Assignment 1 Guidelines

Do this assignment on your own. The assignment is due by **Feb 12, 2026 at 11:59 PM Eastern time**. Zip each of your files into a directory with the naming convention **Lastname_Firstname.zip**. You can also use **tar.gz** (e.g., Lastname_Firstname.tar.gz).

- Make sure that your zip file can be uncompressed and that every file is readable with a text editor. Corrupted files or files that do not follow the naming convention will count as zero points.
- A complete assignment zip file should contain 4 folders (one for each part) with the naming convention **Part_x** where x is the part's number (e.g., Part_1). The folders should contain each part's output files: one for part 1, four files for part 2, and three files for parts 3 and 4, following the instructions below. Each part should have a separate makefile dedicated to the given part of the assignment (e.g., the makefile intended for part 3 should be in the folder for part 3). Failure to report the correct files/folder names will result in points deduction.

Part 1: Bandit_1 (25 pts)

The goal of this assignment is to familiarize yourself with accessing a Linux environment via SSH (that we will discuss later in class), along with developing skills in command line interaction and learning about "capture-the-flag" style wargames.

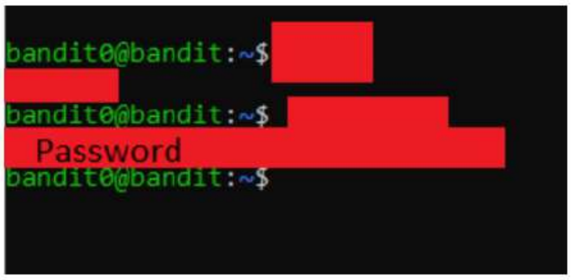
To do this, you will solve the first 5 levels (in other words, log into level 6) on the overthewire.org Bandit challenges.

To track your progress, download the template of the report file of Part 1, and fill in the requested sections. **Take screenshots of the series of commands you ran to find the password for each level. Also, take screenshots of you entering the password to reach the next level.** Your final report should include:

1. A screenshot of the steps taken to obtain the password for the next level up to and including level 6 (i.e., log into level 6, but you do not need to solve for the level 7 password).
2. A screenshot showing the act of logging into all accounts up to and including bandit6 on the OverTheWire bandit server.
3. A brief description of what you did (demonstrate that you understand what the commands you ran did rather than simply restating the list of commands).

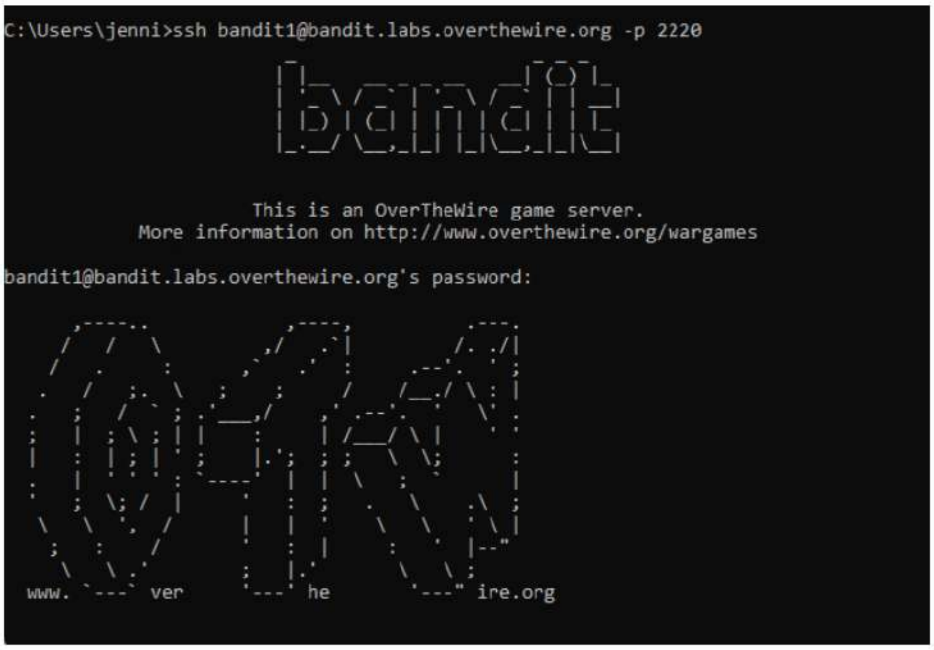
For example, the level 0 -> 1 section of your report may look like the following (without the redactions on the solutions):

1. A screenshot of the steps to obtain the password for the next level (yours should not be redacted and should include the actual password and all steps to obtain it) :



```
bandit0@bandit:~$ [redacted]
[redacted]
bandit0@bandit:~$ [redacted]
Password
bandit0@bandit:~$
```

2. A screenshot of successfully logging into the next level (it is ok if the password is not visible here, but there should be at least a partial banner to demonstrate that the attempt was successful. Also, include the ssh command in the screenshot.):



```
C:\Users\jenni>ssh bandit1@bandit.labs.overthewire.org -p 2220
                         _                     _ _ _
                        | |__   __ _ _ __   __| (_) |_
                        | '_ \ / _` | '_ \ / _` | | __|
                        | |_) | (_| | | | | (_| | | |_
                        |_.__/ \__,_|_| |_|\__,_|_|\__|


                      This is an OverTheWire game server.
            More information on http://www.overthewire.org/wargames

bandit1@bandit.labs.overthewire.org's password:

  www.  `---`  ver      `---' he        `---" ire.org
```

3. In your own words, describe what you did to obtain the password (do not simply list the commands, demonstrate that you understood what those commands accomplished).

Note that Bandit is an open system, and the goal of this assignment is to practice and develop your own skills, so be honorable and do not read walkthroughs. **Failure to report screenshots or manipulated screenshots will result in points deduction.**

Submission Instructions for Part 1

The report file with a detailed description with screenshots of each level should be included in the **Lastname_Firstname** zip file, **Part_1** folder. Points will be deducted for missing content and mislabeled folders.

Part 2: Makefiles (25 points)

All of the coding assignments in this course allow you to write your assignment in C or Python programming language. To allow this, you will need to write a **Makefile** that creates an executable file based on your source code.

In this assignment, you'll practice writing a Makefile for two different types of programs, one a C program and the other a Python program. Errors when executing the Makefile or failure in creating the correct compiled file for the given examples will result in points deducted. Here are some Makefile resources:

- https://www.cs.swarthmore.edu/~newhall/unixhelp/howto_makefiles.html
- https://web.stanford.edu/class/archive/cs/cs107/cs107.1174/guide_make.html
- https://www.gnu.org/software/make/manual/make.html

Note: To test your binaries (**both C and Python**), we will run **`"make -B -f [makefilename]"`** where [makefilename] is the file name for the subsection, so make sure that you follow the requirements for the make file name (the -f flag allows nonstandard names). **Points will be deducted if your makefile does not compile with the given command.** See rubric for the details.

C Program Makefile

The C Program Makefile must compile the example file c_program.c into the executable `c_program`.

Assume that your C Program Makefile is in the same directory as the `c_program.c`, and therefore, when you run `make` in that directory, the compiled program `c_program` is created using `gcc`. Also, your C Program Makefile must recompile the binary when `c_program.c` changes.

Note: Make sure that the resulting binary is called c_program and that the Makefile compiles c_program.c (failure in doing so will result in point deductions).

Python Makefile

The Python Makefile uses the example file python_program.py to create an executable file called `python_program`.

Assume that your Python Makefile is in the same directory as the `python_program.py`, and therefore, when you run `make` in that directory, your Python Makefile will create a `python_program` executable. Also, your Python Makefile must recreate `python_program` when `python_program.py` changes. Note that your makefile should not just run the `python_program.py` script.

There are several ways to approach this. For instance, you can take advantage of the shebang functionality, which allows a file that can be interpreted to be executed. This requires that the file is executable (`chmod +x filename`).

Submission Instructions for Part 2

The Python Makefile as `Makefile.python`, the C Makefile as `Makefile.c` and the example files should be included in the **Lastname_Firstname** zip file, folder **Part_2**. Points will be deducted if the Makefile does not compile with the given command or it produces the wrong output.

NOTE: Avoid using Pip, PipInstaller, or other related programs when submitting a Python makefile.

Part 3: Modified Shift Cipher (25 pts)

As a crypto-breaking warmup, you'll be given a ciphertext that's been encrypted with a modified Shift Cipher. You'll be given the ciphertext, and your goal is to programmatically derive the key (write your own program) and show the decrypted ciphertext. You can submit your code in C or Python.

Get your ciphertext **here**. The first letter of your surname and the first 2, 3, or 4 letters of your first name determine the ciphertext to decrypt (e.g., John Smith = SJO).

Assumptions on the ciphertext:
- No distinctions between lower and upper cases.
- Spaces, commas, points, question marks, and apostrophes are not encrypted.
- Use the English alphabet: ABCDEFGHIJKLMNOPQRSTUVWXYZ

- Special characters, such as `@`, `#`, `$`, `%`, `^`, `&` have been added to obscure the ciphertext. You must account for them in your code. Your code **will be tested with** inputs that do contain the **special characters**.

The output of your program will depend on which method you use to find the key. For example, if you use a brute force attack, the output of your code should include all the possible solutions for the given ciphertext. The key and correct plaintext should be included in the Part 3 report.

IMPORTANT NOTE: your code should be able to break any ciphertext that follows the assumptions above, not only the given ciphertext!

Feel free to double-check your work with online Caesar Cipher breaking tools.

Submission Instructions for Part 3

Submit your code (including comments and a Makefile to generate an executable named `mc_cipher`) and a brief report (max 2 pages) into **Lastname_Firstname.**zip file, folder **Part_3**. Include the makefile for part 3 code **in the Part_3 folder**.

NOTE: Avoid using Pip, PipInstaller, or other related programs when submitting a Python makefile.

The report for Part 3 (template) should include:

- A brief description of what your code does, including instructions on how to run the Makefile and execute the program as in a README file.
- Which method did you use to break the cipher with the assumptions made.
- Describe another method to break the cipher different from the one implemented, and indicate which modifications should be made to your code to accomplish the task (clearly indicate the lines of codes and their modifications).
- The key for the given ciphertext and shift direction (left AND right). For example: "Key = 3 left, 18 right" and the plaintext.

If additional files are needed for the execution of the program, those files must be indicated in the README file (with the description of the usage) and added to the appropriate folder Part_3.

Points will be deducted if 1) The code does not compile, 2) The Makefile is incorrect or does not output the correct compiled file, 3) The code output does not match the correct decrypted text, 4) The information declared in the report does not correspond to the code, 5) Part of the code is missing or incorrect, 6) The code is plagiarized or copied 7) the output does not handle the requested input. See the rubric for the details.

Part 4 — Vigènere Cipher (25 points)

As a second crypto-breaking warmup, you'll be given a ciphertext that's been encrypted with a Vigenère Cipher (as discussed in class). You'll be given the ciphertext, and your goal is to derive the key.

To get your ciphertext **here**. The first letter of your surname and the first 2, 3, or 4 letters of your first name determine the ciphertext to decrypt (e.g., John Smith = SJO).

Assumptions on the ciphertext:

- No distinctions between lower and upper cases.
- Spaces, commas, points, question marks, and apostrophes are not encrypted.
- Use the English alphabet: ABCDEFGHIJKLMNOPQRSTUVWXYZ
- The key is max 4 letters (not necessarily a word)

If additional files are needed for the execution of the program, those files must be indicated in the README file (with the description of the usage) and added to the appropriate folder Part_4.

IMPORTANT NOTE: <u>your code should be able to break any ciphertext that follows the assumptions above, not only the given ciphertext!</u>

Feel free to double-check your work with online Vigenère Cipher breaking tools

Submission Instructions for Part 4

Submit your code (including comments and a Makefile to generate an executable named `v_cipher`) and a brief report (max 2 pages) into **Lastname_Firstname** zip file, folder **Part_4**. Include the makefile for part 4 code **in the Part_4 folder**.

NOTE: Avoid using Pip, PipInstaller, or other related programs when submitting a Python makefile.

The report for Part 4 (template) should include:

- A brief description of what your code does, including instructions on how to run the Makefile and execute the program as in a README file.
- Which method did you use to break the cipher with the assumptions made.
- Describe which modifications should be made to your code if the attacker does not know the key length (clearly indicate the lines of codes and their modifications).
- The key and the plaintext.

Points will be deducted if: 1) The code does not compile, 2) The Makefile is incorrect, 3) The code output does not match the correct decrypted text, 4) The information declared in the report does not correspond to the code, 5) Part of the code is missing or incorrect, 6) The code is plagiarized or copied. See the rubric for the details.

Rubric:

Part 1: Bandit_1 (25 pts)

2.5 points for every correct password getting method/description (2.5*5 (from level 1 to 2 to level 5 to 6) = 12.5 pts)

2.5 points for all logins (2.5*5 (from level 1 to 2 to level 5 to 6) = 12.5 pts)

Note: Points will be deducted for unclear descriptions or lack of screenshots showing the required information.

Part 2: Makefiles (25 points)

C Program Makefile (12.5)

Running make generates an executable c_program from a file c_program.c in the same folder (6.25)

Changing the c_program.c file and rerunning make will generate a new executable c_program based on the changed .c file. (6.25)

Note: For programming sections, if your files do not follow directory and file naming conventions, it will be counted as a zero.

Python Makefile (12.5)

Running make generates an executable python_program from a python_program.py file in the same folder (6.25)

Changing python_program.py and rerunning make will generate a new executable based on the changed.py file (6.25)

Note: For programming sections, if your files do not follow directory and file naming conventions, it will be counted as a zero.

Part 3: Modified Shift Cipher (25 pts)

The correct key and plaintext are given for the assigned ciphertext (2.5 points)

Description of the method used is clear and reasonable (2.5 points)

Description of the method used matches the provided code (5 points)

Description of the alternative method and the modifications are correct (5 points)

The code implements a reasonable method for finding the key and could be used to find keys for other ciphertexts (5 points)

Make file/code/python version correctness (5 pt)

Note: For programming sections, if your files do not follow directory and file naming conventions, it will be counted as a zero.

Part 4: Vigènere Cipher (25 pts)

The correct key is given for the assigned ciphertext (2.5 points)

Description of the method used is clear and reasonable (2.5 points)

Description of the method used matches the provided code (5 points)

Description of the alternative method and the modifications are correct (5 points)

The code implements a reasonable method for finding the key and could be used to find keys for other ciphertexts (5 points)

Make file/code/python version correctness (5 pt)

Note: For programming sections if your files do not follow directory and file naming conventions, it will be counted as a zero.