STACK SMASHING VULNERABILITY AND PROTETION MECHANISMS

IT18361728
H.W.D.T De Silva
Department of Information Systems Engineering
Sri Lanka Institute of Information Technology
Malabe, Sri Lanka

***Abstract*— Rapid growth of technology and huge amount of attack surfaces available in the world paves the way to adversaries to leverage the existing technology and exploiting systems by carrying out highly sophisticated attacks. With compared to them naïve, but still commonly used attack type is stack smashing attack. There intruders inject sophisticated malicious opcodes to the stack and obtain the control flow of the vulnerable program which eventually leads to compromise the underlying system and allows attackers to run arbitrary code in an unauthorized manner without legitimate users' intervention or consent. Return address, stack frame pointer, pointers, arrays of pointers, arrays of characters are well known attack targets used by adversaries and on the other hand ASLR, non-executable stack, canary values, use of memory safe languages and static codes analysis are some protection mechanisms used to defend against stack smashing vulnerability. In this paper what is stack smashing attack, tactics and techniques used to perform attacks and countermeasures against them have been discussed.**

I. INTRODUCTION

When it comes to computing, Random Access Memory is considered as one of the most significant hardware parts of a computer. It acts as a temporary storage location for storing necessary data and pieces of computer instructions enabling the CPU to retrieve them when they are required. Furthermore, a set of machine instructions that are loaded into memory is known as a computer program and the language which is used to write a program is called programming language. Typically people use these computer programs to perform certain tasks in day-to-day life. On the other hand adversaries leverages the vulnerabilities of these programs to gain unauthorized access to systems to perform malicious activities like steal, alter or destroy information and computer systems without the knowledge or consent of legitimate users. Stack buffer overflow or the stack smashing attack is one such most well-known and commonly used critical security vulnerability by the adversaries to gain privileged access to computer systems. In simple words stack smashing means that user can corrupt the stack in such a way as to inject malicious executable code into the stack of the running program and take control of the process. In here it violates the workflow of the stack [1].

This vulnerability is identified and utilized to create critical malwares like Code Red worm, Sasser worm, Zotob worm and Stuxnet worm which were related to high profile computer security incidents in the world. Code Red was initially recognized in July 2001 and it compromised the computer systems which were used to run Microsoft internet information services webserver. This worm used to connect with vulnerable server utilizing default GET request on port 80 and set the payload along with a large number of strings which includes repeated letter 'N' to exploit the buffer overflow vulnerability of the systems. It used to run completely on memory and had the capability to destroy or erase the files resided in the hard drive of compromised systems. Moreover, the estimated loss happened due to this was about $2.62 billion. The Sasser, which is also falls under worm category utilized Local Security Authority Subsystem Service which is a process that responsible for enforcing the security policy on the windows systems like XP and 2000. This also exploited buffer overflow vulnerability and propagated via port 445 or 139. Delta airlines had to cancel number of transatlantic flights due to the failures of computer systems, AFP news agency's' all communications that transmitted through satellite technology were disrupted and blocked for several hours, Failures occurred in X ray machines of a Swedish hospital called Lund University Hospital are some instances of critical damage caused by that worm. Zotob worm is another one which utilized buffer overflow vulnerability identified in plug and play service facility of windows platforms which enables attackers to exploit the systems via remote code execution. Windows XP, 2000 and Windows server 2003 were vulnerable to Zotob worm. After the infection process in Windows 2000 systems zotob creates a backdoor which enables intruders to execute their commands remotely on compromised systems but when it comes to Windows XP systems zotob used them as a platform to propagate to more unpatched vulnerable systems in the network exploiting the same vulnerability. Continuous crashing and reboots, slowing down the performance of the systems are some symptoms which were identified in infected systems. Stuxnet is another well-known malware which was used to disrupt the Iran's nuclear weapon program by US and Israel in 2010. This highly sophisticated worm launched its attack in three phases, initially it compromised windows machines then Siemens step 7 software which was used to program industrial controlling systems which are capable of operating equipment like gas centrifuges and finally the

programmable logic controllers which enabled attackers to spy and cause harm to compromised systems. There Stuxnet worm issued malformed packets via Remote Procedure Call (RPC) protocol to server service on windows computer systems. (RPC protocol was used to request a service from a program installed in another system on a computer network). Through that way Stuxnet worm was able to exploit unpatched systems since buffer overflow occurred when the recipient or the received party tried to process the malicious request crafted by Stuxnet worm. Therefore, aforementioned real-world instances of crucial security incidents proclaim the idea that how critical this stack smashing vulnerability is and the destruction that could happen not only to specific computer systems but also the negative affect that could happen to any nation in the world if it is related to critical infrastructures and how important it is to pay great attention to eliminate this type of vulnerabilities.

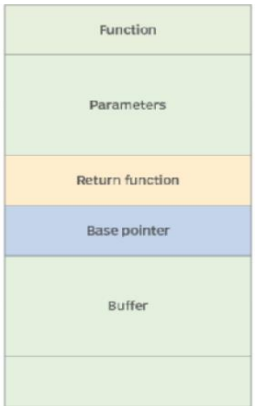
In the next section it is discussed what is stack and the third section describes about what is the basic idea behind the stack buffer overflow attack and how it is carried out in a typical scenario and fourth section describes about the difference between buffer overflow attack and stack smashing attack and fifth section contains well-known attack targets of stack smashing attacks. Possible protection mechanisms are described in the sixth section and the seventh section concludes the everything that is discussed throughout this review paper.

II. WHAT IS STACK

When a program is run, its executable image is loaded into the main memory of the computer in an organized manner. In here executable image referred to the compiled code, which can be divided into five main segments as text, data, bss, heap and stack where each segment represents a special chunk of memory that is allocated for a specific purpose. Since this paper is about stack smashing vulnerability in this section, it is only focused on what is stack and how does it work. When a program calls a function that function will have its own set of passed variables, and the function's code will be at a different memory location in previously mentioned text segment. Since EIP must change when a function is called, the stack is used to remember all the passed variables, the return address of the location EIP should return to after the function execution is finished, and all the local variables used by that function. In here EIP referred to a register which is used to store the memory address of the next instruction to be executed. Furthermore, stack follows LIFO (last in first out) method which means that the last item that is inserted into a stack is the first item to come out of it. When an item is placed into a stack, it is called as pushing, and when an item is removed from a stack, it is called popping. When a new function is called, a new frame is pushed onto the top of the stack and it becomes the active frame. When a function is finished its work, its frame is popped off of the stack, and the frame immediately below it becomes the new, active function. Another key feature related to stack is unlike other segments the stack segment grows backwards which turns out that it starts from high memory addresses and grows towards low memory addresses of the main memory [2],[3].

III. STACK BASED BOF ATTACK

This is generally called as buffer overflow and occurs due to an attempt to write more values than the allocated size of the buffer which is located on the stack. Most of the times this causes to crash the program or to misbehave the program. As previously explained when a new function is called new stack frame will be allocated and all the local variables and arguments will be pushed on to that particular stack frame along with the return address.



General architecture of a stack frame

When program attempts to write more data than the allocated buffer size it starts to overwrite the values of base pointer as well as the return value. When EIP attempts to read the value of the return address and execute it creates a dangling pointer since it is already filled with garbage values instead of valid memory address. When it comes to attack scenario, attackers take this as an opportunity and inject a malicious script and then overwrite the return address with the memory address which points to the beginning of the malicious script. When EIP reads the return address it will start to execute the malicious script which helps attackers to obtain the control flow and gain unauthorized access to compromised system with special privilege levels. This often happens due to the insufficient bound checks of the programs [4],[5].

IV. WHAT IS STACK SMASHING

The difference between in a typical buffer overflow and stack smashing attack is buffer overflow often leads to crash the program creating a sort of denial-of-service attack whereas in the stack smashing intruder deliberately alters existing opcodes which are loaded on to the stack that a legitimate user is not supposed to have access to and injects malicious

machine instructions to obtain the control flow of the victim program which ultimately can be utilized to hijack the control of the underlying system. Apart from that in stack smashing attack the victim program does not get killed or crashed. Moreover, buffer overflow attack falls under stack smashing attack and it is based on vulnerable buffer in a vulnerable program and other than buffer there are several attack targets that can be used to perform stack smashing attacks. Hence it is important to understand the difference between buffer overflow attack and stack smashing attack [6].

V. ATTACK TARGETS

A. Return Address

Typically, when a function calls another function in the program execution time the control flow of the program is passed to the called function. In here the function which calls another function is named as caller and the function which is being called by the caller function named as callee. When this process happens caller function gets terminate and the callee gets executed and after getting executed caller needs to resume its execution. That is when the return address comes into play because it used to hold the memory address of the instruction to return to. Therefore executing the instruction of the memory address which used to reside in the return address program can resume the execution of caller function from the point of where it left off prior to terminate the control flow. If an attacker could modify this original return address value to a memory address of a malicious code when callee finished it directly starts to execute injected code which gives the ability to attacker to take the control [4],[7].

B. Stack Frame Pointer

As previously explained when caller function calls callee function in the program execution time caller function hands over the control of the program to callee. When this happens new stack frame will be allocated. At the same time base address of the caller function and new base address of the callee function will be pushed into the stack in order to create a new frame. Typically, to store function's base memory address register will be used and in 32-bit architectures that register called as ESP register and 64-bit architecture systems it is called as RSP register.

```
#include <stdio.h>

void test()
{
        char buffer[5];
        gets(buffer);
}

int main()
{
        int c =5;
        test();
        while(c > 0)
        {
                printf(" Hello ... ");
                c--;
        }
        return 0;
}
```

```
(gdb) disass main
Dump of assembler code for function main:
   0x0000000000001161 <+0>:     push   rbp
   0x0000000000001162 <+1>:     mov    rbp,rsp
   0x0000000000001165 <+4>:     sub    rsp,0x10
   0x0000000000001169 <+8>:     mov    DWORD PTR [rbp-0x4],0x5
   0x0000000000001170 <+15>:    mov    eax,0x0
   0x0000000000001175 <+20>:    call   0x1145 <test>
   0x000000000000117a <+25>:    jmp    0x1191 <main+48>
   0x000000000000117c <+27>:    lea    rdi,[rip+0xe81]        # 0x2004
   0x0000000000001183 <+34>:    mov    eax,0x0
   0x0000000000001188 <+39>:    call   0x1030 <printf@plt>
   0x000000000000118d <+44>:    sub    DWORD PTR [rbp-0x4],0x1
   0x0000000000001191 <+48>:    cmp    DWORD PTR [rbp-0x4],0x0
   0x0000000000001195 <+52>:    jg     0x117c <main+27>
   0x0000000000001197 <+54>:    mov    eax,0x0
   0x000000000000119c <+59>:    leave
   0x000000000000119d <+60>:    ret
```

A vulnerable program is shown in the first picture where vulnerable gets function is used to obtain user input values and disassembled view of the main function is shown in the second picture. As it represents <+8> line in the disassembled code, value 5 will be moved to the address that is located at the RBP register – 4 memory address which refers to int c =5 line in the c code. In <+44> and <+48> lines also same address is referred to subtract 1 and compare with 0 which related to the condition inside the while loop furthermore it turns out the meaning that c variable's location is accessed using the memory address of RBP. When it comes to attack scenario, if adversary able to modify RSP address value it can use to access fake variables that adversaries placed in the stack which means that program could be tricked to access and execute malicious opcodes which resides in the fake variables instead of valid variable values since local variables accessed or addressed referencing the memory address of RBP or this could cause to malfunction the program [4].

C. Pointers

When a program is executed, data chunks are needed to be copied to perform certain functionalities of the program. Since it is impossible to move the physical memory same data chunks should be copied several times to different locations if same values accessed multiple times during the program execution and it could be computationally exhaustive and expensive task as before copying data chunks, new destinations should be allocated in the memory. As a solution to those problems, pointers were introduced.

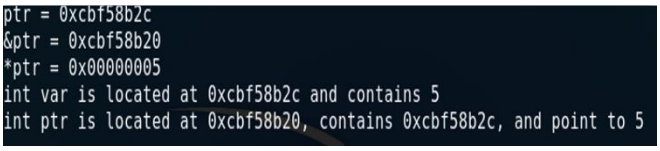
```
#include<stdio.h>

int main(){
        int var = 5;
        int *ptr;

        ptr = &var;

        printf("ptr = 0x%08x\n",ptr);
        printf("&ptr = 0x%08x\n",&ptr);
        printf("*ptr = 0x%08x\n",*ptr);

        printf("int var is located at 0x%08x and contains %d\n",&var,var);
        printf("int ptr is located at 0x%08x, contains 0x%08x, and point to %d
\n\n",&ptr,ptr,*ptr);
}
```



Above pictures manifest that pointers do not contain the values itself and they store the memory addresses of variables. When it comes to attack scenario, adversaries could overwrite the referencing address or pointer that is in the stack itself. Attacker could directly execute the injected malicious code if he is able to modify the value that pointer is holding in the stack because when that pointer is called it is going to reference a fraud memory address that contains malicious code which has been modified by the attacker. On the other hand attacker could modify or overwrite the value of the memory address that the pointer is pointing to and it is considered as indirect pointer overwrite method. Both of these methods ultimately leverage the pointer's behavior that resides in the stack and utilize to perform stack smashing attack by an adversary [4],[8].

D. Arrays of Pointers

If a program function comprises arrays which consist of number of pointers it could become a most obvious attack target of attackers as they if they could modify the content of the memory addresses those pointers are pointing to or modify pointers' content by previously mentioned indirect and direct method. Moreover they can leverage the spatial errors which refers to the situations where pointers could access the memory out of its allocated boundaries and temporal errors which refers to the situations where pointers are used to deallocate objects that has already been deallocated or considered invalid as pointer. Therefore by utilizing those methods attackers tend to perform a stack smashing attacks [4].

E. Arrays of Characters

These types of arrays are the highly targeted attack vectors when it comes to stack smashing attacks like buffer overflow. As previously explained, most of the times attackers leverage the vulnerabilities of functions which used to copy characters like strcpy function and utilize these kind of character arrays to crash or malfunction the program.

VI. PROTECTION MECHANISMS

A. ASLR

Since attackers intentionally change the instructions and data of the executed program in their own way to gain the control flow, there is an obvious requirement to be aware of exact memory addresses of specific targets that they are supposed to modify or where their injected code resides in the memory. But this technique provides a solution for that problem. Here ASLR refers to Address Space Layout Randomization. As the name suggests, in this method the address space locations of major data areas of the process, such as the executable's base and the data section, bss section, stack, heap, and libraries, are randomly organized in the main memory by this technique and prevent attackers from jumping to exploited memory regions in a reliable manner and exploiting memory corruption vulnerabilities [4].

B. Non-executable Stack

Most of the times when performing stack smashing attacks adversaries tend to inject arbitrary code to stack and execute to obtain the control. As a solution to that problem non executable stack concept was introduced where the stack disallows running machine codes which turns out the meaning that attacker either should find a mechanism to disable this protection mechanism or should inject the malicious code in another memory region where it allows code execution. This data execution prevention technique is a hardware feature and XD bit. XN bit and NX bit are couple of instances belong to different computer architectures which segregate memory regions and disables code executions.

C. Canary Values

In this protection mechanism the compilers often place a randomly generated value which is located between local variables and the saved frame pointer. At the startup of the program execution this will be generated and stored in a global variable. When a new function gets called during the execution process copy of the generated canary value would be pushed into the stack. After the function execution prior to the return canary value which is stored in the stack will be compared with the globally stored value. If they differ from each other and failed to verify the integrity the program would be terminated.

D. Static Code Analysis

Most of the times this method is used when a computer program is written in computer languages such as C and C++ which are typically considered as low-level programming languages in computing. Use of these low-level programming languages still remain common despite the well-known memory errors they allow. For instance, the features that make C a desirable language for many system level programming tasks like weak typing, low level access to computer memory and pointers are the same features adversaries misuse to perform memory access violations. Although these violations often cause to crash the programs immediately the symptoms could go undetected. Access errors, buffer over read, invalid page fault, memory leak, stack exhaustion, insufficient bound checks, integer errors, use of uninitialized variables, use after free, double free and null pointer dereferences are some well-known memory errors that could occur while programming by the computer

programmers. As a solution for that this static code analysis mechanism was introduced where the computer programmers manually check and manage memory by specifying when to allocate resources, how much space needs to be allocated and when to free or deallocate the resources. As an example, programmers can identify use of vulnerable functions like strcpy, gets and replace them with functions like strncpy, scanf where they check for necessary bounds. This gives the programmer very fine-grained control over how their implementation uses resources, enabling fast and efficient code not only that but also this type of static analysis helps to address the weaknesses in the source code which are typically lead to stack smashing vulnerabilities. This is normally done in the early development stage before the testing phase. However, this approach is prone to mistake when it comes to complex codes [4],[9].

E. Use of Memory Safe Languages

Another major problem that was encountered which caused to these stack smashing attacks is programs are written using low level programming languages such as C and C++. They also considered as memory unsafe languages among programmers. Moreover, these programming languages do not provide protection against spatial safety errors that occur because pointers access memory regions that are out of their allocated boundaries and temporal safety errors which occur when pointers are used to deallocate objects that has already been deallocated or considered as invalid pointer. As a solution for that it is recommended to use memory safe languages such as python, java, java script, ruby, swift which do not require special modifications, manual boundary checking or any kind of changes. The basic idea behind memory safe languages is that each and every access to the memory is well defined [10].

VII. CONCLUSION

In this paper I described what is stack and how it works, how does typical buffer overflow occur and how it can be utilized to perform stack smashing attack, distinguish the difference between the buffer overflow and the stack smashing attack in computing, several attack targets that are being used most of the times by intruders and various tactics and techniques that can be used to provision robust protection against stack smashing attacks. Moreover, in computing, program is considered as memory safe if all possible executions of that program are memory safe, programming language is considered as memory safe if all possible programs written in that language is memory safe also runtime environment is considered as memory safe it all possible programs which could be run in that environment are memory safe. Therefore it is a significant task to ensure the memory safety of all those three aspects to provide robust protection against stack smashing attacks.

VIII. REFERENCES

[1] Aleph One. Smashing the stack for fun and profit. Phrack, 49, 1996.

[2] T. Chiueh and F.-H. Hsu. RAD: A compile-time solution to buffer overflow attacks. In Proc. of the 21st Int. Conf. on Distributed Computing Systems, Phoenix, AZ, Apr. 2001

[3] Owan, S. Beattie, J. Johansen, and P. Wagle. Point Guard: protecting pointers from buffer overflow vulnerabilities. In Proc. of the 12th USENIX Security Symp., Washington, DC, Aug. 2003.

[4] Yves Younan, Davide Pozza, Frank Piessens, Wouter Joosen: Extended protection against stack smashing attacks without performance loss Yves: Proceedings - Annual Computer Security Applications Conference, ACSAC

[5] Crispin Cowan, Steve Beattie, Ryan Finnin Day, Calton Pu, Perry Wagle, Erik Walthinsen: Protecting systems from stack smashing attacks with StackGuard: Linux Expo

[6] Hiroaki Etoh, Kunikazu Yoda: ProPolice: Protecting from stack-smashing attack Hiroaki Etoh and Kunikazu Yoda ProPolice: Protecting from stack-smashing attacks

[7] Doaa Abdul-Hakim Shehab, Dr. Omar Abdullah Batarfi: RCR for preventing stack smashing attacks bypass stack canaries: Proceedings of Computing Conference 2017.

[8] Alouneh, S., Kharbutli, M., & AlQurem, R. (2013). Stack Memory Buffer Overflow Protection based on Duplication and Randomization. Procedia Computer Science, 21, 250-256.

[9] Crispin Cowan, Calton Pu, et.al.: StackGuard: Automatic Adaptive Detection and Prevention of Buffer-Overflow Attacks: Proceedings of the 7th USENIX Security Conference, San Antonio, Texas, USA, 1998.

[10] Y. Younan, W. Joosen, and F. Piessens. Code injection in C and C++: A survey of vulnerabilities and countermeasures. Technical Report CW386, Departement Computer wetenschappen, Katholieke Universiteit Leuven, July 2004