Hindawi
Advances in Multimedia
Volume 2023, Article ID 9893861, 1 page
https://doi.org/10.1155/2023/9893861

Retraction

Retracted: Detection and Analysis of Man-Machine Interactive Software Vulnerabilities Based on Ultrasonic Data Acquisition and Signal Processing Algorithms

Advances in Multimedia

Received 15 August 2023; Accepted 15 August 2023; Published 16 August 2023

This article has been retracted by Hindawi following an investigation undertaken by the publisher [1]. This investigation has uncovered evidence of one or more of the following indicators of systematic manipulation of the publication process:

(1) Discrepancies in scope
(2) Discrepancies in the description of the research reported
(3) Discrepancies between the availability of data and the research described
(4) Inappropriate citations
(5) Incoherent, meaningless and/or irrelevant content included in the article
(6) Peer-review manipulation

The presence of these indicators undermines our confidence in the integrity of the article's content and we cannot, therefore, vouch for its reliability. Please note that this notice is intended solely to alert readers that the content of this article is unreliable. We have not investigated whether authors were aware of or involved in the systematic manipulation of the publication process.

Wiley and Hindawi regrets that the usual quality checks did not identify these issues before publication and have since put additional measures in place to safeguard research integrity.

We wish to credit our own Research Integrity and Research Publishing teams and anonymous and named external researchers and research integrity experts for contributing to this investigation.

The corresponding author, as the representative of all authors, has been given the opportunity to register their agreement or disagreement to this retraction. We have kept a record of any response received.

References

[1] L. Zhao, "Detection and Analysis of Man-Machine Interactive Software Vulnerabilities Based on Ultrasonic Data Acquisition and Signal Processing Algorithms," *Advances in Multimedia*, vol. 2021, Article ID 7684146, 7 pages, 2021.

Hindawi
Advances in Multimedia
Volume 2021, Article ID 7684146, 7 pages
https://doi.org/10.1155/2021/7684146

Research Article

Detection and Analysis of Man-Machine Interactive Software Vulnerabilities Based on Ultrasonic Data Acquisition and Signal Processing Algorithms

Lei Zhao

Shandong Vocational College of Science and Technology, Weifang 261053, Shandong, China

Correspondence should be addressed to Lei Zhao; hyy@zjnu.edu.cn

Received 27 August 2021; Accepted 22 October 2021; Published 30 November 2021

Academic Editor: Kang Song

With the gradual increase in the informatization, there is much software in various industries, such as data management, business execution, public orientation, and company OA, which greatly facilitates the development of various tasks, but it also brings many hidden dangers. There exist certain vulnerabilities in some software, which have become backdoors to be attacked. In view of these needs and potential hazards, the ultrasonic data acquisition and signal processing algorithms are introduced in this paper, analyzing and grasping the possibility of potentially dangerous paths by combining the instruction addresses and locations of software vulnerabilities, and avoid the existence of these software vulnerabilities through corresponding constraint instructions. The simulation experiment results prove that the ultrasonic data acquisition and signal processing algorithms are effective and can support the detection and analysis of man-machine interactive software vulnerabilities.

1. Introduction

With the continuous development of social economy, informatization has gradually changed the business of various industries, and the data management, business logic, public maintenance, company operation OA, and other software have gradually become variegated and diverse; some run on the PC end, and others run on the mobile side [1]. From the perspective of the running network, some are included in related private networks, internet networks, government affairs networks, and local area networks. These pieces of software are relatively complex and complicated, so the stability of software and the security of resources are extremely important [2, 3]. The existence of hidden loopholes will have an impact on social security, so the security of information is very important, and conducting information security detection is extremely effective [4, 5].

Therefore, during the running process of software, the corresponding program operation needs to be monitored in real time to determine which codes can be executed safely and which codes can be executed directly without symbolic execution [6, 7].

Therefore, during the fixed detection process, setting the corresponding breakpoints for detection can make it clear that these signals are needed during the execution process, but as for detection, the processing and execution of breakpoints are actually an iterative process. By going through all the possibilities of the entire software program, the breakpoint can be understood. However, it should be noted that when the scale or size of software is large enough, it is difficult to fully realize the experience of all the paths, but for most of the paths, there are actually very few vulnerabilities. Therefore, the detection of vulnerabilities can be carried out directly by constructing corresponding test cases, such as stain testing and signal processing, but there are low coverage rate of codes and missing scans in most of these methods. Therefore, it is necessary to increase the detection coverage rate of the program to ensure that the scan of the vulnerability is the safest [8, 9].

Therefore, some scholars replace the corresponding symbolic execution with the corresponding abstract symbol and perform equivalent semantic operations according to the essence of the relevant program, so as to realize the simulation of the program, but because the execution of the abstract symbol is static, the corresponding source code is required, which also caused a lot of misdeclaration of software programs.

In view of the above limitations and requirements, based on ultrasonic data acquisition and signal processing algorithms, searching and analyzing are performed by collecting the corresponding instructions of the dangerous functions through combining and identifying the corresponding signal processing dangerous addresses, meanwhile, constructing the corresponding antitrigger mechanism to avoid further expansion or existence of the corresponding vulnerabilities or dangers, effectively reducing the possibility for triggering of software or programs in a timely and effective manner, aiming to explore the safety and perfection of human-computer interactive software, and further ensuring the effective and stable operation of software.

2. Ultrasonic Data Acquisition and Signal Processing Algorithms

For software vulnerabilities, due to different methods and processes during their generation, the corresponding detection methods are also different. Of course, it should be noted that no special method can completely detect all the vulnerabilities. The detection of specific vulnerabilities requires several corresponding methods. For example, the detection of vulnerabilities that are easily triggered by overflow of values, divisors of 0, etc., needs to be identified and analyzed [10–12].

For the vulnerability detection of software or programs using corresponding signals, as shown in Figure 1, first, an initial input is given. When the target program or software is executed, the input data are first initialized in the symbolized manner. During the execution of the program or software, the constraints of the corresponding mandatory path shall be collected. After the entire program is executed, a set will be obtained for the constraints of the path; in view of the set of mandatory constraints, all the conditions are negated, and the input of corresponding constraints is performed. Through the construction of corresponding test cases, continuous iterations are achieved, while the status of the program or software execution is monitored simultaneously. If the situation exists, it means there may be a problem, and a detailed analysis and exploration of the cause of the breakpoint or the place where the error is reported need to be performed to determine whether it is a vulnerability and related attributes.

If you need to detect a certain type of vulnerabilities, you first need to identify the relevant patterns of this type of vulnerabilities. This requires the aggregation and classification of existing vulnerabilities, and the related vulnerabilities can be obtained through summarization and induction; meanwhile, they can have corresponding characteristics in the form of expression [13, 14]. If there may be

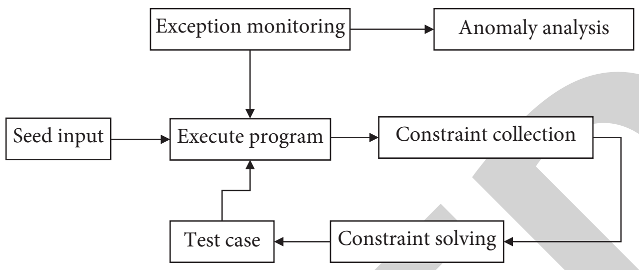

Figure 1: The general process of detecting software vulnerabilities using signal processing.

function insecurity and data overflow (more than 256 or overflow of memory), these functions can be classified as dangerous functions, and the calls of these dangerous functions can also be considered as the use of dangerous paths.

In the process of signal denoising, the selection and quantification of the threshold are very critical; it can be said that it directly affects the effect of denoising. Therefore, in the application, the selection method and quantification rule of the threshold should be determined according to the specific situation [15].

Commonly used threshold processing methods include hard threshold and soft threshold.

$$\tilde{x} = \begin{cases} x, |x| \geq \lambda, \\ 0, |x| < \lambda, \end{cases} \tilde{x} = \begin{cases} x - \lambda, x \geq \lambda, \\ 0, |x| < \lambda, \\ x + \lambda, x \leq -\lambda. \end{cases} \tag{1}$$

The soft and hard threshold compromise method is

$$\tilde{x} = \begin{cases} x - \alpha\lambda, x \geq \lambda, \\ 0, |x| < \lambda, \\ x + \alpha\lambda, x \leq -\lambda. \end{cases} \tag{2}$$

In order to compare the noise reduction effects of different threshold noise reduction methods, the signal-to-noise ratio is used as the evaluation criterion. The formula for the signal-to-noise ratio is

$$\mathrm{SNR} = 10 \times \log \frac{\sum_{i=1}^{N} f^2(i)}{\sum_{i=1}^{N} (s(i) - f(i))^2}. \tag{3}$$

In the formula, $f(i)$ is the original signal; $s(i)$ is the noisy signal; N is the signal length.

The processing of all signals is mainly to use the corresponding signal for searching the program until the dangerous function and the corresponding dangerous path are found. After the dangerous path is found, the corresponding test and detection can be carried out. No detailed instructions are required for other paths. In this way, the calculation amount of the test is reduced, and the scope of the test is reduced, therefore further improving the efficiency of the test. The specific execution workflow is shown in Figure 2. First of all, the specific pattern of the vulnerability needs to be clarified by sorting out and determining the existence of the dangerous function by means of the corresponding pattern, and then the ultrasonic data collection is

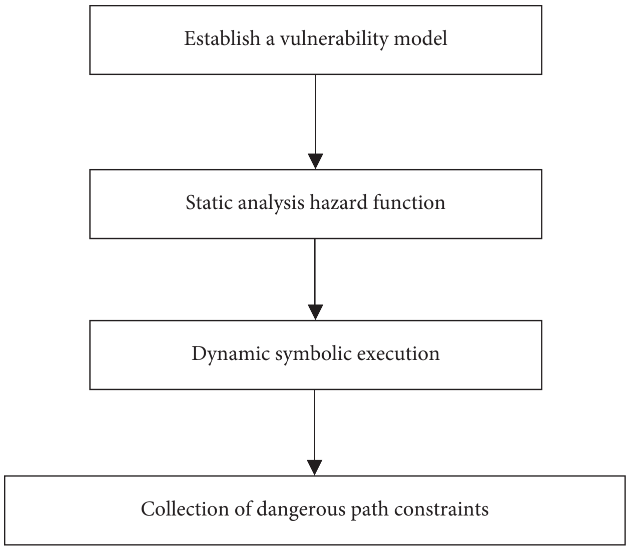
Establish a vulnerability model
Static analysis hazard function
Dynamic symbolic execution
Collection of dangerous path constraints

Figure 2: The execution workflow of guided signal processing.

used to identify and locate the instructional address of the corresponding dangerous function to determine the dangerous function according to the corresponding constraint dataset. Finally, until the program is executed, the corresponding constraints of the dangerous path corresponding to the dangerous function can be summarized.

The input signal is used to initialize the corresponding input file, analyze the branch quality, and run the target program, while the corresponding type analysis technique is used to obtain more type information, specifically as shown in Figure 3.

Through programming, the loopholes in the integer are identified to determine whether there are any conflicts in the existing data types.

3. Binary Program Ultrasonic Data Acquisition

Ultrasonic data acquisition refers to the analysis of the source code or binary code of the program without running the program [16–18]. Through ultrasonic data acquisition, a clear framework understanding of the program can be realized, and the combination of dynamic and static is realized according to the corresponding ultrasonic data acquisition, focusing on solving the problem of false alarm rate [19, 20].

The specific process is shown in Figure 4. First, the list of dangerous functions is sorted out, and corresponding initialization is performed. Those functions listed as dangerous are the objects of focus; secondly, each dangerous function in the list is analyzed and extracted; for each function, the corresponding call address is obtained by calling the corresponding tool. If the address used is reasonable and legal, the call points to the corresponding code list, and finally, the instructional address is printed and marked in red directly.

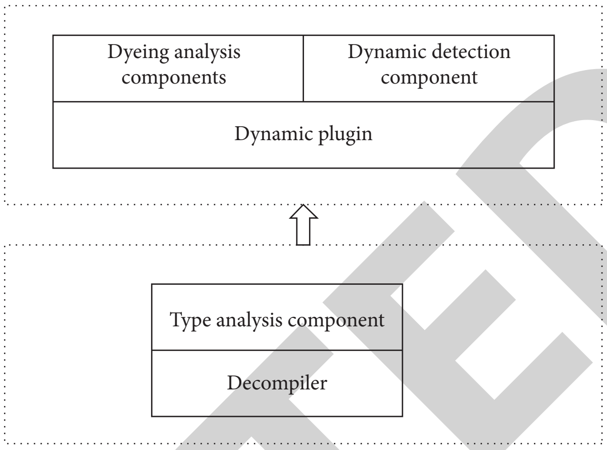

Figure 3: Structural diagram of the detection tool.

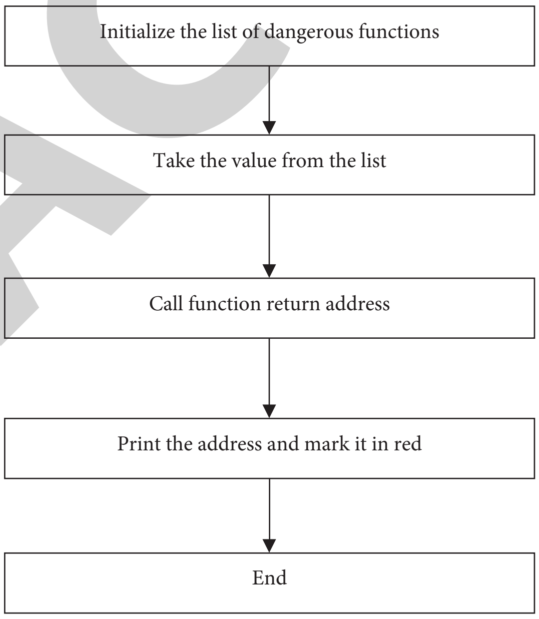
Initialize the list of dangerous functions
Take the value from the list
Call function return address
Print the address and mark it in red
End

Figure 4: Flowchart of the ultrasonic data acquisition algorithm.

4. Formal Description of Integer Vulnerabilities

Integer vulnerabilities may appear in operations related to integer variables, assignment operations, etc., but not these two types of instructions are integer vulnerabilities. It is also necessary to determine whether the value of the integer variable is within the range that its type can represent and whether the value is related to the external input.

4.1. Type Indication. The corresponding types can be effectively classified, which can be divided into two types: width information and symbol information.

Definition 1. Use T to represent the integer type, C_4^2 indicates the width type, and Ts represents the length type.

Definition 2. Assuming $t \in T$, two-tuple C_4^4 records the range of values that t can represent. $C_4^0 + C_4^1 + C_4^3 = 8$ represents the minimum value that t can represent, and $\min_t$ represents the maximum value.

4.2. Modeling Integer Vulnerabilities. For the operations and assignment operations mentioned in the previous article, we restrict them to check whether they constitute integer vulnerabilities. This restriction is our modeling of integer vulnerabilities. We construct the following constraints:

Rule 1
operation (addr)$\longrightarrow$
$\min_{\text{resut}} \leq$ result-value $\leq \max_{\text{result}}$
result_value = loperand opcode roperand
Rule 2
assignment (addr)$\longrightarrow$
$\min_{\text{destination}} \leq$ source − value $\leq \max_{\text{destination}}$
Rule 3
operand's type is bot$\longrightarrow$operand_value ≥ 0

5. Ultrasonic Data Collection and Analysis

For the binary program, the ultrasonic data acquisition method is used to extract the type information from the binary program and construct the suspicious set.

The extended type analysis is as follows: (1) a decompiler is used to convert the binary program into an intermediary language. (2) On the intermediary language, according to the order of decompilation, some specific functions and statements are used to extract information, including arithmetic/logical operations, judgment statements, array subscripts, memory allocation functions, and memory copy functions.

6. Simulation Experiment

The simulation experiment is selected to use the normal operating system; firstly, the program is used to test to determine whether the dangerous path can be identified, and secondly, the real program is used to test.

6.1. Simple Program Test. The test program is a binary program test_strcpy in the ELF format, and part of the source code is as follows:

```
void function (char * str){
char buffer [6];
strcpy (buffer, str);
}
int main (int argc, char * argv[]){
char buf [5] = {0};
int fd, i, num = 0;
fd = open (argv [1], O_RDONLY);
for (i = 0; i ≤ 3; i++)
read (fd,&buf [i], sizeof (char));
if (buf [0] = =  "g") num++;
if (buf [1] = = "o") num++;
if (buf [2] = = "o") num++;
if (buf [3] = = "d") num++;
if (num = = 2) return 2;
if (num = = 4) {function (buf); return 4; }
return 0;
}
```

First, the corresponding string is obtained from the input file to see if it is similar or the same as the dangerous character of the corresponding dangerous function. If the output result of the same number of characters is different, then the dangerous function needs to be called in the corresponding list.

Given the content of the corresponding initialization input file, the dangerous function is used for the test to generate the corresponding test case. The specific calls are shown in Table 1.

The accuracy of this result is analyzed mainly from two aspects as follows: one is whether the total number of paths should be 15, and the other is whether the no. 14 test case will take a dangerous path.

(1) Integer overflow: in view of the integer overflow, the integer overflow is detected through the EFLAGS register. (2) Error in sign: if it is a sign error, it is necessary to check which conflicting operations exist to determine whether the value is a negative number. (3) Assignment truncation: through the detection of assignment statements, the judgment of the operating range is realized.

7. Real Application Testing

The corresponding program is used to test. The specific test results are shown in Figure 5. First, input and initialize a file, and constrain by finding the corresponding dangerous path. Meanwhile, save these constraints, specifically as shown in Figure 5. It can be seen from the results that the restriction of the dangerous path is relatively more complicated than other programs. Meanwhile, it shows that the ultrasonic data acquisition and signal processing algorithm can effectively realize the identification and analysis of the dangerous function and the dangerous path.

Through ultrasonic data acquisition and signal processing algorithms, the dangerous function and dangerous path can be effectively identified, and the corresponding constraint collection is realized. After summary and analysis, the corresponding test case detection and analysis can be

Table 1: Test results.

Test case number	1	2	3	4	5
Content	gaaa	“NUL”oaa	“NULL” “NULL” Ob	“NUL” “NUL” “NUL”d	“NUL”“NUL”od
Test case number	6	7	8	9	10
Content	“NUL” ooa	“NULL”o “NULL”d	Goaa	g“NUL” oa	g“NUL” “NUL”d
Test case number	11	12	13	14	15
Content	G“NUL” od	Good	go “NUL”d	Good	“NUL” ood

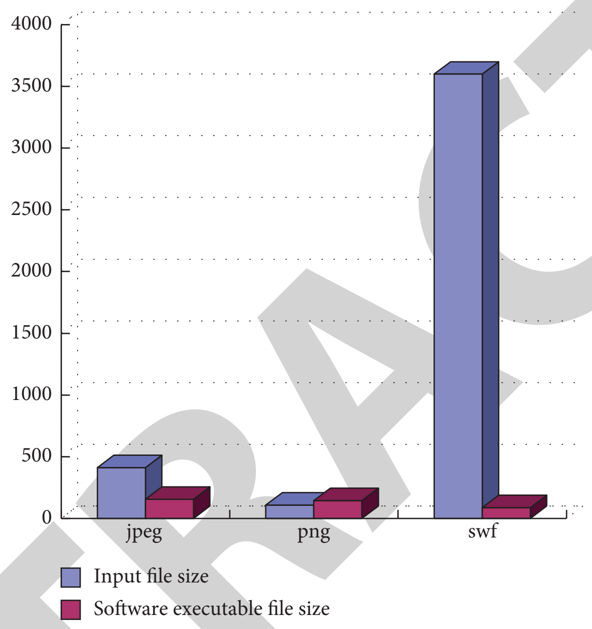

Figure 5: Test results of the real application.

Table 2: The number of suspicious instructions and the accuracy of their type information.

Program name	Size	Integer overflow	Symbol error	Assignment truncation	Total	Accuracy of type information (%)
slocate-2.7	46.3 K	5/6	1/1	1/1	8/8	100
zgv-5.8	284.4 K	22/24	18/18	16/16	58/59	98.3
Python-2.5.2	3.1 M	94/96	22/22	61/67	177/184	96.2
ngiRCd-0.8.1	329.5 K	13/17	13/13	18/19	46/49	93.9
OpenSSH-2.2.1	150.2 K	13/15	1/1	9/9	24/25	96
mpg123-1.7.1	1.02 M	6/8	4/4	1/1	12/13	92.3
rdesktop-1.5.0	562.7 K	2/2	1/1	0/0	3/3	100

Note. x/y: y represents the number of suspicious instructions, and x represents the number of instructions with correct type information.

realized. On this basis, the dangerous path can be further tested until the vulnerabilities are discovered.

8. Suspicious Instruction Set

The static analysis and construction of the instruction set can be realized through the corresponding tools. This is the first step to detect vulnerabilities. Depending on the difference of the program, the results of static analysis are also different. From the results, it can be seen that the static analysis by the ultrasonic data acquisition and signal processing algorithms greatly reduces the amount of computation required by the number of instructions.

9. Accuracy of Type Information Extraction

By comparing the doubtful instructional type information with the corresponding type of the program, it can be found that the matching degree between the two is relatively high, reaching more than 92%. The specific experimental results are shown in Table 2. The main manifestations of the inconsistent types are in the following: (1) it is difficult to ensure consistency between pointer variables and integer variables; (2) it is also difficult to

Table 3: Integer vulnerability detection.

CVE#	Program name	Name of vulnerabilities	Type	Whether detect	Reported vulnerabilities	Real vulnerabilities
2013-0326	slocate	Parse-decode-path bug	Integer overflow	√	2	2
2014-1095	zgv	Multiple integer overflow	Integer overflow	√	22	22
2018-1721	Python	Zlib extension module bug	Symbol error	√	2	2
2015-0199	ngiRCd	List-MakeMask bug	Integer overflow	√	2	2
2011-0144	OpenSSH	Detect-attack bug	Assignment truncation	√	2	2
2019-1301	mpg	Thestore-id3-text bug	Symbol error	√	4	3
2018-1801	rdesktop	Iso-recv-msg function () bug	Integer overflow	√	2	2

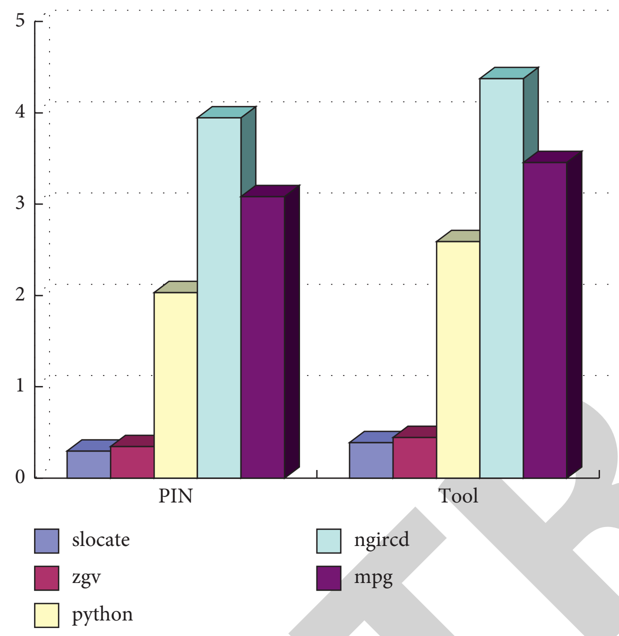

Figure 6: Performance overhead of dynamic detection components.

ensure consistency between pointers and specific arrays, which are easily regarded as unsigned.

Corresponding tools have been tested for effectiveness. Specifically as shown in Table 3, the ultrasonic data acquisition and signal processing algorithms are effective, achieving more than 94% of the detection of vulnerabilities, and only one is underreported.

Through the dynamic performance test, the result is shown in Figure 6. It can be seen from the result that the dynamic detection component is suitable for monitoring the program at runtime. The performance overhead of the dye analysis component is also tested by us, and its performance overhead is about 50 times. Since the dye analysis component can reduce the false alarm rate of detecting integer vulnerabilities, but will introduce a large performance overhead, we provide an interface to provide users with the choice of whether to use the dye analysis component.

10. Conclusions

With the continuous deepening of informatization, for all walks of life, security and stability of software have become the emphasis and difficulty of attention. The algorithms of ultrasonic data acquisition and symbol processing are introduced in this paper, and the dangerous functions are obtained within the program by sorting out the ultrasonic data acquisition methods. On this basis, the signal processing is used to identify and analyze the dangerous path, and continuous iterative analysis to the traversal of the program is performed completely, the corresponding full path efficiency is tested, the full detection of dangerous paths is realized, and the software vulnerabilities are detected. The simulation experiment proves that the algorithms of ultrasonic data acquisition and symbol processing are effective, can effectively identify dangerous paths, and support the detection and analysis of man-machine interactive software vulnerabilities.

Data Availability

The data used to support the findings of this study are available upon request to the author.

Conflicts of Interest

The author declares no conflicts of interest.

References

[1] S. Kim, R. Y. C. Kim, and Y. B. Park, "Software vulnerability detection methodology combined with static and dynamic analysis," *Wireless Personal Communications*, vol. 89, no. 3, pp. 777–793, 2016.

[2] Y. A. Han, A. Sl, and A. Lp, "HAN-BSVD: a hierarchical attention network for binary software vulnerability detection," *Computers & Security*, vol. 5, no. 4, pp. 1–9, 2021.

[3] I.-S. Jeon, K.-H. Han, D.-W. Kim, and J.-Y. Choi, "Using the SIEM Software vulnerability detection model proposed," *Journal of the Korea Institute of Information Security and Cryptology*, vol. 25, no. 4, pp. 961–974, 2015.

[4] C. Chen, H. Xu, and B. Cui, "PSOFuzzer: a target-oriented software vulnerability detection technology based on particle swarm optimization," *Applied Sciences*, vol. 11, no. 3, pp. 1095–1103, 2021.

[5] X. Li, L. Wang, Y. Xin, Y. Yang, Q. Tang, and Y. Chen, "Automated software vulnerability detection based on hybrid neural network," *Applied Sciences*, vol. 11, no. 7, pp. 3201–3209, 2021.

[6] B. Wang and B. Cui, "Ontology-based services for software vulnerability detection: a survey," *Service Oriented Computing and Applications*, vol. 13, no. 4, pp. 333–339, 2019.

[7] L. Wang, X. Li, R. Wang, Y. Xin, M. Gao, and Y. Chen, "PreNNsem: a heterogeneous ensemble learning framework for vulnerability detection in software," *Applied Sciences*, vol. 10, no. 22, pp. 7954–7965, 2020.

[8] S. Liu, G. Lin, and Q. L. Han, "DeepBalance: deep-learning and fuzzy oversampling for vulnerability detection," *IEEE Transactions on Fuzzy Systems*, vol. 28, no. 7, pp. 1329–1343, 2020.

[9] J. Hu, J. Chen, L. Zhang et al., "A memory-related vulnerability detection approach based on vulnerability features," *Tsinghua Science and Technology*, vol. 25, no. 5, pp. 604–613, 2020.

[10] R. Amankwah, P. Kwaku, and S. Yeboah, "Evaluation of software vulnerability detection methods and tools: a review," *International Journal of Computer Applications*, vol. 169, no. 8, pp. 22–27, 2017.

[11] M. Kumar and A. Sharma, "An integrated framework for software vulnerability detection, analysis and mitigation: an autonomic system," *Sādhanā*, vol. 42, no. 9, pp. 1481–1493, 2017.

[12] G. Tang, L. Yang, S. Ren, L. Meng, F. Yang, and H. Wang, "An automatic source code vulnerability detection approach based on KELM," *Security and Communication Networks*, vol. 2021, no. 1, 12 pages, Article ID 5566423, 2021.

[13] H. Hanif, M. Nasir, and M. Razak, "The rise of software vulnerability: taxonomy of software vulnerabilities detection and machine learning approaches," *Journal of Network and Computer Applications*, vol. 179, no. 9, pp. 103–110, 2021.

[14] "A novel deep learning-based feature selection model for improving the static analysis of vulnerability detection," *Neural Computing and Applications*, vol. 5, no. 4, pp. 190–198, 2021.

[15] T. Given-Wilson, N. Jafri, and A. Legay, "Combined software and hardware fault injection vulnerability detection," *Innovations in Systems and Software Engineering*, vol. 16, no. 2, pp. 101–120, 2020.

[16] A. Qasem, P. Shirani, M. Debbabi, L. Wang, B. Lebel, and B. L. Agba, "Automatic vulnerability detection in embedded devices and firmware," *ACM Computing Surveys*, vol. 54, no. 2, pp. 1–42, 2021.

[17] Ch Suryanarayana, "A novel approach using fuzzy sets for detection of vulnerability and imprecision in software estimation and particle swarm optimization for tuning parameters," *International Journal of Applied Engineering Research*, vol. 13, no. 9, pp. 8431–8435, 2018.

[18] W. Qiang, Y. Liao, and G. Sun, "Patch-related vulnerability detection based on symbolic execution," *IEEE Access*, vol. 3, no. 9, pp. 1–10, 2017.

[19] Y. Li, L. Ma, and L. Shen, "Open source software security vulnerability detection based on dynamic behavior features," *PLoS One*, vol. 14, no. 8, pp. 221–230, 2019.

[20] J. Hu, J. Chen, S. Ali et al., "A detection approach for vulnerability exploiter based on the features of the exploiter," *Security and Communication Networks*, vol. 2021, no. 1, 14 pages, Article ID 5581274, 2021.