Homework 2

Due: April 19, 2016 **Points:** 100

Questions

Remember to justify your answers where appropriate.

1. (*10 points*) Consider a computer system with three users: Alice, Bob, and Cyndy. Alice owns the file *alicerc*, Bob the file *bobrc*, and Cyndy the file *cyndyrc*. Alice can read, write, and execute *alicerc*; Bob and Cyndy can read that file. Alice can read *bobrc*; Bob can read and execute it; Cyndy can write it. Finally, Alice has no accesss to *cyndyrc*; Bob can read it; and Cyndy can read and execute it.

   (a) Create the corresponding access control matrix. Represent the read right as "r", the write right as "w", the own right as "o", and the execute right as "x", and assume there are no other rights.
   (b) Cyndy gives Alice permission to read *cyndyrc*, removes Bob's permission to write that file, and Alice removes Bob's ability to read *alicerc*. Show the new access control matrix.

2. (*20 points*) Someone asks, "Since the Harrison-Ruzzo-Ullman result says that the security question is undecidable, why do we waste our time trying to figure out how secure my Linux laptop is?" Please give an answer justifying the analysis of the security of an individual Linux laptop system (or any system, for that matter) in light of the HRU result.

3. (*30 points*) In beginning programming classes, you learned about information hiding, also called encapsulation. Those rules basically state that data structures should be encapsulated so that, if they need to be changed, the interface remains the same, so the user (or calling program) does not know what representation of the abstract data type is used. This can be implemented easily using objects in object-oriented programming languages; in other languages, one must use language constructs carefully (like the use of *static* and files in C).

   As an example, consider the queue library in the Robust Programming handout [Bis11]. The fragile version of the library does not provide information hiding or encapsulation, because the program can access the queue structures directly (through the queue pointers). The robust version does provide encapsulation, because the program cannot access the queue structures, or queue management structure, directly. No pointers or variables are ever exposed.

   Please identify the application security risk in the OWASP reference [Chr11], and the weakness in the CWE reference [OWA13], that best describe the *failure* to encapsulate a data structure as described above.

   The three documents referenced above are available on SmartSite in the Resources > Handouts area.

4. (*40 points*) This problem asks you to implement a buffer overflow attack on a program. In the Resources area of SmartSite (or the Homework area of the nob.cs.ucdavis.edu class web site) is a program *bad.c* (also see below). This program contains a buffer overflow vulnerability; see the call to *gets*(3) at line 13. Your job is to exploit the overflow by providing input to the running process that will cause the program to invoke the function `trap` (which, you may notice, is not called anywhere else). You will know you've succeeded when you run the program, give it your input, and it prints "`Gotcha!`"

   The following questions will help guide you. Please turn in your answers to them, a hex dump of the input you use to call `trap`, and a typescript or screen shot of you running the program *bad*, giving it your input, and showing its output.

   (a) What is the address of the function `trap`()? How did you determine this?
   (b) What is the address on the stack that your input must overwrite (please give both the address of the memory location(s), and their contents)? How did you locate this address?
   (c) What is the address of `buf`?
   (d) The *sled* is the input you give to alter the return address stored on the stack. What is the minimum length your sled must be?

bad.c

This is a listing of *bad.c*.

```
#include <stdio.h>
#include <stdlib.h>

int trap(void)
{
        printf("Gotcha!\n");
        exit(0);
}

int getstr(void)
{
        char buf[12];
        gets(buf);
        return(1);
}

int main(void)
{
        getstr();
        printf("Overflow failed\n");
        return(1);
}
```

Extra Credit

5. (*20 points*) A company publishes the design of its security software product in a manual that accompanies the executable software.

   (a) In what ways does this satisfy the principle of open design? In what ways does it not?
   (b) Given that the design is known, what advantages does keeping the source code unavailable give the company and those who purchase the software? What disadvantages does it cause?

References

[Bis11] Matt Bishop. Robust programming. handout for ECS 153, Computer Security, Mar. 2011.

[Chr11] Steve Christey. 2011 CWE/SANS top 25 most dangerous software errors, Sep. 13, 2011. Available at `http://cwe.mitre.org/top25/`.

[OWA13] OWASP. Owasp top 10 - 2013: The ten most critical web application security risks. Technical report, The Open Web Application Security Project, 2013.