method (CM), improper handling of exception (EO), improper handling of multi-threading processing (MT), improper operation of IO (IO), improper operation of serialization (SO), improper authority control (AC). The internal insecure coding factor can be expressed as follows: IC = {DO, TI, NP, RV, NO, CM, EO, MT, IO, SO, AC}.

III. VULNERABILITY MODEL BASED ON CHEMICAL ABSTRACT MACHINE

We can derive and testify software vulnerabilities by the abstract description of vulnerability attributes and the formal definition of the vulnerability model.

Suppose that there is a software fault called ξ, the attribute of the fault ξ is FaultAtt, and it can be converted into the attribute of the vulnerability, then the vulnerability detection operation can be represented as a function:

Γ：ξ.VulAtt∩VM≠Φ:→ξ∈VM, ξ is a vulnerability, and VulAtt={TypeIno, ReaFeat}, VM={IC, EM}, TypeIno={VC, UG, ST, CS, TC, OM, RS, DL}, ReaFeat={VN, NO, TD, MR, MF, VT, JR, RS, FD, PI}.

A. Chemical Abstract Machine

Chemical abstract machine (CHAM) was firstly proposed by Gerard Berry and others, which was used to describe software architecture in formal method [20]. A chemical abstract machine is specified by defining molecules m, m', etc., solutions S, S', etc., and transformation rules. Molecules are basic elements. Solutions are finite multisets of molecules, written as the set like {m, m', ……}. The transformation relation $S \rightarrow S'$

between the solutions is determined by the transformation rules.

The classification of transformation rules is given here. From the scope of application, the transformation rules can be divided into two types: special rules, which can be applied to certain molecules; general rules, which can be applied to all the molecules. From the scope of reaction effect, the transformation rules can be divided into two types: heating rules, which can break down macromolecules into small molecules; cooling rules, which can synthesize small molecules to macromolecules.

Usually, a transformation rule is expressed as:

$$M_1, M_2, \ldots\ldots, M_k \rightarrow M_1', M_2', \ldots\ldots, M_k'$$

If m_l, m_2, ……, m_k and m', m_2', ……, m_k' are the examples of the molecules, the corresponding transformation rule can be expressed as:

$$m_1, m_2, \ldots\ldots, m_k \rightarrow m_1', m_2', \ldots\ldots, m_k'$$

The multiset union of S and S' is written as $S \cup S'$. Reactions can be performed within any solutions:

$$\frac{S \rightarrow S'}{S \cup S' \rightarrow S' \cup S''}$$

B. CHAM description of vulnerability model

In order to understand the principle of vulnerability better, a CHAM description of the vulnerability model is given with chemical abstract machine to derive and testify software vulnerabilities. The vulnerability model described with CHAM includes the following four parts: (1) The molecular set M, which is divided into three categories: the data element D, the processing element P and the connection element C. The molecular set is relatively static, which belongs to the grammatical level. (2) The initial solution S_0, which represents the initial state. (3) The final solution S_f, which represents the final state. (4) The reaction rule set T, which is the basis of molecular reaction.

CHAM description of vulnerability model is described below:

Molecular Set：

M::= C| P| M◇M

C::= CHAM Syntax

D::= CHAM Syntax

P::= CHAM Syntax

Initial State：

S_0::= CHAM Syntax

Final State：

S_f::= CHAM Syntax

Rules：

T_i::= CHAM Syntax

In the description, the symbol "◇" represents the infix notation, which is used to express the state of processing elements on its input and output behavior. "//" is used to represent two parallel molecules or solutions. M represents molecules, which is composed of P or "◇" or "//" or C. C is the connection element, including Input (D), Output (D), and other operations. Input (D) represents the input operation, and Output (D) represents the output operation. T_i represents the i_{th} reaction rule (in which i=1, 2, 3, ...). S_0 is the initial state, and $S_0 \in S$. S_f is the final state, and $S_f \in S$.

D represents data elements, including: malicious parameters MC, normal parameters NC, final result data set R_Set and so on. The number of D will be flexible according to the actual number of parameters. P represents the processing element, the state of the change can be understood as "from left to right", the connection element before P is the state to be processed, the connection element behind P is the state that has be processed. P contains some main processors: the rule analyzer RUAA, the internal insecure coding analyzer IICA and the external environment factor analyzer EEFA. The details are as follows:

1) The rule analyzer RUAA: RUAA analyze the input parameters to determine whether there are unexpected characters. If there is, the analysis ends and the current code may cause vulnerability. If not, proceed with the steps that follow.

2) The internal insecure coding analyzer IICA: The current code can be analyzed by IICA, and the result set is IR_Set. If $IR\_Set \cap IC \neq \emptyset$, it indicates that there is vulnerability in the current code.

3) The external environment factor analyzer EEFA: The external environment of the current code can be analyzed by EEFA, and the result set is ER_Set. If $ER\_Set \cap EM \neq \emptyset$, it indicates that there is vulnerability in the current code.

(1) Reaction Rules

The main reaction rules are defined as follows based on the vulnerability model:

T1::= Input(Cond)◇RUAA→RUAA◇Input(NC)

T2::= Input(Cond)◇RUAA→RUAA◇Input(MC)

T3::= RUAA◇Input(MC), Output(Cond)◇IICA, Output(Cond)◇EEFA→IICA◇Output(IR_Set), EEFA◇Output(OR_Set)

T4::= RF◇Input(NC), Output(Cond)◇IF, Output(Cond)◇EEFA→IICA◇Output(IR_Set), EEFA◇Output(ER_Set)

Rules T_1 and T_2 represent that the rule processor RUAA gets the parameters and analyzes the type of the parameters, that is, the malicious parameter MC or the normal parameter NC. T_3 represents that if the rule processor RUAA analyzes that the acquired parameter is malicious, the corresponding result set IR_Set and ER_Set would be output after the analysis of the internal insecure coding analyzer IICA and the external environment factor analyzer EEFA. T_4 represents that if the rule processor RUAA concludes that the acquired parameter is normal, the corresponding result set IR_Set and ER_Set would be output after the analysis of the internal insecure coding analyzer IICA and the external environment factor analyzer EEFA. If $IR\_Set \cap IC \neq \emptyset$ or $ER\_Set \cap EM \neq \emptyset$, then the program is vulnerable.

In the process of deriving and testifying software vulnerability, it may be necessary to use the above rules in whole or in part, or some rules based on the current program may need to be customized to perfect the entire formalization process.

IV. System Implementation and Experimental Analysis

A. System implementation

After establishing the vulnerability model based on chemical abstract machine, a CHAM-based vulnerability model analysis system (CHAM-VMAS) is designed and implemented, which can automate the generation of CHAM description, the state transition process and the verification of the model for the given programs. The main steps involved in the model construction of the system are shown in figure 2, firstly load the source program that needs to be analyzed, secondly add the custom reaction rules, thirdly determine which conditions to choose (including whether to consider the external environment factors and the internal insecurity coding factors), then obtain the corresponding CHAM description and state transition process of the program, and finally output the results and conclusions.

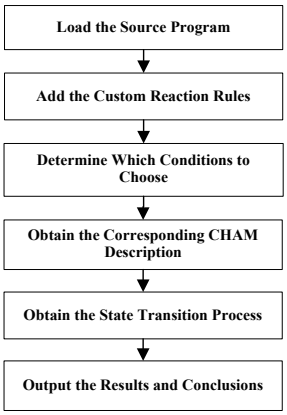

Figure 2 Operation flow of the prototype system.

The prototype system CHAM-VMAS is a windows forms application designed in the Java language. It consists of three components:

1) System Management: In this part, users can add, modify and delete the elements of the model.

2) Model constructing: In this part, users can upload the source code, and select the appropriate conditions according to the specific situation to add custom response rules (including whether to consider the external environment factors and whether to consider the internal insecure coding factors). When the input operation is completed, the system outputs the result set and the conclusion. The results can be exported by clicking the "Export Report" button.

3) Model validation: In this part, users can validate the model by using typical vulnerabilities.

B. Experimental Analysis

Based on the vulnerability model, a vulnerability detection method is integrated into CHAM-VMAS. The main steps of this method are as follows: (1) collect source programs; (2) construct the corresponding CHAM vulnerability model according to the source program; (3) compare the output of the CHAM vulnerability model with the cause and characteristic of the vulnerability, analyze whether there exits vulnerabilities and the number of the vulnerabilities in the program.

In order to further analyze the detection ability of CHAM-VMAS on software vulnerability, comparative experiments among the three testing tools (CHAM-VMAS, findbugs and PMD) are carried out on the 6 sub test case sets in test case set Juliet_Test_Suite_v1.2_for_Java, which includes resource access violation class (RA), arithmetic error class (OE), protection mechanism error calss(PM), the program logic error class (PL), improper handling of resources class (RP) and use the wrong type of structures (SU)). The detailed information of the six test case sets is shown in Table 1, and the experimental results are shown in figure 3. The column "No." represents the number of vulnerabilities for the corresponding test suites.

TABLE I. Information of Six Test Case Sets

Test suites	Description of the vulnerability	No.
RA	buffer overflow, arbitrary address writing, etc.	26
OE	integer overflow, division by zero, improper use of pointer, etc.	21
PM	improper access control, pass message in plain text, etc.	35
PL	improper handling of exception, use incorrect operators, etc	15
RP	improper security log handling, fail to remove sensitive information, etc.	41
SU	formatting string, redundant structure tags, etc.	18

It can be seen from figure 3 that the proposed method CHAM-VMAS has better detection effectiveness for the other five test suites except for test suite PL, followed by findbugs and PMD method.

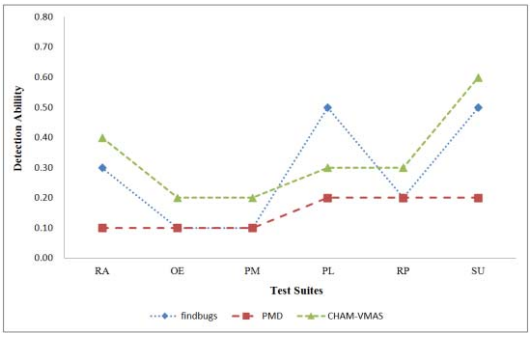

Figure 3 Detection capability comparisons of six sub test case sets.

V. Conclusion

In recent years, information security incidents are increasing due to the malicious use of software vulnerabilities. Software vulnerability analysis is of great significance to practitioners in the industry as it helps reduce the negative impact of vulnerabilities on society. Investigating and analyzing the causes and characteristics of

software vulnerabilities can improve the understanding of software vulnerability. It as well helps computer security experts to search, analyze and find unknown software vulnerabilities to take preventive measures.

This paper proposes a vulnerability model construction method by considering the vulnerability causes and characteristics. The proposed vulnerability model can describe and deduce the software vulnerability by using related concept of chemical abstract machine. Practical examples are subsequently analyzed. Examples show that the established vulnerability model in this paper can express the mechanism of vulnerability effectively, which can help people to further understand the nature of vulnerabilities. Furthermore, a prototype system is also designed and implemented for automating the generation of vulnerability model based on chemical abstract machine, and some experiments were performed. Experiment results show that the system based on the proposed model is more effective than other methods in the detection of software vulnerabilities. The proposed vulnerability model can promote the development of software vulnerability detection technology in future.

ACKNOWLEDGEMENTS

This work is partly supported by National Natural Science Foundation of China (NSFC grant numbers: 61202110 and 61502205), and the project of Jiangsu provincial Six Talent Peaks (Grant numbers: XYDXXJS-016).

REFERENCES

[1] CVND. 2016 CNVD Vulnerability Data Statistics Briefing [EB/OL]. http://www.cnvd.org.cn /webinfo/show/40-40.China, 2017

[2] Aslam T, Krsul I. Use of a taxonomy of security faults. eugene spafford. In Proceedings of the 19th National Information Systems Security Conference, 1996

[3] Krsul I. Software vulnerability analysis. Department of Computer Sciences,Purdue University, 1998

[4] Li P, Cui B. A comparative study on software vulnerability static analysis techniques and tools. In: IEEE International Conference on Information Theory and Information Security, Beijing, Dec.2010, pp.521-524

[5] Cadariu M, Bouwers E, Visser J, et al. Tracking known security vulnerabilities in proprietary software systems. IEEE, International Conference on Software Analysis, Evolution and Reengineering. IEEE Computer Society, 2015, pp.516-519

[6] Zhang S, Caragea D, Ou X. An empirical study on using the national vulnerability database to predict software vulnerabilities. International Conference on Database and Expert Systems Applications. Springer-Verlag, 2011, 6860:217-231

[7] Anand P. Overview of Root Causes of Software Vulnerabilities-Technical and User-Side Perspectives. In: International Conference on Software Security and Assurance (ICSSA), 2016, pp. 70-74

[8] T Scholte, Balzarotti D, Kirda E. Have things changed now? An empirical study on input validation vulnerabilities in web applications. Computers & Security, 31: 344-356, 2012

[9] Tang Y, Zhao F, Yang Y, Lu H, Zhou Y, Xu B. Predicting Vulnerable Components via Text Mining or Software Metrics? An Effort-Aware Perspective. In : IEEE International Conference on Software Quality, Reliability and Security (QRS), 2015, pp. 27-36.

[10] Kapur P, Yadavali V S, Shrivastava A. A comparative study of vulnerability discovery modeling and software reliability growth modeling. 2015 International Conference on Futuristic Trends on Computational Analysis and Knowledge Management (ABLAZE), 2015, pp.246-251

[11] Li H, Kim T, Bat-Erdene M, et al. Software vulnerability detection using backward trace analysis and symbolic execution. International Conference on Availability, Reliability and Security. IEEE Computer Society, 2013, 6(3):446-454

[12] Younis A A, Malaiya Y K, Ray I. Using attack surface entry points and reachability analysis to assess the risk of software vulnerability exploitability.IEEE, International Symposium on High-Assurance Systems Engineering. IEEE Computer Society, 2014, pp.1-8

[13] Anand A, Bhatt N. Vulnerability discovery modeling and weighted criteria based ranking. Journal of the Indian Society for Probability and Statistics, 2016, 17(1):1-10

[14] Chen J F, Chen J M, Huang R B, Guo Y C, Zhan Y Z. An Approach of Security Testing for Third-Party Component based on State Mutation, Security and Communication Networks (SCN), 2016, 9(15): 2827–2842

[15] Tang C L, Dong J Q, Dai D B et al. A similarity query algorithm for sequence pattern. Computer Research and Development, 2011: 132-139 (in Chinese)

[16] Chen J F, Zhu L L, Xie Z B, et al. An effective long string searching algorithm towards component security testing, China Communications, 2016, 13(11): 153-169

[17] Yamaguchi F, Golde N, Arp D, et al. Modeling and Discovering Vulnerabilities with Code Property Graphs. Security and Privacy, 2014, pp.590-604

[18] Singh D, Choudhary J P, De M. An effort to select a preferable metaheuristic model for knowledge discovery in data mining. Inderscience Publishers, 4(1): 57-90, 2015.

[19] Osman A M, Dafa-Allah A, Elhag A A M. Proposed security model for web based applications and services. International Conference on Communication, Control, Computing and Electronics Engineering. IEEE, 2017

[20] Chen J F, Lu Y S, Wang H H. Component security testing approach based on extended chemical abstract machine. International Journal of Software Engineering & Knowledge Engineering, 2012, 22(1):59-83