Averting Buffer Overflow Attack in Networking OS using – BOAT Controller

Vadivel Murugan.P
Research Scholar
Department of Computer Center
Madurai Kamaraj University

M.Alagarsamy
Associate Professor
Department of Computer Center
Madurai Kamaraj University

Abstract:
Today’s Networking Operating System Server software is under continuous scrutiny and attack, whether for fun or for profit. Networking OS Software vulnerabilities that allow the injection and execution of malicious code in persistent Internet connected systems pose serious threats to system security. In a common type of attack, an inimical party induces a software buffer overflow in a prone to the computing devices in order to corrupt a procedure to return addresses and transfer controls to the malicious code. This buffer overflow attacks are often engaged to recruit unaware hosts into Distributed Denial of Service (DDoS) attack networks, which ultimately promote overwhelming buffer overflow attack against victim networks or machines. In spite of current security software countermeasures that they seek to prevent buffer overflow exploits, many systems are in remain vulnerable. The BOAT controller tool is detect and prevent the buffer overflow in networking OS and strongly control the task utilization.

Keywords: Buffer overflows attack, BOAT controller tool, Memory allocation, Stack based buffer overflow, Heap based buffer overflow

I INTRODUCTION

1.1 Background analysis and Problem Statement

A buffer overflow attack, it's an easy to make Waffle File puts it; buffer overflows are the source of “most stealthy data dependent bugs". Until now buffer overflows are more than just a source of thwarting for programmers. Certainly, they can create serious security holes, leading vulnerabilities which can be exploited to achieve a Denial of Service or in some cases, to get a gain access and increased privileges on systems.

The Internet Worm, program code written and released by Robert T. Morris in 1988, deployed the first known buffer overflow attack. The Worm, which infected thousands of computer systems on the Internet, exploited a buffer overflow attack.

Even before the Worm's release, buffer overflow attacks may have known in some circles there is unreliable evidence of buffer overflow attacks dating back to the 1960’s [3]. Even so, prior to 1997, buffer overflow vulnerabilities were only occasionally discovered and never in such numbers as to attract too much attention. In recent years, however, buffer overflow attack have become the most commonly discovered class of vulnerability, and unfortunately, the most widely exploited. In fact of the twenty forth (24) of the forty forth (44) CERT advisories issued since January 1, 1997, have involved some types of buffer overflow attack [10]. Buffer overflow vulnerabilities represent a security problem of computer users, whether they are a programmers, system administrators or users. The Code Red worm creates rigorous problems that buffer overflow vulnerabilities still cause today. Code Red and its variants, which pricked companies over 2001, took advantage of a buffer overflow problem in Microsoft windows 2000, Internet Information Server. The total economic cost of these worms was assessed at $2.6 billion by Computer Economics [13].

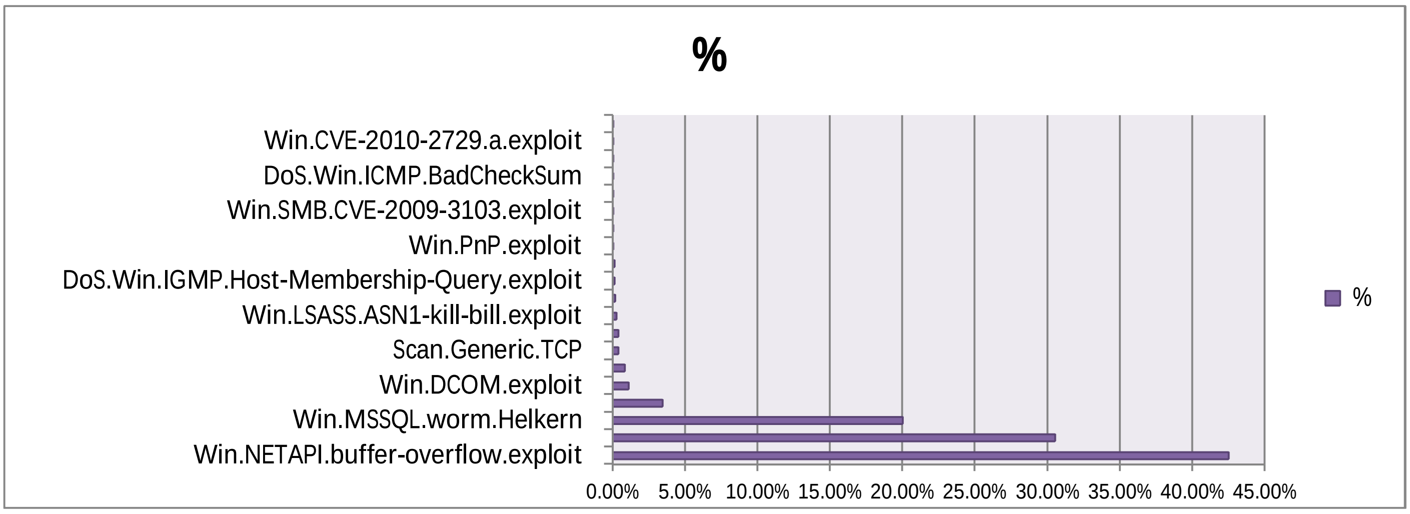
According to US-CERT [4], there were tens of new Buffer overflow attacks appeared each month in 2008. A control-hijacking buffer overflow attack is launched through the overwriting control sensitive data (such as return address, function pointers, GOT entries and the jump buffer) with a new address, called a deviation address, to transfer the execution flow of a program into the code injected are taken by attackers. Stack smashing Buffer overflow attacks and return-into-libc [1], [17], are two most common control-hijacking buffer overflow attack types, especially the former [14].

II BUFFER OVERFLOW ATTACK

2.1 Buffer Overflow Attack methods

The analysis of intrusions in this work concerns a subset of all violations of security policies that would constitute a security intrusion according to definitions in, for example, the Internet Security Glossary [16]. In our context and intrusion or a successful attack aims to change the flow of control, letting the attacker execute arbitrary code. It consider as class of vulnerabilities the worst possible since “arbitrary code” often means starting a new shell. This shell will have the similar access rights to the system as the process attacked. If the process had root access, so the attacker in the new shell, leaving the whole system opens for any kind of manipulation.

Figure 1: In 2010, the IDS system logged 1,311,156,130 number of network attacks. That number was just 220 million in 2009.



The Top 20 malicious programs responsible for network attacks
Table 2: These statistics are based on the detection verdicts of the web antivirus module, and were provided by users of Kaspersky Lab products who gave their consent to transfer their statistical data [11].

Ranking	**Name of the attack**	**No. of unique attacks***	**%**
1	Win.NETAPI.**buffer-overflow.exploit**	55,71,26,500	42.49%
2	DoS.Generic.SYNFlood	40,04,91,518	30.54%
3	Win.MSSQL.worm.Helkern	26,24,43,478	20.02%
4	Scan.Generic.UDP	4,53,43,780	3.46%
5	Win.DCOM.exploit	1,41,34,307	1.08%
6	Generic.TCP.Flags.Bad.Combine.attack	1,06,31,023	0.81%
7	Scan.Generic.TCP	52,38,178	0.40%
8	Win.LSASS.exploit	50,89,038	0.39%
9	Win.LSASS.ASN1-kill-bill.exploit	32,56,429	0.25%
10	DoS.Generic.ICMPFlood	23,41,724	0.18%
11	DoS.Win.IGMP.Host-Membership-Query.exploit	16,41,578	0.13%
12	Win.HTTPD.GET.buffer-overflow.exploit	13,99,613	0.11%
13	Win.PnP.exploit	5,79,249	0.04%
14	Win.EasyAddressWebServer.format-string.exploit	3,84,278	0.03%
15	Win.SMB.CVE-2009-3103.exploit	2,36,122	0.02%
16	Win.WINS.heap-overflow.exploit	1,90,272	0.01%
17	DoS.Win.ICMP.BadCheckSum	1,01,063	0.01%
18	Generic.FTPD.format-string.attack	98,239	0.01%
19	Win.CVE-2010-2729.a.exploit	71,671	0.01%
20	Win.MSFP2000SE.exploit	44,674	0.00%

2.2 Memory Allocation-Stack Based Buffer Overflows

The stack is an area of reserved virtual memory used by applications software. It is an operating system's method of allocating memory. The programmer is not required to give any special instructions in code to expand the memory. However, the operating system performs this task thro guard pages automatically. The given code would store the character array "var" on the stack.

Example:

char var[ ]="Some string stored on the stack memory";

The stack operates in similar to a memory. The information is always pushed onto (added) and popped off (removed) from the top position of the stack. The stack is a method of Last In First Out (LIFO) data structure. Pushing a data item onto a stack causes the current top of the stack to be decremented by four bytes (4B) before the item is placed on the stack. When any information is added to the stack, all the previous data is moved downwards and the new data sits at the top of the stack. Multiple bytes of data can be popped or pushed onto the stack at any given time. Since the current top of the stack is decremented before pushing any item on top of the stack, the stack grows downwards in memory.

A stack frame is a data structure format that is created during the entry into a subroutine procedure (in the terms of C /C++, it's the creation of a functions). The objective of the stack frame is to keep the parameters of the base procedure as is and to pass arguments to the subroutine procedures. The current location of the stack pointer can be accessed at any given time by accessing the stack pointer register (ESP).The current base of a function can be accessed by using the EBP register which is called the Base Pointer or Frame Pointer and the current location of executions can be accessed by accessing the instruction pointer register (EIP).

2.3 Memory Allocation-Heap Based Buffer Overflows

Heap, similar to stack, is a region of virtual memory used by the applications. In every application software has a default heap space. However, different stack, private heap space can be created by programmers via special instructions such as "new ()" or "malloc ()"and freed by using "delete ()" or "free ()". Heap procedures are called when

an application doesn't know the sizes of (or the number of) objects needed in advance, or when an object is very large to fit onto the stack.

Example:
OBJECT * var = NULL;
var = malloc (sizeof (OBJECT));

The operating system Heap Manager operates above the Memory Manager and is responsible for providing functions which allocates or deallocates portions of memory. Every application starts out with a default of 1MB (0x100000) of reserved heap size (view output from dumpbin that follows) and 4kb (0x1000) committed if the table does not indicate the allocation size. Heap grows over time and it doesn't have to be contiguous in memory.

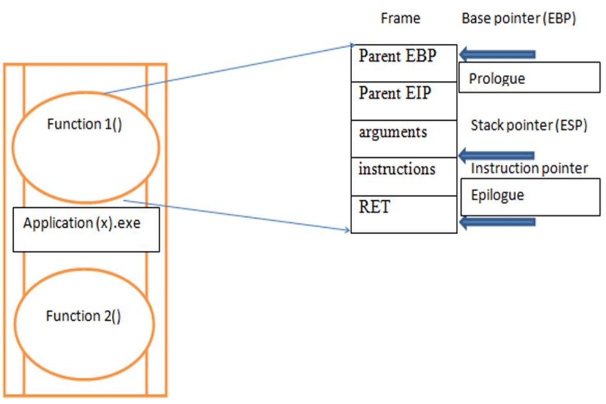


Figure 2: High-Level Memory Layout

2.4 Heap layout

2 Bytes	Size of this block / 8	Control block
2 Bytes	Size of the previous block / 8	
4 Bytes	Flags (8 bit/byte)	
4 Bytes (EAX)	DATA if in use, else previous free block pointer	
4 Bytes(ECX)	DATA if in use, else next free block pointer	

Figure 3: Windows Frame Layout

```
C:\WINDOWS\system32>dumpbin /headers kernel32.dll
<Deleted for brevity>
100000 size of heap reserve (1 MB)
1000 size of heap commit (4k)
<Deleted for brevity>
```

2.5 Heap Structure

Each heap block starts and maintains a data structure to keep track of the memory blocks that are free and the ones that are in use (see Figure 3). Heap allocation has a minimum size of eight bytes, and an additional overhead of eight bytes (heap control block). The heap control block among other things also contains pointers to the next free block. As and when the memory is freed or allocated, these pointers are updated.

III PROPOSED METHODS

3.1 Detecting Techniques

The systemic approach: the programming technology and programmer mistakes or cracker that opens the door for repeated attacks on critical infrastructure software.

It has examined the characteristics of numerous buffer overflow attacks, the main reasons for their attractiveness, and the effectiveness and costs of several defenses against them. Until recently the attackers seemed to have the traditional defenses appeared largely impotent to stop these attacks. The recent appearances of effective defenses that break some of these difficulties give reason for hopefulness that finally the defenders might have a chance to gain against this type of attack.

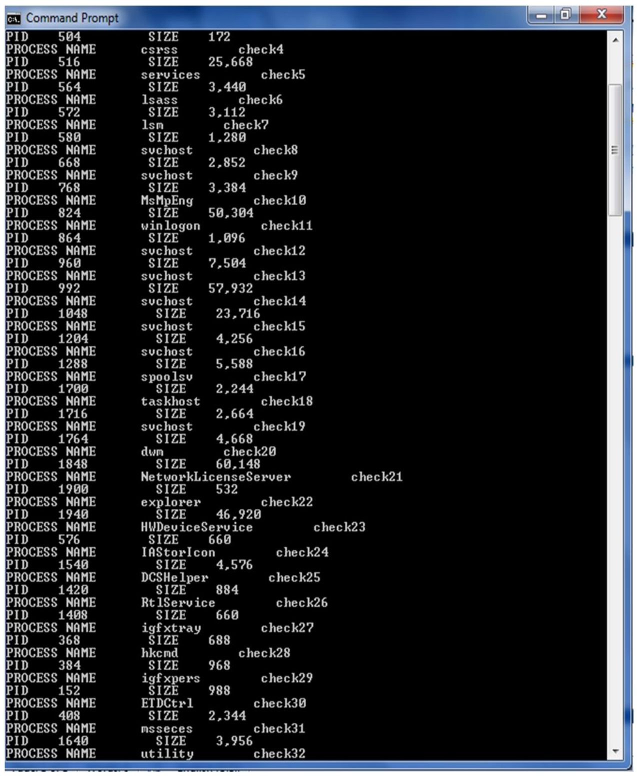
If no tools can resolve absolutely the problem of buffer overflow, but they can reduce the probability of stack smashing attacks. However, code analysis (writing a secure code) is still the best solution to these attacks. If the buffer overflow occurred the BOAT controller tool is helpful to prevent and detect the vulnerability. The application software is control the networking operating system function it takes over the task manager control from the place of *(\Windows\system32\taskmgr.exe).* In operating system the task manager is the application to control the memory allocated by the memory management system.

In task manager the process of several system and application software files are listed in hierarchy, if some application consumes the memory from RAM suddenly, that suspect program is detected by the BOAT controller application software. The application software also do the job of monitoring the process ID, user name and how much KB of memory consumed by the attacking program, every five seconds it refresh the memory and again monitoring the task manager.

IV CONTROL MEASURES

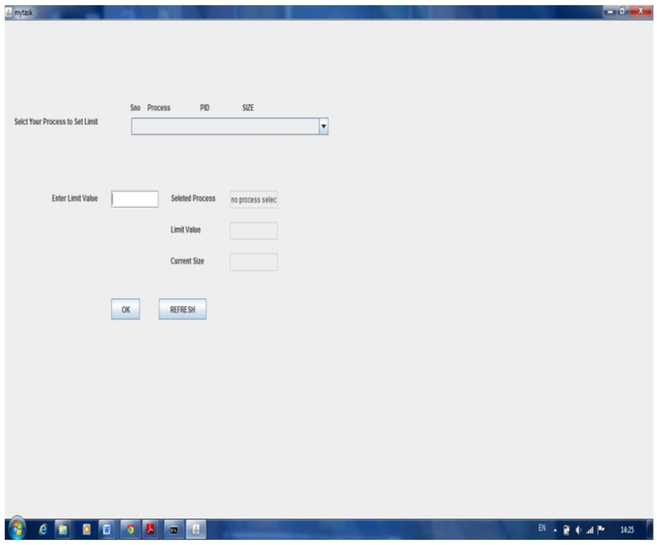
4.1 Buffer Overflow Attack Countermeasures

If the application BOAT controller tool is control the task manager if the memory overrides the program passing an alert to the administrator. If the administrator allocates a memory manually the suspected program may consume the more memory should control by the program. It secures your server from the buffer overflow crash.Select the suspected process with name or process id with memory size, enter the memory limit of value given by the administrator, if the program try to overflow the memory size in particular time the BOAT controller tool control the suspected program.



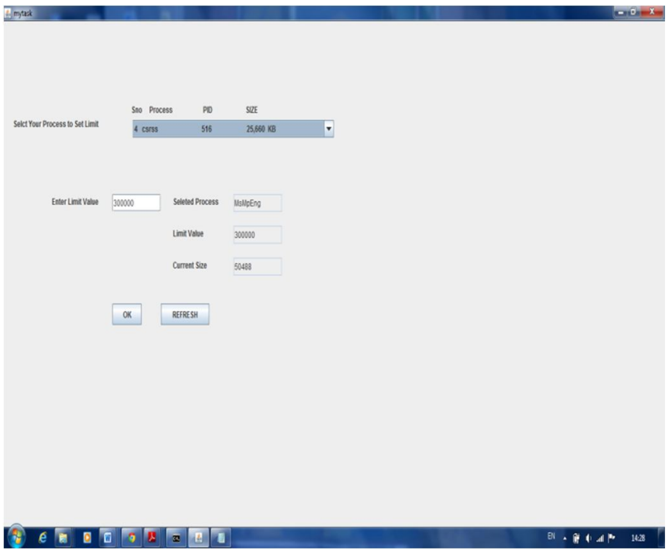
```
PID     504              SIZE    172
PROCESS NAME        csrss           check4
PID     516              SIZE    25,668
PROCESS NAME        services           check5
PID     564              SIZE    3,440
PROCESS NAME        lsass           check6
PID     572              SIZE    3,112
PROCESS NAME        lsm           check7
PID     580              SIZE    1,280
PROCESS NAME        svchost            check8
PID     668              SIZE    2,852
PROCESS NAME        svchost            check9
PID     768              SIZE    3,384
PROCESS NAME        MsMpEng            check10
PID     824              SIZE    50,304
PROCESS NAME        winlogon           check11
PID     864              SIZE    1,096
PROCESS NAME        svchost            check12
PID     960              SIZE    7,504
PROCESS NAME        svchost            check13
PID     992              SIZE    57,932
PROCESS NAME        svchost            check14
PID     1048              SIZE    23,716
PROCESS NAME        svchost            check15
PID     1204              SIZE    4,256
PROCESS NAME        svchost            check16
PID     1288              SIZE    5,588
PROCESS NAME        spoolsv            check17
PID     1700              SIZE    2,244
PROCESS NAME        taskhost           check18
PID     1716              SIZE    2,664
PROCESS NAME        svchost            check19
PID     1764              SIZE    4,668
PROCESS NAME        dwm           check20
PID     1848              SIZE    60,148
PROCESS NAME        NetworkLicenseServer            check21
PID     1900              SIZE    532
PROCESS NAME        explorer           check22
PID     1940              SIZE    46,920
PROCESS NAME        HWDeviceService            check23
PID     576              SIZE    660
PROCESS NAME        IAStorIcon            check24
PID     1540              SIZE    4,576
PROCESS NAME        DCSHelper            check25
PID     1420              SIZE    884
PROCESS NAME        RtlService            check26
PID     1408              SIZE    660
PROCESS NAME        igfxtray           check27
PID     368              SIZE    688
PROCESS NAME        hkcmd           check28
PID     384              SIZE    968
PROCESS NAME        igfxpers           check29
PID     152              SIZE    988
PROCESS NAME        ETDCtrl            check30
PID     408              SIZE    2,344
PROCESS NAME        msseces            check31
PID     1640              SIZE    3,956
PROCESS NAME        utility            check32
```

Figure 4: Running program-task manager is control from the Dos Prompt



Seleted Process: no process selec

Figure 5: The application executed is ready to control the overflow task



Sno Process PID SIZE
4 csrss 516 25,660 KB

Enter Limit Value: 300000
Seleted Process: MsMpEng
Limit Value: 300000
Current Size: 50488

Figure 6: The buffer memory over flow program–selected task will controlled by the program

V CONCLUSION

In this paper, we propose a solution, for a buffer overflow attack the notorious security problem. BOAT controller tool is an accurate adaptive detection mechanism that can recognize and block stack, heap smashing Hence, it can prevent them from damaging the user mode stack from attack. BOAT controller tool is a networking OS-based solution; thus, it does not need to modify the source code of any application programs. BOAT controller tool, prevent the malicious code which consume more memory and secure your system from the vulnerable attack.

VI REFERENCES

1. Aleph One, "Smashing the stack for fun and profit," *Phrack Magazine*, Vol. 7, 1996, http://www.phrack.org/issues.html? issue=49&id=14.,
2. C. Cowan, C. Pu, D. Maier, J. Walpole, P. Bakke, S. Beattie, A. Grier, P. Wagle, Q. Zhang, and H. Hinton. Stack-
Guard: Automatic adaptive detection and prevention of buffer-overflow attacks. In
Proceedings of the 7th USENIX Security Conference, pages 63-78, San Antonio,
Texas, January 1998.
3. Crispin Cowan, Posting to Bugtraq Mailing List, http://geek-girl.com/bugtraq/1999_1/0481.html
4. CERT, http://www.us-cert.gov/.
5. CERT. CERT/CC statistics. http://www.cert.org/stats/cert stats.html, Feb. 2005.
6. C. Cowan. Software security for open-source systems. IEEE Security & Privacy, 1(1):38–45, 2003.
7. D. Larochelle and D. Evans. Statically detecting likely buffer overflow vulnerabilities. In *Proceedings of the 2001 USENIX Security Symposium*, Washington DC, USA, August 2001.
8. E. Rescorla. Is finding security holes a good idea? IEEE Security & Privacy, 3(1):14–19, 2005.
9. H. Etoh. GCC extension for protecting applications from stack-smashing attacks. http://www.trl.ibm.com/projects/security/ssp/, June 2000.
10. http://www.cert.org/advisories.
11. http:// www.securelist.com/en/analysis Kaspersky Security Bulletin 2010 /Statistics, 2010

12. I. Simon. A comparative analysis of methods of defense against buffer overflow attacks. http://www.mcs.csuhayward.edu/˜simon/security/boflo.html, January 2001.

13. J. McCarthy, "Take Two Aspirin, and Patch That System – Now," *SecurityWatch*, August 31, 2001.

14. Li-Han Chen, Fu-Hau Hsu, Cheng-Hsien Huang, Chih-Wen Ou,Chia-Jun Lin And Szu-Chi Liu "A Robust Kernel-Based Solution to Control-Hijacking Buffer Overflow Attacks"Journal Of Information Science And Engineering 27, 869-890 (2011).

15. NIST. ICAT vulnerability statistics.http://icat.nist.gov/icat.cfm?function=statistics, Feb.2005.

16. R. W. Shirey. Request for comments: 2828, Internet security glossary. http://www.faqs.org/rfcs/rfc2828.html, May 2000.

17. SolarDesigner, "Non-executable user stack," http://www.openwall.com/linux.///, J. Pincus and B. Baker, "Beyond stack smashing: recent advances in exploiting buffer overruns," in *Proceedings of IEEE Symposium on Security and Privacy*, 2004,pp. 20-27.

18. T. cker Chiueh and F.-H. Hsu. RAD: A compile-time solution to buffer overflows attacks. In *Proceedings of the 21th International Conference on Distributed Computing Systems (ICDCS)*, Phoenix, Arizona, USA, April 2001.

19. The SANS Institute, "The SANS/FBI Twenty Most Critical Internet Security Vulnerabilities," http://www.sans.org/top20/, October 2002.

20. Vendicator. Stack Shield technical info file v0.7. http: //www.angelfire.com/sk/stackshield/, January 2001.

21. Zitser. Securing software: An evaluation of static source code analyzers. Master's thesis, Massachusetts Institute of Technology, Department of Electrical Engineering and Computer Science, Aug. 2003.