14-741/18-631: Test 2

Thursday, Oct. 22 2020, 80 minutes

Name:

Andrew ID:

Scores

Problem 1 (25 pts max):

Problem 2 (22 pts max):

Problem 3 (28 pts max):

Problem 4 (10 pts max):

Problem 5 (15 pts max):

Total (100 pts max):

Guidelines

- This exam contains 4 problems and is 6 pages long. Check you have all pages. To help us grade as anonymously as possible, please **do not write your name or any other identifying information on any page other than this cover page**.
- Be neat and concise in your explanations. You won't be penalized for using incorrect grammar, but you will get penalized if we can't understand what you are writing.
- Show your work clearly. If your reasoning (in other words, steps) is correct, but your final answer is wrong, you will receive most of the credit. On the other hand, answers without proof, explanation, or argumentation get no credit, *even if they are correct*.
- This exam is open-book. All reference books, class notes, dictionaries and calculators are allowed. **Searching the Internet during the exam constitutes an academic integrity violation and will be punished by failure of the class ("R") and other disciplinary measures at the discretion of the University.**
- Open-book does not mean open communications. Cheating on the exam (including accessing the Internet) automatically results in failure of the class and will be reported to the University administration. We do enforce university and departmental plagiarism policies strictly.
- It is advantageous to partially answer a question than not attempt it at all.
- Good luck!

1 Access Control (25 pts)

A government agency designed a security enhanced version of Linux, called SL. In addition to the underlying UNIX file system protection, SL has additional enhanced features to enforce stricter access control policies. One feature is typed-based enforcement. SL assigns each process and file a type, and admin specifies type-based access control policies in a configuration file. For example, an admin can define two types, os_t for OS processes and files, and app_t for applications' processes and files. Then the following rules in the configuration file says that OS processes can read and write OS files, and OS processes can read and write application files.

```
allow os_t os_t : file {read, write} ;
allow os_t app_t : file {read, write} ;
```

SL has three modes: `Enforcing`, `Permissive`, and `Disabled`. An admin can choose which mode to use. In `Enforcing` mode SL will enforce its enhanced policies on the system. The denied accesses are written to an audit log file. In `Permissive` mode, SL only checks each access against the enhanced policies and no access is denied. However, any policy violation (accesses that would have been denied in `Enforcing` mode) is logged in an audit log. In the `Disabled` mode, the system will not use enhanced policies in any way.

1. (10 pts) In `Enforcing` mode, when an access is not specified by a rule in the configuration file, what do you recommend SL do (deny or allow) and why? Explain which principle(s) of designing access control system that you are following.

2. (10 pts) Why is it useful for SL to offer 3 modes to the admins? Explain which principle(s) of designing access control system come into play.

3. (5 pts) Why is the audit log useful?

2 Multi-Level Security (22 pts)

Most operating systems have strict access control policies for programs downloaded from the Internet, as they could be malicious. Bob wants to apply the idea of multi-level security to restrict downloaded programs. Bob decides to use two labels: `Internet` and `System`. All the programs downloaded from the internet such as a game and their files will be labeled `Internet` and trusted system programs and files such as processes and files belonging to Windows 10 will be labeled `System`. In addition to be considered trusted, System programs and files also may contain secret information. Explain how should Bob use BLP and Biba properties/policies to protect the secrecy and integrity of system processes and files?

3 An Authentication Function (28 points)

Hospital H wants to implement an access control system so only on-duty doctors are allowed to access patients' files. H has a file containing the username and password of all on-duty doctors. An on-duty doctor is required to login with their username and password. Alice implemented the reference monitor for the access control system in C. She wrote the following `auth()` function to authenticate a user using password. The function when called, asks the user to provide username and password. If the inputs match records on file, the function returns 1, indicating that the user is authenticated; otherwise, the function returns 0.

```
1  struct user_record
2  { char name[10]; //username
3    char passwd[10]; //password
4  };
5
6  int auth( )
7  {
8  // input buffer for username
9   char username[100];
10
11 // an array of onduty doctors' username and password,
12 // for simplicity, assume H has only 5 onduty doctors
13  struct user_record onduty[5];
14
15 // input buffer for password
16  char passwd[10];
17
18 // read onduty doctor's username and password from a file on disk
19    ...
20 // input username from the user and store it in the username buffer
21  char c = getchar();
22  int i = 0;
23  while (c != '\n')
24  { username[i] = c;
25    c = getchar();
26    i++;};
27
28 // input password from the user and store it in the passwd buffer
29  i = 0;
30  c = getchar();
31  while (c != '\n')
32    { passwd[i] = c;
33     c = getchar();
34     i++; };
35
36 // look up the username in the onduty array.
37  ...
38  if (...)  // the input username and password match the record
39   return 1; // return 1
40  else // no match found, output error message
41    {printf(username, "is not an authorized user.\n");
42     return 0;}
43 }
```

You are leading the code-review team that is in charge of reviewing Alice's implementation.

1. (5 pts) Draw the stack layout of `auth()` and indicate the direction of stack growth. You can ignore the padding inserted by the compiler and alignment issues.
2. (12 pts) Identify all the vulnerabilities in the above code snippet. Use line numbers when describing those vulnerabilities.
3. (5 pts) How can an unauthorized user compromise the system by interacting with `auth()` and exploiting the vulnerabilities that you discovered without changing the control flow of the program (e.g., jump to injected shellcode or a ROP gadget)? Describe the scenario(s) in detail.
4. (6 points) How can you eliminate those vulnerabilities? Explain in detail your fixes.

4 Canary (10 pts)

Charlie would like to implement a defense mechanism that uses only one stack canary (one memory slot for each stack frame) to protect both the function return address and function pointers on the stack. How should Charlie layout the stack to best achieve this? Please draw the stack layout of a function activation record and explain how the stack canary is useful for protecting both.

5 Shadows (15 pts)

Bob proposes a defense mechanism that works as follows: shadow copies of the return addresses of functions are stored in the designated area on the heap when a function's activation record is created. Upon return, the shadowed return address on the heap is used as the return address.

1. (5 pts) Explain why common buffer overflows can be prevented by this defense. You can use the following simple program to help you explain.

```
void function(){
    char buf[4];
    // mangles saved return address to 0x44434241
    strcpy(buf, "AAAAAAAAABCD");
}
```

2. (10 pts) Bob decides to only allocate 64 bytes to store these shadow return addresses initially. This 64-byte region is allocated at the beginning of each program's heap (the lowest memory addresses of the heap). This designated area can only store 8 return addresses. To protect longer call chains, Bob decides to dynamically allocate additional 64-byte chunks on the heap when necessary and use the last 8-byte to store a pointer to the next 64-byte memory chunk. For example, when the call chain length grows from 7 to 8 (f1,f2,...f8), the initial chunk, let's call it c1, at the time stores the return addresses of f1-f7. A new chunk of 64-byte, let's call c2, is allocated on the heap, the return addresses of f8 is stored in c2, and the beginning address of c2 is stored as the last entry in c1. Does Bob's design have potential vulnerabilities and how could an attacker exploit the vulnerability?