Taxonomy of Source Code Security Defects Based on Three-Dimension-Tree

Zhang Yan, Dong Guowei, Guo Tao, Yang Jianyu

Taxonomy of Source Code Security Defects Based on Three-Dimension-Tree*

ZHANG Yan[1,2,a], DONG Guowei [2,b], GUO Tao [2,c] YANG Jianyu[3,d]

[1]School of Computer Science and Engineering, Beihang University, Beijing, China

[2]China Information Technology Security Evaluation Center, Beijing, China

[3] China Agricultural University, College of Information and Electrical Engineering, Beijing, China

[a] zhangy@cse.buaa.edu.cn+, [b] donggw@itsec.gov.cn, [c] guotao@itsec.gov.cn,[d]ycjyyang@126.com

Keywords: three-dimension-tree; source code; security defect; taxonomy

Abstract. The authors present a new taxonomy for source code security defects based on three-dimension-tree, which considers the information of defect's cause, impact and representation synthetically. Case studies show that a sound system for classifying source code defects could be established with this taxonomy, and it is also good for the prevention and fixing of software vulnerabilities.

Introduction

Most security attacks are caused by the vulnerabilities of application system, and the defects that generated during software design and coding are the main source of them. Source code static analysis is an effective method for vulnerability reduction, because this method could consider the information of path widely and detect program's security defects automatically [1]. Generally, source code defects analysis includes lexical and syntax analysis, intermediate code generation and defect detection, as shown in Fig.1. First, source code is compiled to Abstract Syntax Tree (AST) by lexical and syntax analyzer. Second, AST is transformed into Intermediate Representation (IR), such as Control Flow Graph (CFG), Call Graph (CF), and so on. Finally, IR is checked with defect detection rules and all kinds of analysis techniques, and the results are reported.

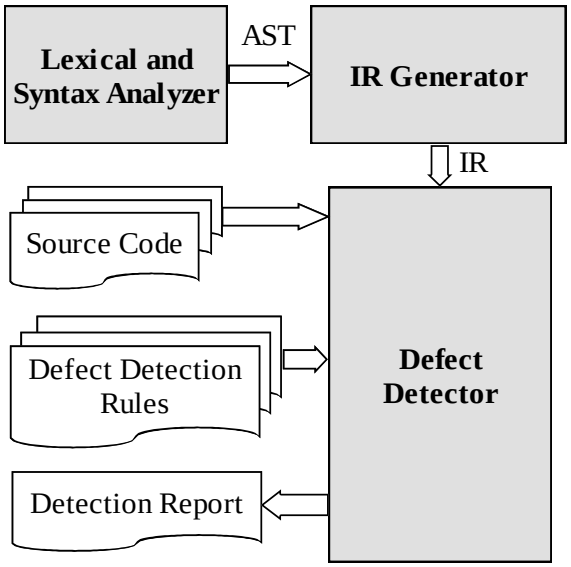

Fig.1 Flow Graph of Source Code Defect Detection

The construction of defect detection rule is an important step in above process, because the rule's description for detect affects analysis result directly. Classification of source code defects is helpful

* This work was supported in part by the National Natural Science Foundation of China (90818021, 41171309).

+ Corresponding Author.

for the refinement of detection rules and the accuracy of analysis. Also, it is better for learning defects' nature and cause. Generally speaking, classification of source code defects is good for the prevention and fixing of new kinds of defects.

At present, there is little taxonomy for source code defects, but the ones for software vulnerabilities are discussed by many researchers. In these works, vulnerabilities are sorted by cause [2-7], threat level [8], impact [4, 9-11], attack mode [5, 6, 11-15], fixing mode [10, 16], location [4, 5, 17], and so on. Although they involve most aspects of vulnerabilities, they are not suitable for source code defects. The main reason is that most of these methods only focus on one attribute of vulnerability.

In this article, we first introduce existing taxonomies of software vulnerabilities, and then present a kind of taxonomy for source code defects based on three-dimension-tree, which considers the information of defect's generation cause, impact and representation synthetically. At last, we sort the defects that are listed in CWE [18] and Fortify [19] with this taxonomy. Case studies show that a sound system for sorting source code defects could be established with this taxonomy, and it is also a guide for the prevention and fixing of software vulnerabilities.

This paper is organized as follow: in section 2, existing taxonomies of software vulnerabilities are introduced; in section 3, the taxonomy of source code defects based on three-dimension-tree is presented; in section 4, two case studies are given; in section 5, this paper is concluded.

Overview of Software Vulnerability Taxonomy

There are different definitions for software vulnerability as to different aspect, such as access control, state space, security strategy, etc [20]. Because of different requirement, existing taxonomies mostly focus on cause, impact, threat level, exploit mode, platform, and so on.

Introduction of Existing Taxonomies. As for Unix system, T. Aslam provided a taxonomy of functional errors based on cause [3]. He divided Unix errors into 4 kinds: design error, environment error, coding error, and configuration error. Design errors are issues that are generated during requirement analysis and software design; Environment errors are caused by the limitation of operation environment, such as errors that are result from compiler or OS defects; Coding errors mainly include synchronization errors, condition verification errors, etc; Configuration errors mainly include install location errors, install parameter errors and install permission errors.

F.B. Cohen presented an attack-mode-oriented taxonomy [12]. He analyzed more than 100 attack sets, and sort vulnerabilities into 18 categories: error and missing, unused value, implicit trustable attack, data spoofing, process bypass, distributed coordination attack, input overflow, Trojan horse, error after data integration, incomplete daemon, unreleased function use, misuse by attack, prohibit audit, failure lead by increasing system load, using network services and protocols, inter-process communication attack, race condition, improper default value.

I. Krsul provided impact-oriented taxonomy [9]. He pointed out that the impact of vulnerability could be divided into direct impact and indirect impact, and he sorted software vulnerabilities into data access, command execution, code execution and denial of service.

As to multi-factor taxonomy, C.E. Landwehr presented a vulnerability taxonomy based on source, time and location, in which source indicates Trojan horse, back door, logic bomb, etc; time refers to the parse that vulnerability takes place in software development life cycle, such as design, coding, maintenance; location means OS level, support software level or hardware level [5]. Upon this work, K. Jiwnani provided a taxonomy based on cause, location and impact for abstracting issues in software development [4]. In his method, cause indicates validation error, domain error, sequence or alias error, etc; Location refers to system initialization, memory management, process management or scheduling; Impact means unauthorized access, root or system access, denial of service, and so on.

D. Wenliang presented a vulnerability-life-cycle-based taxonomy [10]. He defined vulnerability life cycle as the process of "Import-Damage-Fixing", and sorted vulnerabilities by cause, direct impact and fixing mode. Specifically, cause indicates input validation error, permission certificate error, sequence or alias error, etc; Direct impact refers to illegal code execution, illegal target change,

illegal target resource access, denial of service, and so on; Fixing mode means entity false, entity missing, entity misplacing, entity error, etc.

CWE (Common Weakness Enumeration) is a defect dictionary provided by Mitre [18], which is used to provide a general criterion for identifying, reducing and preventing software defects. The last CWE version is 1.11, and it includes more than 800 kinds of defects. In CWE, defects are sorted into 3 classes, which are code defects environment defects and configuration defects, and the code defects includes executable code defects, source code defects and the defects that violates security design principle. Further, source code defects are divided into 14 subclasses: data handling, API abuse, security features, time and state, error handling, indicator of poor code quality, channel and path errors, handler errors, web problems, user interface errors, initialization and cleanup errors, pointer issues, insufficient encapsulation. It's easy to see that the taxonomy for CWE's source code defects is based on cause.

Disadvantages of Existing Taxonomies. Based on the discussion above, we can see that there is little taxonomy for source code defects, and the ones for software vulnerability are not suitable for source code defects. The reasons are as follow:

(1) There is little taxonomy special for source code defects. Source code defects (or coding errors) are often defined as an independent category of software vulnerability, but are not divided deeply [3-5, 10]. In CWE, source code defects are only sorted into 14 simple classes [18];

(2) Some taxonomies could not be used for source code defects classification. For example, the ones based on attack mode or impact mostly consider the factors such as exploiting results [9, 12], which could not directly reflect the information of source code defects;

(3) Existing works could not reflect various aspects of defects. Most of them only consider one attribute of defect, such as cause, impact, which could only represent one aspect of defect. In addition, there are many overlaps between the categories that are generated by these methods. All of these are disbenefit for source code defect analysis.

Taxonomy of Source Code Defects with Three-Dimension-Tree

Classification Attributions of Source Code Defect. In order to regularize the process of source code defect analysis and provide wonderful defect detection rules, we research the taxonomy of source code defects. Based on the analysis above, vulnerabilities are sorted upon different attributes. Similarly, when classifying defects, we could also consider their attributes. After widely studying, we find that programmers often describe defects with their 6 attributes:

(1) Internal cause. This means the issues in source code that are generated during coding, such as the use of dangerous functions;

(2) Intended or unintended subjective cause. Intended defects are imported by developer deliberately, such as logic bomb and undeclared channel, and unintended defects are imported because of programmer's lack of the knowledge of secure coding;

(3) External cause. This attribute mainly focus on the issues generated by the call of external library, for which special environment should be considered;

(4) Impact of defects. This means the direct impact that is caused by source code defect, for example, buffer overflow;

(5) Issues arose in testing or execution. These are the error features that are shown in testing or running, such as I/O errors, calculation errors, logic errors, data handling errors, configuration errors, OS errors, interface errors, global variable errors, system crash, etc;

(6) Developing language. Some defects arise in special language. For example, J2EE configuration errors are special for Java.

We consider the complexity of defects, the extension of taxonomy and the 6 attributes above synthetically, and then provide 3 classification attributes for source code defects, which are cause, impact and representation:

- Cause. This attribute includes the internal, external and subjective causes of defect generation. We have concluded 9 classes of defects in this aspect, which are input issues and validation, API errors, access control and password fail, share and race, exception handle, unsafe code, boundary treatment, configuration errors, malicious code. Details are shown in Tab.1.
- Impact. This attribute is the direct impact that defect produces. We have collected 9 classes and more than 30 subclasses of defects in this aspect, such as overflow, injection, manipulation, web attack, access control, leak, file system, deadlock, and denial of service. Details are shown in Tab.2.
- Representation. This means the form that defect presents in source code, which could also be seen as the form of code with issue, and secure issues may arise when running this code. Of course, some representations are related with special language. We have summarized 13 classes and more than 150 subclasses of defects in this aspect, such as pretreatment, declaration and initialization, expression, integer, float, array, string, memory management, input and output, object oriented, concurrency, as shown in Tab.3.

Cause	Intended	Malicious code	
	Unintended	Environment independent	Input issues and validation
			API errors
			Access control and password fail
			Share and race
			Exception handle
			Unsafe code
			Boundary treatment
		Environment dependent	Configuration errors

Tab.1 Causes of Defects

Impact	Injection	SQL injection
		Command injection
		XML injection
		……
	Overflow	Buffer overflow
		Integer overflow
		……
	Manipulation	Path manipulation
		Configuration manipulation
		……
	Access Control	Password crack
		Poor lock
		Race condition
		……
	Leak	Resource leak
		Memory leak
		Information leak
		……
	File system	File upload
		File include
		……
	……	

Tab. 2 Impacts of Defects

Representation	String Manipulation	Misjudgement of length of string
		Format string
		String iteration
		……
	Semaphore	Single member field
		Semaphore handle
		……
	Expression	Expression is always true
		Expression is always false
		……
	Exception handle	Empty Catch block
		Unhandled exception
		Overly broad throws
		……
	Math	Confusion of math operators
		Mix of mathematical type
		Divided by 0
		……
	Constant	Unreasonable definition of constant
		Out of bounds
		……
	……	

Taxonomy Based on Three- Dimension-Tree. We consider 3 attributes when sorting source code defects. That is, the category of a defect is decided by its cause, impact and representation. From Tab.1, 2, 3, we can see that all of the 3 classification attributes satisfy multi-level containment, which

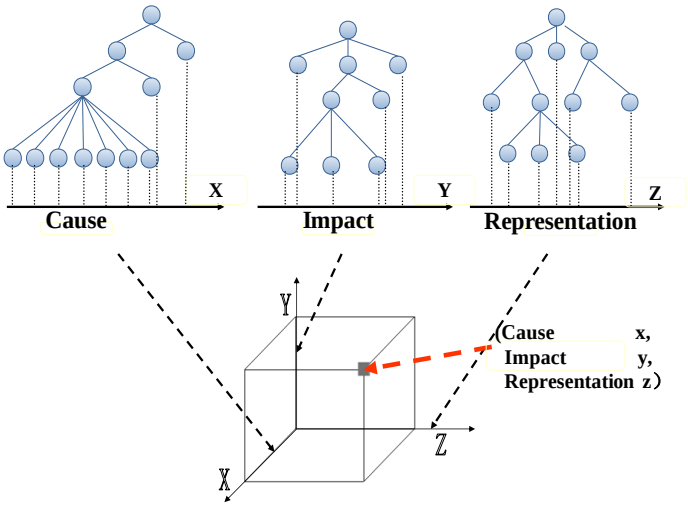

Fig.2 Three-Dimension-Tree

could be described with tree structure. So we represent each attribute with a tree, and the set that includes all leaf nodes of one tree is also the set of corresponding attribute's final categories. Details are shown in Fig.2.

Definition (Three-Dimension-Tree Taxonomy) Let $Tree_{re}$, $Tree_{rt}$,and $Tree_{rp}$ be the trees that represent defect's cause, impact and representation, and $child(n)$ the child node of node n in tree, if $Re=\{re|(re\in Tree_{re}) \wedge (child(re)==\text{Null})\}$, $Rt=\{rt|(rt\in Tree_{rt}) \wedge (child(rt)==\text{Null})\}$, and $Rp=\{rp|(rp\in Tree_{rp}) \wedge (child(rp)==\text{Null})\}$, then $\{(a, b, c) \mid (a\in Re) \wedge (b\in Rt) \wedge (c\in Rp)\}$ is the set of defect categories that are generated by the Three-Dimension-Tree Taxonomy, and each (a, b, c) represents a category of defects.

As we see, this taxonomy could reflect the information and feature of defect in multi-aspects, which is good for the construction of defect detection rules. Besides, it is easy to extend categories in this method. When new defects come out, we should only insert 3 new nodes into 3 trees at suitable positions.

Nomenclature of Defect Categories. In this taxonomy, we name a kind of defects with a triple that includes the names of its corresponding cause, impact and representation, which is intuitive and applied. Defect's features can be reflected by its name, which is good for defects' modification and statistics. For example, we name the kind of defect that is described in code

```
#include <stdio.h>
int main(){
   char fixed_buf[10];
   sprintf(fixed_buf,"Very long format string\n");
   return 0;
}
```

with (Input issues and validation, Buffer overflow, Array out of bounds in Sprintf()).

Case Study

Classification of CWE Defects. We classify about 110 items of source code defects listed in CWE with new taxonomy, and acquire 146 categories. Tab. 4 illuminates 12 of them. The second column indicates the item of defects that corresponding CWE ID denotes.

Following conclusions can be drawn from Tab.4:

id	CWEID	Three-Dimension-Tree Taxonomy		
		Cause	Impact	Representation
1	113	Input issues and validation	HTTP response truncation	HTTP Cookie with incredible data
2	79	Input issues and validation	Cross-site scripting	
3	563	Unsafe code	Dead code	Unused variable
4	563	Unsafe code	Unsafe style	Coverage of independent increment
5	242, 676	API errors	Dangerous Function	Mac OS function
6	242, 676	API errors	Dangerous Function	Unix function
7	404	Unsafe code	False release	
8	404, 772	Boundary treatment	Unreleased Resource	File unclosed
9	103	Input issues and validation	Struts errors	validate() error
10	104	Input issues and validation	Struts errors	No inheritance of Validation
11	105	Input issues and validation	Struts errors	Missing Validator
12	590	Input issues and validation	Misuse of memory	Memory for free() isn't provided by malloc()

Tab.4 12 categories of CWE Defects

(1) Defects' detail information can be presented with new taxonomy. For example, the 1st category of defects is caused by input, and may import the impact of HTTP response truncation. It's representation in code is one HTTP Cookie with incredible data. In addition, the 3rd and 4th categories are described as one item of defects in CWE (Code style and quality, ID is 563), but they are greatly different in impacts and representations, and these differences are embodied in new classification;

(2) The information about platform and language can be presented in new method. For example, the 5th and 6th categories are special for Mac OS and Unix respectively, while the 12th for C/C++;

(3) The attribute of representation in new taxonomy increases the intuition of defect. The 9th, 10th

id	Fortify Defects	Three-Dimension-Tree Taxonomy		
		Cause	Impact	Representation
1	Cross-site scripting	Input issues and validation	Cross-site scripting	Poor Validation
2	Buffer overflow	Input issues and validation	Buffer overflow	Format String(%f/%F)
3	Dead code	Unsafe code	Dead code	Unused variable
4	SQL injection	Input issues and validation	SQL injection	Hibernate
5	Access control	Access control and password fail	Access control	Anonymous LDAP Bind
6	Race condition	Share and race	Race condition	File system access
7	Memory leak	Unsafe code	Memory leak	Memory redistribution
8	System information leak	Boundary treatment	System information leak	Missing Catch Block
9	J2EE misconfiguration	configuration errors	J2EE misconfiguration	Missing Error Handling
10	Object model	API errors	Object model	Just one of equals() and

and 11th categories are all caused by input issues and validation, and all import Struts errors, but their representations are different, which are validate() error, no inheritance of Validation, and missing Validator. It's more intuitionistic.

Classification of Fortify Defects. Fortify is a famous tool for source code security analysis [19]. It could analyze programs in 19 languages. There are more than 300 items of defects in Fortify, and most of them are defined based on the description of defects in CWE and OWASP. We classify 86 items of them with new taxonomy, and get 194 categories. Tab.5 lists 10 of them, and the set of defects that are described with 3 attributes in each line is only a subclass of Fortify defects in the same line. That is to say, classes derived from three-dimension-tree taxonomy are more refined.

Conclusion

In this paper, a new taxonomy for source code security defects based on three-dimension-tree is present, which considers the information of defect's cause, impact and representation synthetically. Case studies show that a sound system for classifying source code defects can be established with this taxonomy, and it is also good for the prevention and fixing of software vulnerabilities.

Upon existing works, we will continue refining classification attributes. The taxonomy in this paper is based on 9 kinds of causes, more than 30 kinds of impacts and more than 150 kinds of representations, but some defects could not be accurately defined, for example, the representations of the 2nd and 4th categories in Tab.4 are blanks. Thus, our next goal is attributes refinement.

References

[1] H. Mei, Q.X. Wang, L. Zhang, and J. Wang: Software Analysis: A Road Map. Chinese Journal of Computers.Vol 32(9)(2009), p.1697-1710

[2] F. Piessens: A Taxonomy of Causes of Software Vulnerabilities in Internet Software. In Proceedings of the 13th International Symposium on Software Reliability Engineering (ISSR'02)(2002)

[3] T. Aslam: A Taxonomy of Security Faults in the Unix Operating System. Technique report TR-95-09, Department of Computer Science, Purdue University, West Lafayette, USA, (1995)

[4] K. Jiwnani, and M. Zelkowitz: Susceptibility Matrix: A New Aid to Software Auditing. IEEE Security and Privacy. Vol.2(2) (2004), p.16-21

[5] C.E. Landwehr, Bull A R, and J.P. McDermott: A Taxonomy of Computer Program Security Flaws with Examples. ACM Computing Surveys. Vol.26(3) (1994), p.211-254

[6] S. Weber, P.A. Karger, and A. Paradkar: A Software Flaw Taxonomy: Aiming Tools at Security. In Proceedings of the 2005 Software Engineering for Secure Systems (SESS'05)(2005)

[7] K. Tsipenyuk, B. Chess, and G. McGraw: Seven Pernicious Kingdoms: A Taxonomy of Software Security Errors. IEEE Security & Privacy. Vol.3(6) (2005), p.81-84

[8] R. Power: Current and Future Danger: A CSI Primer on Computer Crime and Information Warfare. San Francisco, USA: Computer Security Institute(1996)

[9] I. Krsul, E. Spafford, and M. Tripunitara: Computer Vulnerability Analysis. Technique report TR- 47909-1398, Department of Computer Science, Purdue University, West Lafayette, USA(1998)

[10] D. Wenliang, and A.P. Mathur: Categorization of Software Errors that Lead to Security Breaches. In Proceedings of the 21st National Information Systems Security Conference(1998)

[11] M. Bishop: A Taxonomy of Unix System and Network Vulnerabilities. Technical Report CSE-95-8, Dept. of Computer Science, University of California at Davis, Davis(1995)

[12] F.B. Cohen: Information System Attacks: A Preliminary Classification Scheme. Computers and Security. Vol. 16(1) (1997), p.26-49

[13] J.D. Howared: An Analysis of Security Incidents on the Internet 1989-1995. Pittsburgh, USA: Carnegie Mellon University(1997)

[14] K.S. Killourhy, R.A. Maxion, K.M. Tan: A Defense-centric Taxonomy Based on Attack Manifestations. In Proceedings of the 34th IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'04)(2004)
[15] S. Hansman, R. Hunt: A Taxonomy of Network and Computer Attack. Computers and Security. Vol. 24(1)(2005),p.31-43
[16] R.A. DeMillo, and A.P. Mathur: A Grammar-based Fault Classification Scheme and Its Application to the Classification of the Errors of Tex. Technique report, Department of Computer Science, Purdue University, West Lafayette, USA(1995)
[17] A. Bazaz A, J.D. Arthur: Towards a Taxonomy of Vulnerabilities. In Proceedings of the 40th Annual Hawaii International Conference on System Science(2007)
[18] Information on http://cwe.mitre.org/
[19] Information on http://www.fortify.com/
[20] M. Huang, and Q.K. Zeng: Research on Classification Attributes of Software Vulnerability. Computer Engineering. Vol. 36(1)(2010), p.184-186