A Novel Approach for Software Vulnerability Classification

Xiaodan Li, Duke University
Xiaolin Chang, Beijing Jiaotong University
John A. Board, Duke University
Kishor S. Trivedi, Duke University

Key words: software vulnerability classification; aging related vulnerability; attack pattern

SUMMARY & CONCLUSIONS

Software vulnerability analysis plays a critical role in the prevention and mitigation of software security attacks, and vulnerability classification constitutes a key part of this analysis. This paper proposes a new approach for software vulnerability classification, which is based on vulnerability characteristics including accumulation of errors or resources consumption, strict timing requirement and complex interactions between environment and software. We also present seven attack patterns and explore the mapping between vulnerability types and attack patterns. The proposed methods are used to analyze the vulnerabilities and the corresponding attacks reported by *Google Project Zero*. Examples of applying our classification approach to specific vulnerabilities are presented, together with a statistical analysis of the occurrence of different types of vulnerabilities. These results allow us to have a better understanding of software vulnerabilities and how they can be exploited, leading in the future to strategies to better equip programmers to avoid introducing them, and also helping us to formulate effective countermeasures.

We make three observations regarding software vulnerability classification: 1) Mandel vulnerabilities, especially NMVs (Non-Aging-related Mandel Vulnerabilities), account for the largest share of all classified vulnerabilities. 2) It takes more time and complex strategies to fix NMVs. 3) The major goal for attackers is to get elevation of privilege from a target system. The main cause of vulnerabilities is improper validation mechanisms.

1. INTRODUCTION

A software vulnerability is an instance of an error in the specification, development, or configuration of software such that its execution can violate the security policy [1]. It could be exploited by attackers to compromise a system, launch new attacks, and/or acquire sensitive information. The tremendous increase in the number of vulnerabilities discovered and disclosed highlights the necessity of vulnerability analysis, which could give practitioners a better understanding of vulnerability characteristics and their triggering/exploiting conditions, leading to better detection and mitigation strategies. As a key part of the analysis, various vulnerability classification schemes have been proposed based on different criteria such as software development lifecycle (SDLC) phase, affected technology, enabled attack scenarios, disclosure process and so on [2]. However, our literature investigation indicates that there is no research on vulnerability classification in terms of vulnerability characteristics, such as error accumulation process, strict timing requirement and complex interactions between environment and software. Characteristic-based classification of vulnerabilities is helpful for understanding exploitation conditions and then designing effective countermeasures.

This paper proposes an approach to classify vulnerabilities in terms of vulnerability characteristics, which determine the complexity of identifying, fixing and exploiting vulnerabilities. Each type/subtype of vulnerability can be exploited to launch certain type of attacks. For a specific kind of vulnerability, the corresponding attack patterns are similar, and these vulnerabilities can be exploited in fixed ways. Therefore, this paper also generalizes seven attack patterns and proposes the mapping between attack patterns and vulnerability types. The proposed approach not only enables a better understanding of vulnerabilities and related attack patterns, but can also help design effective countermeasures. Our approach is used to analyze vulnerabilities reported in *Google PZ* (*Project Zero*) [3], a Google project for finding vulnerabilities in popular software products.

Our contributions are summarized as follows:

1) A novel vulnerability classification is proposed, which classifies vulnerabilities according to the complexity in vulnerability identification, fixing and exploitation.
2) We propose a generalized set of attack patterns. A mapping between vulnerability types and attack patterns is proposed which facilitates not only the identification of unknown vulnerabilities when their corresponding patterns are known but also the selection of proper countermeasures according to vulnerability types.
3) The impact and cause, and remediation time of different types of vulnerabilities in *Google Project Zero* are studied.

The rest of the paper is organized as follows. Section II presents related work about vulnerability classification.

978-1-5090-5284-4/17/$31.00 ©2017 IEEE

Section III presents our novel approach to classify vulnerabilities. We first present the identified attack patterns. Then the mapping between vulnerability types and attack patterns is introduced. In Section IV, the results of vulnerability classification are presented. Section V concludes the paper.

2. Related Work

A software vulnerability is a security-sensitive bug. Although characteristic-based classification for bugs has been studied in [4], this bug classification scheme is not applicable for classifying vulnerabilities. The main reason is that some attributes for distinguishing bug subtypes cannot be used to classify vulnerabilities.

In [5], vulnerabilities were classified in terms of the software development lifecycle phase when they were introduced. Note that in some situations, there is no clear boundary among phases. Therefore, it is sometimes difficult to determine when a vulnerability is introduced. Moreover, some vulnerabilities are related to multiple phases. In [6], the vulnerabilities were classified in terms of genesis. Firstly, a vulnerability is categorized based on whether it is intentional or inadvertent. Then the intentional vulnerabilities were further classified into malicious or non-malicious ones. In [7]**Error! Reference source not found.**, the direct cause or immediate impact are used to categorize vulnerabilities. The vulnerability taxonomies specific for network protocols were presented by Akyol [8]. In [2], the vulnerabilities were also classified according to whether they are exploitable. In [9], vulnerabilities were classified according to their disclosure process. In [10]**Error! Reference source not found.**, vulnerabilities are classified based on impact, specific attack type, system components targeted, and vulnerability source. In [11], mobile operating system vulnerabilities were classified by elevation of privilege concepts. [12] had conducted extensive research into the taxonomies of attacks and vulnerability. The machine learning models are also applied to classify vulnerability. In [13], the naïve Bayes is used to classify vulnerabilities according to text information. Latent Dirichlet Allocation (LDA) model and SVM are applied to classify vulnerabilities from National Vulnerability Database (NVD) [14]. A classification based model is applied to assess risk of security-critical system [15].

After reviewing existing vulnerability classification schemes, a novel vulnerability classification is proposed. Its advantages should be as follows:

1. The novel approach takes into consideration the complexity of exploitation conditions and characteristics of vulnerabilities. The vulnerabilities are classified both from attackers' and defenders' view.
2. The novel approach can be applied to different kinds of software.
3. The classification of vulnerabilities is completed and there are no overlaps among different categories.

3. Vulnerability Classification

This section first presents vulnerability types we propose and then a vulnerability classification procedure is introduced. At last, the attack patterns and mapping are described.

4.3 3.1 Vulnerability classification

The vulnerability types are defined according to the complexity of identifying, fixing and exploiting vulnerabilities. The types are shown in Figure 1. This classification is based on a similar one used for classifying software bugs [4, 16]. Details of the each type are given in the following.

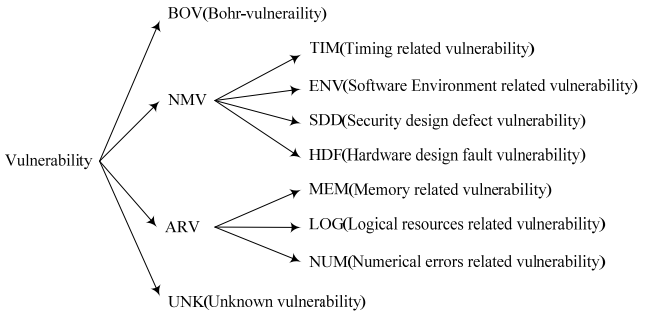

Figure 1-Vulnerability types

a. Bohr-Vulnerability (BOV)

This kind of vulnerability is easy to be identified and/or exploited. If the vulnerability is activated as long as some fixed set of conditions are met, a vulnerability can be regarded as BOV. Thus they can be easily patched by defenders and at the same time easily exploited by attackers on unpatched systems under a well-defined set of conditions.

b. Non-Aging-related Mandel Vulnerability (NMV)

NMV is a kind of vulnerability whose identification and/or exploitation is complex due to dependencies on factors external to the code.

There are four subtypes in NMV:

1) TIM. The success of an attack depends on exploitation of certain race conditions or other strict timing requirements.
2) ENV. The attacker can take advantage of complicated interactions between memory environment and software. (e.g. out of bound operation, improper pointer operation)
3) SDD. The vulnerability is caused by ill-considered security design. When engineers are designing software system, certain security issues are not taken into consideration. The attacker can take advantage of these design defects to exploit vulnerability. (e.g. Improper access control)
4) HDF. Complicated operations are necessary to exploit this kind of vulnerability since it is caused by a hardware design fault. The exploit is related to physical measures such as voltage. (e.g. Flipping DRAM bits)

c. Aging-Related Vulnerability (ARV)

ARV is a kind of vulnerability that can be exploited by attackers to increase failure rate and/or degrade performance as time passes. The vulnerability is related to an accumulation process which makes identification and/or exploitation difficult. The attacker can exploit the vulnerability by requesting memory, hardware resources and logical resources

repeatedly without freeing or by accumulating numerical errors. ARV has the following three subtypes:

1) MEM. The attacker can use this vulnerability to attack software system by requesting memory repeatedly without freeing.
2) LOG. The vulnerability can be exploited by consuming or leaking logical resources repeatedly (e.g., CPU computing time; inodes or sockets) without freeing.
3) NUM. The vulnerability can be utilized via accumulating numeric errors (e.g., integer overflow) which usually leads to other vulnerabilities such as buffer overflow.

d. Unknown Vulnerability (UNK)

UNK denotes the vulnerability which cannot be classified into the first three categories. It is caused by two reasons. Firstly, there is not enough information in the report. Another reason is that some vulnerabilities are too new to have a patch, so the reports are not open to public.

4.4 3.2 Vulnerability classification procedure

A vulnerability is a special kind of bug that can be exploited by attackers to lead to security failures. Our classification is based on three dimensions:

- The nature of the vulnerability
- The complexity of the exploitation process
- The possibility that a hacker can succeed in exploitation

The classification procedure is shown in Figure 2. The order of classification procedure depends on how obvious the characteristics can be identified in the vulnerabilities reports. Firstly, we check if the vulnerability is ARV according to its characteristics and subtypes (MEM, LOG, NUM), since the characteristics of ARV are easiest to identify in reports. If not, we will check if it is NMV (TIM, ENV, SDD, HDF). Otherwise, we will check if the vulnerability is easy to identify and/or to exploit. If so, it is classified as BOV. At last, if the information is not enough to determine its category, it is classified as UNK.

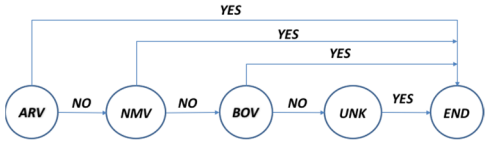

Figure 2-Classification flow chart

4.5 3.3 Attack pattern

The analysis of vulnerability reports indicates that some vulnerabilities can be exploited in certain fixed ways. Seven attack patterns are proposed which include buffer overflow attack, integer overflow attack (width overflow attack and integer signedness attack), time of check to time of use attack, race condition attack, use after free attack, type confusion attack, uninitialization attack and others. The mapping between vulnerability types and attack patterns is given in Figure 3.

4. RESULT ANALYSIS

This section first describes the vulnerabilities database. Then each type of vulnerability is analyzed in terms of cause and impact, time to fix (TTF) vulnerability and its statistics.

4.6

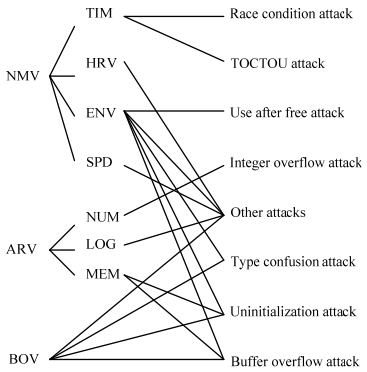

Figure 3-Mapping between vulnerability types and attack patterns

4.7 4.1 Vulnerability source

The vulnerabilities are from a Google security project, project zero [3]. This project aims to find vulnerabilities in popular software products, not only from Google but also from other companies such as Apple, Microsoft, Adobe and so on. An issue is only filed in the database if the Project Zero team believes there is a concrete security concern. We denote these vulnerabilities as the *Google PZ* vulnerability database in the following. 505 vulnerability reports are examined that were recorded from March 10, 2014 to the most recent vulnerabilities. CVE has validated the majority of these vulnerability reports, but the *Google PZ* team provides more detailed information for every vulnerability, such as vulnerability report date, vulnerability fix date, vendor, patch, and ways to exploit the vulnerability. What is more, the *Google PZ* vulnerability database also provides new vulnerabilities which are not yet reported in CVE (e.g., Issue 817). Note that in the *Google PZ* vulnerability database, each vulnerability has an ID and is denoted by "Issue N". N is issue number. For example, Issue 287 denotes stack overflow in printer virtualization under VMware Workstation.

4.8 4.2 Classification examples

In order to illustrate the classification, Table 1 presents examples for each of NMVs, ARVs and BOVs, along with statements from the report that provides information about exploits, errors accumulation and the security failure occurrence. From Table 1, we see that the difference among NMVs, ARVs and BOVs is that the conditions or operations to exploit BOVs are simple, while NMVs, ARVs need complicated, even tricky method to exploit. For example, Issue 141 involves the identification of specific memory locations and precise manipulation of addresses.

Table 1 Classification examples

Software system	Vulnerability ID	Type	Description
Windows	351	NMV/TIM	"If a process is created by a system service while impersonating another user their per-user drive mappings will still be used which could lead to EoP."
Adobe Reader X and XI	141	NMV/ENV	"The type of the crash and the memory context suggests that this is a *use-after-free vulnerability*: object fields and methods are accessed from memory which has been freed and assigned to some other allocation."
Safari(APPLE)	9	NMV/SDD	"Safari sandbox logic error enables reading of arbitrary files. *Don't give the inspector full access to the file system*. "
NaCl sandbox	283	NMV/HDF	"This issue is a placeholder for the PoC for a NaCl sandbox escape for the "rowhammer" DRAM vulnerability. It reports the *physical addresses of victim locations (memory locations where bit flips occur)* and aggressor locations (pairs of memory locations which cause the bit flips when accessed)."
Adobe Reader	249	ARV/MEM	"It is not necessary to have thousands of other subr 12 calls present in the font in verbatim to trigger the condition (although this is also feasible) - the font size can be greatly reduced by using nested subroutine calls to perform the attack. For example, the snippet of PostScript code shown below (consisting of 5 subroutines) inserts as many as *22 * (16 ^ 4) = 1441792 dwords of value 0x41414141 into the destination buffer*"
Linux kernel	88	ARV/LOG	"In the case of deadlock.iso, we point an inode to itself, leading to deadlock. In the case of recurse.iso, we *use a long chain of unique inode references* (100+). Because the resolution of the chain is implemented via recursive functions, we explode the kernel stack."
OSX-Kernel (APPLE)	38	ARV/NUM	"OS X IOKit kernel code execution due to *integer overflow* in IOBluetooth DataQueue (root only)"
Kaspersky	525	BOV	"That's a bug, because if index < SIZEOF_JMP, it will wrap and never exit. I would think it should decrement by 1 not sizeof(jmp) anyway, because jmps do not have to be aligned, but I don't know anything about ExeCryptor - maybe it makes sense."

4.9 Classification results

1) Vulnerability type analysis

In this section, we conduct statistical analysis on classification results with respect to vulnerability types, causes and impacts. Figure 4 - Figure 8 show the results. From these results, we observe that:

- NMVs have the largest share of vulnerabilities, shown in Figure 4. The reason is that most of the exploits are related to virtual address spaces such as stack, heap and so on. The malicious users can get elevation of privilege (EoP) or execute their own code by manipulating control data such as return addresses or function pointers. What is more, the total number of NMVs and ARVs is three times more than that of BOVs, suggesting that the triggering conditions for most of exploitable vulnerabilities are complex and their exploitations are hence random.
- Figure 5 and 6 summarize the proportion of impacts and causes of vulnerabilities. Figure 5 indicates that most common hacking aims at EoP. The malicious users want to control the system instead of just crashing it, which can bring them more benefits. According to Figure 6, the most common causes are improper validation mechanisms. It includes improper input type checking, improper return value type checking, bad bound checking and so on. Use after free (UAF) is also another crucial reason.
- Figure 7 shows that there is a strong predominance of ENVs in NMVs. The ENVs include improper pointer operations and memory management such as use after free, type confusion and uninitialization. The reason is that most of exploits are related to malicious manipulation of address spaces, so hackers need to figure out memory layout in the target computer. As to SDDs (security policy defect), they result from errors in the design stage. Therefore, it is hard to identify and fix SDDs.

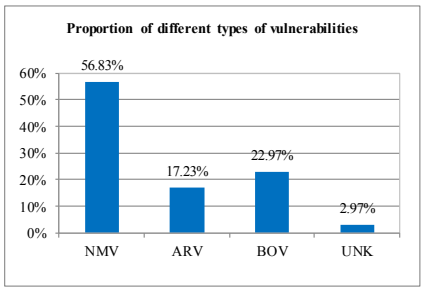

Figure 4-Proportion of different types of vulnerabilities

In Figure 8, we can see that MEMs are 52.9% of total ARVs. MEMs result from repeated requests for memory from the operating system without freeing it. This will keep consuming memory resources until the system crashes. The share of MEMs is large, because most of attacks are launched by manipulating address space. MEMs can be used to conduct

denial of service (DoS) attack and elevate privilege. However, this subtype is different from ENVs (NMVs) due to accumulation process. What is more, NUMs also have a big share of vulnerabilities since integer overflow vulnerability is very common.

2) Time to fix vulnerability analysis

In order to have further understanding of the effect of vulnerabilities on the vulnerability management process, the time to fix vulnerability is analyzed. We measure fixing time for every vulnerability as Time to Fix Vulnerability = closedTime - openTime. This time period includes the time for developers to reproduce reported vulnerabilities, analyze root cause, figure out countermeasures and validate effectiveness of the countermeasures, but does not include time to identify vulnerability [11].

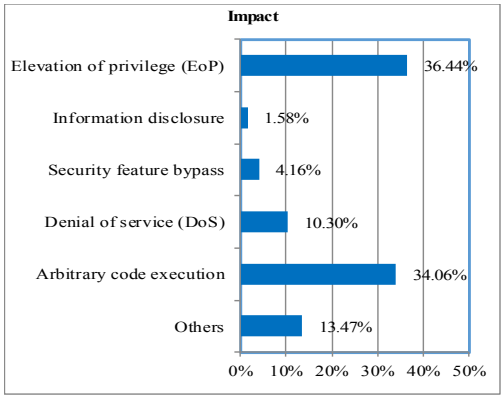

Figure 5-Proportion of impact

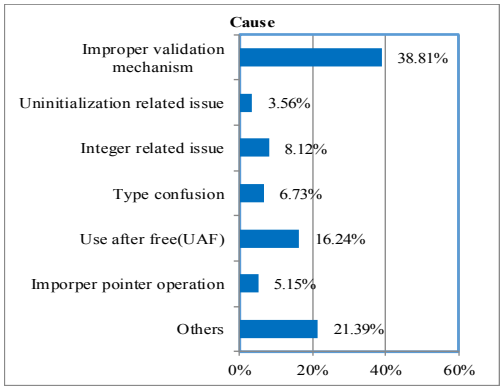

Figure 6-Proportion of causes

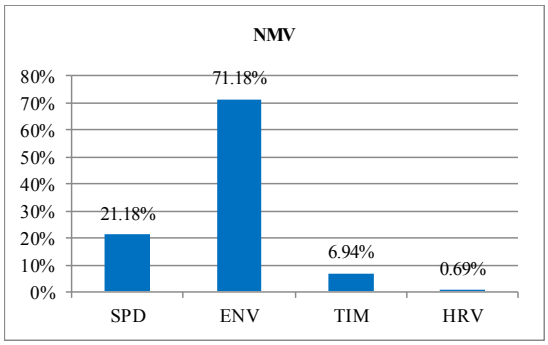

Figure 7-Proportion of subtypes in NMV

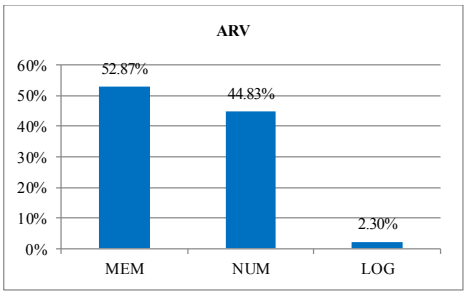

Figure 8-Proportion of subtypes in ARV

The statistics of time to fix vulnerability are summarized in Table 2. The 95% confidence intervals from *t* distribution for mean are also given. Table 2 indicates that the average time to fix vulnerabilities is 67.5 days. As for time to fix BOVs, though they are easy to identify, fixing them also needs much time (66.8 days). In terms of time to fix ARVs, which is contrary to our intuition, it takes the least time to fix. The first possible cause is that its complexity concentrates on identification stage and if the vulnerability is identified, it is easy to fix. For example, software rejuvenation can be used to deal with the problem [4]. What is more, the majority of vulnerabilities which have been studied for many years. Therefore, the countermeasures for them are effective and efficient. As to the time to fix NMVs, they take the longest time to fix. NMVs involve ill-considered security policy, race conditions, complicated operation of memory and undetected hardware defects. Therefore, NMVs are complicated in terms of both identification and fixing stages. According to standard deviation, BOVs has larger standard deviation which means the time to fix BOVs varies a lot. The time to fix NMVs is more centralized compared with BOVs and ARVs. We believe the reason is that the vulnerabilities in BOVs are more specific, so the countermeasures are more varied. However, the NMVs have similarity in exploitation, so the ways to fix vulnerabilities are similar.

Table 2 Summary statistics of time to fix vulnerability

	Standard deviation	Mean(Confidence interval)
Time to fix vulnerability (in days)	34.8	67.5(64.7, 70.3)
Time to fix BOVs (in days)	37.3	66.8(59.9, 73.7)
Time to fix NMVs (in days)	33.1	70.5(66.6, 74.4)
Time to fix ARVs (in days)	36.5	60.6(52.8, 68.4)

5. CONCLUSIONS AND DISCUSSIONS

This paper explores vulnerability classification according

to potential manifestation characteristics of vulnerabilities. In our classification approach, the vulnerabilities are divided into three categories: ARVs (Aging related vulnerability), NMVs (Non aging Mandel Vulnerability) and BOVs (Bohr Vulnerability). We also generalize seven attack patterns and use the vulnerability classification to analyze vulnerabilities reported by the *Google PZ* team. Our findings are as follows:

Mandel vulnerabilities, especially NMVs, account for most of all classified vulnerabilities. Mandel vulnerabilities (NMVs and ARVs) take 74.0% (56.8% and 17.2% respectively) of all classified vulnerabilities. In fact, most of exploits are related to malicious manipulation in memory or address space such as improper pointer operations. NMVs account for the largest share of classified vulnerabilities.

NMVs take more time to fix, and require more complex and specific strategies to be coped with. The general time to fix vulnerability is 67.5 days. As to time to fix NMVs, it takes 70.5 days to fix. NMV involves complex causes such as ill-considered security policy, race condition, complex memory operations and undetected hardware defects. Therefore, it takes more time and complex strategies to deal with NMVs.

The major impact is to get elevation of privilege from a target system. The main cause of vulnerabilities is improper validation mechanism. The predominance of EoP suggests that the major goal of hackers is to get higher privilege from a target system, consequently controlling the system and accessing important information instead of just crashing the system. In order to avoid security failures, more accurate validation mechanisms are needed.

All of our work is based on reports from *Google Project Zero*, so we assume the information from reports is accurate and authentic. The future direction of work is to conduct hacking experiments based on reports for different kinds of vulnerabilities and their attack patterns. The reports in *Google PZ* mainly focus on application vulnerability. The proposed classification schemes will be extended to classify web vulnerabilities.

ACKNOWLEDGEMENTS

This research was supported in part under US NSF grant number CNS-1523994, by the US Navy NEEC grant number N00174-16-C-0036, by IBM under a faculty grant, and by NATO under Science for Peace project number 984425. We thank Dr. Jose M. Martinez for feedback on this work.

REFERENCES

1. Ivan Victor Krsul, “Software vulnerability analysis,” Ph.D. dissertation, Purdue University, 1998.
2. Pascal Meunier: Classes of vulnerabilities and attacks. Wiley Handbook of Science and Technology for Homeland Security, 2008.
3. https://bugs.chromium.org/p/project-zero/.
4. Michael Grottke, Kishor S. Trivedi: Software faults, software aging and software rejuvenation. Journal of the Reliability Engineering Association of Japan, 2005, 27(7): 425-438.
5. Piessens F, A (2002) A Taxonomy of causes of software vulnerabilities in Internet software, Supplementary Proceedings of the 13th International Symposium on Software Reliability Engineering (Vouk, M., ed.), pp. 47-52.
6. Weber S, Karger PA and Paradkar A (2005) A software flaw taxonomy: Aiming tools at security. Software Engineering for Secure Systems (SESS’05).
7. Howard, M, LeBlanc, D and Viega, J (2005) 19 Deadly Sins of Software Security. Emeryville, CA: McGraw-Hill/Osborne.
8. Pothamsetty V, Akyol, BA (2004): A vulnerability taxonomy for network protocols: Corresponding engineering best practice countermeasures. Communications, Internet, and Information Technology 2004. St. Thomas, US Virgin Islands.
9. Christey, S and Wysopal, C (2002) Responsible Vulnerability Disclosure Process. INTERNET-DRAFT "draft-christey-wysopal-vuln-disclosure-00.txt". The Internet Society.
10. Igure, Vinay M., and Ronald D. Williams. "Taxonomies of attacks and vulnerabilities in computer systems." Communications Surveys & Tutorials, IEEE 10.1 (2008): 6-19.
11. Rangwala M, Zhang P, Zou X, et al. A taxonomy of privilege escalation attacks in Android applications. International Journal of Security and Networks, 2014, 9(1): 40-55.
12. Joshi, Chanchala, Umesh Kumar Singh, and Kapil Tarey. "A Review on Taxonomies of Attacks and Vulnerability in Computer and Network System."International Journal 5.1 (2015).
13. Wijayasekara, Dumidu, Milos Manic, and Miles McQueen. "Vulnerability identification and classification via text mining bug databases." In IECON 2014-40th Annual Conference of the IEEE Industrial Electronics Society, pp. 3612-3618. IEEE, 2014.
14. Shuai, Bo, Haifeng Li, Mengjun Li, Quan Zhang, and Chaojing Tang. "Automatic classification for vulnerability based on machine learning." InInformation and Automation (ICIA), 2013 IEEE International Conference on, pp. 312-318. IEEE, 2013.
15. Wang, Tai–ran, Vincent Mousseau, Nicola Pedroni, and Enrico Zio. "Assessing the Performance of a Classification-Based Vulnerability Analysis Model." Risk Analysis 35, no. 9 (2015): 1674-1689.
16. Domenico Cotroneo, Michael Grottke, Roberto Natella, Roberto Pietrantuono, Kishor S. Trivedi: Fault triggers in open-source software: An experience report. ISSRE 2013: 178-187.

BIOGRAPHIES

Xiaodan Li
Department of Electrical and Computer Engineering
Duke University
Box 90291 Durham, NC, 27708-0291, USA

e-mail: xiaodan.li@duke.edu

Xiaodan Li is a second year Phd student in Duke University. In 2012, he received his Master's degree in Industrial Engineering from Beihang University, china.

Xiaolin Chang
School of Computer and Information Technology
Beijing Jiaotong University

xlchang@bjtu.edu.cn

Xiaolin Chang is currently an associate professor in School of Computer and Information Technology at Beijing Jiaotong University. Her current research interests include Cloud data center and Network security.

John A. Board. Jr
Department of Electrical and Computer Engineering
Duke University
Box 90291 Durham NC 27708-0291, USA

e-mail: john.board@duke.edu

John Board is an Associate Professor in Electrical and Computer Engineering, and associate Chief Information Officer for Duke University. His interests are in high performance computing and networking, and embedded systems.

Kishor S. Trivedi
Department of Electrical and Computer Engineering
Duke University
Box 90291 Durham, NC, 27708-0291, USA

e-mail: ktrivedi@duke.edu

Kishor Trivedi is Hudson Chair Professor at Duke University. He heads the Duke High Availability Assurance Laboratory with over 40 years in advancing methods for evaluating performance and dependability in critical industries. Trivedi authored numerous texts and publications on probabilistic modeling, with an upcoming text on dependability engineering with inclusions from collaborators worldwide.