Self Debugging Mode for Patch-Independent Nullification of Unknown Remote Process Infection

Ruo Ando and Yoshiyasu Takefuji

Keio University, Graduate School of Media and Governance,
Endo 5322, Fujisawa, 2528520 Japan
{Ruo, Takefuji}@sfc.keio.ac.jp
http://www.neuro.sfc.keio.ac.jp

Abstract. The rapid increase of software vulnerabilities shows us the limitation of patch-dependent countermeasures for malicious code. We propose a patch-independent protection technique of remote infection which enables each process to identify itself with "being infected" and nullify itself spontaneously. Our system is operating system independent and therefore does not need software rebuilding. Previously, no method for stopping malicious process without recompiling source code or rebuilding software has been proposed. In proposal system, target process is running under self debugging mode which is activated by enhancing debug() exception handler and utilizing MSR debug register. In this paper we show the effectiveness of proposal method by protecting the remote process infection without patching security holes. Implemention of device driver call back function and BranchIP recorder provides the real-time prevention of unregistered worm attack through Internet. In experiment, function test of stack buffer overflow of Win32.SQLExp.Worm is presented. Also CPU utilization corresponding to the number of calling function and some database operations is showed.

Keywords: self-debugging mode, real-time nullification, debug register, improved debug exception handler, branchIP recorder.

1 Introduction

The rapid increase of software vulnerabilities and its exploitation imposes a great burden on network administrators and client users. Recent cyber attacks, worms and virsuses become more sophisticated. Some obfuscation and avoidance techniques which evade network traffic inspection such is called polymorphic and metamorphic coding. These techniques is now applied in malicious code writing.Win32.Evol and Simile virus is valid example which shows us the limitation of naive signature based inspection of network traffic. The rapid spread of Win32.SQLExp.Worm infection also shows the limitation of patch-based coutermeasures for new attack. Previously, instead of signature matching, the adaptive prevention technique has been proposed. Some technique inspect the target file

Y.G. Desmedt et al. (Eds.): CANS 2005, LNCS 3810, pp. 85–95, 2005.

by executing it both on run-time environment and on virtual machine emulation. Another run-time protection is mainly represented by compiler solutions and operating system based method. Openwall Linux kernel patch project is to improve the protection against buffer overflows. OpenBSD also provides the new feature of stack protection technology against buffer overflows which is embedded in the system compiler. Stack-smash protection compilers are used against buffer overflows including the GCC extensions, libsafe, propolice, stackguard, libmib, and MS .net compiler. However, their disadvantage lies in that kernels and software components must be rebuilt.

In this paper we propose the method for the real-time infected process nullfication using improved debug exception handler. This could be described as automated debugging based on improved loader, driver-supplied callback function and debug exception handler. On this system attribute of self-debugging is added to the target process, which makes it possible to control by itself when attacked and infected without the file scanner or IDS.

Figure1 illustrates the concept of automated debug using improved debug exception handler. In conventional debug method, the system needs to launch debugging process and API. The target process can be executed in the memory of debugging process. In proposal method, process does not need their debugging process and API because exception handler is improved so that the debug specified for malicious code is automated in running each process.

In this paper, the proposal system is constructed on 80x86 processor. The 80x86 processor has 20 different exception handlers. Table 1 shows the exception handlers mainly concerned with debugging issues. In this paper we present improved exception handler. Particularly, we enhance debug() function with sig-

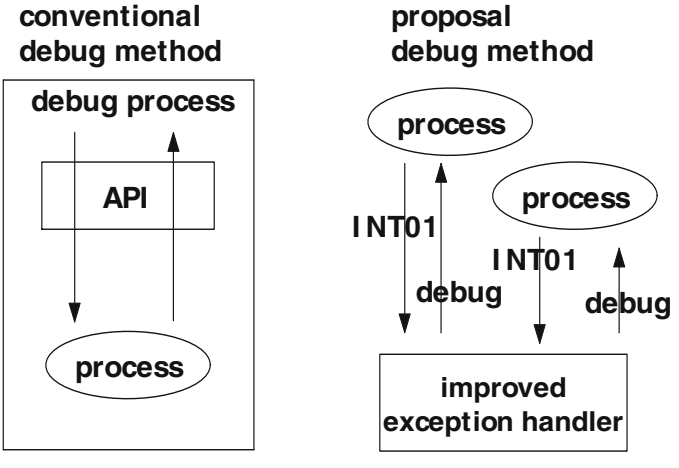

Fig. 1. Automated debugging. By improving exception handler INT01H, the self-debugging attribute is added to the target process. In proposal method, process under inspection does not need debugging process and its memory, which provides sense of infection and self-defense.

Table 1. Comparison of proposal system with previous method

No	Exception	Exception Handler	Signal
1	Debug	debug()	SIGTRAP
2	NMI	nmi()	None
3	Breakpoint	int3()	SIGTRAP
4	Overflow	overflow()	SIGSEGV
5	BoundsCheck	bound()	SIGSEGV
12	StackException	stack_segment()	SIGBUS
14	Page Fault	page_fault()	SIGSEGV

nal SIGTRAP. Debug facilty is called when we set the T flag of eflags or when the address of an instruction fall the range of an active debug register. The concept of proposal method is automated debugging by the implementation of some additional process within the debug() function. To achieve the concept of proposal method, IDT (interrupt descriptor table) must be initialized and overwritten for dealing with the bugs hooked by INT01 insertion. In other words, to enable exception, the kernel should initialize the IDT properly. The correspondence between interrupt or exception vector and the address of each recognized interrupt or exception handler is stored to IDT.

2 Self Debugging Mode

In this section we discuss the self debugging mode added to the target process. Figure2 illustrated the activation of self debugging attribute to the target process. The proposal method is divided into steps. First, we insert the INT01H

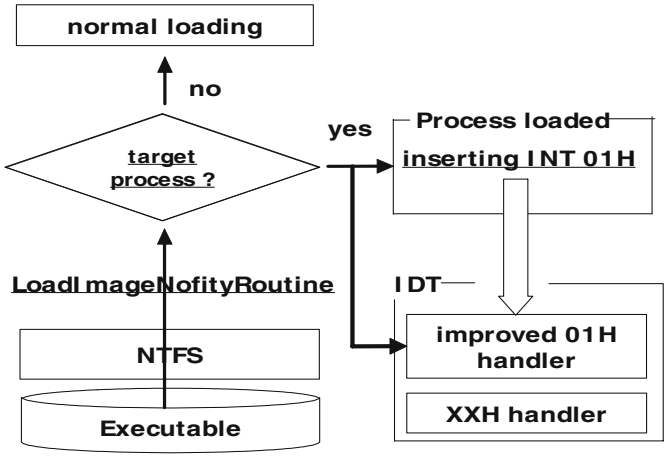

Fig. 2. Changing the self debug mode of the target process. When the executable file is loaded, INT01H debug point is inserted and improved exception handler is registerd to IDT. These steps begin the automated debug mode applying loader processing module such as LoadImageNoiftyRoutine and bin_fmt.

debug instruction into the process like binary translation technique. Second, the enhanced exception handler which is specified for the INT01H software bug hooking is registered to IDT. These steps are executed when the executable is loaded. In detail, we apply LoadImageNotifyRoutine in Win32 or loader processing module such as bin_fmt in linux in order to insert INT01H break point and register the improved exception handler. Regardless of the kind of opearting system, exception handler is speficied for bugs that causes exploitation. Now that we completed this manipulation, the new attribute described as self-debugging is added to the target process loaded. This mode makes it possible for the process to be nullified by itself if it is infected by unregistered malicious code. The detailed facilities of proposal system are discussed in the following.

2.1 Driver-Supplied Callback Function

To implement the proposal concept, we selected driver-based callback function, which is notified whenever an image is loaded for execution. Driver-based callback function is utilized for the identification of loading the target process. Highest-level system profiling drivers can call PsSetImageNotifyRoutine to set up their load-image notify routines. This could be declared as follows, particularly in Win32.

void LoadImageNotifyRoutine (
PUNICODE_STRING FullImageName,
HANDLE ProcessId,
PIMAGE_INFO ImageInfo);

Once the driver's callback has been registered, the operating system calls the callback function whenever an executable image is mapped into virtual memory. When the LoadImageNotifyRoutine is called, the input FullImageName points to a buffered Unicode identifying the executable image file. The argument of list showing handle identities of process has been mapped when we call this function. But this handle is zero if the newly loading image is a driver. If FullImageName, which is input of LoadImageNotifyRoutine matches the name of target process, we go on to call the improved exception handler.

2.2 Debug Register

IA-32 processors provides MSR(model specific registers) for the purpose of recording taken brunches, interrupts and exception. Figure 3 shows the allocation in debugCtlMSR register of Intel P6 family processors. In this paper we focus on last branch interrupt / exceptions flag to save and search the EIP(32 bit instructional pointer). EIP means return address. The most recent taken branches, interrupts and exception are stored in the last branch record stack MSRs. The branch records inform us of branch-FROM and branch-TO instruction address. Concerning F6 family processor, the five kinds of MSR, debugCtlMSR, LastBranchToIP, LastBranchFromIP, LastExceptionToIP and LastExecutionFromIP and available. It is possible to set break points on branches, interrupts and exception and execute single step debugging through these registers. These registers

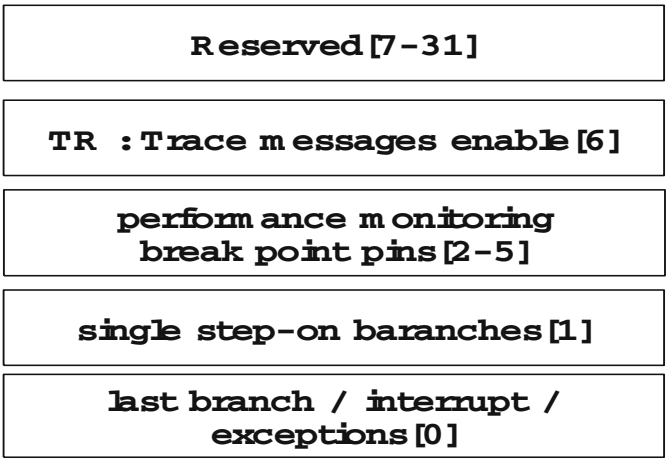

Fig. 3. Allocation in debugCtlMSR register. When the LBR is set, the proceesor records the source and target address at the time the branch instruction or interrupt is taken before the debug exception being generated.

can be used to collect last branch records, to set breakpoints on branches, interrrupts, exceptions and to single step from on branch to the next.

3 Experiments

3.1 Win32.SQLExp.Worm

Recent security incident for the mission critial servers connected to networks has been occurred at very high rates. In mission critical operation, server should be running all the time under many accesses. Once the vulnerability and its exploitation is found, this can be availeble for attackers all over the world. Particularly, the SQL slammer worms, emerged in 2003, caused more then 90 of vulnerable servers all over the world to be infected within a few hours. This worm attack exploits the vulnerability within the vulnerable SQL server. The vulnerable SQL server when it is infected it will send 376 bytes packets via UDP port 1434 to the original attack launcher and other random destinations to propagate the worms. According to the mailing list of snort (open-source instruction detection system), the signature of SQL slammer is as follows at 09:45:52 Sat Jan 25 2003.

0x00b0 89e5 5168 2e64 6c6c 6865 6c33 3268 6b65
..Qh.dllhel32hke
0x00c0 726e 5168 6f75 6e74 6869 636b 4368 4765
rnQhounthickChGe
0x00d0 7454 66b9 6c6c 5168 3332 2e64 6877 7332
tTf.llQh32.dhws2

0x00e0 5f66 b965 7451 6873 6f63 6b66 b974 6f51
_f.etQhsockf.toQ
0x00f0 6873 656e 64be 1810 ae42 8d45 d450 ff16
hsend....B.E.P..

After a while(half a day), the signature is changed as follows:

81 f1 03 01 04 9b xor ecx, 9B040103h
81 f1 01 01 01 01 xor ecx, 1010101h
51 push ecx
9B040103 xor 1010101 = 9A050002 = port 1434 -¿AF_INET

However,this signature describes very specific features of implementation, sending packets to port 1434. Actually, when we implemented and tested the same kind of exploitation by VC++, the binary code is different from the signature above.

This worm exploits the buffer overflow vulnerabilities of MSDE SQL server 2000 unpatched with PORT TCP1433 and UDP1434. In this case, the systematic prevention of buffer overflow should be the first priority. against exploitation of SQL slammer.

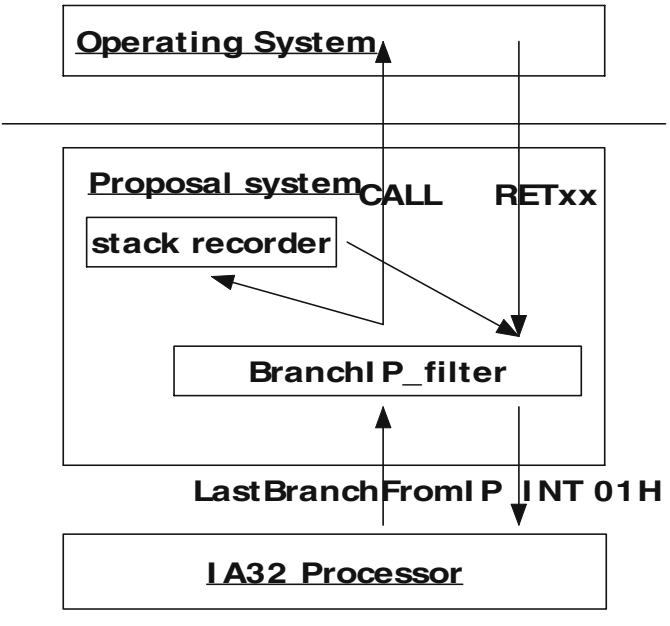

Fig. 4. BranchIP recorder. When the automated debug mode is enabled and the mode of the target process is changed to self debug, the system executes inspection function whenever the Branch instruction is called. BranchIP recoder hook the CALL/RET instrcution and store the saved EIP to stack recorder to check whether the bufferoverflow attack is occurred. The saved EIP is obtained from LastBranchFromIP.

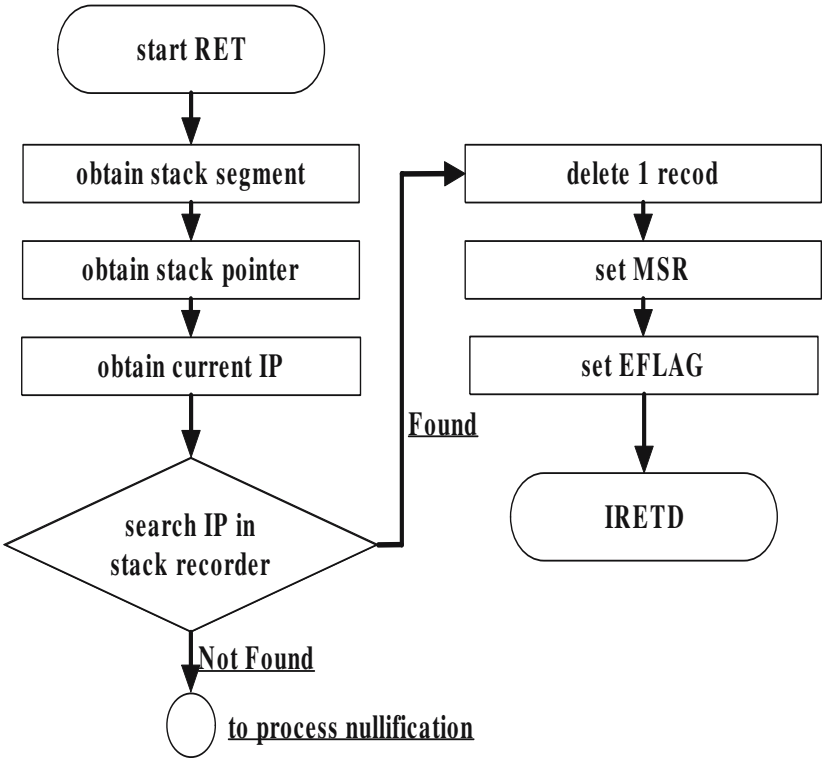

Fig. 5. Flow chart of BranchIP recorder. The self debug mode process checks whether EIP changes after executing local function by using stack recorder. In nested calling function, some saved EIPs are stored to the stack recorder. The proposal method checks the overwriting return address by searching EIP in stack recorder. When the saved EIP is found, this module is terminated in normal mode.

We implemented the Branch IP recorder for inspecting the overwriting EIP after executing function occuring buffer overflow. Figure 4 illustrates the structure of Branch IP recorder. With the memory called stack recorder, we check the transition of EIP while executing function occurring buffer overflow. In experiment, we prototype this system on IA-32 processor because IA-32 family enhanced facilities for debugging are available about inspecting code execution and processor performance. The LastBranchToIP and LastBranchFromIP MSRs are 32-bit registers for recording the instruction pointers for the last branch, interrupt, or exception that the processor took prior to a debug exception being generated. In our system, the inspecting the transition of EIP is possible by monitoring LastBrachFromIP. Branch IP recorder has a FIFO memory for EIP as follows. The inspection flow of saved EIP using Branch IP recorder is shown in Figure5.Although this recording facility has FIFO structure, we can detect the overwriting EIP that is not done in FIFO flow. because proposal system check all stored EIPs when the branch insturction is executed. Experimental result

of table 2-5 shows that our system is effective regrdless of the kinds of buffer overflow signature as long as the target process is running on self debug mode. We succeeded to stop SQL slammer infection on Windows 2000 SP0, without patching security hole of MSDE.

3.2 Performance Measurements

The performance measurements were collected on a Windows 2000 host computer system using Pentium III 1000 MHz with 1024 RAM. Concerning experimental result 5.2, We measured the utilization of MS SQL server 2000.

Table 2 lists CPU utilization corresponding to the number of calling function. We vary the number of executing local function from 100 to 10000. From 100 to 100, utilization is not changed rapidply, comparetively stable about less than 10%. From 1000 to 10000, it was showed that utilzation is not increased linearly. It is showed that the proposal system is effective in the point that utilization is less than 25 % when the function is executed 10000 times without the linear increase.

Table 3 lists CPU utilization of SQL server processing SELECT queries. We vary the number of READ queries from 5 to 30. CPU utilization is 9.14, 17.5 and 47.85, almost doubled corresponding to the number of queries. The performance of proposal system is not reasonable compared with the lower column of unprotected case. Although the utilization differences are caused partly by the

Table 2. CPU utilization corresponding to number of times of calling local funcation(%)

times of calling function	CPU utilization
100	8.75
200	9.12
500	9.6
700	9.93
1000	10.33
5000	14.25
7000	17.31
10000	24.89

Table 3. CPU utilization (%) according to the number of SELECT queries

the number of queries	30	10	5
proposal system enabled	1.48	1.21	0.97
proposal system disabled	1.28	1.08	0.89

Table 4. CPU utilization (%) according to the length of SELECT query

data length(byte)	500	200	50
proposal system enabled	1.89	1.61	1.39
proposal system disabled	1.71	1.37	1.18

Table 5. Comparison of proposal system with previous method

-	rebuild	prevention	utilization
StackGuard	O	O	mid
Bounds Checking	O	O	high
OpenWall	X	O	low
proposal system	X	O	high

implementation of INSERT command, the complementary system is considered below.

Table 4 lists CPU utilization of SQL server processing SELECT queries. We vary the length of READ queries from 50byte to 500byte. The increase of CPU utilization is insignificant in changing the data length while the proposal system is sensitive to the number of query operation as shown in table. This result is caused by the fact that the proposal system inspects the transition of instructional pointer stored in 32bit register (LastBranchFromIP) while in the conventional scanning methods the length of signature changes according to the various kinds of payload of malicious code. In other words, proposal system only requires inspection the value of constant 32bit length, which makes it possible to keep utilization reasonably low.

4 Conclusion

In this paper, we introduce the automated debug technique, utilizing the facility of debug and instruction trace in processor level for real-time malicious process nullification. The conventional anti-virus softwares are all based on stored signatures. Consequently these schemes have the limitation against the unknown exploit occuring buffer overflow and the unregistered attack such as metamorphic and polymorphic viruses and worms Instead of modifying the sophisticated operating system for reducing vulnerabilities, we work out the new attributes for target process called self-debugging. When the process is translated into memory, this mode is activated by enhanced debug() exception handler. For the implementation of proposal system, driver supplied callback function is utilized for the event-driven insertion of self debug facility. The each process which is INT0H break point embeddedi is running on self debugging mode where software bugs is controlled by improved debug() exception handler. In proposal system, we can prevent of the exploitation of every kind of software bugs for which we can write the controlling of exception of INT01H handler. Without rebuilding applications and kernel, the system loading automated debugging technology can contorl infected process to identify its infection and nullify by itself. In experiment, CPU utilization of detecting buffer overflow, CPU time corresponding the number of calling function and some operation of SQL database server was measured and evaluated. Function test of stack buffer overflow of Win32.SQLExp.Worm is also presented.

Table 5 shows the comparison of proposal method and another adaptive protection techniques. The disadvantage of stack guard and bounds checking lies in that kernels and software components must be rebuilt. Concerning OpenWall, the kernel must be rebuild while the application with overflow vulnerability need not recompiling. The proposal system takes advantage in the point that it does not need rebuilding both kernel and application. Our method has also flexibility for all kind of vulnerabilities such stack overflow, heap overflow, race condition and so on as long as we can describe the property of software bugs in debug() exception handler. The proposal system is experimented in MS SQL vulnerability in 2003 and some database operations. The proposal scheme using a new concept of sense of self is based on the automated debugging mode where the execution of malicious code is nullified by autonomous control of target process. The proposed scheme does not need the software-rebuilding, while the existing schemes need the software-rebuilding. Finally, the concept of self debug mode is operating system independent.

Acknowledgement

We are indebted to K.Shoji, T.Kawade and T.Nozaki, by courtesy of Sciencepark Cooporation. Some idea in this paper grew out of the ongoing collaboration with their team.

References

1. Crispin Cowan, Perry Wagle, Calton Pu, Steve Beattie, and Jonathan Walpole: Buffer Overflows - Attacks and Defenses for the Vulnerability of the Decade, DARPA Information Survivability Conference and Expo, 2000.
2. Roesch, M:Snort - lightweight intrusion detection for networks. Proceedings of Thirteenth Systems Administration Conference (LISA '99), pp. 229-238,1999.
3. Symantec Corporation: Bloodhound Technology. http://securityresponse.symantec.com/
4. Gene H. Kim and Eugene H. Spafford:Tripwire A File System Integrity Checker, ACM Conference on Computer and Communications Security , pp. 18-29,1994.
5. Kosoresow, Andrew P. and Steven A. Hofmeyr, "Intrusion Detection Via System Call Traces", IEEE Software,pp 35-40,1997.
6. Zeshan Ghory:Openwall Improving security with the openwall patch ,securityfocus,2002.
7. Linux Openwall project. http://www.openwall.com/
8. David Larochelle and David Evans,Statically Detecting Likely Buffer Overflow Vulnerabilities,2001 USENIX Security Symposium, Washington, D.C., August 13-17, 2001.
9. C.Cowan, C.Pu, D.Maier, J.Walpole, P.Bakke, S.Beattie, A.Grier, P.Wagle, Q.Zhang, and H.Hinton:StackGuard Automatic adaptive detection and prevention of buffer-overflow attacks, In Proc. 7th USENIX Security Conference, pp 63–78,1998.
10. Baratloo, A., N. Singh and T. Tsai, Libsafe: Protecting critical elements of stacks, http://www.research.avayalabs.com/project/libsafe/.

11. J. Bergeron, M. Debbabi, J. Desharnais, M. M. Erhioui, Y. Lavoie and N. Tawbi, "Static Detection of Malicious Code in Executable Programs". Proc. of the International Symposium on Requirements Engineering for Information Security, 2001.
12. Richard W M Jones,Paul H J Kelly: Backwards-compatible bounds checking for array and pointers in C programs, AADEBUG97,1997.
13. Intel Corporation: IA-32 IntelR Architecture Software Developer's Manual, Volume 2A: Insruction Set Reference A-M,2004.
14. Intel Corporation: IA-32 IntelR Architecture Software Developer's Manual, Volume 2B: Insruction Set Reference N-Z,2004.
15. Intel Corporation: IA-32 IntelR Architecture Software Developer's Manual, Volume 3: System Programming Guide,2004.