Name: _____ UNI: _____

COMS W4187: Security Architecture and Engineering
Spring 2013

Rules

- Remember to write your name and UNI on the blue exam book.
- **Important: also write your name on this paper**
- You must turn in *both* the exam sheet and the blue book
- Books and notes are allowed during this examination; computers are not permitted.
- This is a time-limited test. All papers must be turned in 75 minutes after the beginning of the test.
- The total points add up to 100 .
- Good luck, and may the Force be with you.

1		5
2		20
3		15
4		15
5		10
6		15
7		20
Total		100

Questions

1. (5 points) One of the questions we always ask when doing security design or analysis is "who's the enemy?" Why do we ask that?

The identity of the enemy tells us their goals (i.e., what we need to protect) and their abilities (i.e., how strong our defenses should be).

2. (20 points) You're implementing a wiki for a non-networked machine. The author of all changes should be properly identified. What permission and privilege mechanisms would you use? Be specific: for any group of files, what permissions should it have?

The wiki files need to be readable by everyone in the group, but writable only by a privileged program. The privileged program — setuid, on Unix — records the real identity associated with each change. Any log files are write- and probably read-protected.

3. (15 points) Your company is going to deploy a new authentication system. They're trying to decide between passwords and client-side certificates. Your boss says that passwords are better, because client-side certificates require much greater verification of identity if replacement credentials are to be issued. What is your response?

The boss is wrong — it's not an identity credential, it's just for access. The password has the same properties, so it has the same requirements.

4. (15 points) Vendor of secure hardware has come up with an innovative way to generate random numbers for 256-bit cryptographic keys. Next to the computer, there's a small "cage" with four coins inside. There's a camera aimed at the coins. When a key is needed, the user is instructed to invert the cage, thus effectively flipping the coins. The user then presses a button; the camera reads the values of the 4 bits. This is repeated as needed. What are the advantages and disadvantages of this scheme?

Advantage: good physical randomness. Disadvantage: too inconvenient to use.

5. (10 points) Many Cyrillic letters look like letters in the Latin alphabet; however, they have different Unicode values. For example, a Cyrillic A is `U+0410`, a B is `U+0412`, etc., while an ASCII A is `U+41`, B is `U+42`, etc. Why is that of interest in this class?

A web site could appear legitimate in the URL bar, and even have a certificate. But whats signed is the Cyrillic name; it wont be the site you think it is.

6. (15 points) You're designing a multiuse smart card — train fares, library access, cable TV descrambling, anything. It's an open architecture; any vendor who wants to load an application onto it can. Is a containment mechanism a good idea? Why or why not?

You need containment to protect applications from each other.

7. (20 points) The following setUID program fragment copies a user-specified file to a temporary file in `/tmp`. It contains two security holes. Find the holes, and describe what code should be inserted. *Do not* write the actual code; what I'm looking for is an explanation.

```
main(int argc, char *argv[])
{
        FILE *in, *out;
        char tmpname[200];

        if (argc < 1) return 1;
        in = fopen(argv[1], "r");
        sprintf(tmpname, "/tmp/temp.%06d", getpid());
        (void) umask(077);
        out = fopen(tmpname, "w");
        if (in != NULL && out != NULL) copyfile(in, out);

        ...
}
```

The temporary file name is predictable. A random value should be used there, or something like `mkstemp()`. Also, the input file is opened without shedding `setuid` privileges. The `fopen()` call should be surrounded by by code to reset the effective UID (`seteduid()`) and reset it. There is *not* a buffer overflow; processIDs are not nearly large enough to fill a 200-byte buffer.