Vulnerability Coverage for Adequacy Security Testing

Shuvalaxmi Dass
Texas Tech University
Lubbock, Texas, USA
shuva93.dass@ttu.edu

Akbar Siami Namin
Texas Tech University
Lubbock, Texas, USA
akbar.namin@ttu.edu

ABSTRACT

Mainstream software applications and tools are the configurable platforms with an enormous number of parameters along with their values. Certain settings and possible interactions between these parameters may harden (or soften) the security and robustness of these applications against some known vulnerabilities. However, the large number of vulnerabilities reported and associated with these tools make the exhaustive testing of these tools infeasible against these vulnerabilities infeasible. As an instance of general software testing problem, the research question to address is whether the system under test is robust and secure against these vulnerabilities. This paper introduces the idea of "*vulnerability coverage*," a concept to adequately test a given application for a certain classes of vulnerabilities, as reported by the National Vulnerability Database (NVD). The deriving idea is to utilize the Common Vulnerability Scoring System (CVSS) as a means to measure the fitness of test inputs generated by evolutionary algorithms and then through pattern matching identify vulnerabilities that match the generated vulnerability vectors and then test the system under test for those identified vulnerabilities. We report the performance of two evolutionary algorithms (i.e., Genetic Algorithms and Particle Swarm Optimization) in generating the vulnerability pattern vectors.

CCS CONCEPTS

• **Security and privacy** → *Software security engineering*; • **Software and its engineering** → Software configuration management and version control systems;

KEYWORDS

Software Vulnerability Testing, Vulnerability Coverage, Genetic Algorithms (GA), Particle Swarm Optimization (PSO)

ACM Reference Format:
Shuvalaxmi Dass and Akbar Siami Namin. 2020. Vulnerability Coverage for Adequacy Security Testing. In *The 35th ACM/SIGAPP Symposium on Applied Computing (SAC '20), March 30-April 3, 2020, Brno, Czech Republic.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3341105.3374099

1 INTRODUCTION

Software systems and applications are often released with a great number of features and settings. These features and configurations serve their users and the underlying platforms for different purposes such as architectural settings, virtualization, performance, security and access control, privacy, and system level interactions. For instance, MySQL Version 5.5 lists more than 600 configuration parameters categorized into 3 groups namely Server Options, System Variables, and Status Variable References [2]. While these parameters offer great features to their administrators for setting up software systems properly, an improper configuration and setting of such parameters also create loopholes in the systems and thus are vulnerable to certain known or even unknown security attacks (i.e., zero-day vulnerability [4]).

According to the National Vulnerability Database (NVD) [3], as of September 2019, there are 1, 644 records of reported vulnerabilities with assigned CVE numbers. Some of these vulnerabilities are directly the cause of improper settings of the configurations parameters offered as features by the software systems. From the software testing perspective, enumerating all configuration settings and then verifying whether the given software is vulnerable to certain attacks is infeasible.

This paper introduces the concept of "*vulnerability coverage*" as an adequacy criterion for choosing instances of vulnerabilities that the software under test needs to be checked against. The deriving idea is to utilize the Common Vulnerability Scoring System (CVSS) as a means to measure the vulnerability level of the software under test. For instance, a vulnerability with CVE-2019-16383 reported for MySQL has the severity rated as 8.2 out of 10 and it is labeled as high. In this paper, we explore vulnerability coverage, as an adequacy criterion for choosing the vulnerabilities needed to be examined for the software under test. The key contributions of this work are as the following:

(1) Introduce the novel idea of vulnerability coverage to be utilized as an adequacy criterion for testing systems and their configurations thoroughly.
(2) The evolutionary algorithms (i.e., genetic algorithms (GA) and particle swarm optimisations (PSO)) are adapted to generate adequate test inputs with respect to the introduced vulnerability coverage criterion.
(3) The performance of the proposed vulnerability coverage criterion is reported through a case study demonstrating the feasibility of the proposed coverage criterion.

We will be using the words *Vulnerability Vector/Pattern*, *CVSS vector*, *configurations*, interchangeably. This paper is structured as follows: Section 2 reviews the related works. The concept of Common Vulnerability Scoring System (CVSS) is presented in Section 3. The methodology of adapting the evolutionary algorithms is presented in Section 4. The evaluation of the proposed idea performed on a case study is reported in Section 5. Section 6 concludes the paper.

SAC '20, March 30-April 3, 2020, Brno, Czech Republic
© 2020 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-6866-7/20/03.
https://doi.org/10.1145/3341105.3374099

2 RELATED WORK

The work presented in this paper offers some solutions for generating a set of thorough secure configurations for a given system when implementing Moving Target Defense (MTD) strategies [13].

Crouse and Fulp [8] used conventional Genetic Algorithms to implement a Moving Target Defense (MTD) environment to enable security through temporal and spatial diversity in computer configuration parameters.Crouse and Fulp reported that the pool of configurations becomes stale when there are no changes introduced to the set of configurations over a period of time. As a result, GA deals with a limited set of configurations.In this approach, the fitness (security) of aging configurations is reduced by a value (i.e., decay value) based on the time since they were last active. Such weak configurations are eventually replaced by more secure ones.

Lucas et al. [10] described a framework for implementing MTD at host-level. Their framework uses evolutionary-inspired Genetic Algorithm (GA) to generate secure configurations. They evaluated their framework using two qualitative measurements: Fitness score and pairwise Hamming distance (i.e., diversity).

The use of genetic algorithms in generating a thorough set of test inputs has been discussed and modeled in literature. For instance, Andrews et al. [5] used genetic algorithms to enable random testing more effective. A similar approach is adapted here to produce a better test inputs for the purpose of maximizing the coverage level of test pool using the evolutionary algorithms.

3 COMMON VULNERABILITY SCORING

The Common Vulnerability Scoring System (CVSS) provides a way to capture the principal characteristics of a vulnerability and produce a numerical score reflecting its severity. The scoring system also provides a textual representation of the semantic of the calculated score. The numerical score can then be translated into a qualitative representation (e.g., low, medium, high, and critical) to help organizations to properly assess and prioritize their vulnerability management processes [1].

CVSS is composed of three main metric groups: (1) Base, (2) Temporal, and (3) Environmental, each consisting of a set of sub-metrics. Without loss of generality and to demonstrate the feasibility of the proposed approach, the GA and PSO algorithms are only applied to the Base metric group. Additional reason that this paper focuses on the Base metric is due to the fact that the Base metric quantifies the essential characteristics of a vulnerability, which remains unchanged across different environments and over time. The Base metric consists of two sub-main metrics:

(1) Exploitability Metrics. It describes the "how" part of the attack that is being captured, which depends on the characteristics of the vulnerable components. This metric consists of:

- *Attack Vector (AV).* It reflects the proximity of the attacker to attack the vulnerable component. The more the proximity required to attack the component, the harder it is for the attacker. The attack vector takes on four values: Network (N), Adjacent (A), Local (L) and Physical (P).
- *Attack Complexity (AC).* This metric reflects the resources and conditions that are required to conduct the exploit on the vulnerable component. The more the number of conditions to be met, the higher the degree of complexity of attack is. It takes on two values: Low (L), and High (H).
- *Privileges Required (PR).* This metric represents the level of privileges required by an attacker to successfully launch an exploit. The lesser the level is, the easier the attack is. It takes on three values: None (N), Low (L), and High (H).
- *User Interaction (UI).* It reflects whether the participation of the user is required for launching a successful attack. The attack becomes difficult if the user interaction is mandatory. This metric takes on two values: None (N) and Required (R).

(2) Impact Metrics. These metrics reflect the characteristics of the impacted components. They consist of:

- *Availability Impact.* It measures the severity of the attack on the availability of the impacted component. The metric takes on these values: None (N), Low (L), and High (H).
- *Integrity Impact.* It measures the severity of the attack on the integrity of the impacted component. The metric takes on three values: None (N), Low (L), and High (H).
- *Confidentiality Impact.* It measures the severity of the attack on the confidentiality of the impacted component. It takes on the following values: None (N), Low (L), and High (H).

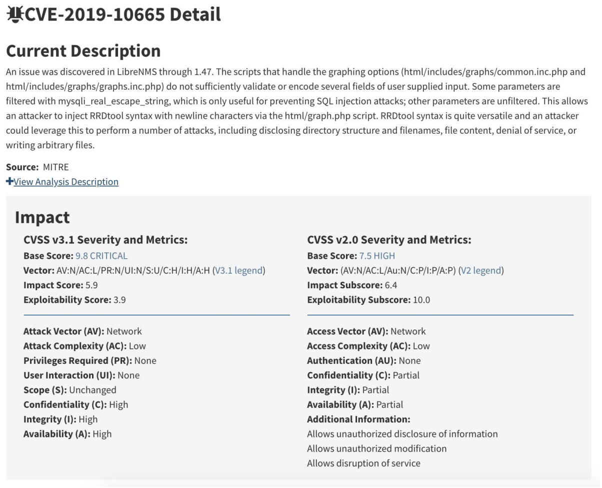
(3) Scope Metrics. There is also a vector called *"scope"* which describes the scope of the attack (i.e., whether the attack on the vulnerable component consequently impacted the resources beyond its means). It takes on two values: Unchanged (U) and Changed (C).

CVSS incorporates all of the aforementioned metrics in a formula to calculate the vulnerability score. The lower the Base score is, the harder it is for the execution of an exploit on the vulnerable component. Furthermore, the score is measured using a certain number of features. For instance, Figure 1 shows the CVSS score along with the vulnerability vector for CVE-2019-10665 reported for MySQL. As the figure shows, the severity of this vulnerability is rated as 9.8 out of 10 and it is labeled as a critical one. The generated format of the vulnerability/CVSS vector is `[AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H]` where [1] :

- `AV:N` indicates that the Attack Vector (AV) of such vulnerability is set at the Network (N) level.
- `AC:L` implies that the Attack Complexity (AC) of this vulnerability is Low (L).
- `PR:N` shows that the Privileges Required (PR) for launching an attack based on this vulnerability is None (N).
- `UI:N` reports that the User Interaction (UI) and involvement in enabling launching a successful attack is None (n).
- `S:U` shows the Scope (S) of the attack is Unchanged (U).
- `C:H` indicates that the risk of losing the Confidentiality (C) of data when this attack occurs is High (H).
- `I:H` implies that the risk of losing the Integrity (I) of data when this attack occurs is High (H).
- `A:H` indicates that the risk of losing the Availability (C) of data when this attack occurs is also High (H).

4 METHODOLOGY

We compared two different evolutionary algorithms for the vulnerability pattern generation targeting a certain level of CVSS score, as fitness function. The optimization algorithms that we implemented



CVE-2019-10665 Detail

Current Description

An issue was discovered in LibreNMS through 1.47. The scripts that handle the graphing options (html/includes/graphs/common.inc.php and html/includes/graphs/graphs.inc.php) do not sufficiently validate or encode several fields of user supplied input. Some parameters are filtered with mysqli_real_escape_string, which is only useful for preventing SQL injection attacks; other parameters are unfiltered. This allows an attacker to inject RRDtool syntax with newline characters via the html/graph.php script. RRDtool syntax is quite versatile and an attacker could leverage this to perform a number of attacks, including disclosing directory structure and filenames, file content, denial of service, or writing arbitrary files.

Source: MITRE
+View Analysis Description

Impact

CVSS v3.1 Severity and Metrics:
Base Score: 9.8 CRITICAL
Vector: AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H (V3.1 legend)
Impact Score: 5.9
Exploitability Score: 3.9

Attack Vector (AV): Network
Attack Complexity (AC): Low
Privileges Required (PR): None
User Interaction (UI): None
Scope (S): Unchanged
Confidentiality (C): High
Integrity (I): High
Availability (A): High

CVSS v2.0 Severity and Metrics:
Base Score: 7.5 HIGH
Vector: (AV:N/AC:L/Au:N/C:P/I:P/A:P) (V2 legend)
Impact Subscore: 6.4
Exploitability Subscore: 10.0

Access Vector (AV): Network
Access Complexity (AC): Low
Authentication (AU): None
Confidentiality (C): Partial
Integrity (I): Partial
Availability (A): Partial
Additional Information:
Allows unauthorized disclosure of information
Allows unauthorized modification
Allows disruption of service

Figure 1: The description of CVE-2019-10665 for MySQL.

were: 1) Genetic and evolutionary Algorithms (GA), and 2) Particle Swarm Optimization (PSO).

We chose the CVSS score as a measure of fitness function because the diversity of its parameters enable us to adapt typical optimization and search techniques such as evolutionary algorithms for addressing this problem. Moreover, some other researchers have also adapted these types of greedy algorithms to address similar problems in test input generations in the context of random testing (e.g., [5]). Each parameter in the configuration is assigned a score vector that represents the vulnerability it contains along with its severity. The score is modeled after the Common Vulnerability Scoring System (CVSS) vector and provides a method for measuring the security level of an individual configuration parameter setting. The vector can serve as a foundation to estimate the number of possible vulnerabilities of a certain configuration. Moreover, to account for the diversity in the configuration generation, we also calculated the Hamming distance, which measures how different one pattern (i.e., configuration) is from another.

4.1 Genetic Algorithm

We developed a python-based genetic algorithm script to generate a CVSS vector string pool with the best fitness score (i.e., = 2.0). We call it a "*string*" since each vector is treated as a string in our implementation. We set the number of iterations = 50 and population size = 100. The algorithm mainly comprises of five parts:

(1) *Configuration Generation*: An initial pool of 100 possible CVSS vector strings was generated randomly.
(2) *Fitness Score*: The Python library, called `CVSS3`, implements the Base metric score it was utilized to calculate the CVSS scores. The vector strings were assigned their CVSS scores to be their fitness scores if the former lied between 2.0 and 5.5, otherwise, they were assigned 100.
(3) *Breeder's Selection*: This type of selection not only chooses the best solutions (i.e., vectors with lower score) of the previous generation but also selects some random ones to avoid converging soon to a local minima.
(4) *Crossover*: For crossover, we took the simplest way of randomly switching the values of corresponding metrics among the two parent vector strings.
(5) *Mutation*: We performed mutation on the CVSS vector configurations by randomly picking one vector field and changing its value by randomly selecting from permissible values.

4.2 Particle Swarm Optimization Algorithm

We also implemented the PSO algorithm based on the CVSS scores for the purpose of comparing the performance of PSO with GA on generating a set of secure configurations. In order to enable the comparison unbiased, we kept the best score, number of iterations and the size of population similar to the ones set for GA.

The PSO implementation is similar to GA. However, unlike GA, PSO is easier to implement as it has no evolution operators such as crossover and mutation. In PSO, the potential solutions, are called particles. The algorithm mainly deals with two parameters: (1) `pbest_fitness` and (2) `particle_vel`, which contain the initial `pbest` fitness and velocity values for every particle in the swarm, respectively. The velocity for each particle measures how far is its fitness score (`pbest`) from the best score.

The PSO algorithm returns a count of particles with scores = 2.0 or velocity = 0 in every iteration. We set the target global best value as 10.0, `particle_velocity` (i.e., Hamming Distance) between the range $[0, 8]$ as 0 and 8 are the minimum and maximum number of differences that might exist between two particles, respectively. The `fitness_range` is set in the range of $[2, 10]$ where 2.0 is deemed as the best fit and 10.0 is considered to be poorly unfit.

The *Particle Initialization* follows a similar strategy as described for the configuration generation in GA. The algorithm continues to find the best fitness value and velocity for every particle until it reaches a threshold limit. In each iteration, the algorithm assigns the `pbest_fitness` (particle best) values as their CVSS scores `cvss_fit` only when the fitness is better (i.e., in our case, lesser the better) than its current `pbest` fitness value. It then assigns the `gbest` (global best) value of the swarm to be the best `pbest` value obtained so far by any particle in the population. After finding its two best fitness values, each particle updates itself in a similar fashion as GA where the configurations are mutated whenever their current velocity values are greater than their previous ones. For instance, if 'AV' vector field gets chosen randomly, then any one value will be randomly picked from the {H, L, N, A} set.

5 EVALUATION AND DISCUSSION

We evaluated the performance of the two evolutionary algorithms through a case study. Table 1 reports the percentages of number of CVSS vulnerability vectors produced for different CVSS score ranges across 100 runs for both GA and PSO scripts. We observed that the values came out to be almost similar.

	[2.0]	(2.0, 3.0]	(2.0, 4.0]	(2.0, 5.0]
GA	3.505	22.956	34.972	38.567
PSO	3.387	23.144	34.088	39.379

Table 1: The percentage instances of CVSS vectors produced by GA and PSO for the different score ranges in 100 runs.

#	VENDOR	CVE ID	VULNERABILITY DESCRIPTION	Score
1	Mysql	CVE-2019-14939	An issue was discovered in the mysql (aka mysqljs) module 2.17.1 for Node.js. The LOAD DATA LOCAL INFILE option is open by default	2.1
2	Mysql	CVE-2016-7440	The C software implementation of AES Encryption and Decryption in *wolfSSL* (formerly CyaSSL) before 3.9.10 makes it easier for local users to discover AES keys by leveraging cache-bank timing differences	2.1
3	Oracle	CVE-2014-6551	Unspecified vulnerability in Oracle MySQL Server 5.5.38 and earlier and 5.6.19 and earlier allows local users to affect confidentiality via vectors related to CLIENT:MYSQLADMIN	2.1
4	Oracle	CVE-2012-3160	Unspecified vulnerability in the MySQL Server component in Oracle MySQL 5.1.65 and earlier, and 5.5.27 and earlier, allows local users to affect confidentiality via unknown vectors related to Server Installation	2.1
5	Mysql	CVE-2006-4031	MySQL 4.1 before 4.1.21 and 5.0 before 5.0.24 allows a local user to access a table through a previously created MERGE table, even after the user's privileges are revoked for the original table, which might violate intended security policy	2.1

Table 2: The number of vulnerabilities found for CVSS pattern: `AV:L/AC:L/PR:N/S:N/C:P/I:N/A:N`**.**

A possible explanation of obtaining similar results for both GA and PSO is that the number of vector fields for CVSS (i.e., Base level) is limited to eight. As a result, there are not too many options for GA or PSO to select from. Therefore, both algorithms converge to the same values quickly because the search space is very small. Considering all the other metrics in CVSS might provide a larger search space for the algorithms.

To illustrate the effectiveness of the introduced adequacy criterion for security testing, we looked up the CVE website [7] and identified different vulnerabilities whose CVSS patterns matched the vulnerability pattern produced by GA and PSO. As an example, based on the CVSS pattern: `AV:L/AC:L/PR:N/S:N/C:P/I:N/A:N` (In C: P , P stands for Partial or Low as per the website), table 2 shows multiple CVEs of vulnerabilities we found in the product MySQL for different vendors along with their description and CVSS score with the exact CVSS vector pattern matching.

6 CONCLUSION AND FUTURE WORK

This paper introduced the concept of "`vulnerability coverage`" as an adequacy criterion for security and vulnerability testing of software applications. The deriving idea is to utilize Common Vulnerability Scoring System (CVSS) as a fitness metric and identify a set of vulnerability vector patterns that achieves a certain level of CVSS score. The generated set can be used for adequacy testing of underlying system in which all or representative sets of vulnerabilities with similar vulnerability vector patterns will be selected for further inspection of the system under test. The paper compared two evolutionary-based algorithms namely Genetic Algorithms and Participle Swarm Optimization and the results indicated a similar results obtained by these two greedy algorithms.

The novel idea of vulnerability adequacy criterion as introduced in this paper needs further attentions. To our best knowledge, an adequacy criterion based on vulnerability coverage does not exist. There are several other features that need to be investigated including other metrics incorporated into CVSS and National Vulnerability Database (NVD) including temporal and environmental metrics. Furthermore, the idea needs tool supports and further empirical studies to thoroughly search the NVD database for reported vulnerabilities with exact pattern matching property for security testing purposes and investigate the effectiveness of such adequacy criterion.

The usefulness of Bayesian approaches have been discussed extensively in the literature [11]. These probabilistic reasoning approaches can be adapted in the context of uncertainty analysis for implementing adaptive security testing in dynamic domains (e.g., reinforcement reasoning [6]). It is possible to apply these learning-based algorithms along with temporal properties and dependencies and then adapt deep learning-based approaches [12] to address the problem. In the presence of existence of some constraints in the configuration settings, the problem can be formulated as a constraint satisfaction problem and the generation of test inputs using symbolic executions [9].

ACKNOWLEDGMENT

This research work is supported in part by a funding from National Science Foundation under grant numbers 1516636 and 1821560.

REFERENCES

[1] Access 2019. Common Vulnerability Scoring System v3.0: Specification Document. https://www.first.org/cvss/v3.0/specification-document.

[2] Accessed 2019. MySQL Version 5.5 Documentation. https://dev.mysql.com/doc/refman/5.7/en/server-option-variable-reference.html.

[3] Accessed 2019. National Vulnerability Database. https://nvd.nist.gov/.

[4] Faranak Abri, Sima Siami-Namini, Mahdi Adl Khanghah, Fahimeh Mirza Soltani, and Akbar Siami Namin. 2019. Can Machine/Deep Learning Classifiers Detect Zero-Day Malware with High Accuracy?. In *IEEE BigData*.

[5] James H. Andrews, Felix Chun Hang Li, and Tim Menzies. 2007. Nighthawk: a two-level genetic-random unit test data generator. In *IEEE/ACM International Conference on Automated Software Engineering*. 144–153.

[6] Moitrayee Chatterjee and Akbar Siami Namin. 2019. Detecting Phishing Websites through Deep Reinforcement Learning. In *Annual Computer Software and Applications Conference, COMPSAC*. 227–232.

[7] MITRE Corporation. 1999. CVE Security Vulnerability Database. https://www.cvedetails.com/

[8] M. Crouse and E. W. Fulp. 2011. A moving target environment for computer configurations using Genetic Algorithms. In *Symposium on Configuration Analytics and Automation*. 1–7.

[9] Marcel Heimlich and Akbar Siami Namin. 2019. TestLocal: just-in-time parametrized testing of local variables. In *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing, SAC*. 1874–1877.

[10] Brian Lucas, Errin W. Fulp, David J. John, and Daniel Cañas. 2014. An Initial Framework for Evolving Computer Configurations As a Moving Target Defense. In *The Annual Cyber and Information Security Research Conference*. 69–72.

[11] Akbar Siami Namin and Mohan Sridharan. 2010. Bayesian reasoning for software testing. In *the Workshop on Future of Software Engineering Research*. 349–354.

[12] Sima Siami-Namini, Neda Tavakoli, and Akbar Siami Namin. 2019. The Performance of LSTM and BiLSTM in Forecasting Time Series. In *IEEE BigData*.

[13] Jianjun Zheng and Akbar Siami Namin. 2019. A Survey on the Moving Target Defense Strategies: An Architectural Perspective. *Journal of Computer Science and Technology* 34, 1 (2019), 207–233.