CSCI 3330 Assignment 1

Deadline: September 18, 2025, 10:00pm

1 Background

In this assignment, we will be further exploring Unix permissions, entropy, and randomness. Specifically, by completing this assignment, you will:

- Configure least-privilege access on multiuser Unix systems.
- Quantify password strength using Shannon entropy.
- Recognize and exploit weak randomness in software to understand real-world risk—and how to avoid it.
- Communicate security reasoning with justified calculations and reproducible code.

2 AI policy

You may use AI to learn background concepts. You may not use AI to generate, verify, or iterate on solutions for this assignment.

Allowed:

- Concept definitions and explanations: "What does 'uniformly at random with replacement' mean?", "What is Shannon entropy?", "What does the `time.time()` function return?", "What is a Unix timestamp?"
- Generic examples not derived from this assignment: "Show how to compute entropy on a contrived example," "Walk through directory permissions `r` versus `x` with examples."
- Non-generative tooling: man pages, official documentation, calculators, programming languages, and trying something for yourself.

Handle with care:

- Tools which improve grammar or clarity.
- IDE autocompletion or documentation lookup that does not inject solution content.

Not allowed:

- Any hint, outline, or solution for these specific questions (including rephrased or "similar" versions), e.g., "Walk me through Q2 using 3,790 emojis," "Write a script to brute-force seeds around 1507530118," or "Check my answers."
- Pasting assignment text or numbers into AI to tailor examples.
- Using AI to generate any text, code, calculations, or any other artifacts that you submit.

Ask me questions instead! When in doubt, don't use AI, because **you won't have access to it on tests**, for which these assignments are your preparation.

3 Questions

1. Your Bowdoin shell account (accessible via ssh at `shell.bowdoin.edu`) allows free web hosting for all content in the `$HOME/public_html/` directory. Such content is served by the `tildesites.bowdoin.edu` server. For example, all files in `/home/j.knockel/public_html/csci3330f25/` are automatically served here.

   You wish to configure your permissions so that the web server can see your files but also so that other users on the server can access as few as possible (i.e., according to the principle of least privilege). Specifically, you have the following preferences:

   - `$HOME/public_html`: traversable but not listable
   - `$HOME/public_html/profile.jpeg`: readable
   - `$HOME/public_html/pets/`: traversable and listable
   - `$HOME/public_html/pets/new-kitten.jpeg`: readable

   What is the minimum number of permissions you can grant to each of the following paths so that the above files can be served according to your preferences?

   (a) `$HOME`
   (b) `$HOME/public_html`
   (c) `$HOME/public_html/profile.jpeg`
   (d) `$HOME/public_html/pets/`
   (e) `$HOME/public_html/pets/new-kitten.jpeg`

   For each, answer in the form of `rwxrwxrwx`, e.g., `rw----r--`. Assume that all files should be readable and writable by you and that all directories and their contents should be traversable, listable, and writable by you. (Recall

that a directory is *traversable*, i.e., its files can be accessed, if it has the `x` permission, and recall that the web server must be able to traverse (`x`) every directory down to a file to be able to read the file itself.) Assume also that the web server's user is "www-data" and only group membership is "www-data" and that all files and directories including and in your `$HOME` directory are owned by your username (e.g., `j.knockel`) and the group `cs`.

5. Your password generator is configured to generate passwords with eight lowercase alphanumeric characters (i.e., characters spanning "a" through "z" and "0" through "9") uniformly at random with replacement. Your coworker says that choosing a password with four emojis is more secure. Is she correct? Assume you are using the Unicode 16.0 standard which specifies 3,790 single-code-point emojis. Specifically: What is the entropy (in bits) of your password generating scheme? What is the entropy of your coworker's? Is your coworker's more secure? Justify your answers.

6. Your intern overheard your conversation with your coworker and says that his generator, which produces password phrases made up of five English words, must be more secure than both of yours because every phrase has at least 20 letters and therefore an entropy of at least 94 bits. Is his reasoning sound? Why or why not?

7. You investigate the source code of your intern's generator and find that it generates each phrase as follows:

   ADJECTIVE + NOUN + TRANSITIVE VERB + ADJECTIVE + NOUN.

   For example, one such generated password phrase is as follows:

   UntidySoupShookRecentSediment

   Each word is chosen uniformly at random with replacement from a dictionary of words of that part of speech. The dictionary used by the generator contains 1,500 adjectives, 2,000 nouns, and 1,000 transitive verbs. What is the entropy (in bits) of your intern's generator? Did his generator turn out to be more secure than yours after all? Do you think his is easier to remember than yours?

8. In an attempt to show off how secure his password is, your boss reveals to you his password and asks you to calculate the entropy of it. Why is this a conceptually invalid question? Is there another information theoretic measure he could have meant instead of entropy?

9. What is the entropy of flipping a fair coin? What is the entropy when, instead, the coin is weighted such that it is heads with $\frac{3}{5}$ probability and tails with $\frac{2}{5}$ probability? Show your work. Password generators typically choose characters uniformly at random, but could we improve password generators by weighing some characters more than others?

10. You lost the password to your bitcoin wallet containing $3 million USD of bitcoins. However, there is hope. When reverse engineering the code of your password generator, you discovered that it randomly generates passwords as follows:

```
import random
import string
import time

def generate():
    ts = int(time.time())
    r = random.Random(ts)
    password = ''
    for _ in range(32):
        password += r.choice(string.ascii_lowercase)
    return password
```

You check and find that your wallet was created on October 9, 2017 2:21:58 AM EDT, which corresponds to a Unix timestamp of 1507530118. You remember that you generated your password around that time.

Your password, when recovered, will correspond to the following SHA256 hash:

```
1e38ba2a745042f0dbd46639a3947951bc2ceff965f8b2c1a858cd6bc4cfc72b
```

That is to say, for the correct `password`,

```
    hashlib.sha256(password.encode('utf-8')).hexdigest() == \
'1e38ba2a745042f0dbd46639a3947951bc2ceff965f8b2c1a858cd6bc4cfc72b'
```

What is the password to your bitcoin wallet, and at what time (Unix timestamp) was it generated? How many seconds before or after the wallet creation was the password generated? On a typical Unix machine, what would be a better seed source than the one used by your password generator? As part of your answer, submit the code that you wrote to crack your bitcoin wallet password.

(This exercise is based on a real story.)

4 Submission

Email your answers in a single `*.txt` or `*.pdf` file attachment to my email address in the Syllabus. If you would like to submit handwritten material, you may take a photo or scan your work as long as it is clearly legible. Such photos or scans must be included in your single submitted `*.pdf`, not as external files.