Introduction

In this Assignment, I will cover 3 type of buffer overflow, Integer Overflow, Format String Overflow and strncpy Overflow. In this assignment mainly cover on malformed input from the attacker and how to avoid the attack. Besides, will need to know about the basic knowledge of the buffer overflow.

Example 1

Integer Buffer Overflow is mean an integer number store in buffer as a binary and the binary form of number are too large and cause the overflow happen. My example will show as below.

```
int main(void)
{
int val;
printf("type number: ");
scanf("%d", &val);
printf("val = %d (%d)\n", val + 1, val);
return 0;
}
```

In my example code, if I give the number as 234567 it will return 234568, that is fine in this case. Because it hasn't hit the maximum of the integer. But if I give 2147483647 in to this program, it will return -2147483647, the result is not matched to my expect at all. The reason of this will happen is because of the integer store in the buffer is based on 32bit, which mean 2147483647 in integer and 7fffffff in hexadecimal. In binary form of 7fffffff it will look like 0111 1111 1111 1111 1111 1111 1111 1111. And in my example code plus 1 in the binary form will become 1111 1111 1111 1111 1111 1111 1111 1111. That's the reason of the result will become -2147483647. In my solution in this example will show below.

```
int main(void)

{
unsigned int val;
printf("type number: ");
scanf("%u", &val);
printf("val = %u (%d)\n", val+1, val);
return 0;
}
```

In my solution, I change the integer value to unsigned, in default if the code doesn't mention the value will become a signed integer. The different between signed and unsigned value is signed integer included negative value and unsigned integer only have positive value. In a signed integer if the value larger than 2147483647, all will become negative value, it is not an accurate result. But if I using unsigned integer, in this case It giving back an accurate result. The range of the signed integer value is 2147483647 to -2147483646, but unsigned value giving 2 to power 32 which is 0 to 4294967295, and it is double when compare to signed value.

In my solution code using unsigned integer, if I give 2147483647 it will return 2147483648, this is a accurate value as I expect. In this example I'm using signed and unsigned value, I don't thing is a clever way to solving problem but is a simple way to avoid integer overflow. It give double spacing to programmer in some of the situation. However, if the value larger than 4294967295 it also will cause overflow. For example, if I give a value 23789098765456789 it will return 762385814 and 0x2d711595 in hexadecimal form.

Example 2

String formatting could know as an attack that using to against haven't declare output data type, such as integer, character, floating point, etc. My example show at below.

```
int main()
{
        char password[1];
        printf("Enter your password:\n");
        scanf("%s", &password);
        printf(password);
        printf("\n");
}
```

In this code, I was found two vulnerability, the first one is the buffer overflow, in the "char password[10]", if I give a "password" as "avshg2" for this code, it will print out a exactly same "password" back, but I your giving too more character it could lead to segmentation fault or bus error. The reason it will lead to the bus error is because it trying to write at the non-writable area in the memory address.

The second problem is when I type in something like "%x%x%x%x%x%x%x%x%x%x%x" and it return "1eccd67700eccd48c00e6137530078259c807825782578259d80e2" and segmentation fault. The segmentation fault mean that the data is locating to the memory address

it not supposed to get in. which mean the data is returning the data outside of the password's memory address and it is also known as the information leak.

```
int main()
{
	char password[100];
	printf("Enter your password:\n");
	scanf("%s", &password);
	printf("%s",password);
	printf("\n");
}
```

In the solution, I specific the output data, to make sure the data wouldn't have other option. `

Example 3

In this example, show at the below is about using the strncpy function could cause over buffer in the program, in this example I have 15 array for the buffer. Therefore, if i input the character under 14 character that will be fine for the program to execute, but if input over 14 character the program could run in to buffer overflow.

```
void copy(char *userId)
 {
	char buff[15];
	strncpy(buff,userId,14);
	printf("User Id is : %s \n", buff);
 }
int main()
{
	char Id[15];
	printf("Enter the ID : \n");
	scanf("%s", Id);
	copy(Id);
	return 0;
}
```

For the solution, I was adding the argument for the function called "len", this argument is for specific how many character can copy from the source to the destination. in this example, when I input the data into "Id" and put the "Id" in to the "copy" function. Inside the "copy" function, strncpy will copy the string from the argument and copy to the "buff" and specific to only take the first 10 character. Therefore, if I give data "123123123123123123" to the program, it will return "12312312312312312", so it won't cause buffer overflow. The reason I set the length to 14 and not 15 is because of C code doesn't have string, string in C code is reference to a set of character. in that set of character contain a string terminator. If a string has length of 15, that string can only contain 14 character and the last byte is the terminator. If I set the length in the function strncpy to 15, it will cause buffer overflow. When I give "123123123123123123", and it will output "123123123123123?k?]? "

Reference

1. OSWP (2010), *Reviewing Code for Buffer Overruns and Overflows,* Available at: https://www.owasp.org/index.php/Reviewing_Code_for_Buffer_Overruns_and_Overflows [Available on 24th Feb 2017]
2. OSWP (2012), *Format String Attack,* Available at: https://www.owasp.org/index.php/Format_string_attack [Available on 24th Feb 2017]
3. cdf.toronto.edu, printf and scanf format codes, Available at: http://www.cdf.toronto.edu/~ajr/209/notes/printf.html#keyletter [Available on 24th Feb 2017]