SC3010 Computer Security

Tutorial 3 – Software Vulnerability Defenses

1. Circle the correct answers in the following questions.

   1) Which of the following security features require hardware support?
      (i) Linux ExecShield, as it requires the hardware to control if each memory page is executable ✓
      (ii) ARM MTE, as it requires hardware tags to be attached to the memory regions. ✓
      (iii) ASLR, as it requires the hardware to determine if the address space of each program is randomized ✗

      [✓ A. (i) and (ii)] ✓
      B. (i) and (iii) ✗
      C. (ii) and (iii) ✗
      D. All of the above ✗

   2) Which statement is false about code reuse attack?

      A. The attacker does not need to inject malicious code. ✓
      [✓ B. The attacker does not need to compromise the return address.] ✓
      C. The attacker can chain multiple library functions for execution. ✓
      D. The attacker needs the information of the system's libraries in order to reuse the code. ✓

2.1 The return address can be overwritten to return to any code already loaded. For instance, the attacker may be able to cause a return into the libc execve() function with"/bin/sh" as an argument

   3) Which one of the following C functions has the lowest risk in terms of software vulnerabilities?

      A. gets
      B. strcpy
      [✓ C. strncat] ✓
      D. sprintf

2.1Non exectuable memory protection does not work when attacker does not inject malicious code into the stack. This happens when Code Reuse attack is used where the code in the program is reused without injecting the code. The return address can be replaced with the address of dangerous library function such that when the function returns, the dangerous function will execute. Also, multiple library functions can be chained.

2. Answer the following questions.

   1) Why are non-executable stack and heap not enough to defeat buffer overflow attacks?

   2) Distinguish three types of fuzzing techniques?

   3) What is a code reuse attack? refer to qn 1

2.3 Instead of injecting the malicious code into the memory, the attacker can compromise the control flow to jump to the existing library for attacks

3. For string copy operation, strncpy is regarded as "safer" than strcpy. However, improper use of strncpy can also incur vulnerabilities. Please describe the problem in the following program, and what consequences it will cause.

2.2 Fuzzing techniques:
perturb the input in a heuristic way
1)Mutation based fuzzing: A corpus of inputs are collected to explore as many states as possible. Inputs are modified and substituted randomly. The program is then run on inputs for crashes
2)Generation based fuzzing: A specification of input format is converted into a generative procedure. Test cases are generated and random perturbations are introduced. The program is then run on inputs for crashes
3)Coverage guided fuzzing: Traditional fuzzing strategies are used to create new test cases by mutating the input. The program is then tested and the code coverage is measured. Code coverage is used as a feedback to craft input for uncovered code

3. dest array is of size 8 and strncpy is trying to copy 8 characters. One character needs to be reserved for '/0'. This leads to buffer overflow. As a result, there might be corruption of program data, unexpected transfer of control, memory access violation and execution of code chosen by attacker.
strncpy does not automatically add the NULL value to dest if n is less than the length of string src. So it is safer to always add NULL after strncpy. In this program, dest does not have a terminator at the end. Then the function strlen will keep counting, which can cause segmentation fault.

4.1 When the canary value is random, it cannot be guessed by the attacker. When a stack buffer overflows into the function return address, the canary is overwritten. So, every time the function returns, check whether the canary value has been changed. If so, stack buffer overflows and the program is aborted.

```
#include <stdio.h>
#include <string.h>
int main() {
  char src[] = "geeksforgeeks";

  char dest[8];
  strncpy(dest, src, 8);
  int len = strlen(dest);

  printf("Copied string: %s\n", dest);
  printf("Length of destination string: %d\n", len);

  return 0;
}
```

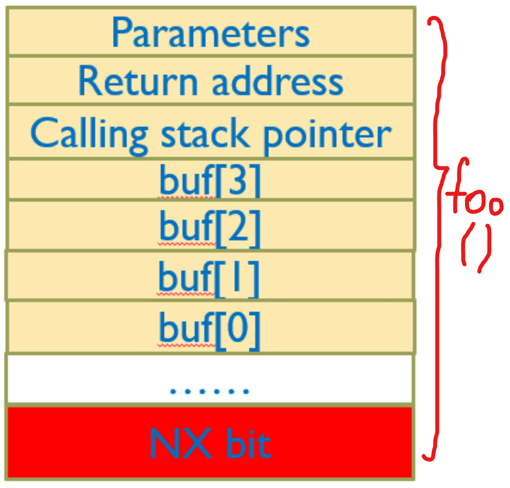
no terminator and
strlen will keep counting

4. StackGuard is a mechanism for defending C programs against stack-based buffer overflows. It detects memory corruption using a canary, a known value stored in each function's stack frame immediately before the return address.
   1) In some implementations, the canary value is a 64-bit integer that is randomly generated each time the program runs. Explain why this prevents the basic form of buffer-overflow attacks
   2) What is a security drawback to choosing the canary value at compile time instead of at run time?
   3) If the value must be fixed, what will be a good choice?
   4) List an attack which can defeat the StackGuard.

4.2 Choosing a canary value at compile time means the canary value is fixed once the program runs and can be guessed through brute force or reverse engineering, and perform the buffer overflow attack with that value.

5. One possible solution to defeat buffer overflow attacks is to set the stack memory as Non-executable (NX). This is usually achieved by the OS and the paging hardware. However, imagine that a machine does not support the non-executable feature, then we can implement this functionality at the software level. The compiler can allocate each stack frame in a separate page, and associate a software-manipulated NX bit at the bottom of this page, controlling if this page (stack frame) is non-executable. The structure of a stack frame is shown in the Figure below. Since the NX bit is at the bottom of the memory page, the buffers inside this stack frame is not able to overwrite this bit to make it executable. Describe a buffer overflow attack that can still overwrite NX bits.

4.3 A good choice for a fixed value is a terminator canary. These include \0, newline, linefeed and EOF. This is as string functions will not copy beyond terminator. Thus, the attacker cannot use string functions to corrupt stack.



5. refer to qn2.1

Although the buffers inside the frame cannot overwrite the NX bit. It can call another function, whose buffer can be used by the attacker to overwrite the NX bit. Then the caller's frame will become executable.

4.4 An attack that can defeat StackGuard is stack smashing attack.

The attacker can overwrite the function pointer instead of the return address

```
foo(){
   bar();
}
bar(){
  cahr buf[4];
}
```

parameters
return address
calling stack pointer
buf[...]
...
NX bit

bar()