Buffer Overflow Attack and Defense Techniques

Sabah M. Alzahrani

Department of Computer Science, College of Computers and Information Technology, Taif University,
Taif P.O. Box 11099, Taif, 21944, Saudi Arabia

Summary
A buffer overflow attack is carried out to subvert privileged program functions to gain control of the program and thus control the host. Buffer overflow attacks should be prevented by risk managers by eradicating and detecting them before the software is utilized. While calculating the size, correct variables should be chosen by risk managers in situations where fixed-length buffers are being used to avoid placing excess data that leads to the creation of an overflow. Metamorphism can also be used as it is capable of protecting data by attaining a reasonable resistance level [1]. In addition, risk management teams should ensure they access the latest updates for their application server products that support the internet infrastructure and the recent bug reports [2]. Scanners that can detect buffer overflows' flaws in their custom web applications and server products should be used by risk management teams to scan their websites.
This paper presents an experiment of buffer overflow vulnerability and attack. The aims to study of a buffer overflow mechanism, types, and countermeasures. In addition, to comprehend the current detection plus prevention approaches that can be executed to prevent future attacks or mitigate the impacts of similar attacks .

Key words:
Buffer; Overflow; Cybersecurity; Stack; Defense; Attack; Shellcode.

1. Introduction

Internet usage over the years globally has experienced exponential growth due to the benefits associated with its utilization to governments, corporations, and individuals. However, the interconnected computer systems have led to the discovery of various software vulnerabilities, which can be exploited by unscrupulous individuals or organizations.
Furthermore, the most prevalent vulnerability is the buffer overflow attack, which in most cases is activated by the input that is explicitly designed to execute malicious code. Additionally, the recent infamous buffer over attacks includes I love you attacks, Blaster, and the SQL Slammer, all of which were unexpected behaviors that exist in particular programming languages. Likewise, the inability of a program to store large amounts of data in a buffer is the main reason why hackers utilize buffer overflow attacks. Thus, when a program attempts to store excess data than what it was made to store, the extra information overflows into other buffers in most cases which are not considered good by most experts as the buffer's original data may be overwritten [3]. Contemporary hackers have been disguising buffer overflow attacks as viruses intending to illegally access information.

2. Literature Review

Buffer overflows are a common occurrence in most organizations today, and weakness is created by the vulnerability in cases where memory near a buffer is overwritten which should not be unintentionally or deliberately adjusted in a program. Some buffer overflow attack causes are logical errors that arise while implementation is being carried out, using unsafe library functions, and a lack of input filters. In situations where a buffer overflow attack occurs, the program either loses its stability or collapses [4]. Most attackers do not carry out buffer overflow attacks to cripple the program but rather to overwrite the stack's essential values so that their malevolent unsigned codes can be executed. Because they target web servers, web applications, and desktop applications that are used by most organizations, buffer overflows are considered to be extremely dangerous.

The attack usually occurs to destroy the memory, where it comprises of these memory sections such as the stack that is responsible for storing local variables such as the inside functions and arguments. Another area is the data area, which comprises the data segment that consists of the static or global variables previously started by the programmer [3]. Furthermore, another data area is the BSS segment, which is known as the Block started by the symbol. It comprises the uninitialized global variables, which can be initialized to zero that occurs before the program execution. Moreover, the heap is the data area segment where it is the space utilized for dynamic memory allocation when there is a program execution underway by malloc(), calloc(), realloc(), and free(). Likewise, there is the text segment that

Manuscript received December 5, 2021
Manuscript revised December 20, 2021
https://doi.org/10.22937/IJCSNS.2021.21.12.30

comprises the program executable code plus it is usually read-only.

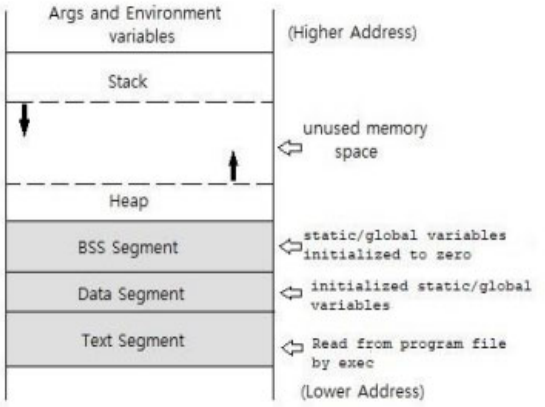

Fig. 1 Memory program layout description

There exist two types of buffer overflow attacks namely stack-based and heap-based attacks, which can have devastating effects on the functioning of computers. In addition, these attacks lead to the memory space reserved for the program being usually flooded by the attacker in a heap-based attack. Furthermore, heap overflow attacks are those where the buffer that is to be overwritten is allocated in the memory's heap portion, where the data writing to the memory is done without the data undergoing the bound checking processes. Equally, the stack, which is a memory potion reserved to store addresses and data for the program, is targeted and taken advantage of by the attacker in the stack-based attack [5]. Similarly, the stack is then forced to partially crash by the attacker which forces the execution of the program to start from a malicious program address from the attacker. Besides, other types of attack entail integer overflows.

3. Methodology

In this paper, a program with a buffer-overflow vulnerability is used; then develop a scheme to exploit the vulnerability and finally gain the root privilege. The environment has been applied on pre-built Ubuntu 16.04 VM, which can be downloaded from the SEED website and these tasks are explained in the SEED website as well [6].
First, since the buffer-overflow attack is difficult in Ubuntu and other Linux distributions, it has to turning off the countermeasures by disable them. These systems use address space randomization for randomizing the start address of heap and stack as well. Thus, it is difficult to guessing the exact addresses. This can be done by the following commands:

```
sudo sysctl -q kernel.randomize_va_space
```

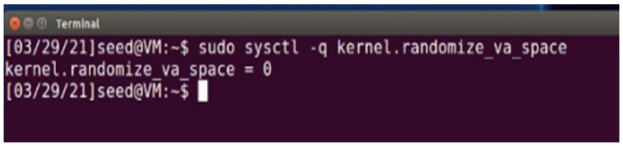

Fig. 2

Then, turn off the interspace random

```
sudo sysctl -w kernel.randomize_va_space=0
```

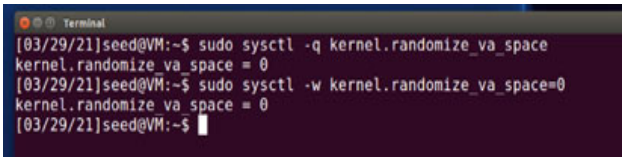

Fig. 3 Turn off the interspace random

The victim program is a Set-UID program, and the attack relies on running /bin/sh, thus the countermeasure in /bin/dash makes the attack more difficult. Therefore, /bin/sh will link to another shell which does not have a countermeasure. To install a shell program by the following command.

```
ls -l /bin/sh  &sudo IN -sf/bin/zsh/bin/sh
```

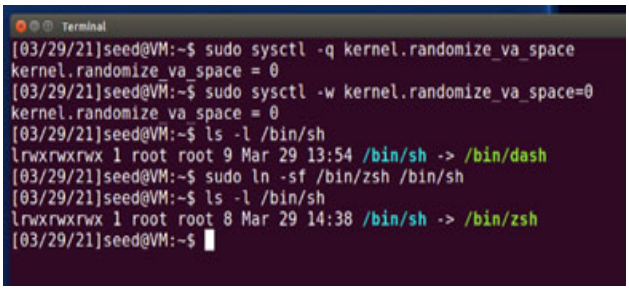

Fig. 4 Install a shell program

After that, the vulnerable program (stack.c) is used. This program has a buffer-overflow vulnerability. The aim to exploit this vulnerability and get the root privilege by writing the following commands.

```
gcc -fno-stack-protector -z execstack stack.c -o stack
```

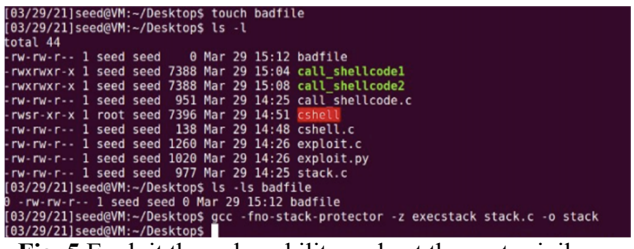

Fig. 5 Exploit the vulnerability and get the root privilege

The (stack.c) is program that has a buffer overflow vulnerability. Firs,t program reads an input from the file(badfile). Second, passes the input to another buffer in the function called bof(). The maximum length is 517 bytes of original input. However, the buffer in bof() is less than 517 on BUF SIZE bytes. The buffer overflow will occur here since strcpy() function does not check the boundaries. In addition, this program is a root-owned Set-UID program, thus if a normal user can exploit this buffer overflow vulnerability, the user able to gain a root shell. It have to create the contents for (badfile). Thus the vulnerable program copies the contents into its buffer, therefore a root shell can be spawned.

```
./stack
```

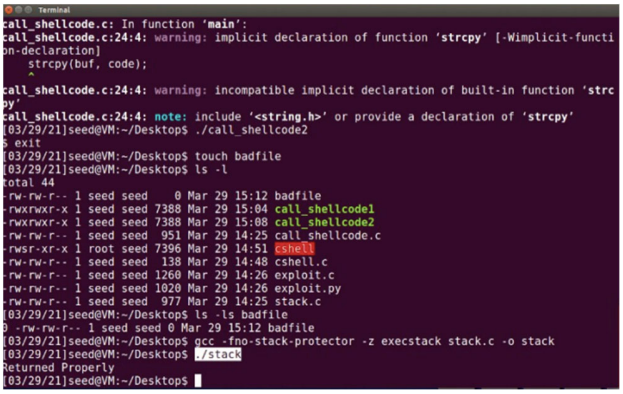

Fig. 6 The vulnerable program copies the contents into its buffer

It has to turn off the StackGuard and the non-executable stack protections using the -fno-stack-protector and "-z execstack". In addition, it has to make the program a root-owned SetUID program. The following are the commands used.

```
sudo chown root stack
sudo chmod 4755 stack
 ls -l stack
```

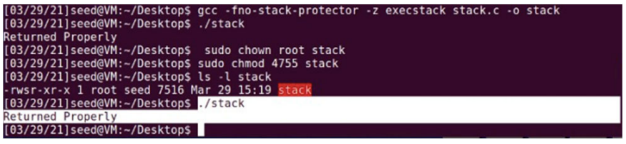

Fig. 7 Make the program a root-owned SetUID program-1

```
gcc -g -fno-stack-protector -z execstack stack.c -o stack_dbg
gdb ./stack_dbg
```

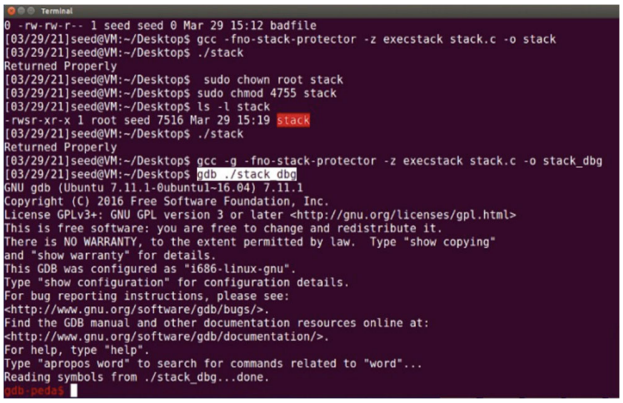

Fig. 8 Make the program a root-owned SetUID program-2

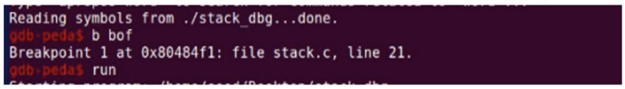

Fig. 9 Make the program a root-owned SetUID program-3

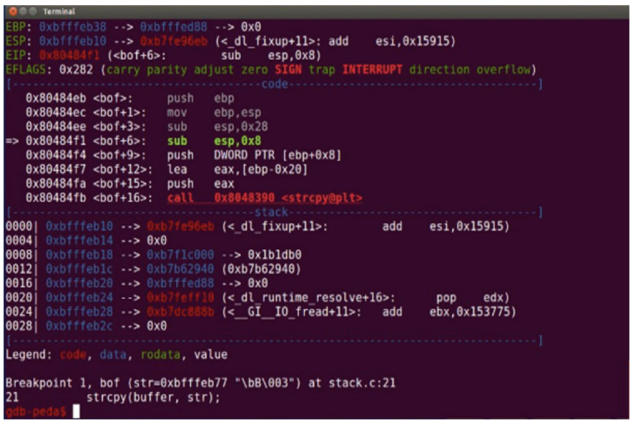

Fig. 10 Make the program a root-owned SetUID program-4

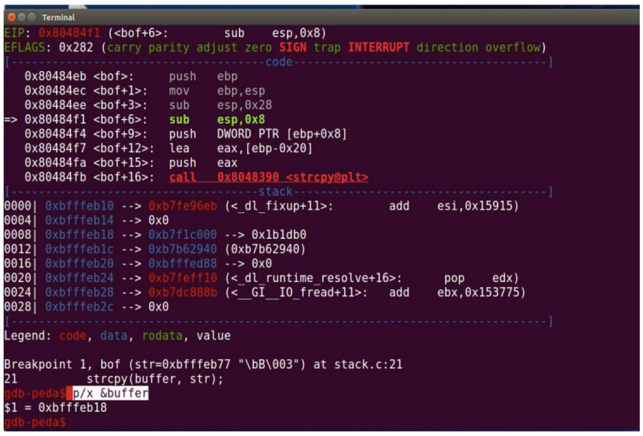

Fig. 11 Make the program a root-owned SetUID program-5

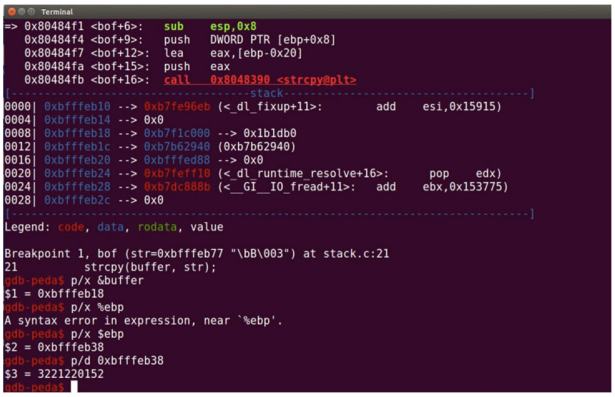

Fig. 12 Make the program a root-owned SetUID program-6

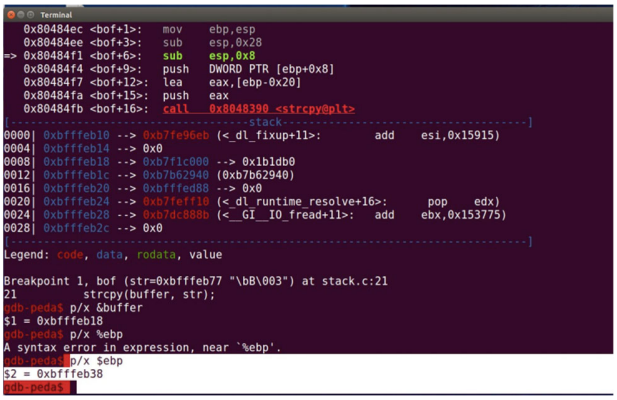

Fig. 13 Make the program a root-owned SetUID program-7

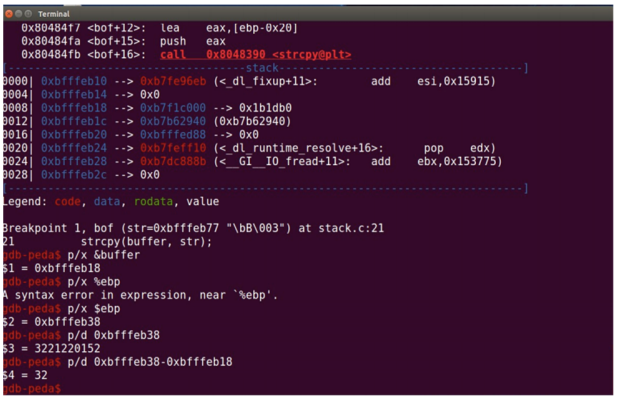

Fig. 14 Make the program a root-owned SetUID program-8

After that, now it have to exploiting the vulnerability using code called (exploit.py). This code is to construct contents for the file (badfile).

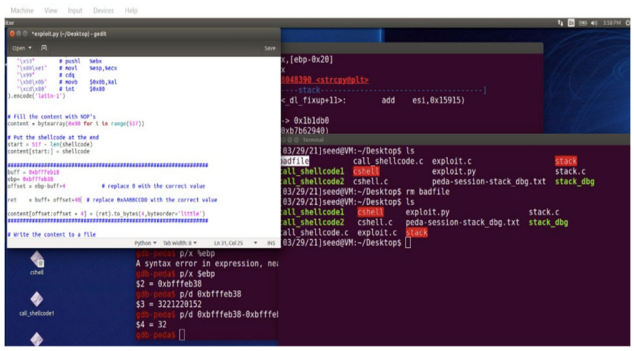

Fig. 15 This code is to construct contents for the file (badfile) - 1

```
python3 exploit.py
bless badfile &>/dev/null
```

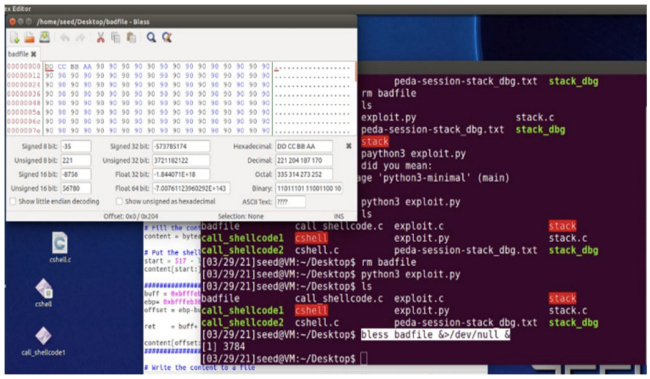

Fig. 16 This code is to construct contents for the file (badfile)-2

Then, modifying the C code, and compile it.

Fig. 17 Modifying the C code, and compile it

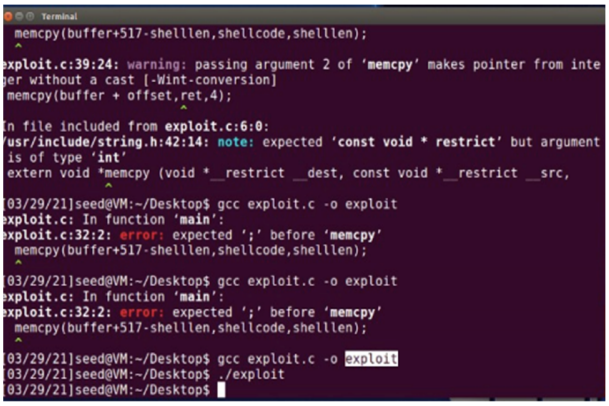

Fig. 18 Compile C code

```
vbindiff badfile badfilepy
```

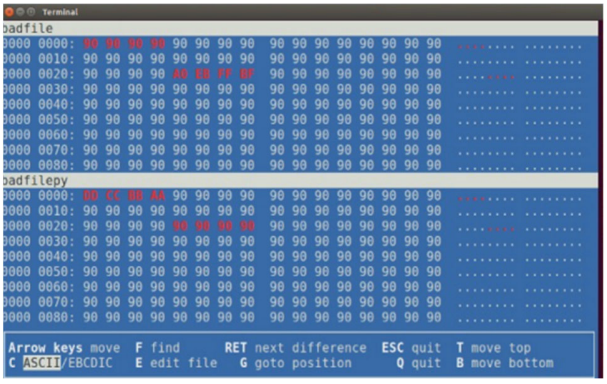

Fig. 19 Construct contents (badfile)

Now, lets defeating dash's Countermeasure by change the real user ID of the victim process to zero. This has to be before invoking the dash program by invoking setuid(0) before executing execve() in the shellcode. Thus, first change the /bin/sh symbolic link, this set back to :/bin/dash

```
sudo ln -sf /bin/zsh /bin/sh
```

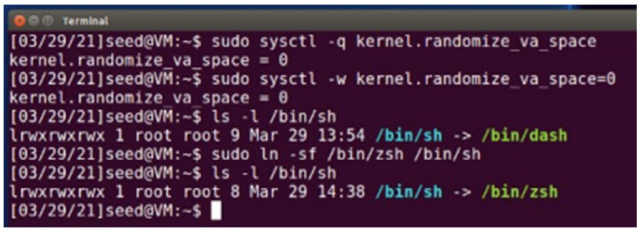

Fig. 20 Change the /bin/sh symbolic link

Now, lets defeating address randomization by using brute-force approach. This can be done by turn on the Ubuntu's address randomization.

```
sudo sysctl -w kernel.randomize_va_space=0
```

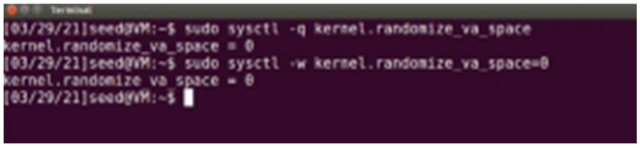

Fig. 21 turn on the Ubuntu's address randomization.

Now, lets turn on the StackGuard protection by compile the program without the -fno-stack-protector option. Also, it should be o turn off the address randomization.

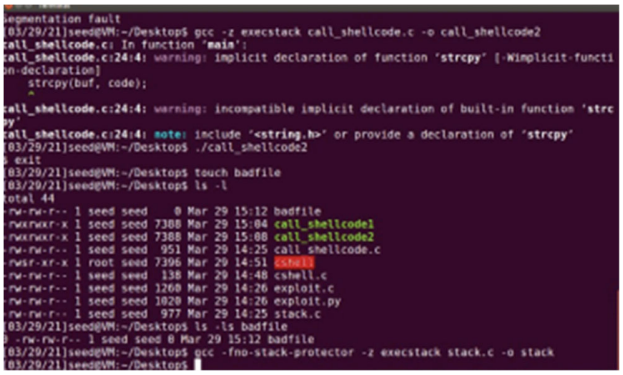

Fig. 22 Turn on the StackGuard protection

Now, recompile the vulnerable program using the noexecstack option. This scheme will make such attack very difficult.

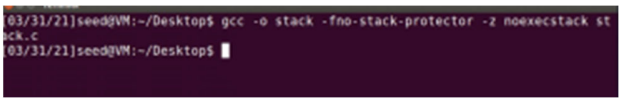

Fig. 23 No attack can occur in this schema.

4. Discussion

Due to buffer overflow attacks becoming quite common in contemporary times, both computer experts need to understand the methods of preventing them. To achieve this, programmers must first ensure no buffer overflows occur in their programs [7]. Besides, this can be achieved through utilizing programming languages that do not result in buffer overflows like Java, NET, PHP, Python, and PERL.
Fortunately, in cases where a program is released and demonstrates this vulnerability, software developers can make patches that can address some of the bugs. When the initial development of these is tools taking place, programmers can use additional programs like LibSafe, StackGaurd, and StackShield to screen errors. Additionally, developer training and code auditing can be used to resolve the vulnerabilities that make buffer overflows possibly. Computer experts should utilize systems that use non-executable stacks to protect their systems from a stack overflow. Furthermore, screening of code should be carried out to ensure no junk characters exist and the code is not too long. If programmers use an out-of-date or vulnerable language, they should ensure that they use updated patches, the principle of least privilege, and compilers that can protect the program from overflows. Lastly, checking of exceptions should always be carried out while factoring in the language used and how it supports this function.

5. Conclusion

Buffer overflow attacks should be prevented by risk managers by eradicating and detecting them before the software is utilized. This paper presents an experiment of buffer overflow vulnerability and attack. The aims to study of the buffer overflow mechanism, types, and countermeasures. Buffer overflow attacks should be prevented since its become a critical attack against many organizations.

References

[1] Chiamwongpaet, Sirisara, and Krerk Piromsopa. "Boundary Bit: Architectural Bound Checking for Buffer-Overflow Protection." ECTI Transactions on Computer and Information Technology (ECTI-CIT) 14.2 (2020): 162-173. Fan, X., Cao, J.: *A Survey of Mobile Cloud Computing*. ZTE Communications 9(1), 4–8 (2011)

[2] Di, Bang, et al. "Efficient Buffer Overflow Detection on GPU." IEEE Transactions on Parallel and Distributed Systems 32.5 (2020): 1161-1177.

[3] Mihailescu, Marius Iulian, and Stefania Loredana Nita. "Brute Force and Buffer Overflow Attacks." *Pro Cryptography and Cryptanalysis with C++ 20*. Apress, Berkeley, CA, 2021. 423-434.

[4] Nicula, Ștefan, and Răzvan Daniel Zota. "Exploiting stack-based buffer overflow using modern day techniques." *Procedia Computer Science* 160 (2019): 9-14.

[5] Sah, Love Kumar, Sheikh Ariful Islam, and Srinivas Katkoori. "An efficient hardware-oriented runtime approach for stack-based software buffer overflow attacks." *2018 Asian Hardware Oriented Security and Trust Symposium (AsianHOST)*. IEEE, 2018.

[6] Seedsecuritylabs.org. 2021. *Buffer-Overflow Vulnerability Lab*. [online] Available at: <https://seedsecuritylabs.org/Labs_16.04/Software/Buffer_Overflow/> [Accessed 12 November 2021].

[7] Wang, Zhilong, et al. "To detect stack buffer overflow with polymorphic canaries." *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 2018.

Sabah Alzahrani received the B.Sc. degree in Computer Science from Taif University, Saudi Arabia, in 2007. the M.Sc. degree and Ph.D degree. in computer and information systems engineering, from Tennessee State University, United States in 2015 and 2018 respectively. He is currently an Assistant Professor with department of Computer Science, College of Computers and Information Technology, Taif University, Taif, Saudi Arabia. Her research interests include the Internet of Things, Cyber Security, Computer Networking, Cloud, and Big Data.