Process Analysis of Buffer Overflow Based on Dynamic Binary Platform

Kang Fei, Dong Pengcheng, Shu Hui, Sun Jing
China National Digital Switching System Engineering and Technological Research Center
Zhengzhou, China, 450002

***Abstract*—Based on dynamic binary platform, an analysis method for buffer overflow is described in detail and a prototype system is implemented. Overflow detection based on exception capture, control flow analysis, and memory status checks is implemented according to the principle of buffer overflow exploits. By monitoring memory read and write instructions, control transfer instructions, system obtains call sequences and data transfer flow. Through analysis of memory data and function calls, we locate code lead to overflow. Experimental results show that the system can detect overflow, and accurately position code segment which results in buffer overflow. The prototype system is of important value in efficient analysis of 0day and rapid patching software.**

Keywords-Buffer Overflow, Dynamic Binary Platform, Dynamic Detection,Overflow Positioning, Control Flow Analysis

I. INTRODUCTION

Network attacks carried out by buffer overflow has been a serious threat to information security. Researchers have carried out in-depth study for detection of buffer overflow, and have proposed a variety of detection methods.

Buffer overflow process analysis is composed of detection of overflow, location of the code results in overflow. The methods of overflow analysis include manual analysis and automated analysis method. Analysts can debug overflow exploits by setting breakpoints. But it's inefficient, and requires experience. Also shellcode may use anti-debugging strategy. It's subject to great restrictions. Automated detection technique includes detection of the source code, the compiler checks and dynamic taint tracking. Compiler checks method have been proposed, such as StackGuard, GS and SafeSEH etc. StackGuard protect return address by adding Canary, SafeSEH saves legitimate exception handling. Compiler checks can be easily bypassed by exploit technologies. Meanwhile, detection technology based on compiler checks is designed to defend buffer overflow while program is running, therefore, analysts are not able to obtain better support for the process analysis.

Overflow detection based on dynamic taint tracking is the research focus in recent years. The development of dynamic binary platform provides a strong technical support for overflow detection. DynamoRIO [1], Valgrind [2], Pin[3] have gradually developed. To determine overflow, it tracks input data and analysis the relationship between input data and program execution. Several prototype systems, such as TaintCheck [4], Argos [5] and VirtualVAE [6] have been implmented. For tracking the input data, it results in greater performance overhead. The methods of determine input data is to capture network data packets, therefore these systems are primarily for network software, lacking of support for local software.

This paper presents dynamic buffer overflow process analysis method based on DynamoRIO. By tracking the execution flow of the program in the assembly instruction-level, we analyse control flow instructions, catch the exception event, and check the memory status to detect the buffer overflow. Based on the results of the detection stage, we monitor the data flow, memory read and write instructions and control transfer instructions to get sequence of function calls and the process of data dissemination. By analyzing the relationship between memory, data and function calls, we can locate code which result in buffer overflow.

Compared with other methods, the prototype system has the following advantages. First, overflow detection is realized by control flow instructions monitoring, exception capturing and memory status checking, which avoids performance overhead due to track a large amount of input data, and is of well support for the local software; second, the positioning of the reasons for overflow is implemented at two levels of monitoring the system library function calls and memory read and write instructions.

II. DETECTION AND LOCATION MECHANISM

A. DynamoRIO

DynamoRIO is a dynamic binary platform for both Windows and Linux, which works between application programs and operating system. It has full control over the target program. The target program is divided into basic blocks. Basic block is a sequence of instructions which ended with control transfer instruction (CTI). CTI includes the return, jump or function call instructions, etc. DynamoRIO copies a basic block to the code cache each time, DynamoRIO instrument instructions by adding, modifying, or transforming logic to instruction within each basic block. It also has well performance of supporting for large scale software with high efficiency.

B. Detection of Buffer Overflow

Stack overflow is caused by not properly checking the data length while writing data into stack memory. To exploit stack overflow, you can overwrite a function return address in stack or make use of SEH mechanism. Return address is pushed onto the stack frame by CALL.

Published by Atlantis Press, Paris, France.

Ret pops the return address to the EIP register when function returns. Therefore, we can instrument on the CALL and RET to compare the return address to detect buffer overflow.

Buffer overflow attacks Based on SEH, usually covers the exception registration structure, and triggers the abnormally to excute shellcode. Thus, to detect SEH-based overflow attacks, we can follow steps as below:

Step 1: Capture the error occurred time.

We can get the time the exception occurred by instrumenting Windows API KiUserExceptionDispatcher, which is the starting point in user mode.

Step2: Detection of overflow

The most important structure in overflow exploit is exception registration structure.

Algorithm 1：Detection of Buffer Overflow

```
Input: BasicBlock
Output: TRUE || FALSE
Variables:
Addrcall: return address
ExceptionOccur: exception event occurs
HeapManageCall: call heap manage function event
SEHChainIntegrityCheck(): Check the integrity of SEH Handle Chain
1. Begin:
2       Foreach instr ∈ BasicBlock
3        switch instr
4                Case CALL:
5.
        InsertAddrList(Addrcall );
6.               Case RET:
7.                       If(Addrret ==Addrcall)
8.
        DeleteAddrList(Addrcall );
9.                       Else {
10.
        CrashAddr = GetCrashAddr();
11.                                      Return FALSE; }
12.      hEvent = EventCapture();
13.   switch hEvent
14.              Case ExceptionOccur:
15.     SEHChainIntegrityCheck( );
16.              Case HeapManageCall:
17.              entry = GetFirstEntry( );
18.              While( !( entry->flags && 0x10 ) ) {
19.              If( entry->Flink->Blink!= entry || entry->Blink->Flink!=entry ){
20.
        CrashAddr = GetCrashAddr( );
21.                                      Return FALSE; }
22.                       entry = entry->Blink;
23.     End
```

Figure 1. Detection of Buffer Overflow

In most of the heap overflow exploits, the critical step is to overwrite the heap block management structure, such as Flink or Blink. Windows uses two-way linked list Freelist or one-way linked list Lookaside to manage free heap block. Heap overflow usually occurs in the heap allocation release or merge. Attacker will get the opportunity to access to any address in memory to write data. And the functions related with allocation and release include malloc / free, C + + new / delete and the WIN32 API function HeapAlloc / HeapFree, etc. But ultimately it will call ntdll.dll's RtlAllocateHeap / RtlFreeHeap to complete the operation. In accordance with the above analysis, overflow detection process is shown in Figure 1.

C. *Implementation of Buffer Overflow Location*

Buffer overflow may be triggered while processing a string or memory copy. Based on the detection result, we track execution and data flow. Through tracing library function call sequence and instruction execution flow, we position code cause overflow in the levels of library function and the assembly instructions.

In the library function calls level, we analyze dynamic link library which is loaded while program is running to access the export function, and added them to the information database of library functions, and track the execution flow to monitor library function call process to get parameters and build library function call sequence diagrams.

In assembly level, we trace memory read and write instructions. For both memory read and write instructions, we instrument to obtain the source address and destination address, and analyze the data in source area. We can establish data transfer diagram, combined with library function call flow, we are able to give a data dissemination process, and position code which leads to overflow.We implment overflow positioning strategy as below.

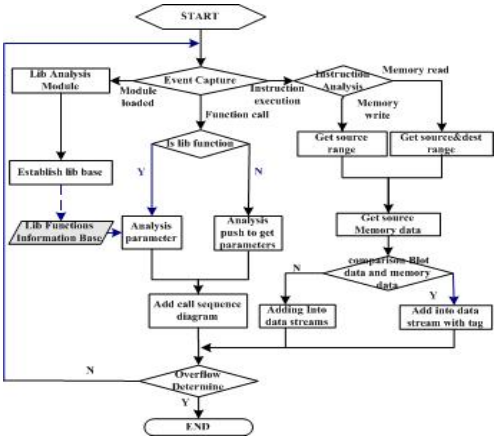

Figure 2. Location of Buffer Overflow

III. SYSTEM IMPLEMENTATION FRAMEWORK

Based on the above method, we implement the prototype system. Figure3 shows the overall framework. System is composed of overflow detection module and positioning module. DynamoRIO provide environment to monitor target program. The input is the process to be inspected, and output includes overflow type, occurred point, and the code leads to overflow.

Overflow detection module determines the occurrence of overflow, which is achieved in two steps: event capture and overflow determine. Event capture instruments at the

possible timing of overflow. We instrument to capture call,ret, exception event, and heap management functions; and checks data list related with overflow.

Positioning module consists of three parts. Dll analysis module analyzes the dynamic link library loaded, and stores the information of the library function into information database in a certain format.

Library functions call analysis module instruments to get parameters when library function is called. Memory instruction analysis module instruments to get read and write memory instrumentions, and analyzes related address range in memory. By analyzing the data memory address range, we determine the existence of taint data. If tainted data is found, we mark the code segment.

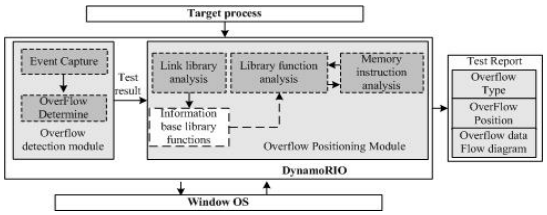

Figure 3. System Framework

IV. Data Analysis and Performance Evaluation

A. Data Analysis

To test the validity of the system, we test multiple overflow examples including Adobe Reader, Office, WinRar and IE, and achieved good results. We analyze CVE-2010-0188 which is an Adobe Reader exploit in detail.

Overflow detection results are shown in figure 4. It shows that there is a stack overflow, which is triggered in ACROFORM.dll, at 0x20cb2c66, The original return address 0x20cb4210 is covered by 0x070072f7

```
C:\Program Files\Adobe\Reader 9.0\Reader\plug_ins\AcroForm.api
TAG 0x20CB2C66
+0  1.3 33 c0      xor  eax, eax
+2  1.3 c9         leave
+3  1.3 c3         ret
END 0x20CB2C66
ESP: 0x0012d6fc ADDR: 0x20CB4210 NADDR: 0x070072f7
```

Figure 4. Overflow Detection Result

Positioning results are shown in figure 5. When program arrived at 0x209d3151 in AcroForm.dll, it calls memcpy for copying data without checking length. Finally, memory write instruction rep movs covered return address, which is pushed into stack by call instruction at 0x20cb420b.

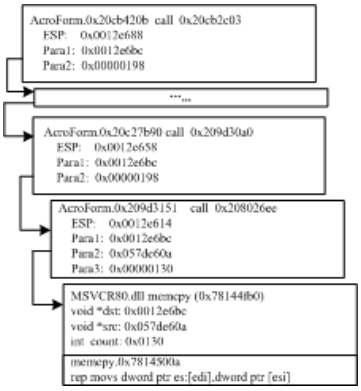

Figure 5. Function Call Sequence Diagram

B. Performance Evaluation

Table I shows part of the test results. The results show that there are no false positives.

We evaluate performance using exploit examples, including Office2003, AdobeReader9.1 and WinRar3.6. The environment is XP SP3, Intel Pentium M, 2.0GHz, 2G memory. Reference time is the execution time of the overflow

TABLE I. Part of The Test Results

CVE	software	Test result		Conformity
		type	module	
CVE-2006-3086	Excel 2003	stack	Hlink.dll	√
CVE-2007-2193	ACDSee9.0	stack	ID_X.apl	√
CVE-2007-1922	Winamp5.5	heap	IN-MOD.dll	√
CVE-2010-0805	IE 6.0	Stack	TDC.DLL	√

samples. The results shows that performance cost is 5-12 times in overflow detection module, and 11-22 times in Positioning

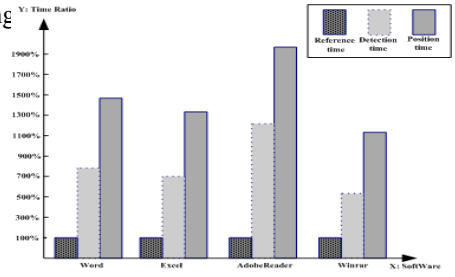

Figure 6. Performance Evaluation

V. Summaries

The results show that the prototype system can effectively detect buffer overflow and accurately analyze the overflow reason. The system has an important role in efficient analysis of vulnerability and fast patching software. The system performance need to be further improved, shellcode detection and extraction is also an important aspect

in the analysis process of overflow. We will focus on shellcode detection, and further optimize system performance.

REFERENCES

[1] D L Bruening. Efficient, Transparent, and Comprehensive Runtime Code Manipulation[D]. USA: Massachusetts Institute of Technology, 2004.

[2] Nicholas Nethercote. Dynamic binary analysis and instrumentation[D]. University of Cambridge, 2004.

[3] Chi-Keung Luk, Robert Cohn, Robert Muth, et al. Pin: building customized program analysis tools with dynamic instrumentation[C]. New York, USA: Proceedings of the 2005 ACM SIGPLAN conference on Programming language design and implementation, 2005: 190-200.

[4] J. Newsome, D. Song. Dynamic Taint Analysis for Automatic Detection, Analysis, and Signature Generation of Exploits on Commodity Software[C]. San Diego, USA: Proc. of the12th Annual Network and Distributed System Security Symp. 2005

[5] Georgios Portokalidis, Asia Slowinska, Herbert Bos. Argos: an Emulator for Fingerprinting Zero-Day Attacks[C]. Leuven, Belgium: EuroSys'06. 2006:15-28.

[6] Wang Chunlei, Wen Van, Dai Yiqi. A Software Vulnerability Analysis Environment Based on Virtualization Technology[C]. Beijing: IEEE International Conference on Wireless Communications, Networking and Information Security (WCNIS2010). 2010: 620 – 624