A STUDY ON THE WEAKNESS ANALYSIS FOR BINARY CODE IN EMBEDDED ENVIRONMENTS

YUNSIK SON, JUNHOJEONG, SEMAN OH

1Dep. of Brain and Cognitive Engineering, Korea University, Seoul, Rep. of Korea
2Electronic Commerce Institute, Dongguk University, Gyongju, Rep. of Korea
3Dep. of Computer Science and Engineering, Dongguk University, Seoul, Rep. of Korea
E-mail: ‘sonbug@korea.ac.kr, ‘yanyenli@dongguk.edu, ‘smoh@dongguk.edu

Abstract- Today's software is developed by collaborating with various levels of people rather than being developed by superior individuals. Also, recent software development is increasingly being developed using open source and third party libraries. On the other hand, research shows that when using third party libraries, security analysis and testing are less than 50% in the development process. For those reasons, security incidents caused by the use of such third-party libraries are continuously increasing, such as HeartBleed, ShellShock, POODLE, and DROWN, but security of binary libraries is very difficult to verify. In this paper, we analyze major security weaknesses based on the major CVEs generated in the embedded SW environment and use them for security weakness analyzer and intermediate language design for effective security weakness analysis of binary code in the future.

Index Terms- Binary Code, Binary exploitation, Embedded Environments, Weakness Analysis, Vulnerability Analysis.

I. INTRODUCTION

Today's software embraces various types of security weaknesses from vulnerable developers, or already developed libraries, as well as security weaknesses intended by malicious developers. If this software does not address the security vulnerabilities, this vulnerability could develop into a serious software problem.

For this reason, a systematic and detailed development methodology that removes security weaknesses according to security weakness classification and analysis such as CWE/SANS TOP 25 and CERT Secure Coding in SDLC (Software Development Life Cycle) is applied[1,2]. Therefore, general security weakness analysis is a way to eliminate security weaknesses beforehand, in source code mainly by inputting the source code. Therefore, there is a need for a way to analyze security weaknesses against binary code.

Among them, embedded SW is an important security environment such as medical, vehicle, and weapon system. Therefore, it is more important because security vulnerability can cause serious problems directly related to human life.

Security weakness analysis based on binary code was performed through reverse engineering, but due to the inefficiency of implementation method, static analysis method based on intermediate language recently appeared[7-10]. These techniques fall into two categories: converting binary code using an intermediate language used by the compiler, or converting binary code through a new intermediate language design.

When designing such a new intermediate language, it should be designed considering the main security weakness of the object to be analyzed.

Therefore, this paper categorizes and analyzes major security weaknesses that originate from the security vulnerabilities that may exist in the code for the security weakness analyzer and the intermediate language design for the effective security weakness analysis of the embedded SW environment binary code.

In section 2 introduces security weaknesses and vulnerabilities, and in section 3 introduces key analysis results for embedded platform-specific CVEs. In section 4 introduces major security weaknesses that can be exist in embedded binary environment code. Finally, we summarize the important points to be considered for an effective security weakness analyzer and an intermediate language design.
II. SECURITY WEAKNESS AND VULNERABILITY

Software security weakness is a vulnerability that should be removed when developing software, and it is a potential security vulnerability that could cause cyber attacks such as software defects and errors. The meaning of a security vulnerability is that it is already a problem in software operation. Security weaknesses do not immediately become security vulnerabilities, but security vulnerabilities can arise from security weaknesses. Therefore, it is possible to classify security weaknesses that cause security vulnerabilities in a specific environment and to identify the main threats from security vulnerabilities.

To systematically manage these vulnerabilities, MITER is writing CVE (Common Vulnerabilities Exposures)[11]. CVE is a standardized list of security vulnerabilities and other information security exposures that are created and managed to facilitate data sharing through security vulnerability databases and security tools. Therefore, in order to accomplish this, each security vulnerability is assigned a unique ID. When describing a certain vulnerability, it can be easily conveyed by simply expressing it as CVE ID rather than a long description, and it is easy to search for vulnerabilities.

Although the types of security vulnerabilities handled by CVE are classified into 13 types as shown in Fig. 1, some of them are vulnerabilities specific to web service environment, in this paper, we analyze CVE in embedded environment to analyze security weakness in embedded SW environment.

IV. A CVE ANALYSIS BY PLATFORM

Unlike general PC system or server system, embedded system is used for medical, vehicle, and weapon systems, if there is a problem in the system, it can be caused by a problem that is directly related to the human life. In this paper, we analyze the characteristics of vulnerabilities occurring in the embedded system environment and classify the main security weaknesses that have generated the vulnerabilities. Therefore, based on the CVE items related to the embedded SW, CVEs generated from VxWorks, Windows CE, Linux Kernel, and Linux Kernel RT as main embedded platforms are analyzed and summarized in Table 1.

Table 1. The number of reported vulnerabilities by Platform

<table>
<thead>
<tr>
<th>Platform</th>
<th>Denial of Service</th>
<th>Execute Code</th>
<th>Overflow</th>
<th>Directory Traversal</th>
<th>Bypass Something</th>
<th>Gain Information</th>
<th>Gain Privilege</th>
<th>Memory Corruption</th>
</tr>
</thead>
<tbody>
<tr>
<td>VxWorks</td>
<td>7</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Windows CE</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Linux Kernel</td>
<td>906</td>
<td>66</td>
<td>255</td>
<td>2</td>
<td>90</td>
<td>234</td>
<td>199</td>
<td>83</td>
</tr>
<tr>
<td>Linux Kernel RT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Of the 13 vulnerability types, the most common vulnerabilities that can occur in embedded SW could be classified into DoS, Execute Code, Overflow, Memory Corruption, Directory Traversal, Bypass Something, Gain Information, and Gain Privilege[12]. Among them, Execute Code, DoS and Overflow vulnerabilities are more vulnerable to the accumulated vulnerabilities than the most frequently occurring XSS vulnerabilities in web services. Especially, Execute Code is twice as many as XSS. In addition, since these three vulnerabilities are very serious in terms of damage, it is very important to analyze the security weaknesses that generate the vulnerabilities in embedded SW.

Fig. 1. The number of the CVE reporting by Type of Vulnerabilities

Of the 13 vulnerability types, the most common vulnerabilities that can occur in embedded SW could be classified into DoS, Execute Code, Overflow, Memory Corruption, Directory Traversal, Bypass Something, Gain Information, and Gain Privilege. Among them, Execute Code, DoS and Overflow vulnerabilities are more vulnerable to the accumulated vulnerabilities than the most frequently occurring XSS vulnerabilities in web services. Especially, Execute Code is twice as many as XSS. In addition, since these three vulnerabilities are very serious in terms of damage, it is very important to analyze the security weaknesses that generate the vulnerabilities in embedded SW.
They proposed a method that can detect the buffer overflow pain phenomenon. If overflow problem occurs, execution flow is determined, but direct attack is difficult, and corner case could exist. Wang et al. proposed a method to detect this phenomenon. They proposed to convert X86 binary file into PANDA intermediate language through the IntScopeand check the problem using Symbolic Execution and Taint Analysis method[9]. Finally, Null Pointer Dereference is a memory error that occurs when a pointer is NULL and the program is executed using that pointer. Zhang et al. have used static analysis to detect the use of invalid pointers in binary files[10]. After converting the binary file into an intermediate language, they have presented a formalized method and algorithm that uses pattern analysis of existing security weaknesses. They developed a prototype tool to add-on IDA Pro [9] to apply to the actual binary file and analyzed the performance of the tool.

Recently, studies on the analysis of binary code security weakness mostly use a method of converting binary code into an intermediate language. Although there are advantages that it is not hardware dependent using intermediate languages, many existing intermediate languages such as REIL and LLVM have been used[14,15]. However, it is unclear whether these languages can be used effectively for certain specific security weaknesses. Therefore, it is important to consider the characteristics of specific security weaknesses and to design an intermediate language that can be effectively used for analysis.

V. THE ANALYSIS OF THE MAJOR SECURITY WEAKNESS IN EMBEDDED ENVIRONMENTS

This section describes major security vulnerabilities and introduces detection methods. Buffer overflow is one of the most common security weaknesses and is a common security problem for stack and heap buffers. This problem is often caused by using vulnerable functions to execute commands without checking the size of the buffer in the source code. In order to detect buffer overflows in binary code, Rawat et al. uses a reverse engineering framework called BinNavi[7]. The code in the initial binary state is converted into assembly code using IDA Pro and converted into intermediate language using REIL[13,14]. After that, they checked the size between the source buffer and the destination buffer using the call graph and control flow graph provided in REIL and detect the buffer overflow using a formal buffer overflow pattern. They proposed a tool called BinaryReviser that is not only detects the buffer overflow by directly analyzing the binary file, but also provides the ability to remove the vulnerability through code patching.

Use After Free is the most common security weakness to date, this occurs because the memory is used after releasing the memory. Recently, Feist et al. have analyzed the security weakness by converting a binary file into a REIL-based intermediate language [8]. In this study, GUEB (Graph of Use after Exploit Binary) tool is proposed and analyzed to see if there is a dereference part in allocating and releasing the heap area by visualizing the control flow graph.

Integer type variable overflow is not usually used for direct attack but it uses integer type variable to determine execution path in branch or loop inside program. If overflow problem occurs, execution flow prediction is difficult, and corner case could exist. Wang et al. proposed a method to detect this phenomenon. They proposed to convert X86 binary file into PANDA intermediate language through the IntScopeand check the problem using Symbolic Execution and Taint Analysis method[9].

CONCLUSION

Through this study, we were able to identify vulnerabilities such as DoS, Code Execution and Overflow that are mainly occurring in the embedded SW system. We also analyzed the security weaknesses that cause these vulnerabilities. A lot of studies analyzing security weaknesses from binary code have shown that they use intermediate languages, as a result, we have seen the need for an intermediate language design that is effective in analyzing specific security weaknesses.

Therefore, we will design an effective security weakness analyzer and intermediate language for binary code considering security weaknesses that occur in the embedded SW environment.

REFERENCES

A Study On The Weakness Analysis For Binary Code In Embedded Environments


[12] https://www.cvedetails.com


★★★