A static comprehensive analytical method for buffer overflow vulnerability detection *

Shao Bilin, Yan Jiafen, Bian Genqing, Zhao Yu, Song Dan
School of Management
Xi'an University of Architecture and Technology
Xi'an, China

***Abstract*—Buffer overflow vulnerability is a widespread and dangerous security problem. Detecting buffer overflow vulnerability has great research value in information security area. This paper proposes a static comprehensive analytical method for buffer overflow vulnerability detection. Firstly, this method adopts many kinds of static detection tools for detecting the source codes and producing their own detecting reports. Secondly, comprehensive analysis is implemented to evaluate the reliability weights of detecting tools by training process with detection results, and further optimize the detection results. This training process can improve the efficiency of discovering buffer overflow vulnerabilities with lower rate of omissions and misstatements. The experimental results show that compared with single static detection methods, the rates of both false alert and missed alert decrease significantly.**

Keywords—Buffer overflow; vulnerability detection; comprehensive analysis; reliability weights

I. INTRODUCTION

Recently, the comprehensive utilization technology of buffer overflow vulnerabilities, Trojan horse, computer virus and Botnet brings new challenges to information security. Buffer overflow is one of the most effective and common network attack means with strong concealment and destructive. Therefore, strengthening the detection and prevention against buffer overflow vulnerability has a great research value. In existing methods for vulnerabilities detection, the static detecting methods focus on finding overflow vulnerability in the stage of software development, in order to improve the reliability and security of softwares[1]. In contrast, dynamic detection methods focus on the post-defense after the release stage of the software, and play the role of defending overflow vulnerability against illegal exploitation. However, dynamic detection methods may lead to terminate the execution of the program. In this sense, the static detection methods in the process of software development may be more significant than the dynamic detection methods. Because of the limitation of the rule base and one-sidedness of static detection methods, adopting single detection tool may usually cause high false alarm rate and false negative rate. Therefore, this paper proposes a novel detecting method to synthesize the results produced by multiple static detection tools comprehensively, so as to verify the results, correct mistakes and reduce the false negative rate and false alarm rate.

II. STATIC METHODS FOR DETECTING BUFFER OVERFLOW VULNERABILITY

Considering the differences between the detection methods, the existing static detection tools based on source code can be assorted to two categories: static analysis and program verification [2].

A. Static analysis [3]

Static analysis scans the codes directly to get the syntactic and semantic information, and then compares them with the predetermined characteristic database of vulnerabilities and safety rule database to detect vulnerabilities.

1) Lexical analysis

Lexical analysis needs to scan the source program, partition the programs by word symbols, match with the characteristic database of the tool, and then find out the possible buffer overflow vulnerabilities in the program. These methods often only check some known vulnerabilities in the code, it is easy to cause misinterpretations and omissions.

2) Type inference

By automatically deriving the types of variables and functions in the program, this method determines whether accessing the variables and functions is illegal [2].This method does not take into account the execution conditions and the order of the codes, so the processing speed is very fast. However, due to the infiniteness of constraint variables, so it can only be used to detect specific types of buffer overflow vulnerability.

3) Rules check

There are some common security rules in general, also known as the vulnerability model. Rule checking method is used to describe these rules with a specific syntax and compare with the behavior of the code.

The representative tools of the static analysis and their advantages and disadvantages are shown in TABLE I.

*The Natural Science Foundation of China under Grant No. 61073196, No.61272458; the Natural Science Foundation of the Shaanxi Province under Grant No. 2014JM2-6119; the Science and Technology Foundation of Yulin under Grant No. 2014cxy-12.

TABLE I. CLASSIFICATION OF STATIC ANALYSIS

Detection method	Representative tools	Advantages and disadvantages
Lexical analysis	ITS4[7]、Flawfinder[8]、RATS	High efficiency; analysis is not accurate, false negative rate is high
Type inference	CQual、CCured、Percent-S	Able to handle large-scale programs with high efficiency; can check Limited vulnerability
Rules check	LCLint、Splint	Able to dealing with large-scale programs with high efficiency; restricted by rules, only can analyze specific types of vulnerabilities

B. *Program verification*

The program verification [6] can obtain formalized computer programs or models by abstracting, and then detect the vulnerability by verifying formal program or model.

1) Model checking

Firstly, model checking methods construct the abstract model of the program based on state machine or digraph. Secondly, these methods traverse the abstract model. Because model checking needs to enumerate all possible states, it can only model one property of the program.

2) Theorem proving

The theorem proving methods verify the logic formulas with a variety of determining processes. During these analyses, the logic formulas are transformed from the safety standards and the semantic transformation of the codes. This method can guarantee that there is no buffer overflow vulnerabilities through validation program. However, depended on the information of calling procedure provided by researchers, the degree of automation is not high enough.

3) Symbolic execution

Symbolic execution mainly converts the value logic of the variable into the abstract symbol, simulates path sensitive program control flow, and then tests whether there is the possibility of error by constraint solution.

The representative tools of the program verification and their advantages and disadvantages are shown in TABLE II.

TABLE II. CLASSIFICATION OF PROGRAM VERIFICATION

Detection method	Representative tools	Advantages and disadvantages
Model checking	MOPS、UNO	Able to do accurate detections; may cause the state space explosion[9]
Theorem proving	CSSV、Eau Claire	Using strict reasoning proof to control detection, false alarm rate is low; the lack of applicability in some formula reasoning
Symbolic execution	BOON、ARCHER	Able to find the subtle logic errors in the program; the number of possible paths increases rapidly with the increasingsize of the program

The theoretical basis of program verification methods is more stringent than the static detection methods, so the false alarm rate is generally low. However, due to its high operating costs, most of the detection efficiency is lower. Static analysis can get high efficiency, but its false positive rate and false negative rate are relatively high. If the program verification is directly integrated with the static analysis, it may affect the whole efficiency of the detection [7].Therefore, this paper is committed to integrate a variety of static analyses. The combination of program verification methods will be the next research objective.

III. INTEGRATED DETECTING METHOD

A. *The main idea*

The rule bases used for different static detection tools for detecting buffer overflow vulnerability are different, so vulnerabilities they can find exist differences. According to the literatures, it is pointed that flaw finder finds most of vulnerabilities, but not completely includes vulnerabilities found by RATS and ITS4 aimed to the same software package. The quantity of vulnerabilities found by RATS and ITS4 have a smaller difference, but with many disjoint sets [8]. Splint is adopted to detect modular for the C language [9].

If different tools are integrated into the same system to conduct a comprehensive detection, the different results of vulnerability detecting methods will mutually complement, and the rate of false positive and false negative will decline. The process of the integrated detection method is shown in Fig. 1.

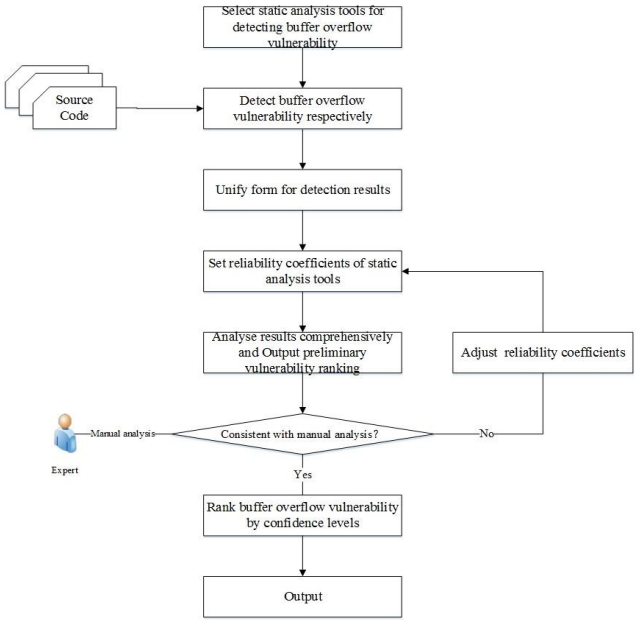

Fig. 1. The process of the integrated detection method

The method is mainly divided into three steps: preprocessing, synthesize detection results and train tool's credibility weight.

1) Step 1

In the preprocessing, the integrated detection tools need to be selected firstly. Because of the differences between the formats of the reports generated by different tools, the formats of these reports should be transformed into a unified format.

2) Step 2

Aimed to synthesize detection results, there are two mainly problems should be solved. One is the unifying the risk level of vulnerability. Another one is the problem of calculating train tool's Credibility weight.

3) Step 3

In the tool's reliability weight training, the mainly task is obtaining the difference between the output and actual vulnerabilities based on users' feedback and modifying tool's Credibility weight based on the differences.

Suppose there are n buffer overflow vulnerability detection tools, the credibility weights of the i-th tool($1 \leqslant i \leqslant n$) is *w(i)*. The reported correct number of vulnerabilities is *TS(i)*. *FS(i)* represents the number of the error vulnerabilities reported by detecting tools. *LOE* means the actual vulnerable number of measured source code. *TP(i)* represents positive reported rate. The false positive rate is *FP(i)*. *FN(i)* represents false negative rate. *RE(i)* means result set. According to the meaning of false positive rate and false negative rate, some formulas can be given as follows:

$$TP(i) = \frac{TS(i)}{LOE} \tag{1}$$

$$FP(i) = \frac{FS(i)}{FS(i)+TS(i)} \tag{2}$$

$$FN(i) = 1 - \frac{TS(i)}{LOE} \tag{3}$$

The three assumptions mentioned former can be described as follows:

(1) $RE(1) \cup RE(2) \cup \ldots \cup RE(n) \geq RE(i), 1 \leq i \leq m$

(2) $\prod_{i=1}^{n} TP(i) < min\{TP(i)\}$

(3) $\prod_{i=1}^{n} FP(i) < min\{FP(i)\}$

B. Preprocessing

The preprocessing is mainly to prepare for the comprehensive testing. First, the set of static detection tools which will be integrated should be configured. The tools which will be selected should rely on the type of static analysis. The selection criteria are open source code, widely used, and the principle adopted by the detection methods. Due to the detection results found by RATS and ITS4 with little difference, ITS4 and Flawfinder are chosen from lexical analysis tools. In type inference, the CQual is chosen because CCured is not a completely static detection tool. In check rules, Splint is chosen, which is a new version of LCLint .

After detecting by the batching tools, a number of testing reports are produced, the format of these reports have differences which hinder the combination of detection results. Reports produced by different tools are quite different but contains some common information, such as filename, line numbers, function of vulnerability, risk level, and vulnerability descriptive information. According to above information, the integrated system can unify the different test reports into a unified description which is a sequence of multi-tuples including detection tools' name, filename, line number, defective function, risk level and description [10]. After unifying the format of the report, the comprehensive test section will be executed.

C. Comprehensive analysis of the test results set

Synthesize detection results is the core part of the whole approach. After obtaining preprocessed results, unifying different risk level of vulnerability is needed at first. Secondly, the comprehensive detecting method deduces the comprehensive marks of all vulnerabilities with the reliability weights of different detection tools.

1) Unify vulnerability risk level

Different detection tools have their own ways to quantify risk level of vulnerability. For instance, Flawfinder divides the risk levels into five levels:1,2,3,4,5. ITS4 has four levels: Some Risk，Risky，Very Risky，Urgent[13]. Literature [1] shows a hierarchical mapping for risk uniform. For instance, the level of vulnerability risk used in ITS4 is mapped into five levels: *f(some risk)=1* ， *f(risky)=2* ， *f(very risky)=4* ， *f(urgent)=5*.However, this method is too simple to show difference of vulnerability risk levels. So this paper improves the algorithm and show as follows:

Assume the vulnerability risk level of i-th tool in the comprehensive detection is expressed as *c(i)*($1 \leqslant i \leqslant n$).In this comprehensive detection, different risk level of vulnerability should be quantized into a uniform five levels. To ensure the gap between the adjacent levels is equal after mapping, equivalent mapping should be implemented for the tools with different risk levels. For example, In Flawfinder， *f(1)=1*， *f(2)=2* ， *f(3)=3* ， *f(4)=4* ， *f(5)=5*; In ITS4 ， *f(some risk)=(5/4)*1*， *f(risky)=(5/4)*2*， *f(very risky)=(5/4)*3*， *f(urgent)=(5/4)*4*，the gap between the adjacent levels is 5/4. Some others tools use the same way to deal with level mapping. After level mapping, the estimated values of vulnerability risk by different detection tools are transformed by the unified vulnerability risk level.

2) Calculation of the credibility weight of detection tools

Tools' reliability represent the importance of each tool in the process of comprehensive detecting results. As mentioned above, we can know that in the comprehensive detection, the set of tools' credibility weight $\{w(i)|i=1,2\ldots m\}$ must follows these formulas:

（1）$0 \leqslant w(i) \leqslant 1$， $i=1,2\cdots n$

（2）$\sum_{i=1}^{m} w(i) = 1$, m is the number of detection tools.

Tools' credibility weights are measured by the false positives rate and false negatives rate in comprehensive detection. Assume the false positives rate of the i-th tool for detecting the source code j-th is $FP^j(i)$, the false negatives rate is $FN^j(i)$, ($0 \leqslant FP^j(i) \leqslant 1$, $0 \leqslant FN^j(i) \leqslant 1$, $i=1,2\cdots n$， $j=1,2\cdots m$). Due to inversely proportional relationship between the tools' credibility weights and the rate of false positive, the higher false alarm rate is, the lower tool's reliability weight is, so do the rate of false negative.

For calculating the credibility weights of detection tools, some special situation should be considered as followed:

(1) When the false positives rate $FP^j(i)$ and false negative rate $FN^j(i)$ are close to 0, the tool's credibility weight should tend to 1;

(2) When the false positives rate $FP^j(i)$ and false negative rate $FN^j(i)$ are close to 1, the tool's credibility weight should tend to 0;

(3) When one of the false positive rate $FP^j(i)$ and false negative rate $FN^j(i)$ tends to 0, the other tends to 1, the tool's credibility weight should be tends to 0.

Generally, when the false positives rate $FP^j(i)$ and false negative rate $FN^j(i)$ increase, its credibility weight should decrease. Conversely, when the false positives rate $FP^j(i)$ and false negative rate $FN^j(i)$ decrease, its credibility weight should increase. From the above, Formula 4 is used to calculate weight $w^j(i)$:

$$w^j(i) = \left(1 - FP^j(i)\right)\left(1 - FN^j(i)\right) \quad (4)$$

The formula accords with the above special cases and also conforms to the general trend of the change. After calculating the value $w^j(i)$ during all source codes detection, this paper proposes formula 5, a similar way of standardization, to calculate the credibility weight $w(i)$ of each detection tool:

$$w(i) = \sum_{j=1}^{n} w^j(i) \Big/ \sum_{i=1}^{m}\sum_{j=1}^{n} w^j(i) \quad (5)$$

The credibility weight $w(i)$ of each detection tool can be calculated by formula 5. For the same vulnerability z, its comprehensive estimated value of security risk should be weighted with $w(i)$ and calculated by the formula 6.

$$E(z) = \sum_{i=1}^{m} c(i) \times w(i) \quad (6)$$

Finally, the system will rank vulnerabilities based on comprehensive estimated values and tools' credibility weight.

D. Train the tool's credibility weight

Experts' knowledge is needed in the tool's credibility weight training. Firstly, analyst need to find the vulnerability in source codes and evaluate the authenticity of detection results given by static tools. Secondly, the differences between manual analysis and tools' results feedback to comprehensive detection process to adjust each tools' credibility weight. If one tool produce a large number of false vulnerabilities, its weight will be reduce, vice versa. This training process can make credibility weight more reasonable. After adjustment, this process will be iterated until the difference between the test results and the actual situation under a certain threshold.

IV. EXPERIMENTS

Aimed to buffer overflow vulnerabilities detection, This paper select three open softwares, including WU-FTPD 2.5.0，Net-tools 1.4.6 and Pure-FTPD 1.0.17a, to verify the effectiveness of detecting tools for overflow vulnerabilities. The experimental environment is bases on win7, MYSQL5.0, VC++6.0, and Cygwin. The basic information of the test code is shown in Table 3. LOP is the total lines of code. LOC is the total lines of code except blank lines and comments. LOE is the number of actual vulnerability. LOE/LOC represents the vulnerability density in the code.

TABLE III. THE BASIC INFORMATION OF THE TEST CODE

Program	LOP	LOC	LOE	LOE/LOC(%)
WU-FTPD	20772	13582	64	0.471
Net-tools	10878	4146	50	1.206
Pure-FTPD	29275	25230	1	0.004

Due to untrusted external input data, the security risk caused by the format string vulnerability is the same as threats by buffer overflow vulnerabilities. Therefore, the detecting objects considered in experiments include buffer overflow vulnerabilities and format string vulnerabilities. Each detection results not only contain the buffer overflow vulnerabilities and the format string vulnerability, but also contain other vulnerabilities. However, this research only considers the first two vulnerability types. Due to the large computational cost, vulnerabilities' comprehensive estimated value greater than or equal to 3.0 will be chosen to analysis. The results of different tools are shown in the following tables (TS means the true number of detection and FS means the false number of detection):

TABLE IV. DETECTION RESULTS BY ITS4

Program	TS	FS	LOE	FP(%)	FN(%)
WU-FTPD	32	71	64	68.93	50.00
Net-tools	29	22	50	43.13	42.00
Pure-FTPD	0	71	1	100	100

TABLE V. DETECTION RESULTS BY FLAWFINDER

Program	TS	FS	LOE	FP(%)	FN(%)
WU-FTPD	48	156	64	76.47	25.00
Net-tools	36	79	50	68.70	28.00
Pure-FTPD	0	161	1	100	100

TABLE VI. DETECTION RESULTS BY SPLINT

Program	TS	FS	LOE	FP(%)	FN(%)
WU-FTPD	5	14	64	73.68	92.18
Net-tools	2	6	50	75.00	96.00
Pure-FTPD	0	40	1	100	100

TABLE VII. DETECTION RESULTS BY CQUAL

Program	TS	FS	LOE	FP(%)	FN(%)
WU-FTPD	7	40	64	85.11	89.06
Net-tools	4	11	50	73.33	92.00
Pure-FTPD	0	49	1	100	100

TABLE VIII. DETECTION RESULTS BY COMPREHENSIVE DETECTION

Program	TS	FS	LOE	FP(%)	FN(%)
WU-FTPD	50	97	64	65.99	21.87
Net-tools	42	45	50	51.72	16.00
Pure-FTPD	0	37	1	100	100

From Table Ⅳ-Ⅷ, the comprehensive detection is better than other detection methods. Its false negative rate and false alarm rate are low.

It is noteworthy that the Pure-FTPD has only one security hole, and four tools do not detect the hole. Therefore, the false negative rate and false positive rate of comprehensive detection

are 100%.In comprehensive detection mechanism, Net-tools has higher false positives rate than ITS4, but its false negative rate is significantly reduced. If the comprehensive detection select higher threshold, the rate of false positive and false negative will further decline. It shows that this comprehensive detection method has high efficiency, scalability, human-interactive and strong practicability compared to other methods.

V. CONCLUSION

An effective static comprehensive analytical method for buffer overflow vulnerability detection is proposed in this paper. The method integrates multiple static analysis tools for buffer overflow vulnerabilities detection. In order to reduce the rate of false positives and false negatives effectively, the multiple detection results is synthesized to authenticate and complement the detection results produced by different tools. The future work is integrating different static detection tools for buffer overflow vulnerabilities based on program verification. It should also be feasible in theory.

REFERENCES

[1] Pengfei Gou.Research on detection technology of buffer overflow vulnerability [D].University of Electronic Science and Technology of China，2011.

[2] ZHANG Lin, ZENG Qing-kai. Static Detecting Techniques of Software Security Flaws [J]，Computer Engineering，2008,34(12).

[3] Xia Yiming. Security Vulnerability Detection Study Based on Static Analysis[J]. Computer Science,2006,33(10):279-283.

[4] Kurshan R P. program Verification[J].Notices of the American Mathematical Society,2000,47(5):534-545.

[5] Viega J, Bloch J T. ITS4: A Static Vulnerability Scanner for C and C++ Code[C]//Proc. of Annual Computer Security Applications Conference.[S.I.]:IEEE Computer Society Press,2000.

[6] Flawfinder[EB/OL]. (2006-05-01). http://www.dwheeler.com/flawfinder.

[7] XU Lu-lu，ZHANG Li-ping，GUO Yue. Hierarchical Static Test Method in Code Analysis [J].JISUANJI YU XIANDAIHUA，2013，217（9）：58-65.

[8] Davide Pozza, Riccardo Sisto, Luca Durante, et al. “Comparing Lexical Analysis Tool for Buffer Overflow Detection in Network Software” [C]. The first Int. Conf. on Communication System Software and Middleware, New Delhi, India. January 2006,IEEE.

[9] Li Xiaonan.Design and implementation of software security vulnerability static detection platform based on data comprehensive analysis [D]. University of Electronic Science and Technology of China，2011.

[10] KONG Deguang,ZHENG Quan,CHEN Chao,etal.Source Code Static Analysis Technology for Vulnerability Detection Based on Data Fusion [J].Journal of Chinese Computer Systems，2008,29（6）：1109-1112.