Buffer Overflow Vulnerability Detection Based on Unsafe Function Invocation

Xianda Zhao*, Shuguang Huang, Zulie Pan and Huang Hui
College of Electronic Engineering, National University of Defense Technology, Hefei 230031, China

*Corresponding author e-mail: zmkm_xd@163.com

Abstract. Buffer overflow vulnerabilities are widespread in software programs and pose a serious security threat. In order to effectively mitigate buffer vulnerabilities, we proposed a buffer overflow vulnerability detection technique based on unsafe function invocation. By extracting the typical characteristics of the insecure function call operation, a vulnerability code characteristic model is constructed, and the model is used as a guide to use the data flow analysis method for vulnerability detection. Experimental results show that our method can effectively detect buffer overflow problems caused by unsafe function invocation in programs.

1. Introduction
The continuous change and wide spread of computer technology make the software industry develop with each passing day. Software plays an increasingly important role in the in-depth development of national economy, national defense construction, energy industry, medical and health care, and is inseparable from people's life [1]. With the continuous development of diversified user demands, the software system becomes more and more complex. At the same time, due to the incomplete security considerations of the software architecture, software vulnerabilities also increase, posing a serious security threat to the economic and social development [2].

In recent years, buffer overflow vulnerability presents a high incidence, which is not optimistic [3,4]. According to the latest release of CWE [5], the first 25 most dangerous types of software errors in 2019 can be seen, the type of improper operation restrictions within the memory buffer range (cwe-119) ranks the first. We make further statistical analysis on the development trend of the number of security vulnerabilities and the proportion of buffer overflow vulnerabilities in CVE [6] over the years from 1999 to 2018, as is shown in figure 1. It can be seen that in the past 20 years, the number of vulnerabilities has increased year by year on the whole, and the number of vulnerabilities has increased sharply in recent years. At the same time, the proportion of buffer overflow vulnerabilities is growing rapidly.
At present, there are various binary program vulnerability detection technologies, which can be divided into static analysis [7] and dynamic analysis [8] from the perspective of software operation. Dynamic analysis technology completes the vulnerability detection process through the actual running program. It can obtain the relevant information of the actual running time of the program and has a good accuracy. Static analysis technology can directly detect the target program without providing the execution environment required for the program to run. With high path coverage, it can build control flow and other information in a relatively comprehensive way without incurring runtime overhead. And it has the ability to find program defects that cannot be detected by dynamic analysis technology. Therefore, we adopt static analysis method to detect buffer overflow vulnerability caused by unsafe functions.

A major cause of the buffer overflow problem is that some of the string-manipulation functions and printing functions in the C library functions, such as strcpy(), strcat(), sprintf(), and gets(), can easily cause buffer overflow vulnerabilities if the boundary checking code is not included when the program calls them. We adopt static analysis method to detect buffer overflow caused by unsafe function invocation. First, the typical characteristics of unsafe function call operations are extracted and the representation characteristics of vulnerability code are described by formal language. Then, a buffer overflow vulnerability detection method based on unsafe function call was proposed. The parameters of unsafe function were tracked through analysis, and whether the parameters could be controlled during the detection process was analyzed. Finally, the effectiveness of the method is proved by experiments.

2. Unsafe function invocation feature model

2.1. Unsafe Function Call Feature Extraction

The lack of security checks on buffer boundaries in programs is the main cause of buffer overflow problems. According to the library function call of unsafe functions, based on the existing research results and the analysis of a large number of examples, the characteristics of overflow vulnerability caused by the use of such functions are summarized, and the characteristics of vulnerability are extracted as follows:

• Call the data copy class unsafe function. As a direct cause of buffer overflow problems, unsafe function invocation and irregular operations are important factors.
• Buffer copy does not check input size. Not checking the length of the source string and the length of the destination buffer before calling the string copy class function can easily lead to buffer overflow problems.

• The source buffer data is not constant. According to the analysis of actual vulnerability characteristics, in the process of data copy, if the data in the source buffer is a constant string and is not controlled by the user, the possibility of overflow is very small. On the contrary, for example, if the data of the source buffer is passed through the parameters of other functions and is extremely large, it is easy to cause buffer overflow. In the detection process, the accuracy and pertinence of vulnerability detection can be improved according to this condition.

• Copy length is not constant. In the actual buffer copy, if the copy length is constant and not controlled by the user, the possibility of overflow is very small. Therefore, when using such functions, the value of the function parameter that contains the copy length should be checked to further prevent buffer overflow problems. In the detection process, it is also important to check the copy length controlled by the user.

2.2. Unsafe Function Call Feature Extraction

Through the analysis of the expression form of buffer overflow type provided by CWE and a large number of examples of buffer overflow vulnerability programs, it can be seen that the unsafe function call process that causes buffer overflow vulnerability has relatively deep code expression characteristics. Abstract the generic characteristics of vulnerability code reasonably, construct the structure or attribute feature model of potential vulnerability code, and formally describe the extracted characteristics of vulnerability code, which can deepen the understanding of the connotation of vulnerability causes, and then carry out vulnerability detection process with the guidance of vulnerability feature model.

We represent the buffer overflow vulnerability caused by unsafe function call as a five-tuple composed of the unsafe function \(\text{func}\), the corresponding parameter \(\text{parm}\) of the function, the function operation \(\text{op}\), whether the parameter length compares \(\text{check}\) and \(\text{cond}\), as is shown below.

\[
\text{UNSAFE\_FUNC\_TO\_BOF} < \text{func, parm, op, check, cond} >
\]

\[
\text{func} \in \text{BUF\_FUNC}
\]

\[
\text{parm} \in \left\{ \begin{array}{l}
\{\text{src\_size, dest\_size, count}\} \\
\{\text{src\_size} \in \text{SIZE, dest\_size} \in \text{SIZE, count} \in \text{SIZE}\}
\end{array}\right.
\]

\[
\text{op} \in \text{BUF\_OP}
\]

\[
\text{check} \in \text{CHECK}
\]

\[
\text{cond} \in \text{BOF\_COND}
\]

\(\text{BUF\_FUNC}\) represents the C language library function that may cause buffer overflow vulnerability; \(\text{SIZE}\) represents the \(\text{SIZE}\) of the function's source buffer and destination buffer, and copy length (if it exists). \(\text{BUFF\_OP}\) represents the string operation completed by the function; \(\text{CHECK}\) is a Boolean type, indicating whether the boundary \(\text{CHECK}\) is carried out during the C language library function call. \(\text{BOF\_COND}\) represents the condition that the source buffer or copy length satisfies to cause a buffer overflow problem under the current \(<\text{func, parm, op, check}>\).

It should be noted that the buffer overflow vulnerability detected by our proposed method generally refers to buffer overflow "defect", and it is not tested whether the defect can be exploited. The code defect causing buffer overflow in this paper is considered as the buffer overflow vulnerability.
According to the characteristics of buffer overflow vulnerability caused by unsafe functions extracted, the determination process of buffer overflow vulnerability caused by unsafe function invocation is defined as follows:

Definition (determination of buffer overflow vulnerability caused by unsafe function call):
Buffer overflow vulnerability is caused by unsafe function call in the program. If and only if the unsafe function call operation meets the condition for vulnerability, that is:

\[ \text{func} \in \text{BUF}_\text{FUNC} \land \text{check} = \text{false} \land \text{cond} \in \text{BOF}_\text{COND} \]  

(2)

It can be seen that unsafe function invocation can be judged as potential buffer overflow holes if the following 3 sufficient conditions are met:
Condition 1: the program invocation the unsafe function and does not check the boundary.
Condition 2: the source buffer can be controlled by the user, that is, it can be controlled by the attacker's input.
Condition 3: copy length is controllable by the user, that is, it can be controlled by the attacker's input.

3. Unsafe function feature detection
According to the unsafe function call characteristic model, the unsafe function call detection process is shown in figure 2.
• Search and match the location of the call unsafe function in the locator program (function address starting position).
• Trace the source and destination buffers of unsafe functions, and determine the size and location of the source and destination buffers through a traceback program.
• Determine whether the source buffer data is external controllable to the user, and provide a basis for the next detection process.
• According to the characteristics of buffer overflow vulnerability defined based on unsafe function, it is determined whether buffer overflow vulnerability will occur.

![Figure 2. Unsafe function call feature detection process.](image-url)
4. Experiment
The experimental environment is configured as follows: Intel(R) Core(TM) i5-2400 CPU @ 3.10ghz, 8G memory, and Windows 7 64-bit operating system. The CVE-2017-11882 [9] public vulnerability program was selected as the test case. CVE-2017-11882 is a typical vulnerability caused by unsafe function invocation. The vulnerability is caused by a stack buffer overflow when the formula editor component (eqnedt32.exe) did not check the length of the formula font name (Font Name data) while reading ole data containing MathType. The experimental results show that 26 potential vulnerabilities have been found, among which 3 are real ones, which verifies the effectiveness of our method.

5. Conclusion
In this paper, the buffer overflow problem caused by unsafe function invocation is detected, and the validity of the method is verified by experiments. This method USES static data flow analysis technology, and the next step can be combined with dynamic analysis technology to improve the effectiveness of vulnerability detection.

References
[1] Cousot P, Giacobazzi R, Ranzato F. Program Analysis Is Harder Than Verification: A Computability Perspective [C] // International Conference on Computer Aided Verification. 2018.
[2] Ji T, Wu Y, Wang C, et al. The Coming Era of AlphaHacking?: A Survey of Automatic Software Vulnerability Detection, Exploitation and Patching Techniques [C] //2018 IEEE Third International Conference on Data Science in Cyberspace (DSC). IEEE, 2018: 53-60.
[3] Ren J, Zheng Z, Liu Q, et al. A Buffer Overflow Prediction Approach Based on Software Metrics and Machine Learning [J]. Security and Communication Networks, 2019, 2019.
[4] Shao SH, Gao Q, Ma S, Duan FY, Ma X, Zhang SK, Hu JH. Progress in research on buffer overflow vulnerability analysis technologies. Ruan Jian Xue Bao/Journal of Software, 2018, 29 (5): 1179–1198 (in Chinese). http://www.jos.org.cn/1000-9825/5504.htm
[5] 2019 CWE Top 25 Most Dangerous Software Errors [EB/OL]. [2019-09-18]. https://cwe.mitre.org/top25/archive/2019/2019_cwe_top25.html.
[6] CVE [EB/OL]. [2019–07–16]. https://cve.mitre.org/cve/.
[7] Zhang J, Zhang C, Xuan JF, Xiong YF, Wang QX, Li B, Li L, Dou WS, Chen ZB, Chen LQ, Cai Y. Recent progress in program analysis. Ruan Jian Xue Bao/Journal of Software, 2019, 30 (1): 80-109 (in Chinese). http://www.jos.org.cn/ 1000-9825/5651.htm.
[8] Andriesse D, Chen X, Veen V D, et al. An In-Depth Analysis of Disassembly on Full-Scale x86/x64 Binaries [C] // USENIX Security Symposium. 2016.
[9] CVE-2017-11882|Microsoft Office Memory Corruption Vulnerability [EB/OL]. [2017–11–29] https://portal.msrc.microsoft.com/en-US/security-guidance/advisory/CVE-2017-11882.