ConfVD: System Reactions Analysis and Evaluation Through Misconfiguration Injection

Shanshan Li, Wang Li, Xiangke Liao, *Member, IEEE*, Shaoliang Peng, Shulin Zhou, Zhouyang Jia, and Teng Wang

***Abstract*—In recent years, misconfigurations have become one of the major causes of software system failures, resulting in numerous service outages. What is worse, misconfigurations are also costly to diagnose and troubleshoot. This remains a great challenge for sysadmins (system administrators) to detect, diagnose, or troubleshoot these misconfigurations. Unlike software bugs, misconfigurations are more vulnerable to sysadmins' mistakes. Developers and researchers are attempting to improve system reactions to misconfigurations to ease the burden of sysadmins' diagnoses. Such efforts would greatly benefit from the techniques that can comprehensively detect bad system reactions through injected misconfigurations. Unfortunately, few such studies have achieved the above goal in the past, primarily because they only relied on generic alterations and failed to find a way to systematically generate misconfigurations. In this paper, we study eight mature open-source and commercial software packages and summarize a fine-grained classification of option types. Based on this classification, we use Augmented Backus–Naur Form to summarize and extract syntactic and semantic constraints of each type. In order to generate comprehensive misconfigurations in the test systems, we propose misconfiguration generation methods for our constraints. We implement a tool named Configuration Vulnerability Detector (ConfVD) to conduct misconfiguration injection and further analyze the systems' reaction abilities to various misconfigurations. We carried out comprehensive analyses upon Apache Httpd, MySQL, PostgreSQL, and Yum. The results of our analysis show that our option classification covers 96% of 1582 options from the above-mentioned systems. Our constraints are more fine grained than previous works and their accuracy was found to be 91% (ascertained by manual verification). Our technique could improve generic alteration approaches without constraints, and we found that ConfVD could find nearly three times the bad reactions that were found by ConfErr. In total, we found 65 bad reactions from the systems being tested and our fine-grained constraints contributed 27.7% more bad reactions than techniques only using coarse-grained constraints.**

***Index Terms*—Constraints, misconfiguration, system reactions, testing.**

I. Introduction

NOWADAYS, system administrators (sysadmins) are faced with an ever-increasing complexity of configuration. This is reflected in the large and still increasing number of configuration parameters as well as various configuration constraints and consistency requirements [1]. For instance, popular open-source software systems (such as MySQL, Httpd, and PostgreSQL) each have more than 200 configuration options of various types. It is hardly surprising that misconfiguration has become a critical issue. Several research groups [2]–[5] have revealed that misconfigurations are largely responsible for the deterioration of software reliability. One report [6] indicated that misconfigurations caused 30% of all failures in a commercial storage system. Meanwhile, popular commercial systems (like Microsoft Azure [7], Amazon web services [8], and Facebook service [9]) have suffered from misconfigurations (e.g., outages) in recent years. The seriousness of the misconfiguration problem is often underestimated and has caused such companies major financial losses every year. According to Computing Research Association's report [10], 60%–80% of the capital outlay in IT departments was spent on administrative expenses, where system configuration is one of the major operations [11].

Unfortunately, diagnosis of misconfigurations is troublesome. This is mainly explained as follows:

1) Root causes of misconfigurations are highly mixed. Fig. 1(a) indicates that misconfigurations result not only from human mistakes but also inappropriate software implementation. Even though misconfigurations are always made by sysadmins, inappropriate software implementation [e.g., "misconfiguration B" in Fig. 1(a), the system rolls back the sysadmin's setting without any notification] has given rise to avoidable sysadmin misunderstandings and system failures.
2) Misconfigurations are hard to detect before they are triggered. Fig. 1(b) gives an example of how software Yum failed to detect a latent configuration error. In this case, the value of the option "cachedir" was incorrectly set, but Yum still worked until it was called upon to access the local cache. Although regression testing is widely used in software development processes, misconfigurations are more likely to be introduced by sysadmins than developers. This is why so many misconfigurations have been missed by the checkers.

Manuscript received July 14, 2017; revised January 25, 2018 and May 12, 2018; accepted August 5, 2018. Date of publication September 24, 2018; date of current version November 29, 2018. This work was supported in part by National Natural Science Foundation under Grant 61690203 and Grant 61532007 and in part by National 973 Program (2014CB340703) of China. This paper was presented at the 21st International Conference on Evaluation and Assessment in Software Engineering, Jun. 15–16, 2017, Karlskrona, Sweden [36]. Associate Editor: Z. Chen. *(Corresponding author: Wang Li.)*

The authors are with the National University of Defense Technology, Changsha 410073, China (e-mail: shanshanli@nudt.edu.cn; liwang2015@nudt.edu.cn; xkliao@nudt.edu.cn; pengshaoliang@nudt.edu.cn; zhoushulin@nudt.edu.cn; jiazhouyang@nudt.edu.cn; wangteng13@nudt.edu.cn).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TR.2018.2865962

Misconfiguration A: symbolic_lin=0 /* MySQL */
Root cause: typo mistakes of administrator: omission ‘k’ in “symbolic_link”
Misconfiguration B: symbolic_link=yes
Root cause: system can’t recognize the string “yes” and rollbacks the value of the parameter to “0” without notification to users.

(a)

Misconfiguration: /* Yum */
cachedir=/nonexistent/directory
Symptoms:
Yum will fail if it needs to access the local cache, but works well if not

(b)

Misconfiguration: /* MySQL */
socket=/nonexistent/filename
Symptoms:
MySQL fails after startup and prints logs: “Can’t start server : Bind on unix socket: Address already in use”

(c)

Fig. 1. Challenges in diagnosing misconfiguration. (a) Complicated misconfiguration root cause. (b) Study case of Yum. Misconfigurations is latent in system until triggered by specified functions. (c) Study case of MySQL. Poor feedbacks of MySQL obstruct the diagnoses of the misconfiguration.

3) The lack of feedback also obstructs the diagnosis of misconfigurations. Fig. 1(c) shows that MySQL reports an error message “can’t start server,” which is not helpful for diagnosing misconfigurations. Sysadmins cannot debug system problems like developers can, and without adequate guidance, there does not seem much likelihood of sysadmins solving misconfigurations on their own.

~~The severity of misconfigurations has inspired many research efforts. Most of them concentrate on detecting and diagnosing misconfigurations. Different approaches (such as program analysis approaches [12]–[15]) use data flow to automatically diagnose misconfigurations. Statistical approaches (such as [16], STRIDER [17], and EnCore [18]) diagnose misconfigurations by learning configuration rules from configuration files.~~

~~Although~~ the previous approaches have significantly improved the situation, the fundamental issue of misconfigurations probably lies in the process of configuration design and implementation. In terms of in-house testing, insufficient research has been undertaken on detecting system bad reactions (also known as vulnerability) in handling misconfigurations. ConfErr [19] and SPEX [20] analyze system reactions by generating and testing misconfigurations, with the aim of enhancing systems’ abilities to fight against these misconfigurations. In this paper, we refer to such works as “Misconfiguration Testing” approach.

~~Researchers~~ and developers would benefit greatly from a mature misconfiguration testing approach that conducts a comprehensive analysis of a system’s ability to react to a misconfiguration. A good software system reaction, which means error indication and error handling, would greatly ease the burden of sysadmins diagnosis of misconfigurations. Unfortunately, few such studies have been conducted. Despite being a pioneer in this field, ConfErr relies on generic alterations to generate misconfigurations, which weakens its capability to analyze system reactions. SPEX, taking it a step further, infers five main categories of constraints (e.g., the condition with which the correct configuration must be consistent), but it is coarse-grained in its option types; therefore, as a result of poor diversity, SPEX does not propose a systematic way of generating those misconfigurations. Consequently, the variance in the reactions of misconfigurations cannot be observed by researchers and developers.

~~With~~ the aim of improving software reliability, we have implemented a tool known as Configuration Vulnerability Detector (ConfVD) to conduct error injection tests on software systems. In order to more comprehensively study systems abilities to react to misconfigurations, ConfVD uses misconfiguration generation methods to help inject misconfigurations into the targeted system in a more systematic way. Furthermore, we have also tried to classify the systems reactions to misconfigurations, such as failures with inadequate diagnostic messages [21].

In this paper, we have summarized and classified 1593 software configurations from eight mature open-source and commercial software systems. Based on these classifications, we have then used normal forms, such as Augmented Backus–Naur Form (ABNF) [22], to summarize and extract the fine-grained option constraints. ABNF is a popular technical specification to define format syntax, and it balances compactness and simplicity. Different from RE, ABNF allows us to separately define each part of the option value, which helps us generate more comprehensive syntactic misconfiguration. Furthermore, by violating these constraints, we have proposed misconfiguration generation methods to generate and inject a variety of misconfigurations into systems. We have then analyzed the statistical characteristics of different system reactions and tried to reveal some design problems related to misconfiguration. Based on these problems, we have then made some suggestions for improving software reaction ability.

The contributions of this paper to the field are as follows:

1) In order to generate effective constraints for each type of configuration option, we have summarized a comprehensive classification based on a large number of configuration options from eight mature open-source and commercial software packages. Our classification is tree based and can be easily extended. Based on this classification, we have proposed syntactic and semantic constraints for each type. Our analysis results show that our option classification covers 96% of 1582 options from Httpd, Yum, PostgreSQL, and MySQL.
2) In this paper, we have considered option constraints, both syntactically and semantically. Our constraints use ABNF, which is more fine grained than previous techniques like EnCore and has been found to be consistent with 91% of real constraints using manual verification. Furthermore, in order to generate misconfigurations in a more systematic way, we have proposed misconfiguration generation methods for our constraints. The experimental results show that our fine-grained constraints find 27.7% more bad reactions than techniques using coarse-grained constraints.

TABLE I
NUMBERS OF OPTIONS WITH CLASSIFICATION, BY APPLICATIONS

Software	IP-address	Boolean	Mode	Path	Email	Count	Permission	Port	Domain Name	Name	Others	-
Squid-3.5	11	73	46	26	4	111	0	7	11	34	16	339
Nginx-1.10.3	7	114	124	65	1	186	5	0	7	104	24	637
Redis-3.2.8	2	13	9	6	0	33	0	1	0	0	6	70
Nagios-XI-5.6.3	0	33	10	17	2	37	0	0	0	16	8	123
Lighttpd-1.4.45	1	12	6	5	0	12	0	0	1	14	4	55
Puppet-4.10	0	45	1	77	0	19	0	1	6	57	2	208
SeaFile-6.0.4	0	15	2	1	0	13	0	1	3	1	2	38
Vsftpd-3.0.3	2	73	2	19	0	16	0	4	2	1	2	123

3) We have implemented the ConfVD tool to conduct misconfiguration injection and analyze system reaction abilities. Based on these results, ConfVD has revealed bad reactions and implementation problems in systems. We have evaluated the capability of ConfVD of finding bad system reactions and the results have shown that ConfVD finds nearly three times the bad reactions found by generic alteration approaches.
4) We have defined three types of system reactions to misconfigurations. Based on these types, we have calculated the distribution of system reactions and have analyzed the reasons for these reactions. We found that path misconfigurations might be difficult for systems to diagnose due to a lack of checking of the constraints, both syntactically and semantically. Our experimental results show that adequate configuration syntax checking after startup can effectively help diagnose misconfigurations.

The remainder of this paper is organized as follows: We present constraints generation in Section II. In Section III, we explain the process of misconfiguration generation. The analysis is explained in Section IV, and Section V details our experience and practice. Our related work is presented in Section VI, and we conclude this study in Section VII.

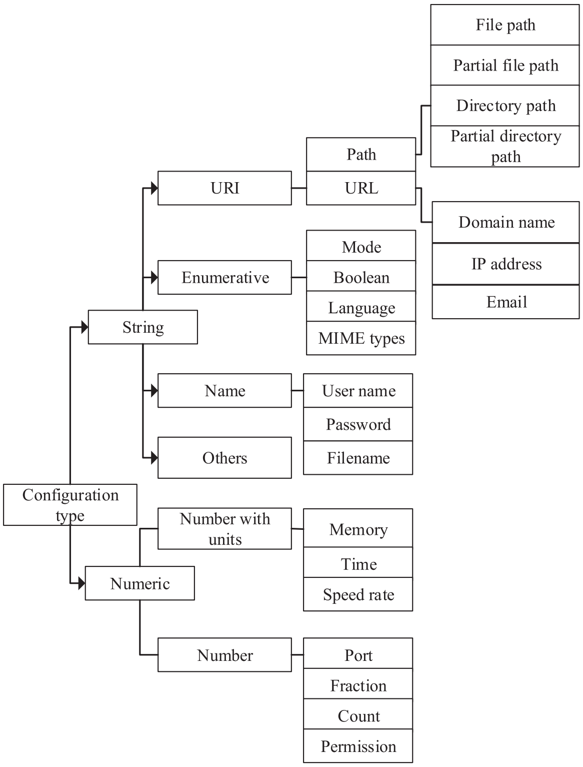


Fig. 2. Type classification.

II. CONSTRAINTS GENERATION

To understand configuration constraints (i.e., specification of configuration requirements) better, we study 1593 options in configuration files from eight widely used software packages in this section. In order to generate corresponding constraints for misconfiguration injection, we are attempting to classify these options by their types. Eight representative software systems in their field are selected, such as Squid, Nginx, Redis, Nagios, Lighttpd (core), Puppet, SeaFile, and Vsftpd. To extract the relative information (e.g., option name, the default value, official descriptions, etc.) for options, we manually investigate the configuration files as well as related official documents. A study [23] reveals that nowadays, a large proportion of configurations are in the form of key-value pairs. Since our study targets key-value format configuration and to make our study uniform, we transform options from other formats to key-value pairs.

A. Type Taxonomy

Although Rabkin and Katz [23] have studied type taxonomy, they focused on extracting configuration related code. However, our objective is to infer option-related constraints for misconfiguration injection. In order to consider all the options we find in configurations, a sufficiently fine-grained classification is required. As we encounter a new option, we count the number of occurrences of each option type. Table I statistically illustrates the main types of options in this paper. These configuration options are obtained mainly from documents such as guide book or sysadmin manual. If we can not access such information from the above documents, we also consider configuration files or the source code to speculate on options' types.

As shown in Table I, it can be obviously found that most options are well represented by a few sets of types. In Table I, "other" represents the program-specific option types that are never seen in other programs (e.g., "name=Fedora $releasever - $basearch Debug" in yum.conf). We use "other" in the classification for the reason that it is nontrivial to analyze those program-specific options, and we are aimed at proposing a common classification for generating misconfigurations. However, it does not mean that "other" options cannot be handled. In Section III, we propose two methods to achieve this goal. Fig. 2

TABLE II
COVERAGE RATE OF CLASSIFICATION

Software	Options	Coverage rate
Httpd	564	543(96.2%)
MySQL	671	643(95.8%)
PostgreSQL	273	270(98.9%)
Yum	74	71(95.9%)
Total	1582	1527(96.5%)

illustrates our classification as a tree. This classification tree is also scalable and we can easily supplement it with new option types.

The evaluation of our classification is based on 1582 options from other four open-source software systems (Httpd, MySQL, PostgreSQL, and Yum) as we carefully check whether each option can be well classified into Fig. 2. As Table II illustrates, the overall coverage rate [i.e., the proportion of the options if its type can be found in Fig. 2 (excluding “Others”)] is as high as 96.5%, and at least 95.8% for each system.

Although the classification in this paper is reasonably effective, we still meet some difficulties. First of all, we find that some options do not have a single type. Take MySQL’s option “LogFile” as an example: “LogFile” usually can be set with Path type by default; however, it also allows URL type for remote invocations. To remove the ambiguity, we only consider the default values’ type for the option. What is more, in our study, some software systems use their specific configurations, which are hard to be transformed into key-value pairs. For example, Httpd considers the options as directives, and with lots of arguments. In this case, we can not easily convert them into key-value pairs, so we only consider each argument’s type of option. “Others” type in Fig. 2 is a problem too, but, as mentioned above, it is trivial to analyze those program-specific options. For this problem, a practical solution is that our classification scales well and sysadmins can easily incorporate any new types into it.

配置的约束使用作者之前的工具获得

B. Type Constraints Inference

To comprehensively analyze and evaluate system reaction, our injected misconfigurations should be as complete as possible. For that purpose, we infer each configuration type’s fine-grained constraint, both from inherent constraint or domain knowledge [e.g., request for comments (RFC) documents]. Formers [20] infer constraints from software source code. Unfortunately, there are signs that various constraints remain in the source code [18], [20], [24], [25], after we analyze a large number of open-source software packages. Thus, it poses a great challenge to infer those constraints from source code. In this paper, on the basis of option’s relationship with the execution environment, we find that constraints are combined with syntactic aspect and semantic aspect. For example, option of type PORT is with the syntactic constraint that must be set to an integer between 0 and 65535. At the same time, it is not allowed to use the occupied port, which is a semantic constraint, because it needs the information not only from system itself but also the execution environment.

Aimed at generating misconfigurations by violating those syntactic rules, this paper uses type-specific predefined patterns to express option type’s commonly used standardization. Previous work like EnCore [18] used regular expression (RE) to express the option type’s string patterns. Similar but different to EnCore, our syntactic constraints can be expressed by augmented Backus–Naur forms (short for ABNF) [22], which is a popular Internet specifications in RFC. As shown in Table III, the pattern of ABNF is composed of several elements. Besides, ABNF uses common patterns such as RE to define such elements. Table III illustrates the details of our syntactic constraints as simplified ABNFs. We adopt this design mainly because ABNF provides a discrete pattern for each element; thus, we could define each element’s constraints to obtain fine-grained constraints of a certain option type. We use an example to show ABNF’s superiority over RE. Suppose there is an option “memory = 16M.” Testers may want to test misconfigurations like “−1 M” (bad number), “16 C” (bad unit). In ABNF, option “memory” can be described using elements: memory = number unit, while rule of “number” is “[0–9]+ ” and “unit” is “[KMG].” Thus, we can mutate each element to generate misconfigurations like bad number and bad unit. Besides, we can define the range for the element. For “number,” it can be from 0 to 64, for example, we can generate misconfiguration “65M” to violate this constraint. However, when we use RE “[0–9] + [KMG]” to describe the option’s syntax previously, these features are not supported.

We also improve the ABNF design by adding rules like valid range for the element in the digital form, which can be found in Table III. Our inference can use program analysis [24] for system-specific constraints such as extracting the value range of the type Count. Owing to space limitations, we show here simplified ABNF as descriptions of syntactic constraints. Elements “DIGIT,” “ALPHA,” “HEXDIG,” and “OCTET” are all from core rules of ABNF (RFC2234).

Different from syntactic constraints, in this paper, semantic constraints are demanded to describe the complicated relationship between systems and their execution environments. For this purpose, we divide the semantic constraint into two parts: constraints for the option’s value and the environment-related attributes for the option’s type. Table IV lists the constraints for the option’s value according to its type. We take an option of URL type, for example, to show these constraints cannot be well expressed by syntactic form. URL option’s value can be judged as correct or wrong, depending on whether it is consistent with its syntactic form. However, semantic constraints like whether it is accessible depends on system’s execution environment, such as the network states. The semantic constraints are also critical to help generate misconfigurations and deserve our high attention.

To reflect the requirements from the execution environment, we define the environment-related attributes for option’s type in this paper. As listed in Table V, these attributes are assigned to different option types, and they refine the requirements from execution environment, for example, when a URL option should satisfy the constraint “The URL should be reachable” in Table IV, we can use attribute “Is URL forbidden by firewall”

TABLE III
SYNTACTIC CONSTRAINTS

Option Type	Syntactic Constraints			
	ABNF pattern	Elements	Rule of Element	
File (Directory) Path	1* ⟨“/” DirString⟩ [“/”]	DirString	[\w.-]+	
Partial File (Directory) Path	DirString *⟨“/” DirString⟩ [“/”]	DirString	[\w.-]+	
Domain Name	[Scheme “://”] ServerName[“:”Port]	Scheme	(“telnet”/“https”/“http”/“ftp”/)	
		ServerName	(\w)+((\.\w+)+)	
		Port	See option type “Port”	
IP Address	Int 3⟨“.” Int⟩	Int	1*3DIGIT	[0-255]
Email	Sysadminname“@”ServerName	SysadminName	(\w)+(\.\w+)*	
		ServerName	See element “ServerName”	
Mode	Mode	Mode	(“value1”/“value2”/“value3”/)	
Boolean	Bool	Bool	(“on”/“off”/“yes”/“no”/“true”/“false”)	
Language	Lang	Lang	2ALPHA	
MIME-types	Type“/”SubType	Type	(“application”/“image”/“audio”/“text”/)	
		SubType	1*(HEXDIG/“-”)	
Memory	Number Unit	Number	1*DIGIT	[min-max]
		Unit	(“K”/“M”/“G”/“T”/“KB”/“GB”/“TB”/“B”)	
Time	Number Unit	Number	1*DIGIT	[min-max]
		Unit	(“s”/“min”/“h”/“d”/“ms”)	
Speed Rate	Number Unit	Number	1*DIGIT	[min-max]
		Unit	(“bps”/“Mbps”/“Kbps”)	
Count	Int	Int	1*DIGIT	[min-max]
Fraction	Fraction	Fraction	1*DIGIT“.” 1*DIGIT	[min-max]
Port	Int	Int	1*DIGIT	[0-65535]
Permission	Octet	Octet	1*OCTET	[0-777]
Sysadminname	SysadminName	SysadminName	[a-zA-Z][a-zA-Z0-9]*	
Password	Pwd	Pwd	N/A	
Filename	FileName	FileName	[\w -]+.[\w -]+	

TABLE IV
SEMANTIC CONSTRAINTS

Option Type	Semantic Constraints	Resources
Path	The path should be existent	File System
Path	The path should be readable/writable/excutable	File System
URL	The URL should be reachable	Network
IP address	The IP should be accessible	Network
Email	The email should not be existent	Network
Domain name	The domain name should not be existent	Network
Port	The port should not be occupied	Services
Language	The language should be existent	Services
MIME type	The MIME type should be existent	Services
Memory	The memory should be less than available memory	Hardware
Speed rate	The speed rate should be less than available bandwidth	Hardware
Sysadminname	The sysadminname should be from root group	Security

in Table V with an boolean value “No” to describe it. This paper uses environment information and domain knowledge to infer such semantic constraints. This paper is inspired by EnCore [18], and we classify those attributes into six main resource types (Network, Services, Hardware, File System, Security, and Environment Variable).

After inferring the configuration constraints from four mature software systems, we evaluate the effectiveness of our syntactic constraints by checking whether related documentation descriptions are consistent with constraints inferred. By manual inspection, Table VI shows that our syntactic constraints work well and are consistent with 91% of 1582 options from four popular software systems. Although EnCore [18] also uses predefined patterns to speculate option types, syntactic constraints in this paper are described by improved ABNF. As we take both the string pattern and the data range (e.g., the value set of Enumeration) into consideration, thus, our syntactic constraints appear to be more flexible and fine grained. SPEX [20] infers five main constraints (e.g., Basic type constraint, Data range constraint, etc.) from the source code but is not fine-grained enough in Semantic-type constraint. This paper might not include some other system-specific constraints. However, since we infer such constraints for misconfiguration generation instead of precise configuration constraints analyses, we consider the inaccuracy to be acceptable.

III. MISCONFIGURATION INJECTION

In this section, we answer two questions: How can we use these constraints to generate misconfigurations and how can we test system reactions with misconfigurations? To address these problems, we propose a tool called ConfVD to conduct misconfiguration injection and analyze system reactions to a variety of misconfigurations.

A. *Misconfigurations Generation*

To generate misconfigurations for injection, we propose misconfiguration generation methods based on the constraints. The whole misconfiguration generation process can be seen as three main parts: First, ConfVD parses configuration files into

TABLE V
ENVIRONMENT-RELATED ATTRIBUTES FOR OPTION TYPE

Option Type	Environment-related Attribute	Type	Description	Resources
Path	Owner	String	Owner of Path	File System
	Group	String	Group name of Path	
	Permission	Octal	Permission of Path	
	Contents	String	Contents of path	
	HasSymLink	Bool	If has a symbol link	
	Type	Enum	What's type of Path	
URL	Forbidden	Bool	Is URL forbidden by firewall	Network
	Type	Enum	What's type of URL	
IP address	Forbidden	Bool	Is IP forbidden	Network
	Type	Enum	What's type of IP	
Port	Sysadmin	String	What's the sysadmin of port	Services
	Type	Enum	What's type of port	
Language	IsSupported	Bool	Is Language supported by system	Services
MIME type	IsSupported	Bool	Is MIME type supported by system	Services
Memory	Allocation	Enum	Strategy of allocation	Hardware
Sysadminname	GroupName	String	Group name of Sysadminname	Security

TABLE VI
PROPORTION OF OPTIONS CONSISTENT WITH SYNTACTIC CONSTRAINTS

Software	Options	Satisfied
Httpd	564	466(82.6%)
MySQL	671	641(95.5%)
PostgreSQL	273	265(97.0%)
Yum	74	68(91.9%)
Total	1582	1440(91.0%)

TABLE VII
EXAMPLES OF HOW SYNTACTIC MISCONFIGURATIONS ARE GENERATED

Mutation Operations	Example: MemSize=64MB				
	Before		After		Result
	Number	Unit	Number	Unit	
Substitute	64	MB	64	Ma	64Ma
Add	64	MB	64	Mba	64Mba
Convertcase	64	MB	64	mb	64mb
Delete	64	MB	64	M	64M
Deyond range	64	MB	129	MB	129MB
Change digit type	64	MB	64.5	MB	64.5MB
Disorder	64	MB	Unit	Number	MB64
Cut out	64	MB	Number		64
Repeat	64	MB	Number	Unit Unit	64MBMB

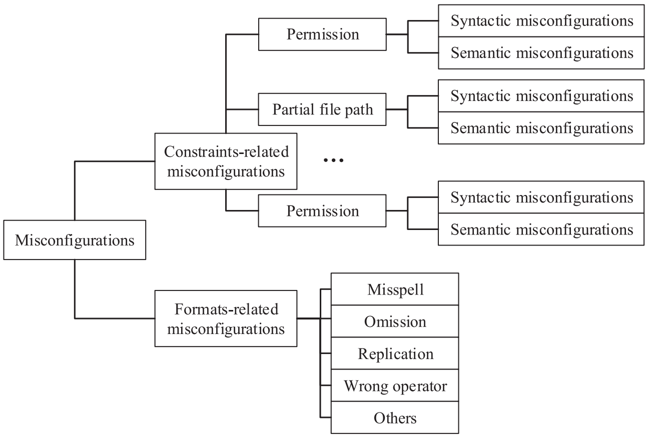

Misconfigurations
Constraints-related misconfigurations
Permission
Syntactic misconfigurations
Semantic misconfigurations
Partial file path
Syntactic misconfigurations
Semantic misconfigurations
...
Permission
Syntactic misconfigurations
Semantic misconfigurations
Formats-related misconfigurations
Misspell
Omission
Replication
Wrong operator
Others

Fig. 3. Misconfiguration generation methods.

structured data (e.g., XML files are used in ConfVD to store the configuration information such as key, value, etc.) to easily manipulate the configuration. Second, it modifies structured original data to generate misconfigurations using generation methods. Finally, these modified data are assembled again to generate new configuration files with misconfigurations. As illustrated in Fig. 3, the misconfiguration generation methods we use can be classified into two main categories: constraint-related misconfiguration and format-related misconfigurations.

1) Constraint-related misconfiguration: ConfVD can generate misconfigurations via violating syntactic constraints. First, after inferring options' constraints, we can obtain the ABNF for each option and each ABNF contains at least one element. Second, ConfVD takes mutation operations on such ABNFs to generate mutated elements. Third, mutated elements are aggregated with other unchanged ones to generate a new option value as a candidate syntactic misconfiguration. Finally, all the candidate syntactic misconfigurations will be checked with specific ABNFs to make sure they violate the syntactic constraints. Hence, a new syntactic misconfiguration will be generated.

Mutation operations have a variety of types divided into element-level and form-level operations. For element-level operations, we randomly substitute, add, convert the case, or delete a single character in each element of ABNF. Considering the element in digital form, we generate misconfigurations beyond its range and modify the digit type it belongs to (e.g., we use float number when it should be an integer). Form-level operations only consider ABNFs with multiple elements; in this case, we disorder, cut out, or repeat one of the elements from the ABNFs. The elements after these two operations are called mutated elements.

Table VII illustrates an example to explain how ConfVD generates syntactic misconfigurations. For the option "MemSize," its type is memory, and its ABNF consists of two elements: "Number" and "Unit." We demonstrate element-level mutation operations such as "substitute," "add," "convert case," and "delete" on the element "Unit" and operations "beyond range" and "change digit type" on the element "Number." In fact, each element in ABNF should be processed by these operations to generate mutated elements. After these element-level

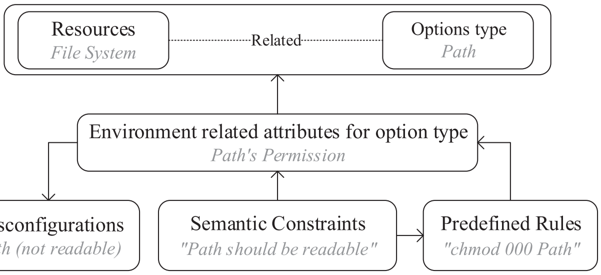


Fig. 4. Generation of semantic misconfigurations.

TABLE VIII
EXAMPLES OF FORMAT-RELATED MISCONFIGURATIONS

Format-related rules	Example	
	Before	After
Omission	EnableLog = On	= On
Misspelling	EnableLog = On	EnableLogs = On
Delete the existing value	EnableLog = On	EnableLog =
Change the case of text	EnableLog = On	enablelog = On
Wrong operator	EnableLog = On	EnableLog : On
Delete the operator	EnableLog = On	EnableLog On
Wrong section structure	[mysqld]	[mysqld
Wrong section name	[mysqld]	[mysqlf]

operations, form-level operations like "disorder," "cut out," and "repeat" are also shown in Table VII. To help understand how mutated elements generate a misconfiguration, we take the result of the "substitute" operation as an example: after the mutation operation, the value of the element "Unit" of "MB" is changed to "Ma," then we aggregate the elements "Number" and "Unit," so we get a candidate syntactic misconfiguration of option "MemSize" as "64Ma." Finally, this candidate syntactic misconfiguration will be checked to see whether it is consistent with the ABNF of "MemSize," and the answer is no, so a syntactic misconfiguration comes out. It also should be noted that, for "Others" type options, since they do not have any syntactic constraints, there still are two methods for them to generate misconfigurations. First, ConfVD can use generic alterations on option values, such as randomly changing the strings. Second, sysadmins can supplement a new option type to our type classification; at the same time, syntactic constraints can also be defined to help generate misconfigurations.

Generating misconfigurations with constraints defined by ABNF helps ConfVD to simulate the situations where syntactic errors occur, and it provides comprehensive and systematic misconfigurations for testing system reactions.

Similar to syntactic misconfigurations, ConfVD generates semantic misconfigurations using semantic constraints. To generate misconfigurations, resource-related predefined rules are designed to simulate the violation of semantic constraints, as shown in Fig. 4. Here we use Path type as an example, attribute "permission" is related to the semantic constraint "The file should be readable." To violate the semantic constraint like "The file should be readable," we predefine rules to modify these attributes. For example, we run command like "chmod 000 path" to change the attribute "permission." Since our purpose is to change the environment attributes to simulate semantic misconfigurations, ConfVD changes not only option value but also execution environment to maximally reproduce the scenarios when semantic misconfigurations occur.

The processing model of ConfVD for generating semantic misconfigurations is well customized and scalable. ConfVD already provides a collection of rules for generating common misconfigurations and sysadmins can add other custom rules.

2) Format-related misconfiguration: In format-related rules, considering the fact that configuration files need to meet the format requirements, ConfVD can generate misconfigurations as listed in Table VIII, by simulating sysadmins' common mistakes, such as omission or misspelling, while editing those complex configuration files.

B. Testing

使用软件自身的测试框架

The same misconfiguration may cause different system reactions owing to the different program states. To unify the testing process, ConfVD uses software's own test framework [26]–[29] to test injected misconfigurations. To simulate the situations when sysadmin meets the misconfigurations, ConfVD runs a sequence of test scripts such as "launching the server" or "creating a database." Before testing the injected misconfigurations, to eliminate the interference, ConfVD should confirm that system must be in the proper state by checking whether the system can pass all test cases.

When testing a misconfiugration, first, ConfVD replaces the original configuration file(s) with the one(s) containing errors, and then launches the system. If the system successfully starts up, ConfVD would run test scripts persistently until system failure happens or system passes all the test scripts. During the testing process, ConfVD records the system state by monitoring the information such as system logs or console output. Finally, ConfVD analyzes these information to evaluate the system's reaction ability of handling misconfigurations.

IV. ANALYSIS

In this section, we analyze and evaluate the system reaction ability from the results of the misconfiguration injection by ConfVD. Table IX illustrates the software systems and their information in our studies. Generating misconfigurations for all those options will lead to the computational complexity's explosive exponential growth. In order to avoid this issue, we applied a stratified random sampling on configuration options. In detail, according to the type classifications in Fig. 2, we divide options into several subgroups. For each group, options are randomly sampled with a fraction of the total population. As the number of options with different types vary greatly among systems, this sampling method will ensure that estimates can be made with equal accuracy in different option types of configuration, and that comparisons of samples with different types can be made with equal statistical power. As shown in Table IX, we finally sample 113 options and generate 1273 corresponding misconfigurations.

TABLE IX
SYSTEMS ANALYZED

Software	LoC	Options	Misconfigurations	Reactions (Duplicated ones removed)
Httpd	148K	29	328	78
MySQL	1.2M	26	318	113
PostgreSQL	757K	33	352	10
Yum	38K	25	275	15

As our purpose is to expose as many configuration vulnerabilities as possible, ConfVD generates massive and various misconfigurations. However, there can be redundancy in these misconfigurations. In other words, two different misconfigurations generated by the ConfVD may share the same software reaction. With this noise, analyzing the distribution of software reactions to all injected misconfigurations makes little sense. Therefore, our analysis of system reactions should be based on misconfigurations that do not include those duplicated ones. In our implementation, we eliminate those duplicated misconfigurations by checking whether the related log messages are the same. We first set the systems log verbosity option to the highest to get more details of the system reactions. After collecting the logs, we formalize them and remove the variables for comparison. In the end, we get clusters of similar software reactions. The results are also listed in Table IX.

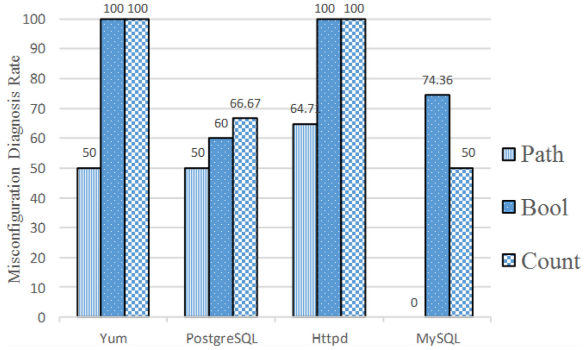


Fig. 5. Misconfiguration diagnosis rate of different systems with three kinds of misconfiguration.

A. System Reaction Ability Analysis

To evaluate the system reaction ability of handling misconfigurations, we classified the reactions into three types: good reaction, bad reaction, and no reaction. A good reaction is that, given a particular misconfiguration, the system can precisely locate the root causes of failures or anomalies using console information, log messages, etc. A bad reaction would be a system's failure to resolve the problems, only providing vague diagnostic messages, or even ending up with a silent failure. Finally, no reaction occurs when the system has no reaction at all to the misconfigurations due to the robust design or inadequate testing for triggering the latent configuration errors [30].

The overall results of the system reactions are listed in Table X. In order to help readers better understand different system reactions, taking a further step, we analyze the root causes of each reaction.

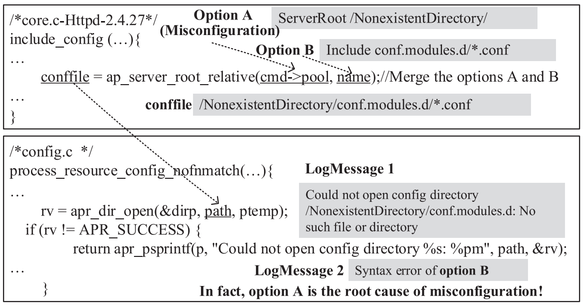
1) *Good reactions:* About 34.56% explicitly locate the misconfigurations by printing log messages that contain the line number or the name of misconfiguration. Most of these reactions happened when system checks the validity of the configuration, primarily during system startup.
2) *Bad reactions:* About 29.95% triggered the exceptions but failed to locate the misconfigurations, mainly because options were not checked, or the checking condition failed to capture the error. Even though they were detected on exception, these reactions may be obscure or mislead sysadmins in their diagnosis. For example, when Httpd was misconfigured incidentally by adding the option "Listen 80" twice in the configuration file, the logs after failure printed "Address already in use" and "Could not bind to address," which may confuse sysadmins. Even worse, bad reactions may obstruct the diagnosis. We found that bad reactions (e.g., crashes, hangs, and silent failures) were caused by improper exception handling or a lack of configuration checking.
3) *No reactions:* About 35.49% passed all the tests without raising any exception. There are two reasons: The first case is the robust design, which allows systems to legalize these misconfigurations. For instance, PostgreSQL allows both "key value" and "key = value" formats, which would avoid format-related misconfigurations. In the second case, there may be latent configuration errors (LC errors) [30]. Such options are not checked during initialization; hence, we used various test cases to expose as many errors as we could.

B. Misconfiguration Diagnosis Analysis

In this section, we analyze and evaluate different types of misconfigurations and their corresponding reactions. Fig. 5 shows that misconfigurations are from three widely used option types, i.e., Path, Boolean, and Count. We consider that a reaction with an explicit indication of misconfigurations represents a good reaction after system failure. Accordingly, we define "misconfiguration diagnosis rate" as the proportion of good reactions after failures caused by misconfigurations. Results show that MySQL fails to locate Path-related misconfigurations. When compared with Path, misconfigurations related to Boolean are much easier to locate. The diagnosis rate of Httpd and Yum reach 100%, while there is only 60% for PostgreSQL and 74.36% for MySQL. The main reason is for MySQL only reporting the exception captured in systems without any location information about misconfigurations. The diagnosis rates for Count are even higher than those of Boolean in Httpd (100%), Yum (100%), and PostgreSQL (66.67%). Similar to the reason for the Boolean

TABLE X
OVERALL RESULTS OF SYSTEM REACTIONS

Abbr.	Yum	Httpd	PostgreSQL	MySQL	Sum	Ratio
Good Reaction	7	26	5	37	75	34.56%
Bad Reaction	3	11	3	48	65	29.95%
No Reaction	5	41	3	28	77	35.49%
Sum	15	78	11	113	217	100%



```
/*core.c-Httpd-2.4.27*/
include_config (…){
…
      conffile = ap_server_root_relative(cmd->pool, name);//Merge the options A and B
…
}
```

Option A (Misconfiguration) ServerRoot /NonexistentDirectory/

Option B Include conf.modules.d/*.conf

conffile /NonexistentDirectory/conf.modules.d/*.conf

```
/*config.c  */
process_resource_config_nofnmatch(…){
…
      rv = apr_dir_open(&dirp, path, ptemp);
   if (rv != APR_SUCCESS) {
            return apr_psprintf(p, "Could not open config directory %s: %pm", path, &rv);
…
      }
```

LogMessage 1

```
Could not open config directory
/NonexistentDirectory/conf.modules.d: No
such file or directory
```

LogMessage 2 Syntax error of **option B**

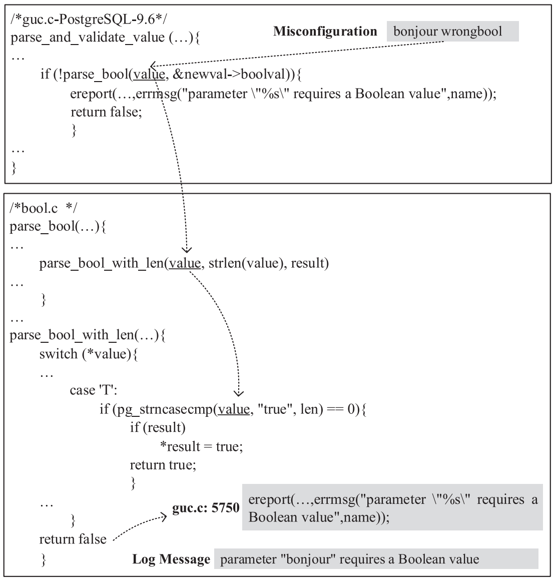
In fact, option A is the root cause of misconfiguration!

Fig. 6. Case study of Path misconfiguration.

misconfiguration, MySQL still has a low proportion (50%) in this respect.

Among these three types, Path misconfigurations are the hardest to be diagnosed by systems, and even experienced developers may fail to gracefully handle these misconfigurations. Fig. 6 shows a study case of Path misconfiguration that occurred in Httpd. We used ConfVD to inject misconfiguration into the option "ServerRoot" with a nonexistent directory. Httpd only checked the syntax, but not the semantic constraints (e.g., whether the directory is existent). Then, Httpd generated variable "conffile" by merging option "ServerRoot" and option "Include." When Httpd tried to open "conffile," it failed and printed log messages "Could not open...." It then indicated that there was a syntax error in option "Include." The bad reactions found in Httpd come from two main sources: 1) the options of "Path" type sometimes need concatenation with other paths. Thus, misconfigurations can propagate through data flow and have a large scope of impact. 2) Semantic constraints are harder to check. The popular practice involves trying to open the path and using the return code to judge whether the operation is successful, but this does not explain the root cause of the misconfiguration.

Types of misconfigurations with simple constraints (e.g., Boolean, Mode) have a high diagnosis rate, mainly because verifying the constraints of these types can be done easily. Fig. 7 illustrates how PostgreSQL parses values for Boolean options. For Boolean options, PostgreSQL uses the parse_bool function to check that each Boolean option is corrective. It in fact uses the "parse_bool_with_len" function to make a comparison between option value and legal values. As shown in Fig. 7, misconfiguration can be located and resolved easily. We surmise that this is because such constraints have no correlation with the environment. Therefore, it can be easily judged whether the value of an option is valid or not.



```
/*guc.c-PostgreSQL-9.6*/
parse_and_validate_value (…){
…
      if (!parse_bool(value, &newval->boolval)){
            ereport(…,errmsg("parameter \"%s\" requires a Boolean value",name));
            return false;
            }
…
}
```

Misconfiguration bonjour wrongbool

```
/*bool.c  */
parse_bool(…){
…
      parse_bool_with_len(value, strlen(value), result)
…
      }
…
parse_bool_with_len(…){
      switch (*value){
      …
            case 'T':
                  if (pg_strncasecmp(value, "true", len) == 0){
                        if (result)
                              *result = true;
                        return true;
                        }
      …
            }
      return false
      }
```

guc.c: 5750 ereport(…,errmsg("parameter \"%s\" requires a Boolean value",name));

Log Message parameter "bonjour" requires a Boolean value

Fig. 7. Case study of Bool misconfiguration.

Thus, using more simple constraint options in configuration is highly recommended for developers in order to reduce the potential misconfigurations. Furthermore, when faced with a failure, sysadmins require more reasons than symptoms of failures (e.g., exceptions). We recommend that developers point out the root causes instead of only recording what happened in the system and console log messages.

Thus, we highly recommend developers to use simple constraints to describe options, to prevent sysadmins introducing misconfigurations. Moreover, sysadmins require more reasons than symptoms of failures (e.g., exceptions) in the systems. To make systems more user-friendly, developers should try to point out the root causes of failures, instead of only recording what happened during the failures.

C. Capability Analysis

To demonstrate the need for the fine-grained constraints we have proposed herein, we have analyzed ConfVDs capabilities by answering two questions: How does ConfVD improve generic alteration approaches without constraints, and is there the necessity for fine-grained constraints in misconfiguration injection?

1) How ConfVD Improves Generic Alterations Approaches Without Constraints: A straightforward approach to evaluate the reaction ability of systems is testing with generic alterations. ConfErr [19] relies on using generic alterations to original

TABLE XI
CAPABILITY OF CONFVD AND CONFERR IN FINDING BAD SYSTEM REACTIONS

	# of undiagnosed misconfigurations				
	Httpd	MySQL	PostgreSQL	Yum	Total
ConfVD	11	48	3	3	65
ConfErr	4	19	0	2	25

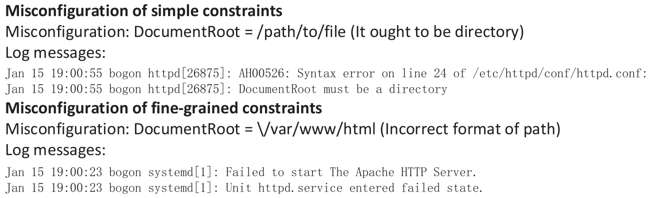
TABLE XII
COMPARISON OF CONFVD WITH A VARIANT THAT ONLY USES COARSE-GRAINED CONSTRAINTS

	# of undiagnosed misconfigurations				
	Httpd	MySQL	PostgreSQL	Yum	Total
ConfVD	11	48	3	3	65
Variant	9	32	3	3	47

configuration options (e.g., omissions, substitutions, and case alterations of characters). We chose ConfErr because ConfErr is not guided by constraints. In this section, we show the improvements of ConfVD compared with generic alterations approaches like ConfErr.

We evaluate the tools' capability of finding bad system reactions by counting the numbers of bad reactions they find. In Table XI, the number of the undiagnosed misconfigurations (i.e., misconfigurations of which systems fail to find the root causes of failures) found by the two tools is presented. These misconfigurations could possibly have been avoided if ConfVD or ConfErr had been used to evaluate the system reactions and harden the system against misconfigurations. Our results reveal that ConfVD finds more undiagnosed misconfigurations in all four systems than ConfErr, and undiagnosed misconfiguations found by ConfErr only account for 38.46% of the ones found by ConfVD. It should be noted that all the misconfigurations generated by ConfErr could have been found by ConfVD using options' constraints. The inefficiency in ConfErr is for the reason that widely used mature systems are able to detect the violation of configuration formats, but to diagnose constraint-related misconfigurations, it requires not only domain knowledge but also environmental information, which enables ConfVD to find the potential bad reactions of systems.

2) Is There a Necessity for Fine-Grained Constraints in Misconfiguration Injection: State-of-the-art techniques like SPEX [20] use coarse-grained constraints to help generate misconfigurations. However, those coarse-grained constraints (such as data type and value range) are insufficient for generating misconfigurations. In order to demonstrate the need for fine-grained constraints, we evaluated a variant of ConfVD by removing the fine-grained constraint-related misconfigurations. Like SPEX, this variant simply generates misconfigurations only using coarse-grained constraints (e.g., Basic type constraint, Data range constraint, etc.), these constraints are inferred from source code, and they do not concern the semantic-type-specific characteristics. We used our own implementation of the technique. We were unable to use SPEX because it assumes the availability of source code and needs annotations from domain-specific knowledge. The experimental results are shown in the "Variant with Simple Constraints" column of Table XII. The variant without fine-grained constraints only found 47 undiagnosed misconfigurations, a reduction of 27.69% from the 65 of ConfVD. By contrast, ConfVD uses fine-grained constraints to generate and inject these comprehensive misconfigurations into targeted systems to test their reactions.



Misconfiguration of simple constraints
Misconfiguration: DocumentRoot = /path/to/file (It ought to be directory)
Log messages:

```
Jan 15 19:00:55 bogon httpd[26875]: AH00526: Syntax error on line 24 of /etc/httpd/conf/httpd.conf:
Jan 15 19:00:55 bogon httpd[26875]: DocumentRoot must be a directory
```

Misconfiguration of fine-grained constraints
Misconfiguration: DocumentRoot = \/var/www/html (Incorrect format of path)
Log messages:

```
Jan 15 19:00:23 bogon systemd[1]: Failed to start The Apache HTTP Server.
Jan 15 19:00:23 bogon systemd[1]: Unit httpd.service entered failed state.
```

Fig. 8. Two examples of system reactions to misconfigurations.

As illustrated in Fig. 8, the top misconfiguration violates the simple constraint: the option type of "DocumentRoot" is a directory path. Httpd detected this misconfiguration. However, bad reactions occurred when faced with the bottom misconfiguration as it violated the fine-grained constraints. Httpd easily found the misconfiguration of simple constraints but failed to identify fine-grained ones. Our study confirms that a comprehensive test for systems against misconfigurations would improve the quality assurance.

ConfVD's fine-grained constraints were extremely useful for Httpd and MySQL but did not affect the results for PostgreSQL and Yum. We surmise that this is because although many misconfigurations are injected, fine-grained constraints were not necessary in the 72.3% of cases in which the misconfigurations of simple constraints could find bad system reactions. However, that may not always be enough. Therefore, it remains important to use fine-grained constraints to generate misconfigurations and find bad system reactions that might discourage sysadmins.

D. Validity Limitations

There are several major limitations in terms of the validity of our analyses.

1) Although the eight software systems we studied in Section II are both mature and large, our classification, which is on the basis of them, may not be representative of other systems. In order to verify the generalization of our option type classification and avoid the issue of overfitting, the configuration options we used for classification are taken from eight open-source software packages, while those used for validation in Section II-A are taken from other four open-source software packages.
2) Only a few sets of popular system options are considered in our studies. Some system options may not be considered in our studies, which may influence our results. Therefore, we choose those most frequently used options and believe that these common options are representative of the options in most situations.
3) All the analyses and evaluations are on the basis of manually checking the system and console logs, which may introduce errors. To ensure the correctness, we double-check all the results.
4) Our misconfiguration generation method may omit some kinds of misconfigurations. To reduce this limitation, we plan to analyze more sysadmin reports from software projects and improve our methods.

V. Experience and Practice

In this section, we provide some advice and best practices based on what we have observed from the systems evaluated for misconfiguration issues.

Avoiding inconsistency: There are lots of good habits in handling unit inconsistency. For instance, PostgreSQL always assigns numbers to options with units (e.g., option like "max_stack_depth = 100 kB"). To the opposite, a lack of necessary explanations or ambiguous descriptions in configuration may raise sysadmins' confusion and trap them into making misconfigurations.

Making configuration simple and easy: Sysadmin-friendly configurations are also meaningful for reducing the occurrences of misconfigurations. Our study finds many ambiguous descriptions of configuration options in sysadmin guidebooks. Xu *et al.* [1] reveal a lot of useful findings, which leads to a few guidelines for simplifying configuration. They also study configuration navigation as an intermediate solution to help sysadmins understand configuration.

Early checking: During our analysis and evaluation, we conduct a study on the source code of these systems under evaluation, and find that PostgreSQL manages the system configuration using a consistent interface. For example, it uses function "parse_and_validate_value" to check each value of the option right after the system startup and reports any misconfigurations it found. Early checking helps sysadmins effectively detect misconfigurations during system's startup.

Friendly comment: Through those comments in configuration files, PostgreSQL explicitly informs sysadmins of usage of specified options (e.g., "#1s-600s," "#defaults to "localhost"; use "*" for all"). Such comments can also be found in other systems, but there is still a large number of software systems lacking in friendly comment. Adequate comments could guide sysadmins to configure options more efficiently and correctly.

VI. Related Work

Recently, aimed at solving misconfiguration problems, researchers have focused on detecting [18], [31] and troubleshooting [12], [14], [15], [32]–[34] misconfigurations. Although it is helpful to understand the misconfigurations and improve the reliability of systems, only a few efforts [19], [20] have focused on evaluating system reaction ability of handling misconfigurations.

Misconfiguration testing: Aimed at improving software reliability, we can analyze and evaluate systems reactions to misconfigurations. ConfErr [19] is a pioneer in misconfiguration testing. It relies a Generic Error Modeling System framework to produce human errors (e.g., typo, copy–paste mistakes, etc.) in configurations. However, ConfErr can only generate deficient misconfigurations without constraints, which hinders the evaluation of system reactions. Another work, SPEX [20], pushes the boundary of the ConfErr, as it infers constraints from the source code to identify misconfiguration vulnerabilities by injecting misconfigurations that violate those constraints. However, its coarse-grained constraints of option types result in poor diversity in the injected misconfigurations. On the basis of fine-grained constraints of option type classification, ConfVD generates comprehensive misconfigurations, which means we could effectively analyze and evaluate the system reaction ability of handling misconfigurations.

Detecting and troubleshooting misconfigurations: Misconfiguration detection [18], [31] refers to checking the potential configuration errors before the misconfigurations manifest, while misconfiguration troubleshooting [12], [14], [15], [32]–[34] is carried out in response to an already happened misconfiguration. EnCore [18] infers the configuration rules between systems and the executing environments to detect misconfigurations using a machine-learning method. Aimed at automatically diagnosing the root causes of performance problems, X-ray [12] proposes a technique for troubleshooting the potential misconfigurations. At the same time, the major contribution of this paper is proposing fine-grained constraints from option types classification. These option type classification based constraints can guide sysadmins and researchers to detect potential misconfigurations. Meanwhile, misconfiguration troubleshooting can also benefit from our work, since we have a comprehensive analysis and evaluation of system reactions.

System reactions study: There are also some other research works on system reactions to misconfigurations. ConfDiagDetector [21] focuses on detecting inadequate diagnostic messages for misconfigurations. Based on 546 real-world misconfigurations, Yin *et al.* [6] conducted a comprehensive empirical misconfiguration characteristic study. Xu *et al.* [35] studied on access-denied messages, showing that many of today's software systems miss the opportunities for providing adequate feedback information, imposing unnecessary obstacles to correct resolutions. This paper also benefits from these works' insights on system reaction's characteristics.

Configuration characteristic study: The former has drawn lots of efforts on characteristics on configuration that provides us with fundamental thesis. Rabkin and Katz [23] propose a classification of configuration option types in several Java applications for automatically extracting the related source code. Our classification differs from Rabkin's work mainly in the objectives: their main purpose is to extract those options from source code, while we intended to use these constraints for misconfiguration injection. Thus, this paper requires a more comprehensive classification to infer as fine-grained constraints as possible. SPEX [20] and EnCore [18] have a further research on configuration constraints using program analysis and machine learning. Zhou *et al.* [24], [25] focus on the characteristics of option constraints by investigating source code.

VII. Conclusion

As misconfiguration has become one of the most serious issues, to analyze and evaluate the system reaction ability of handling misconfigurations, in this paper, we propose a method to infer type-related constraints for misconfiguration injection. We study eight mature systems to summarize a comprehensive classification of option types. On the basis of this classification, we use ABNF to extract fine-grained constraints of each type. To generate comprehensive misconfigurations in the test systems, we propose misconfiguration generation methods for our constraints. We implement a tool named ConfVD to conduct misconfiguration injection and further analyze and evaluate the system reaction ability of handling various misconfigurations. Our analysis result shows that our option classification covers 96% of 1582 options from Httpd, Yum, PostgreSQL, and MySQL. Our constraints are more fine grained than SPEX and Encore, and the consistency was found to be 91% through manual verification. Our technique could improve generic alterations approaches without constraints, and we found that ConfVD could find nearly three times the bad reactions found by ConfErr. In total, we found 65 bad reactions from the systems under test and our fine-grained constraints contributed 27.7% more bad reactions than techniques only using coarse-grained constraints.

References

[1] T. Xu, L. Jin, X. Fan, Y. Zhou, S. Pasupathy, and R. Talwadker, "Hey, you have given me too many knobs!: Understanding and dealing with over-designed configuration in system software," in *Proc. Joint Meet. Found. Softw. Eng.*, 2015, pp. 307–319.

[2] J. Gray, "Why do computers stop and what can be done about them," Tandom Comput., Cupertino, CA, USA, Tech. Rep. TR-85.7, vol. 30, no. 4, pp. 88–94, 1985.

[3] K. Nagaraja, F. Oliveira, R. Bianchini, R. P. Martin, and T. D. Nguyen, "Understanding and dealing with operator mistakes in internet services," in *Proc. Conf. Symp. Oper. Syst. Des. Implementation*, 2004, pp. 61–76.

[4] D. Oppenheimer, A. Ganapathi, and D. A. Patterson, "Why do internet services fail, and what can be done about it?" in *Proc. 4th Usenix Symp. Internet Technol. Syst.*, 2003, pp. 1–1.

[5] L. Barroso, J. Clidaras, and U. Hoelzle, "The datacenter as a computer: An introduction to the design of warehouse-scale machines," *Synthesis Lectures Comput. Architecture*, vol. 8, no. 3, p. 154, 2009.

[6] Z. Yin, X. Ma, J. Zheng, Y. Zhou, L. N. Bairavasundaram, and S. Pasupathy, "An empirical study on configuration errors in commercial and open source systems," in *Proc. ACM Symp. Oper. Syst. Princ.*, Cascais, Portugal, Oct. 2011, pp. 159–172.

[7] Y. Sverdlik, "Microsoft: Misconfigured network device led to azure outage," 2012. [Online]. Available: http://www.datacenterdynamics.com/focus/archive/2012/07/microsoft-misconfigured-network-device-led-azure-outage. Accessed on: Jan. 29, 2017.

[8] A. Team, "Summary of the amazon EC2 and amazon RDS service disruption in the US east region," 2011. [Online]. Available: http://aws.amazon.com/message/65648. Accessed on: Jan. 29, 2011.

[9] J. Robert, "More details on today's outage," 2010. [Online]. Available: https://www.facebook.com/notes/facebook-engineering/more-details-on-todays-outage/431441338919," Accessed on: Jan. 29, 2017.

[10] G. Fitzgerald, "Grand research challenges in information systems," *Computing*, vol. 23, no. 3, pp. 337–344, 2003.

[11] T. Xu and Y. Zhou, "Systems approaches to tackling configuration errors: A survey," *ACM Comput. Surv.*, vol. 47, no. 4, pp. 1–41, 2015.

[12] M. Attariyan, M. Chow, and J. Flinn, "X-ray: Automating root-cause diagnosis of performance anomalies in production software," in *Proc. Usenix Conf. Oper. Syst. Des. Implementation*, 2012, pp. 307–320.

[13] M. Attariyan and J. Flinn, "Automating configuration troubleshooting with dynamic information flow analysis," in *Proc. Usenix Conf. Oper. Syst. Des. Implementation*, 2010, pp. 1–11.

[14] A. Rabkin and R. Katz, "Precomputing possible configuration error diagnoses," in *Proc. IEEE/ACM Int. Conf. Autom. Softw. Eng.*, 2011, pp. 193–202.

[15] S. Zhang and M. D. Ernst, "Automated diagnosis of software configuration errors," in *Proc. Int. Conf. Softw. Eng.*, 2013, pp. 312–321.

[16] H. J. Wang, J. C. Platt, Y. Chen, R. Zhang, and Y. M. Wang, "Automatic misconfiguration troubleshooting with peer pressure," in *Proc. Conf. Symp. Oper. Syst. Des. Implementation*, 2004, pp. 17–17.

[17] Y. M. Wang *et al.*, "Strider: A blackbox, state-based approach to change and configuration management and support," in *Proc. Usenix Conf. Syst. Admin.*, 2003, pp. 159–172.

[18] J. Zhang *et al.*, "Encore: Exploiting system environment and correlation information for misconfiguration detection," in *Proc. Int. Conf. Archit. Support Program. Lang. Oper. Syst.*, 2014, pp. 687–700.

[19] L. Keller, P. Upadhyaya, and G. Candea, "ConfErr: A tool for assessing resilience to human configuration errors," in *Proc. IEEE Int. Conf. Dependable Syst. Netw. FTCS DCC*, 2008, pp. 157–166.

[20] T. Xu *et al.*, "Do not blame users for misconfigurations," in *Proc. 24th ACM Symp. Oper. Syst. Princ.*, 2013, pp. 244–259.

[21] S. Zhang and M. D. Ernst, "Proactive detection of inadequate diagnostic messages for software configuration errors," in *Proc. Int. Symp. Softw. Test. Anal.*, 2015, pp. 12–23.

[22] E. D. Crocker, "Augmented BNF for syntax specifications: ABNF," 1997. [Online]. Available: https://tools.ietf.org/html/rfc2234," Accessed on: May 2, 2018.

[23] A. Rabkin and R. Katz, "Static extraction of program configuration options," in *Proc. Int. Conf. Softw. Eng.*, Waikiki, Honolulu, HI, USA, May 2011, pp. 131–140.

[24] S. Zhou, X. Liu, S. Li, W. Dong, X. Liao, and Y. Xiong, "Confmapper: Automated variable finding for configuration items in source code," in *Proc. IEEE Int. Conf. Softw. Qual. Rel. Secur. Companion*, 2016, pp. 228–235.

[25] S. Zhou *et al.*, "Easier said than done: Diagnosing misconfiguration via configuration constraints analysis: A study of the variance of configuration constraints in source code," in *Proc. Int. Conf. Eval. Assess. Softw. Eng.*, 2017, pp. 196–201.

[26] PostgreSQL, "PostgreSQL 9.6.1 documentation," 2017. [Online]. Available: https://www.postgresql.org/docs/9.6/static/pgbench.html.

[27] Apache, "Apache http test project," 2017. [Online]. Available: http://httpd.apache.org/test/," Accessed on: Jan. 29, 2017.

[28] MySQL, "The mysql test framework," 2017. [Online]. Available: https://dev.mysql.com/doc/mysqltest/2.0/en/," Accessed on: Jan. 29, 2017.

[29] Yum, "Qa:testcase yum basics," 2017. [Online]. Available: http://fedoraproject.org/wiki/qa:testcase_yum_basics," Accessed on: Jan. 29, 2017.

[30] T. Xu *et al.*, "Early detection of configuration errors to reduce failure damage," in *Proc. Usenix Conf. Oper. Syst. Des. Implementation*, 2016, pp. 619–634.

[31] V. Ramachandran, M. Gupta, M. Sethi, and S. R. Chowdhury, "Determining configuration parameter dependencies via analysis of configuration data from multi-tiered enterprise applications," in *Proc. Int. Conf. Autonomic Comput.*, Jun. 15–19, 2009, Barcelona, Spain, pp. 169–178.

[32] B. Aggarwal, R. Bhagwan, T. Das, S. Eswaran, V. N. Padmanabhan, and G. M. Voelker, "Netprints: Diagnosing home network misconfigurations using shared knowledge," in *Proc. 6th USENIX Symp. Netw. Syst. Des. Implementation*, 2009, pp. 349–364.

[33] M. Attariyan and J. Flinn, "Using causality to diagnose configuration bugs," in *Proc. Usenix Annu. Tech. Conf.*, Boston, MA, USA, 2008, pp. 281–286.

[34] J. Mickens, M. Szummer, and D. Narayanan, "Snitch: Interactive decision trees for troubleshooting misconfigurations," in *Proc. 2nd Usenix Workshop Tackling Comput. Syst. Probl. Mach. Learn. Techn.*, 2007, Art. no. 8.

[35] T. Xu, M. N. Han, L. Lu, and Y. Zhou, "How do system administrators resolve access-denied issues in the real world?" in *Proc. CHI Conf. Human Factors Comput. Syst.*, 2017, pp. 348–361.

[36] W. Li, S. Li, X. Liao, X. Xu, S. Zhou, and Z. Jia, "Conftest: Generating comprehensive misconfiguration for system reaction ability evaluation," in *Proc. 21st Int. Conf. Eval. Assess. Softw. Eng.*, 2017, pp. 88–97.

Shanshan Li is an Associate Professor with the Department of Computer Science, National University of Defense Technology, Changsha, China. She has published more than 50 papers. Her main research interests include empirical software engineering, with a particular interest in software quality enhancement, defect prediction, and misconfiguration diagnosis.

Prof. Li is a member of ACM. She is a recipient of several awards including the Distinguished Paper Award in Saner 2018, Spotlight paper in *Transactions on Parallel and Distributed System*, etc.

Wang Li received the B.S. and M.S. degrees from the School of Computer Science, National University of Defense Technology, Changsha, China, in 2015 and 2017, respectively, where he is currently working toward the Ph.D. degree.

His main research interests include software engineering, software reliability, software configuration, and so on.

Xiangke Liao (M'15) received the B.S. degree from the Department of Computer Science and Technology, Tsinghua University, Beijing, China, in 1985, and the M.S. degree from National University of Defense Technology, Changsha, China, in 1988.

He is currently a Full Professor and the Dean of the School of Computer, National University of Defense Technology. His research interests include parallel and distributed computing, high-performance computer systems, operating systems, software reliability, cloud computing, and networked embedded systems.

Prof. Liao is a member of the ACM.

Shaoliang Peng is the Executive Director with the National Supercomputing Center in Changsha, Changsha, China, and is an Adjunct Professor of BGI, and National University of Defense Technology, Changsha, China. He was a Visiting Scholar with CS Department, City University of Hong Kong from 2007 to 2008 and at BGI Hong Kong from 2013 to 2014. His research interests include high-performance computing, bioinformatics, big data, virtual screening, and biology simulation. He has published dozens of academic papers on several internationally influential journals, including *Science*, *Nature Communications*, *Cell AJHG, Genome Biology, Cancer Research*, ACM/IEEE TRANSACTIONS, *BIBM*, and so on.

Shulin Zhou received the B.S. and M.S. degrees from the School of Computer Science, National University of Defense Technology, Changsha, China, in 2014 and 2016, respectively, where he is currently working toward the Ph.D. degree.

His main research interests include software engineering, software reliability, operating system, and so on.

Zhouyang Jia received the B.S. and M.S. degrees from the School of Computer Science, National University of Defense Technology, Changsha, China, in 2013 and 2015, respectively, where he is currently working toward the Ph.D. degree.

His main research interests include software engineering, software reliability, operating system, and so on.

Mr. Jia is a student member of the ACM.

Teng Wang received the B.Eng. degree in software engineering from National University of Defense Technology, Changsha, China, in 2017, where he is currently working toward the Master's degree at the College of Computer Science.

His research interests include program analysis, software quality, and software security.