Midterm Exam

CS161 Computer Security, Fall 2008

Name _____

Email address _____

Score _____ / 55 points

Please do not open this exam until you are instructed to begin work. Until then, you may read these instructions and fill in your name and email address in the provided blanks.

For each short answer question, write your answer in the space beneath the question. If you need more space, you can use the back of another page (being careful to label it with the number of corresponding question).

If you have questions during the exam, please raise your hand and one of the exam administrators will come to you.

1. Security Goals

(5 points) Please classify each of the following as a violation of *(A) confidentiality, (B) integrity, (C) availability, or (D) non-repudiation* :

(a) *(1 point)* A copies B's homework

(b) *(1 point)* A crashes B's operating system

(c) *(1 point)* A changes the amount on B's check from 100 to 1000

(d) *(1 point)* A forges B's signature on a land acquisition contract

(e) *(1 point)* A registers the domain name PrenticeHall.com and refuses to let the publishing house buy or use the domain name.

2. True/False with Reason

(8 points) For each of the statements below, state whether they are true or false. Explain your answer with a short justification, at most 2 sentences long.

(a) *(2 points)* T/F: In a C program, format string bugs alone cannot be exploited for control-hijacking attacks.

(b) *(2 points)* An OS running on a new Intel processor marks the stack memory pages as non-executable using NX bit feature[1].

T/F : This mechanism prevents attackers from getting a root shell by exploiting stack-based buffer overflows.

[1]Briefly, the NX bit (no-execute bit) is a flag added to every memory page indicating whether the contents of the page can be executed or not. When NX bit is set for a page, the processor never executes any byte in the page

(c) *(2 points)* T/F : Consider an idealized, perfectly random hash function h, i.e h is selected uniformly at random from all functions mapping $\{0,1\}^*$ to $\{0,1\}^k$. Given $y = h(x)$ for some x, the expected value for the number of attempts required to find any preimage x', such that $h(x') = y$, is on the order of $2^{k/2}$.

(d) *(2 points)* T/F : Consider the function $f(x) = x^2 - 1 \ (mod\ p)$, where p is a large prime and x is an integer between 1 and $(p-1)$. $f(x)$ is a pre-image resistant function.

3. Memory Errors and Defenses

(10 points) The following code runs on a 32-bit x86 based Linux workstation. The stack grows downwards, and the C compiler allocates four bytes for the saved frame pointer register in the start of the stack frame (activation record) of a function, followed by the space for the local variables of the function. You can assume that no extra space is allocated for saved registers and no alignment/padding bytes are added to the activation record.

```
1:  struct bar   { unsigned int a;
2:                 unsigned int b;
3:                 int c;       };
4:  union foobar { struct bar arr[20];
5:                 char buf[160]; };
6:  int func (int i, char* p, int data) {
7:      union foobar x;
8:      if (((i<0) || (i>20)) && p)
9:          strncpy (&x.buf[0], p,
10:                   sizeof(unsigned int)*160);
11:      else {
12:          x.arr[i].b  = data;
13:          x.arr[i].c  = data; }
14: }
```

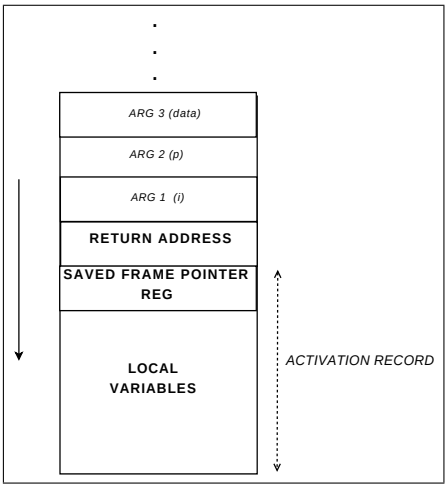

Figure 1: (Left) Vulnerable code (Right) Stack frame layout for function `func`

(a) *(6 points)* This code is vulnerable to a control hijacking attack in at least 2 different ways. Write 2 invocations of `func` with appropriate parameters that result in 2 different attacks. In both attacks, return from the function `func` transfers control directly to the `sleep` system call code, located at address `0x80484cc`. Different attacks here means that the target control variable is overwritten at different line numbers in the two attacks.

(b) *(4 points)* Does a canary based defense protect against both of these attacks. Explain why or why not? You can assume that the compilers stores a 4 byte canary right after (or immediately below) the saved frame pointer in the activation record.

4. User/Kernel Pointer Bugs

(5 points) Suppose you find the following soundcard device driver code present in the source code of your operating system. The parameters `cmd` and `arg` contain user controlled data passed to this code through a device specific `ioctl` system call.

```
struct stream {struct streamdata *databuf; int size;};
struct msg    {struct streamdata *databuf; .... };
struct msghdr {struct stream * streams;  int num_streams; ...};
....
1: int sound_ioctl (unsigned int cmd, unsigned long arg)
2:  {
3:  ...
4:      switch (cmd)  {
5:        case SOUND_MIXER_WRITE_IGAIN :
6:           if (copy_from_user (&msg_sys, (struct msghdr *)arg, sizeof(msghdr)))
7:           return -EIO;
8:    ...
9:    struct msg ** mlist = kmalloc (sizeof(struct msg*) * 6, ...);
10:    parge_msg (msg_sys, mlist, 5);
11:  }
12:
13:  void
14:  parse_msg (struct msghdr msg_sys, void **msglist, int size) {
15:    if (size >= 0) {
16:      while (size && msglist)
17:      {
18:        ...
19:        msglist[size] = kmalloc (sizeof (struct msg), ...);
20:        ...
21:        if(copy_from_user (msglist[size]->databuf,
22:                           msg_sys.streams[size].databuf,
23:                           sizeof (struct streamdata)))      {
24:               res = -EIO;  return;       }
25:        ...
26:        size--;
27:      } ... } ... }
```

You can assume that `copy_from_user (char *dest, char* src, size_t size)` is a function that checks that the 'src' pointer points to valid (allocated) memory in user space, before copying 'size' bytes of data pointed to by 'src' to memory pointed to by 'dest'. It checks that 'dest' points to valid memory in the kernel space. If successful, it returns `0` or else it returns a non-zero value. `kmalloc` can be assumed to have the same interface as the `malloc` user-level C library function.

Identify a user/kernel pointer bug in the above code. Precisely point out the user controlled pointer expression that is accessed unsanitized, and the line number where the unsafe access occurs.

5. Fuzzing

(8 points) Consider the following program:

```
1.  int f (int x) {
2.    return 2 * x;
3.  }
4.
5.  int target (int x, int y) {
6.    int buf[40];
7.
8.    if ((x >= 0) && (y >= 0)) {
9.       if (f(x) == x + 10) {
10.        if (y < 16) {
11.        buf [x + f(y)] = 0;
12.        }
13.      }
14.   }
```

(a) *(2 points)* Suppose we employ blackbox fuzzing to function `target`, to check if it vulnerable to a buffer overflow. Blackbox fuzzing creates new test cases in each run with random values for the parameters to `target` with the goal of crashing the program. How many test cases (approximately) are needed to generate a crash at line 11.

(b) *(4 points)* Suppose we use a symbolic execution based automatic test case generation technique for whitebox fuzzing. Let us denote the inputs `x` and `y` with symbolic variables x_0 and y_0.

Show the path constraints P on the symbolic input necessary to reach line number 11.

Write the security assertion Q to prevent a buffer overflow at line 11, in terms of the symbolic inputs.

(c) *(2 points)* By solving the constraints $(P \wedge \neg\, Q)$, give an instance of values for x_0 and y_0 that are necessary and sufficient to cause a buffer overflow on line 11.

6. One-Time Pad

(10 points) In this problem, your task is to encrypt a message $M \in \{0, 1, 2, 3, 4\}$ using a shared random key $K \in \{0, 1, 2, 3, 4\}$.

(a) *(5 points)* Suppose you do this by representing K and M using three bits (000, 001, 010, 011, 100) each, and then XORing the two representations. Given a ciphertext 101, what can you infer about the corresponding plaintext? Do you think this scheme has the same security guarantees as the one-time pad? Explain.

(b) *(5 points)* Suggest an alternate encryption algorithm E using only the arithmetic addition $(+)$ and modulus (mod) operations, that performs the above task, providing the same gaurantees as a one-time pad scheme. You should not change the message space $\{0, 1, 2, 3, 4\}$ or the key space $\{0, 1, 2, 3, 4\}$. Instead, design your encryption algorithm such that $E(K, M)$ is a secure encryption of M, when K and M are distributed as above.

7. Secret Sharing

(9 points) Alice is the owner of a company. In a new public key encryption scheme (known as ElGamal's encryption scheme), Alice randomly selects her private decryption key x (where x is an integer such that $1 \leq x < (p-1)$) and computes her public encryption key as $y = g^x \ (mod\ p)$, where p is a large prime and g is a publicly known generator of $\mathbb{Z}_p^*$. $\mathbb{Z}_p^*$ is the set of non-negative integers below p that are relatively prime to p.

Encryption. To encrypt a message $m \in \mathbb{Z}_p^*$, Bob first picks a random integer $1 \leq k < p-1$ such that $\gcd(k, p-1) = 1$ and obtains Alice's public key y. Next he computes $r = g^k \ (mod\ p)$ and $s = my^k \ (mod\ p)$, and sends $C = (r,s)$ as the ciphertext.

Decryption. Alice decrypts the ciphertext (r, s) as follows : She computes $r' = r^x \ (mod\ p)$, and then computes t such that $r't \equiv 1 \ (mod\ p)$. She recovers m by computing $st \ (mod\ p)$.

(a) *(3 points)* Show that the computation $st \ (mod\ p)$ indeed recovers m.

(b) *(6 points)* Suppose Alice wishes to split her private key x into three shares for three board of directors of her company. Construct a secret sharing scheme such that given a ciphertext C encrypted using the above ElGamal encryption scheme, the three board of directors can together decrypt the ciphertext C without any of them explicitly reconstructing x in the process. Additionally, any two of the three board of directors together will not learn anything more about x than what is publicly known and will not be able to decrypt C.

(b.1) Show behow how Alice would split x into the three shares.

(b.2) show how the three board of directors together decrypt C.