CS 491/591 Practice Problems

Fall 2015

1. Basic Notions of Security (10 points)

List the three basic properties for security below. For each property, give an example of an attack that denies or degrades it and a defense that maintains it.

- **C**
- **I**
- **A**

2. Ring 0 vs UID 0 (10 points)

Explain the difference between running in ring 0 on x86 and running as UID 0 in Linux. Give an example of something that each one enables, but the other does not.

3. x86 Instructions (10 points)

(a) What is the binary opcode for the nop ("no op") instruction?

(b) Give two different x86 assembly instructions that will zero the contents of register `%eax`, without using a move instruction such as `movl` or `movb`.

(c) What is the x86 assembly instruction for making a system call?

4. UNIX Permissions (10 points)

Alice, Bob, and Ted have user accounts on a Linux system. Use the provided list of group memberships and the directory listing to answer the following questions.

Listing 1: /etc/group

```
users: alice, bob, ted
insiders: alice, bob
```

Listing 2: Directory Listing from `ls -l`

```
-r-xr-sr-x   root   insiders   /bin/check
drwx--x---   bob    users      /home/bob
drwxrwx---   bob    insiders   /home/bob/private
-rwsr-x---   bob    insiders   /home/bob/private/prog
-rw-------   bob    insiders   /home/bob/private/file.txt
drwx--x---   alice  users      /home/alice
drwxr-x--x   ted    users      /home/ted
```

(a) Create a Lampson-style access matrix showing the access each user has to each resource in the system *if the user can subvert the programs* `check` *and* `prog`.

(b) How is your answer to part (a) different from what was intended in the system?

(c) Describe a series of steps that could be used to exploit this difference to steal or corrupt a protected resource.

5. Program Layout in Memory **(10 points)**

```
#include <stdlib.h>

int test(int i) {
  int rc = i;
  if( i < 0 )
    rc = -1 * rc;
  return rc;
}

int func(int a, int b) {
  int num_i = 16;
  int num_c = test(a+b);
  int *array = (int *) malloc(num_i * sizeof(int));
  char *buf = (char *) malloc(num_c * sizeof(char));
  return 0;
}

int main(int argc, char* argv[])
{
  int x = 2;
  int y = 4;
  int z = func(x,y);
  return 0;
}
```

For the program above, give plausible virtual addresses for these variables:

1. `y`
2. `z`
3. `num`
4. `func`
5. `*buf` (That is, the contents of the pointer `buf`)

6. Address Space Layout Randomization **(10 points)**

Now suppose the system in the previous problem uses a version of ASLR where the code and data sections have 12 bits of randomization and the stack has 16 bits of randomization.

(a) Give another set of plausible addresses that could occur after the program is re-randomized. Your answers here should be consistent with your previous answers.

1. `y`
2. `z`
3. `num`
4. `func`
5. `*buf` (That is, the contents of the pointer `buf`)

(b) If the program is re-randomized after every attempted attack, how many attempts, on average, will a return-oriented programming attack require before the adversary can execute his first ROP gadget? Why?

7. Virtual Memory **(10 points)**

Consider the following C program running on 32-bit x86 Linux with 4KB pages.

```
#include <stdlib.h>

int mult(int factor) {
  return (factor * 1024);
}

char* allocate(int nkbs) {
  int nbytes = mult(nkbs);
  char *ptr = (char *) malloc(nbytes);
  return ptr;
}
```

```
int main(int argc, char* argv[])
{
  char *c = allocate(2);
  char *d = allocate(3);

  return 0;
}
```

(a) How many pages does the OS allocate to a process running this program?

(b) Fill in one row of the table for each page in the process' address space.

Contents (code, data, etc)	**Permission Bits (r/w/x)**	**Virtual Page Number**	**Physical Page Number**

(c) How many additional pages (including page directory and page tables) must the OS allocate in order to run this program and manage its virtual memory?