1. With respect to security, it's been said that complexity, extensibility,

and connectivity are the "trinity of trouble" [143]. Define each of these

terms and explain why each represents a potential security problem.

2. What is a validation error, and how can such an error lead to a security

flaw?

3. Provide a detailed discussion of one real-world virus or worm that was

not covered in the text.

4. What is a race condition? Discuss an example of a real-world race

condition, other than the mkdir example presented in the text.

5. One type of race condition is known as a time-of-check-to-time-of-use,

or TOCTTOU (pronounced "TOOK too").

a. What is a TOCTTOU race condition and why is it a security issue?

b. Is the mkdir race condition discussed in this chapter an example

of a TOCTTOU race condition?

c. Give two real-world examples of TOCTTOU race conditions.

6. Recall that a canary is a special value that is pushed onto the stack

after the return address.

a. How is a canary used to prevent stack smashing attacks?

b. How was Microsoft's implementation of this technique, the /GS

compiler option, flawed?

7. Discuss one real-world example of a buffer overflow that was exploited

as part of a successful attack.

8. Explain how a heap-based buffer overflow works, in contrast to the

stack-based buffer overflow discussed in this chapter.

9. Explain how an integer overflow works, in contrast to the stack-based

buffer overflow discussed in this chapter.

10. Read the article [311] and explain why the author views the NX bit as

only one small part of the solution to the security problems that plague

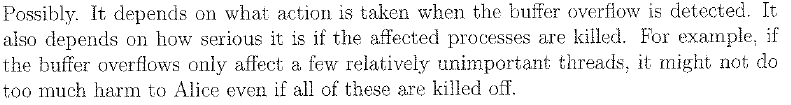
computers today.

11. As discussed in the text, the C function strcpy is unsafe. The C function

strncpy is a safer version of strcpy. Why is strncpy safer but not safe?

12. Suppose that Alice's system employs the NX bit method of protecting against buffer overflow attacks. If Alice's system uses software that is known to harbor multiple buffer overflows, would it be possible for Trudy to conduct a denial of service attack against Alice by exploiting one of these buffer overflows? Explain.

NX bit means there are flags in certain areas of memory as non-executable or executable, which stops an attack from running code in a non-executable region.



13. Suppose that the NX bit method of protecting against buffer overflow

attacks is employed.

a. Will the buffer overflow illustrated in Figure 11.5 succeed?

b. Will the attack in Figure 11.6 succeed?

c. Why will the return-to-libc buffer overflow example discussed in

Section 11.2.1.2 succeed?

14. List all unsafe C functions and explain why each is unsafe. List the

safer alternative to each and explain whether each is safe or only safer,

as compared to its unsafe alternative.

15. In addition to stack-based buffer overflow attacks (i.e., smashing the

stack), heap overflows can also be exploited. Consider the following C

code, which illustrates a heap overflow.

int mainO

{

int diff, size = 8;

char \*bufl, \*buf2;

bufi = (char \*)malloc(size);

buf2 = (char \*)malloc(size);

diff = buf2 - bufi;

memset(buf2, '2', size);

printfCBEFORE: buf2 = %s ", buf 2) ;

memset(bufl, º ' , diff + 3);

printf("AFTER: buf 2 = \*/.s ", buf 2) ;

return 0;

}

a. Compile and execute this program. What is printed?

b. Explain the results you obtained in part a.

c. Explain how a heap overflow might be exploited by Trudy.

16. In addition to stack-based buffer overflow attacks (i.e., smashing the

stack), integer overflows can also be exploited. Consider the following

C code, which illustrates an integer overflow [36].

440 SOFTWARE FLAWS AND MALWARE

int copy .something (char \*buf, int len)

{

char kbuf[800];

if(len > sizeof(kbuf))

{

return -1;

}

return memcpy(kbuf, buf, len);

}

a. What is the potential problem with this code? Hint: The last

argument to the function memcpy is interpreted as an unsigned

integer.

b. Explain how an integer overflow might be exploited by Trudy.

17. Obtain the file overflow.zip from the textbook website and extract

the Windows executable.

a. Exploit the buffer overflow so that you bypass its serial number

check. Turn in a screen capture to verify your success.

b. Determine the correct serial number.

18. Consider the following protocol for adding money to a debit card.

(i) User inserts debit card into debit card machine.

(ii) Debit card machine determines current value of card (in dollars),

which is stored in variable x.

(m) User inserts dollars into debit card machine and the value of the

inserted dollars is stored in variable y.

(iv) User presses enter button on debit card machine.

(v) Debit card machine writes value of x + y dollars to debit card and

ejects card.

Recall the discussion of race conditions in the text. This particular

protocol has a race condition.

a. What is the race condition in this protocol?

b. Describe a possible attack that exploits the race condition.

c. How could you change the protocol to eliminate the race condition,

or at least make it more difficult to exploit?

19. Recall that a trojan horse is a program that has unexpected functionality.

11.7 PROBLEMS 441

a. Write your own trojan horse, where the unexpected functionality

is completely harmless.

b. How could your trojan program be modified to do something malicious?

20. Recall that a computer virus is malware that relies on someone or something

(other than itself) to propagate from one system to another.

a. Write your own computer virus, where the "malicious" activity is

completely harmless.

b. Explain how your virus could be modified to do something malicious.

21. Recall that a worm is a type of malware similar to a virus except that

a worm propagates by itself.

a. Write your own worm, where the "malicious" activity is completely

harmless.

b. Explain how your worm could be modified to do something malicious.

22. Virus writers use encryption, polymorphism, and metamorphism to

evade signature detection.

a. What are the significant differences between encrypted worms and

polymorphic worms?

b. What are the significant differences between polymorphic worms

and metamorphic worms?

23. This problem deals with metamorphic software.

a. Define metamorphic software.

b. Why would a virus writer employ metamorphic techniques?

c. How might metamorphic software be used for good instead of evil?

b-Signature-based detection provides a relatively simple and efficient method for detecting known viruses. At present, most antivirus systems rely primarily on signature detection. Metamorphic viruses are potentially one of the most difficult types of viruses to detect. Such viruses change their internal structure, which provides an effective means of avoiding signature detection. Previous work has shown that a specific and straightforward metamorphic engine can generate viruses for which reliable detection using “static analysis” is NP-complete

24. Suppose that you are asked to design a metamorphic generator. Any

assembly language program can be given as input to your generator,

and the output must be a metamorphic version of the input program.

That is, your generator must produce a morphed version of the input

program and this morphed code must be functionally equivalent to the

input program. Furthermore, each time your generator is applied to the

same input program, it must, with high probability, produce a distinct

metamorphic copy. Finally, the more variation in the metamorphic

copies, the better. Outline a plausible design for such a metamorphic

generator.

442 SOFTWARE FLAWS AND MALWARE

25. Suppose that you are asked to design a metamorphic worm, where each

time the worm propagates, it must first produce a morphed version of

itself. Furthermore, all morphed versions must, with high probability,

be distinct, and the more variation within the metamorphic copies, the

better. Outline a plausible design for such a metamorphic worm.

26. A metamorphic worm that generates its own morphed copies is sometimes

said to "carry its own metamorphic engine" (see Problem 25). In

some situations it might be possible to instead use a standalone metamorphic

generator (see Problem 24) to produce the metamorphic copies,

in which case the worm would not need to carry its own metamorphic

engine.

a. Which of these two types of metamorphic worms would be easier

to implement and why?

b. Which of these two types of metamorphic worms would likely be

easier to detect and why?

27. A polymorphic worm uses code morphing techniques to obfuscate its decryption code while a metamorphic worm uses code morphing techniques to obfuscate the entire worm. Apart than the amount of code that must be morphed, why is it more difficult to develop a metamorphic worm than a polymorphic worm? Assume that in either case the

worm must carry its own morphing engine (see Problems 25 and 26).

28. In the paper [330] several metamorphic malware generators are tested.

Curiously, all but one of the generators fail to produce any significant

degree of metamorphism. Viruses from each of these weak metamorphic

generators are easily detected using standard signature detection

techniques. However, one metamorphic generator, known as NGVCK,

is shown to produce highly metamorphic viruses, and these successfully

evade signature detection by commercial virus scanners. Finally,

the authors show that, in spite of the high degree of metamorphism,

NGVCK viruses are relatively easy to detect using machine learning

techniques—specifically, hidden Markov models [278].

a. These results tend to indicate that the hacker community has, with

rare exception, failed to produce highly metamorphic malware.

Why do you suppose this is the case?

b. It might seem somewhat surprising that the highly metamorphic

NGVCK viruses can be detected. Provide a plausible explanation

as to why these viruses can be detected.

c. Is it possible to produce undetectable metamorphic viruses? If so,

how? If not, why not?

11.7 PROBLEMS 443

29. In contrast to a flash worm, a slow worm is designed to slowly spread

its infection while remaining undetected. Then, at a preset time, all

of the slow worms could emerge and do something malicious. The net

effect would be similar to that of a flash worm.

a. Discuss one weakness (from Trudy's perspective) of a slow worm

as compared with a flash worm.

b. Discuss one weakness (also from Trudy's perspective) of a flash

worm compared with a slow worm.

a-:high probablility of detection

b-**Flash worm** designed to infect entire Internetalmost instantly. Searching for vulnerable IP addresses is theslow part of any worm attack

30. It has been suggested that from the perspective of signature detection,

malware now far outnumbers goodware(Software that is available free of charge but that encourages the user to make a monetary donation ). That is, the number of signatures

required to detect malicious programs exceeds the number of

legitimate(جائز) programs.

a. Is it plausible(qabilee fakar) that there could be more malware than legitimate

programs? Why or why not?

b. Assuming there is more malware than goodware, design an improved signature-based detection system.

31. Provide a brief discussion of each of the following botnets. Include a

description of the command and control architecture and provide reasonable

estimates for the maximum size and current size of each.

a. Mariposa

b. Conficker

c. Kraken

d. Srizbi

32. Phatbot, Agobot, and XtremBot all belong to the same botnet family.

a. Pick one of these variants and discuss its command and control

structure.

b. These botnets are open source projects that are distributed under

the GNU General Public License (GPL). This is highly unusual

for malware—most malware writers are arrested and jailed if they

are caught. Why do you suppose that the authors of these botnets

are not punished?

33. In this chapter, the claim is made that "botnets are ideal for use in

various attack-for-hire scenarios." Spam and various DoS attacks are

the usual examples given for the uses of botnets. Give examples of other

types of attacks (other than spam and DoS, that is) for which botnets

would be useful.

444 SOFTWARE FLAWS AND MALWARE

34. After infecting a system, some viruses take steps to cleanse the system

of any (other) malware. That is, they remove any malware that has

previously infected the system, apply security patches, update signature

files, etc.

a. Why would it be in a virus writer's interest to protect a system

from other malware?

b. Discuss some possible defenses against malware that includes such

anti-malware provisions.

35. Consider the code that appears in Table 11.5.

a. Provide pseudo-code for a linearization attack on the code in Table

11.5.

b. What is the source of the problem with this code, that is, why is

the code susceptible to attack?

36. Consider the code in Table 11.5, which is susceptible to a linearization

attack. Suppose that we modify the program as follows:

int main(int arge, const char \*argv[])

{

int i;

boolean flag = true;

char serial[9]="S123N456\n";

i f ( s t r l e n ( a r g v [ l ] ) < 8)

{

printf("\nError try again.\n\n");

exit(O);

}

for(i = 0 ; i < 8; ++i)

{ i f ( a r g v [ l ] [ i ] != s e r i a l [ i ] ) flag = false;

}

i f ( f l a g )

{

printf("\nSerial number is correct!\n\n"); }

}

Note that we never break out of the for loop early, yet we can still

determine whether the correct serial number was entered. Explain why

this modified version of the program is still susceptible to a linearization

attack.

11.7 PROBLEMS 445

37. Consider the code in Table 11.5, which is susceptible to a linearization

attack. Suppose that we modify the program so that it computes the

hash of the putative serial number and we compare this hash to the

hash of the actual serial number. Is this modified program susceptible

to a linearization attack? Explain.

38. Consider the code in Problem 36, which is susceptible to a linearization

attack. Suppose that we modify the program so that it computes a

random delay within each iteration of the loop.

a. This program is still susceptible to a linearization attack. Why?

b. An attack on this modified program would be more difficult than

an attack on the code that appears in Problem 36. Why?

39. Consider the code in Table 11.5, which is susceptible to a linearization

attack. Suppose that we modify the program as follows:

int main(int arge, const char \*argv[])

{

int i;

char serial[9]="S123N456\n";

if(strcmp(argv[l], serial) == 0)

{

printf("\nSerial number is correct!\n\n");

}

}

Note that we are using the library function stremp to compare the input

string to the actual serial number.

a. Is this version of the program immune to a linearization attack?

Why or why not?

b. How is stremp implemented? That is, how does it determine

whether the two strings are identical or not?

40. Obtain the Windows executable contained in l i n e a r . z i p (available at

the textbook website).

a. Use a linearization attack to determine the correct eight-digit serial

number.

b. How many guesses did you need to find the serial number?

c. What is the expected number of guesses that would have been

required if the code was not vulnerable to a linearization attack?

41. Suppose that a bank does 1000 currency exchange transactions per day.

a. Describe a salami attack on such transactions.

b. How much money would Trudy expect to make using this salami

attack in a day? In a week? In a year?

c. How might Trudy get caught?

42. Consider the code in Table 11.5, which is susceptible to a linearization

attack. Suppose that we modify the program as follows:

int main(int arge, const char \*argv[])

{

int i;

int count = 0;

char serial[9]="S123N456\n";

i f ( s t r l e n ( a r g v [ l ] ) < 8)

{

printf("\nError try again.\n\n");

exit(0);

}

for(i = 0; i < 8; ++i)

{

if(argv[l][i] != serial[i])

count = count + 0;

else

count = count + 1;

}

if(count == 8)

{

printf("\nSerial number is correct!\n\n");

}

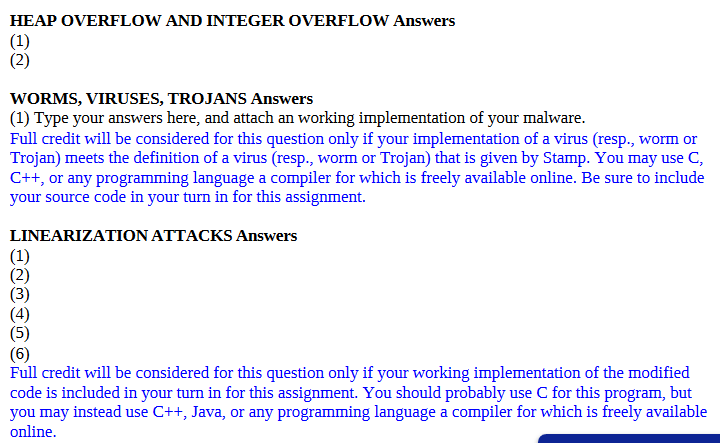
}

Note that we never break out of the for loop early, yet we can still

determine whether the correct serial number was entered. Is this version

of the program immune to a linearization attack? Explain.

43. Modify the code in Table 11.5 so that it is immune to a linearization attack. Note that the resulting program must take exactly the same amount of time to execute for any incorrect input. Hint: Do not use any predefined functions (such as stremp or strnemp) to compare the input with the correct serial number.



44. Read the article "Reflections on Trusting Trust" [303] and summarize

the author's main points.