International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 05 Issue: 05 | May-2018 www.irjet.net p-ISSN: 2395-0072

Ideology of Buffer Overflow Exploits

Riha Maheshwari

6th Semester MCA - ISMS

Abstract – Buffer Overflow has been around for a very long time. *Buffer Overflow Attacks are simple and very easy to implement which allows malicious code to execute with the administrator privileges. It is probably the best form of software security vulnerability known in last 10 years. Buffer Overflow attacks are the real concern for Computer systems in today's Internet. A big part of security threats would be removed if the buffer overflow vulnerability could be eliminated. To mitigate buffer overflow vulnerability, it is very important to understand how buffer overflow actually works, the danger they possess to your application and what are the technique that an attacker can use to exploit these vulnerabilities. In this paper, I will mostly focus on Stack – Based Buffer Overflow Attack and SEH Based Buffer Overflow Attack.*

Key Words: Buffer Overflow Exploit, Stack- Based Buffer Overflow Attack, SHE-Based Buffer Overflow Attack.

1.INTRODUCTION

Buffer Overflow Exploits are also referred to as Stack Smashing, which are very common attacks performed from earlier Windows XP age. The exploits are being combined with the malware, resulting in very complex attack and it is even difficult for the antivirus to detect. A buffer is a part of memory which can be allocated to store anything like character, decimals, integers, etc. The buffer-overrun occurs when the amount of data entered into the buffer is more than it was allocated to handle. The extra information gets overflowed and overwrites the other pointers.

Buffer Overflow attacks can be performed in the program using vulnerable programming languages but many of them are prone to these kind of attacks. The extent of such attacks however would be different which would depend on the type of programming language that would be used to write the program. For instance, the code is generally not susceptible to buffer overflow which are written in Perl and JavaScript. However, a program which are vulnerable to buffer overflow attack written in C, C++, Assembly or Fortran could allow the malicious attacker to compromise the full system. The focus on the technique used in Buffer Overflow is mainly on Stack-Based and SHE-Based.

1.1 OVERVIEW

Year	Description
1988	Morris Internet Worm
1995	Stack Overflow exploit for NCSA was published.
1996	Smashing the Stack for Fun and Profit was published.
2001	Code Red Worm
2003	SQL Slammer Worm
2004-2005	Sasser Worm
2008-09	Overflow in Windows RPC
2009-2010	Stuxnet
2010-12	Print Spooler and LNK overflows

Table 1.1. Overview of Buffer Overflow Attacks

2. WINDOWS MEMORY STRUCTURE

The Memory Structure of Windows Operating System have multiple sections that can be broken down in different components. To understand the depth of writing exploits and taking advantage of poor coding, we first need to understand all these components. So let us understand all the components in detail. The following figure shows a basic representation of the Memory Structure / Memory Map of Windows Operating System.

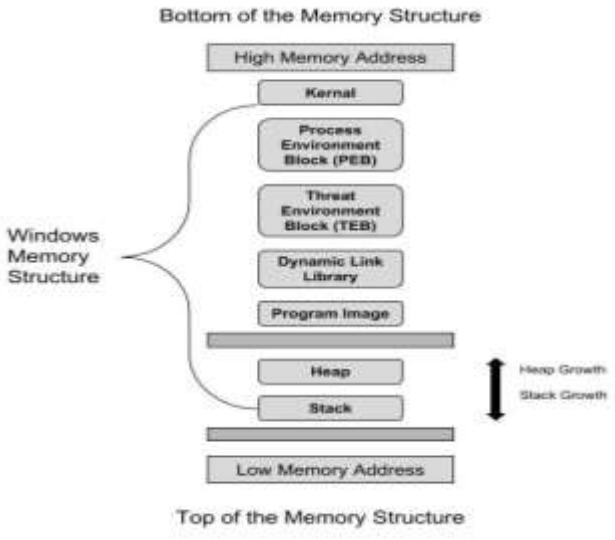

Figure. Windows Memory Structure

2.1 STACK AND HEAP

Stack is a programmable concept known as a Last in First Out (LIFO) structure. Items that are inserted onto the stack are pushed onto it, and items that are run or removed from the stack are popped off of it. We can think of it as a stack of plates or books. To effectively remove books, we have to take them off the top by one or by sets. The stack is used to allocate short-term storage for local variables in an ordered manner and that memory is subsequently freed at the termination of the given function, unlike the heap, where

each process can have multiple threads and memory allocation for global variables is relative arbitrary and persistent. Each function or thread has its own stack frame. That stack frame size is fixed after the creation and at the conclusion of the function the stack frame is deleted.

To better understand the difference between the heap and the stack movement, let's see the below figure, which shows the adjustment as memory is allocated for global and local resources.

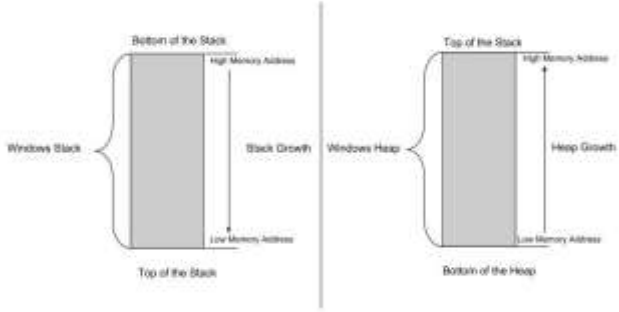

Figure. Stack and Heap

2.2 PROGRAM IMAGE

In a memory Program image is the portion where the executable resides. This includes the sections i.e. .text, .data, .rare which contains the executable code or CPU instructions, contains the program's global data, contains non-executable resources, including icons, images, and strings, respectively. We just need to put the program image where the actual executable is stored in memory. Portable Executable (PE) is the defined format for the executable, which contains the executable and the DLL.

2.3 DYNAMIC LINK LIBRARY

Shared code libraries like Dynamic Link Libraries (DLLs) are mostly used in Windows programs which allows efficient code reuse and memory allocation. These DLLs are also called modules / executable modules which occupies a part of the memory space. DLLs are similar to executables, but they cannot be called directly, and instead they have to be called by an executable. At its core, the idea of DLLs is to provide a method for the capabilities to upgrade without requiring the entire program to be recompiled when OS is updated. This means that even if other components are going to be in different memory locations, many core DLLs will stay in the same referenced locations. Remember, programs require specific callable instructions and many of the foundational DLLs are loaded into the same regions of memory.

2.4 THREAT ENVIRONMENT BLOCK(TEB) & PROCESS ENVIRONMENT BLOCK(PEB)

The Process Environment Block (PEB) is where non kernel components of a running process are stored. Information that is needed by systems that should not have access to kernel components is stored in memory. Some Host Intrusion Prevention Systems (HIPS) monitor activities in this memory region to see if malicious activities are taking place. The PEB contains details related to the loaded DLLs, executables, access restrictions, and so on.

A Thread Environment Block (TEB) is spawned for each thread that a process has established. The first thread is known as the primary thread and each thread after that has its own TEB. Each TEB share the memory allocations of the process that initiated them, but they can execute instructions in a manner that makes task completion more efficient. Since writeable access is required, this environment resides in the non-kernel block of the memory.

2.5 KERNEL

This is the area of memory reserved for device drivers, the Hardware Access Layer (HAL), the cache and other components that programs do not need direct access to. The best way to understand the kernel is that this is the most critical component of the OS. All communication is brokered as necessary through OS features. The attacks we are highlighting here do not depend on a deep understanding of the kernel. Additionally, a deep understanding of the Windows kernel would take a book of its own. After defining the memory locations, we have to understand how data is addressed within it.

3. WORKING OF STACK BASED BUFFER OVERFLOW

Stack Based Buffer Overflow occurs due to the insufficient boundary checks. The attack is possible from where the program accepts the input, can be anywhere. It includes the input fields, command line argument, sockets, files, etc. When detecting Stack Overflows, we should be able to overwrite the Extended Instruction Pointer (EIP).

Stack Based Buffer Overflow occurs due to the insufficient boundary checks. The attack is possible from where the program accepts the input, can be anywhere. It includes the input fields, command line argument, sockets, files, etc. When detecting Stack Overflows, we should be able to overwrite the Extended Instruction Pointer (EIP).

To understand what we are trying to do with the writing of the exploit, we must understand what is happening in memory. We are going to inject data into an area of memory where there was no bound checking. This usually means that a variable was declared a specific size, and when data was copied into that variable there was no verification that the data would fit in it before copying. This means that more data can be placed in a variable than what was intended. When that happens, the excess data spills into the stack and overwrites saved values. One of those saved values includes the EIP.

When calling any function, the Extended Instruction Pointer(EIP) is saved on the stack for later use. When the function returns, this saved address is used to determine the location of the next executed instruction. Thus by

overwriting the EIP, we can add a differ address to point it to our payload. The buffer will be filled with more than the reserved characters, which will allow it to overwrite the EIP successfully.

Thus, we are going to flood the stack with a variety of characters to determine the area we need to overwrite. First, we will start with a large set of A's. The values we see while viewing our debugger data will tell us where on the stack we have landed. The differences in character types will help us better determine what size our unique character test needs to be. Now after getting a general idea of where the EIP is, we can generate a unique pattern with the size of the A's. This unique pattern will be injected back into the vulnerable program. We can then take the unique value that overwrites the EIP register and compare it to our pattern. We determine how far down our large unique pattern that value falls and determine that is how much data is needed be pushed onto the stack to reach the EIP.

In simple the following is a pictorial diagram that will explain how the stack overflow works.

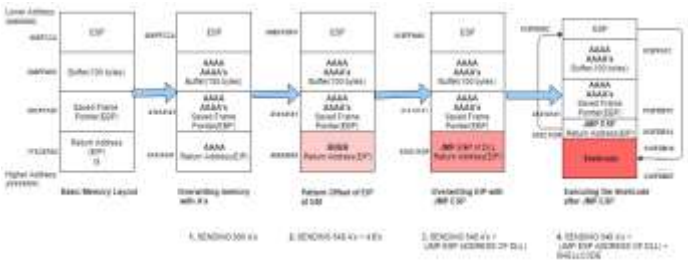

Figure. Pictorial View of working of Buffer overflow in memory

Once we have identified where the EIP is, we can locate the instruction we want to reference in the EIP by examining the DLLs. Remember, DLLs that are a part of the program itself will be more portable, and your exploit will work in more than one version of Windows. Windows OS DLLs make writing exploits easier, because they are omnipresent and have the required instructions you are looking for.

In this version of the exploit, we are trying to Jump to the ESP as the available space is there, and it is easy to build an exploit to take advantage of it. If we were using one of the other registers, we would have to look for an instruction to jump to that register. We will then have to determine how much space is available from the manipulated register down to the EIP. That will help determine how much data needs to be filled in that area of the stack, as our shellcode will only fill in a small part of that area.

Here, we will fill our shellcode with No Operations (NOPs). The NOPs should be inserted between the shellcode and the EIP are to offset the injected shellcode. So, they are loaded in appropriate chunks, when instructions are loaded into the registers. Otherwise, the shellcode will be out of place. Finally, the sled that is loaded last onto the stack is there to take up the rest of the space, so when the Jump to ESP is called the code slides down from the top to the actual shellcode.

4. WORKING OF SHE-BASED BUFFER OVERFLOW

An exception handler is a part of code which is coded when designing an application. SEH handles all the exceptions that occurs during the runtime. By default, Windows has an exception handler (SEH) which is coded to catch an exception, if solvable then solve the error or generate an error message**.** Exception Handler (SEH) and Next Exception Handler (nSEH) are the pointers to the SEH that are added to the stack. The order is reversed (nSEH and then SEH) due to the flow of the Windows stack. After the exception occurs, we see that the value of ESP is located 8 bytes before the value of SEH.

Let us understand how SEH Based Buffer Overflow works. The following is the pictorial diagram of the working of SEH Based Buffer Overflow.

Figure. SEH Based Buffer Overflow

Our goal at this point is to successfully overflow the buffer, and overwrite the SHE and nSEH that would point towards our shellcode thus executing it.

The OS walks the SEH Chain and each Exception Handler (SEH) is checked to see if it can handle the exception (by calling the exception callback function and examining the details found in the exception and context records). If not, Exception Continue Search is returned and it moves to the address of the next record (pointed to by Next SEH) and continues down the chain until it finds a suitable exception handler or hits the last, default handler(FFFFFFFF).

5. MITIGATION AND RECOMMENDATION

- **Windows**

Since, security have become the most important part from an organization point of view. Security Measures must be taken to ensure our data are secure and is private. Below are some of the security measures that should be taken for the protection from the Buffer Overflow attack.

The following are the techniques that are discussed below:

- DEP - DEP stands for Data Execution Prevention. DEP is a security feature within the Operating System that

helps to prevent damage from security threats and virus by preventing malicious code from running on the system. Since harmful Programs try to attack Windows by running malicious code from the memory.

Thus, to solve this issue Windows introduced DEP, which marks all these memory locations as non-executable. In short, we need to remember that DEP will make it significantly harder to run exploits from the memory. For the Security purpose we should enable DEP from the Server side to protect the Server.

Visit the following link to view in detail: https://radiojitter.wordpress.com/2018/04/17/buffer-overflow-mitigation-part-6/

- ASLR

ASLR is used to for the security purpose, which involves the randomization of the base address of an executable and the position of stack, heap in a process address space. The randomization of the memory address gives an advantage of not knowing the location of actual required address. Thus it does not remove the vulnerability from the system but makes it more challenging for an attacker to exploit this vulnerability.

Visit the following link to view in detail: https://radiojitter.wordpress.com/2018/04/17/buffer-overflow-mitigation-part-6/

- SafeSEH

SafeSEH is a protection mechanism introduced by Windows in which exception handlers are validated, registered and stored. To ensure it is safe the addresses are checked prior to executing a given exception handler. A POP+POP+RET address comes from a module compiled with SafeSEH that is used to overwrite an SEH record and will not appear in the table and the SEH exploit will fail. To prevent SHE based attacks SafeSEH is very useful and effective.

Visit the following link to view in detail: https://radiojitter.wordpress.com/2018/04/17/buffer-overflow-mitigation-part-6/

- Stack Cookies/GS protection

The stack cookies will only be added by the compiler to minimize the impact of the performance, if string buffer is there in the functions or using _allocate is used to allocate function in the stack. If the buffer contains 5 bytes or less the protection is not activated.

In a buffer overflow attack, we attempt to overwrite the EIP with the address of our shellcode. But before the EIP is overwritten with our data, the cookie is overwritten as well, thus resulting the exploit useless. Thus the application dies once the function epilogue finds that the cookie is changed.

4. CONCLUSION

In this paper, we explored on how the Stack Based and SEH Based buffer overflow actually works. It is not only a programming error, but can also be exploited to execute arbitrary code on the system. Security Measures are also provided in this project to safeguard from the attack.

ACKNOWLEDGEMENT

I take this opportunity to express my deep gratitude and appreciation of all those who encouraged me to successfully complete the journal.

With profound sense of gratitude and regards, I acknowledge with great pleasure the guidance and support extended by, Mr. Priyashloka Arya from RadioJitter Concept Labs for his accomplishment and valuable information, direction and sense of perfection to work. He had been main source of inspiration for completion of work and strengthening confidence.

I would also thank my parents for their understanding & encouragement, my friends, one and all those who supported me and helped me.

REFERENCES

[1] Attacks and Defenses for the Vulnerability of the Decade

[2] Stack Based Overflow: Detect & Exploit, Morton Christiansen

Sites Referred:

- https://www.corelan.be/index.php/2009/07/19/exploit-writing-tutorial-part-1-stack-based-overflows/
- https://www.corelan.be/index.php/2009/07/23/writing-buffer-overflow-exploits-a-quick-and-basic-tutorial-part-2/
- https://www.corelan.be/index.php/2009/07/25/writing-buffer-overflow-exploits-a-quick-and-basic-tutorial-part-3-seh/
- https://radiojitter.wordpress.com/2018/04/16/buffer-overflow-exploit-part-1/
- https://radiojitter.wordpress.com/2018/04/17/buffer-overflow-mitigation-part-6/