

# CS 7349

## Data and Network Security

### Homework #3

Name: Bingying Liang  
ID: 48999397

Due: Mar 20 2024

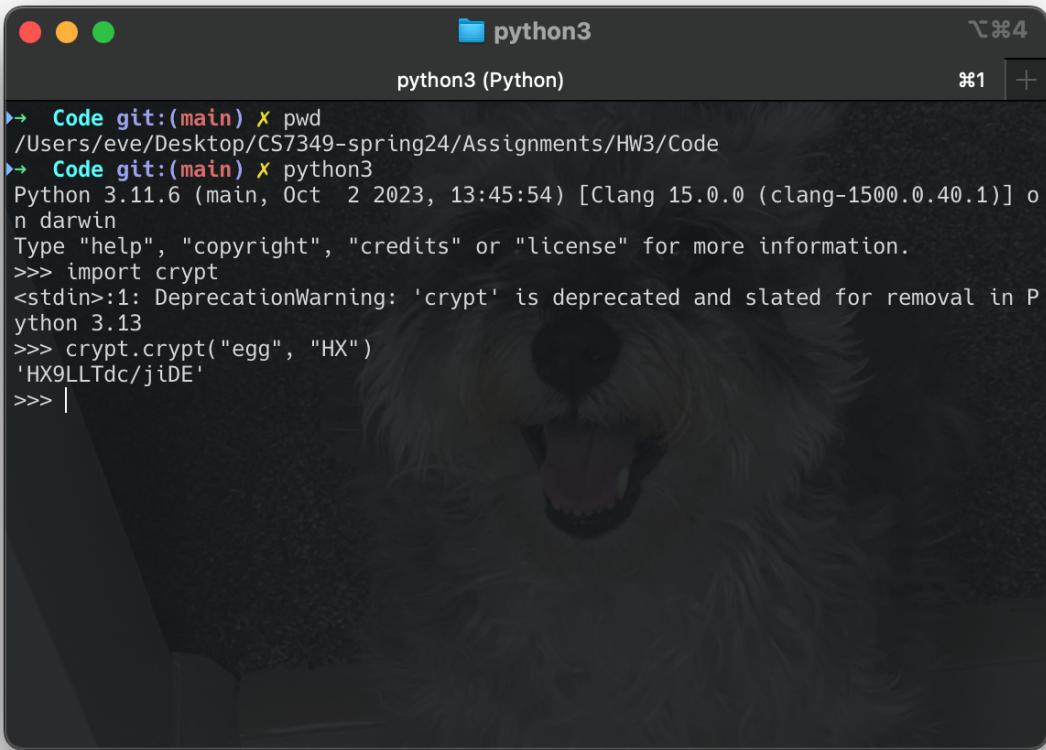
Name: Bingying Liang   
**What to Submit**

Your final submission shall be a single pdf document that includes this document, screen captures of your exercises plus your answers to each of the written questions (if any). Note that you are expected to clearly label each section so as to make it clear to the instructor what files and data belong to which exercise/question.

Collaboration is expected and encouraged; however, each student must hand in their own homework assignment. To the greatest extent possible, answers should not be copied but, instead, should be written in your own words. Copying answers from anywhere is plagiarism, this includes copying text directly from the textbook. Do not copy answers. Always use your own words. For each question list all persons with whom you collaborated and list all resources used in arriving at your answer. Resources include but are not limited to the textbook used for this course, papers read on the topic, and Google search results. Note that ‘Google’ is not a resource. Don’t forget to place your name on the document.

#### Exercise 1: UNIX Password Cracker

The goal of this exercise is to write a password cracker for the UNIX file system. UNIX stores all passwords in the file /etc/passwd. Well, it doesn’t store the password itself. Instead, it stores a *signature* of the password by using the password to encrypt a block of zero bits prepended by a *salt* value with a one-way function called `crypt()`. The result of the `crypt()` function is stored in the /etc/passwd file. For example, for a password of “egg” and salt equal to “HX”, the function `crypt('egg', 'HX')` returns HX9LLTdc/jiDE.



The screenshot shows a macOS terminal window titled "python3". The window has three tabs at the top: "python3 (Python)", "⌘1", and a plus sign. The main pane displays the following Python code:

```
>>> Code git:(main) ✘ pwd
/Users/eve/Desktop/CS7349-spring24/Assignments/HW3/Code
>>> Code git:(main) ✘ python3
Python 3.11.6 (main, Oct 2 2023, 13:45:54) [Clang 15.0.0 (clang-1500.0.40.1)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import crypt
<stdin>:1: DeprecationWarning: 'crypt' is deprecated and slated for removal in Python 3.13
>>> crypt.crypt("egg", "HX")
'HX9LLTdc/jiDE'
>>> |
```

When you try to log in, the program `/bin/login` takes the password that you typed, uses `crypt()` to encrypt a block of zero bits, and compares the result of this function with the value stored in the `/etc/passwd` file.

The security of this approach rests on both the strength of the `crypt()` function and the difficulty in guessing a user's password. The `crypt()` algorithm has proven to be highly resistant to attacks. Conversely, the user's choices for passwords have been found to be relatively easy to guess, with many passwords being words contained in the dictionary.

To write our UNIX password cracker, we will need to use the `crypt()` algorithm that hashes UNIX passwords. Fortunately, the `crypt` library already exists in the Python 2.79 standard library (on UNIX-based operating systems). (Note: for Windows-based operating systems, you will need to find the correct way to import the UNIX `crypt()` algorithm.) To calculate the encrypted UNIX password signature, we simply call the function `crypt.crypt()` and pass it the password and salt as parameters. This function returns the signature as a string.

A simple dictionary attack involves computing the possible signatures generated for each word in the dictionary with a range of salt values.

Let's create our first password cracker using a dictionary attack.

- 1) Create a file called `cracker.py`. Start your program by reading in the `HW2-passwords.txt` file and, for each password found in the file, iterate through each dictionary word found in the `HW2-dictionary.txt` file and appropriate salt value. Report out the password found, if any, for each user. If no password is found, indicate that no password was found.
- 2) Using literature review, identify from where you can retrieve the salt value used in generating the signature.

### Solution:

- 1) The code and result are in the following[1]:

```

eve@Eves-Air:~/Desktop/CS7349-spring24/Assignments/HW3/Code
..ents/HW3/Code (-zsh)

▶ Code git:(main) ✘
▶ Code git:(main) ✘ ls
HW3dictionary.txt HW3passwords.txt cracker.py
▶ Code git:(main) ✘ cat cracker.py
# Import the crypt library for hashing functions
import crypt

# Define a function to attempt cracking a single encrypted password
def attempt_password_crack(encrypted_password):
    # Extract the first two characters of the encrypted password as the salt
    salt = encrypted_password[:2]
    # Open the dictionary file
    with open('HW3dictionary.txt', 'r') as dictionary_file:
        # Iterate over each word in the dictionary file
        for word in dictionary_file:
            # Remove the newline character at the end of each word
            word = word.strip('\n')
            # Encrypt the word from the dictionary using the extracted salt
            encrypted_word = crypt.crypt(word, salt)
            # If the encrypted word matches the user's encrypted password
            if (encrypted_word == encrypted_password):
                # Password found, print the result and return
                print("[+] Found Password: " + word)
                return True
    # If no password is found after going through the dictionary
    print("[-] Password Not Found.")
    return False

# Define the main function
def main():
    # Open the password file
    with open('HW3passwords.txt') as password_file:
        # Iterate over each line in the password file
        for line in password_file:
            if ":" in line:
                # Extract the username and the encrypted password
                user = line.split(':')[0]
                encrypted_password = line.split(':')[1].strip()
                # Print which user's password is currently being attempted
                print("[*] Cracking Password For: " + user)
                # Call the function to attempt cracking the password
                attempt_password_crack(encrypted_password)

    # Execute the main function when the script is run directly
if __name__ == "__main__":
    main()
▶ Code git:(main) ✘ python3 cracker.py
/Users/eve/Desktop/CS7349-spring24/Assignments/HW3/Code/cracker.py:2: DeprecationWarning: 'crypt' is deprecated and slated for removal in Python 3.13
    import crypt
[*] Cracking Password For: victim
[+] Found Password: egg
[*] Cracking Password For: root
[-] Password Not Found.
▶ Code git:(main) ✘ |

```

- 2) In the process of UNIX system password encryption, the salt is an essential component used alongside the user's plaintext password to generate an encrypted password hash. The purpose of the salt is to enhance the complexity and uniqueness of the encryption. In UNIX password storage, the salt typically comprises the first two characters of the encrypted password hash[2]. The way to retrieve the salt[3], extraction from the Encrypted Password: The salt is the first two characters of the stored encrypted password hash. For example, if the encrypted password is "HX9LLTdc/jiDE", the salt would be "HX".

## Exercise 2 : Zip File Password Cracker

The goal of this exercise is to write a zip file extractor and password cracker. For this exercise, we will use the zipfile library. You may view information about the zipfile library in Python 2.79 by issuing the command `help('zipfile')` to learn more about the library. Pay close attention to the `extractall( )` method. You may use this method to extract the contents from a zip file.

```
python3 (Python)
python3 (Python)
>>> |
```

```

python3 (less)
extractall(self, path=None, members=None, pwd=None)
    Extract all members from the archive to the current working
    directory. `path` specifies a different directory to extract to.
    `members` is optional and must be a subset of the list returned
    by namelist().

getinfo(self, name)
    Return the instance of ZipInfo given `name`.

infolist(self)
    Return a list of class ZipInfo instances for files in the
    archive.

mkdir(self, zinfo_or_directory_name, mode=511)
    Creates a directory inside the zip archive.

namelist(self)
    Return a list of file names in the archive.

open(self, name, mode='r', pwd=None, *, force_zip64=False)
    Return file-like object for `name`.

    name is a string for the file name within the ZIP file, or a ZipInfo
    object.

    mode should be 'r' to read a file already in the ZIP file, or 'w' to
    write to a file newly added to the archive.

    pwd is the password to decrypt files (only used for reading).

    When writing, if the file size is not known in advance but may exceed
    2 GiB, pass force_zip64 to use the ZIP64 format, which can handle large
    files. If the size is known in advance, it is best to pass a ZipInfo
    instance for name, with zinfo.file_size set.

printdir(self, file=None)
:
```

Let's begin the process of writing a zip file password cracker.

- 1) Write a quick script to test the use of the zipfile library. After importing the library, instantiate a new ZipFile class by specifying the filename of the password-protected zip file (*evil.zip*). utilize the `extractall()` method and specify the optional parameter for the password (*secret*). Execute your script and turn in the code and output.
- 2) Use the `except Exception` exception handler to catch exceptions and print them out when an incorrect password is used. Execute your script with an incorrect password and exception handler and turn in the code and output.
- 3) Write a script that performs a dictionary attack on the password protected zip file. Execute your script and turn in the code and output. Be sure to provide user feedback on exceptions thrown.

#### Solution:

- 1) The code and result are in the following[4]:

```
eve@Eves-Air:~/Desktop/CS7349-spring24/Assignments/HW3/Code/evil .HW3/Code/evil (-zsh)
↳ Code git:(main) ✘
↳ Code git:(main) ✘ ls
HW3_evil.zip      HW3dictionary.txt HW3passwords.txt  cracker.py      zip_cracker.py
↳ Code git:(main) ✘ cat zip_cracker.py
import zipfile

def extract_zip(zip_file_name, password):
    try:
        # Instantiate a ZipFile object
        with zipfile.ZipFile(zip_file_name, 'r') as zip_ref:
            # Use the extractall method and provide the password
            zip_ref.extractall(pwd=bytes(password, 'utf-8'))
            print("Extraction successful. Contents extracted.")
    except Exception as e:
        print(f"An error occurred: {e}")

# Specify the filename of the password-protected zip file
zip_file_name = 'HW3_evil.zip'
# Specify the password
password = 'secret'

# Call the function to extract the zip file
extract_zip(zip_file_name, password)
↳ Code git:(main) ✘ python3 zip_cracker.py
Extraction successful. Contents extracted.
↳ Code git:(main) ✘ ls
HW3_evil.zip      HW3passwords.txt  evil
HW3dictionary.txt cracker.py      zip_cracker.py
↳ Code git:(main) ✘ cd evil
↳ evil git:(main) ✘ ls
evil.jpg          note_to_adam.txt
↳ evil git:(main) ✘ [
```

- 2) The code and result are in the following:

The screenshot shows a terminal window on a Mac OS X desktop. The title bar reads "eve@Eves-Air:~/Desktop/CS7349-spring24/Assignments/HW3/Code/evil". The command line shows the user navigating through a directory structure, running a script named "zip\_cracker.py", and then listing files in a folder named "evil". The terminal output includes the Python code for the "zip\_cracker.py" script, which defines a function to extract a zip file using a password. It handles various exceptions like "RuntimeError" for incorrect passwords and "BadZipfile" for corrupted files. The user then runs the script with a wrong password ("wrong\_password") and lists the contents of the resulting directory.

```

eve@Eves-Air:~/Desktop/CS7349-spring24/Assignments/HW3/Code/evil
..HW3/Code/evil (-zsh)

↳ Code git:(main) ✘ ls
HW3_evil.zip      HW3passwords.txt  evil
HW3dictionary.txt cracker.py        zip_cracker.py
↳ Code git:(main) ✘ rm -r evil
↳ Code git:(main) ✘ ls
HW3_evil.zip      HW3passwords.txt  zip_cracker.py
HW3dictionary.txt cracker.py
↳ Code git:(main) ✘ cat zip_cracker.py
import zipfile

def extract_zip(zip_file_name, password):
    # try:
    #     # Instantiate a ZipFile object
    #     # with zipfile.ZipFile(zip_file_name, 'r') as zip_ref:
    #     #     # Use the extractall method and provide the password
    #     #     zip_ref.extractall(pwd=bytes(password, 'utf-8'))
    #     #     print("Extraction successful. Contents extracted.")
    # except Exception as e:
    #     print(f"An error occurred: {e}")
    try:
        # Instantiate a ZipFile object
        with zipfile.ZipFile(zip_file_name, 'r') as zip_ref:
            # Use the extractall method and provide the password
            zip_ref.extractall(pwd=bytes(password, 'utf-8'))
            print("Extraction successful. Contents extracted.")
    except RuntimeError as e:
        # Catching exceptions related to incorrect password
        if 'Bad password' in str(e):
            print("Incorrect password provided.")
        else:
            print(f"Runtime error occurred: {e}")
    except zipfile.BadZipfile:
        print("File is not a zip file or the zip file is corrupted.")
    except Exception as e:
        # Catching other general exceptions
        print(f"An error occurred: {e}")

# Specify the filename of the password-protected zip file
zip_file_name = 'HW3_evil.zip'
# Specify an incorrect password
password = 'wrong_password'

# Call the function to extract the zip file
extract_zip(zip_file_name, password)
↳ Code git:(main) ✘ python3 zip_cracker.py
Incorrect password provided.
↳ Code git:(main) ✘ ls
HW3_evil.zip      HW3passwords.txt  evil
HW3dictionary.txt cracker.py        zip_cracker.py
↳ Code git:(main) ✘ cd evil
↳ evil git:(main) ✘ ls
↳ evil git:(main) ✘ ls -la
total 0
drwxr-xr-x  2 eve  staff   64 Mar 14 15:14 .
drwxr-xr-x@ 10 eve  staff  320 Mar 14 15:14 ..
↳ evil git:(main) ✘ |

```

3) The code and result are in the following:

```

eve@Eves-Air:~/Desktop/CS7349-spring24/Assignments/HW3/Code/evil
..HW3/Code/evil (-zsh)

→ Code git:(main) ✘ ls
HW3_evil.zip      HW3passwords.txt  evil          zip_cracker3.py
HW3dictionary.txt cracker.py       zip_cracker.py
→ Code git:(main) ✘ rm -r evil
→ Code git:(main) ✘ ls
HW3_evil.zip      HW3dictionary.txt HW3passwords.txt  cracker.py      zip_cracker.py  zip_cracker3.py
→ Code git:(main) ✘ cat zip_cracker3.py
import zipfile

def dictionary_attack(zip_file_name, dictionary_file):
    # Attempt to open the dictionary file and read passwords
    try:
        with open(dictionary_file, 'r') as dict_file:
            passwords = dict_file.readlines()
    except FileNotFoundError:
        print("Dictionary file not found.")
        return
    except Exception as e:
        print(f"An error occurred while reading the dictionary file: {e}")
        return

    # Initialize the ZipFile object
    try:
        with zipfile.ZipFile(zip_file_name, 'r') as zip_ref:
            for password in passwords:
                try:
                    # Attempt to extract the zip file with the current password
                    zip_ref.extractall(pwd=bytes(password, 'utf-8'))
                    print(f"Success! File extracted with password: {password}")
                    return
                except RuntimeError as e:
                    # Handling incorrect password exception
                    continue
                except zipfile.BadZipfile:
                    print("File is not a zip file or the zip file is corrupted.")
                    return
    except FileNotFoundError:
        print("Zip file not found.")
    except Exception as e:
        print(f"An error occurred while processing the zip file: {e}")

    # Specify the filename of the password-protected zip file
zip_file_name = 'HW3_evil.zip'
# Specify the dictionary file containing potential passwords
dictionary_file = 'HW3dictionary.txt'

# Call the function to perform a dictionary attack on the zip file
dictionary_attack(zip_file_name, dictionary_file)
→ Code git:(main) ✘ python3 zip_cracker3.py
Success! File extracted with password: secret
→ Code git:(main) ✘ ls
HW3_evil.zip      HW3passwords.txt  evil          zip_cracker3.py
HW3dictionary.txt cracker.py       zip_cracker.py
→ Code git:(main) ✘ cd evil
→ evil git:(main) ✘ ls
evil.jpg          note_to_adam.txt
→ evil git:(main) ✘

```

### Exercise 3: Port Scanner

The goal of this exercise is to learn about port scanners for networked systems.

First, create a simple Python-based port scanner. Using the socket library, you will create a script that iterates through a range of IP addresses, and, for each IP address, will identify the active ports available for that IP address. At least ports corresponding to telnet, ftp SSH, smtp, http, imap, and https services should be scanned and identified.

Second, download and install the nmap port scanning software from nmap.org. Utilize nmap to identify the operating system and the open ports of devices on a range of IP addresses.

### Solution:

First[5],

```

eve@Eves-Air:~/Desktop/CS7349-spring24/Assignments/HW3/Code
..ents/HW3/Code (-zsh)

▶ Code git:(main) ✘ ls
HW3_evil.zip      HW3passwords.txt  evil          zip_cracker.py
HW3dictionary.txt cracker.py       portScan.py    zip_cracker3.py
▶ Code git:(main) ✘ cat portScan.py
import argparse
from socket import *
from threading import Thread, Semaphore

screenLock = Semaphore(value=1)

def connScan(tgtHost, tgtPort):
    try:
        connSkt = socket(AF_INET, SOCK_STREAM)
        connSkt.connect((tgtHost, tgtPort))
        connSkt.send(b'ViolentPython\r\n') # Sending bytes object
        results = connSkt.recv(100).decode('utf-8') # Decoding to string
        screenLock.acquire()
        print(f'[+] {tgtPort}/tcp open')
        print(f'[+] {results}')
    except Exception as e:
        screenLock.acquire()
        print(f'[-] {tgtPort}/tcp closed')
    finally:
        screenLock.release()
        connSkt.close()

def portScan(tgtHost, tgtPorts):
    try:
        tgtIP = gethostbyname(tgtHost)
    except:
        print(f'[-] Cannot resolve '{tgtHost}': Unknown host')
        return
    try:
        tgtName = gethostbyaddr(tgtIP)
        print(f'\n[+] Scan Results for: {tgtName[0]}')
    except:
        print(f'\n[+] Scan Results for: {tgtIP}')
    setdefaulttimeout(1)
    for tgtPort in tgtPorts:
        t = Thread(target=connScan, args=(tgtHost, int(tgtPort)))
        t.start()

def main():
    parser = argparse.ArgumentParser(usage='%(prog)s -H <target host> -p <target port[s]>')
    parser.add_argument('-H', dest='tgtHost', type=str, help='specify target host')
    parser.add_argument('-p', dest='tgtPort', type=str, help='specify target port[s] separated by comma')
    args = parser.parse_args()

    tgtHost = args.tgtHost
    tgtPorts = args.tgtPort.split(',')

    if tgtHost is None or tgtPorts[0] is None:
        print(parser.usage)
        exit(0)

    portScan(tgtHost, tgtPorts)

if __name__ == "__main__":
    main()
▶ Code git:(main) ✘ python3 portScan.py -H google.com -p 21,22,23,80

[+] Scan Results for: rt-in-f100.1e100.net
[+] 80/tcp open
[+] HTTP/1.0 400 Bad Request
Content-Type: text/html; charset=UTF-8
Referrer-Policy: no-referrer
Content-Length: 142
Content-Type: text/html; charset=UTF-8
[+] 23/tcp closed
[+] 22/tcp closed
[+] 21/tcp closed
▶ Code git:(main) ✘ |

```

Second[6],

```
brew install nmap
Running `brew update --auto-update`...
==> Auto-updated Homebrew!
==> Updated Homebrew from 4.2.9 (b1fdb59479) to 4.2.12 (a3e5e3f7a0).
Updated 5 taps (homebrew/cask-fonts, homebrew/services, playcover/playcover, homebrew/core and homebrew/cask).
==> New Formulae
actions-batch          gptscript           phodav
appwrite                inlyne              pkl
bluez                  kubeshark          prjtrellis
bpftop                 liborigin           scala@3.3
c4core                 libsqli             sigi
cargo-fuzz              lsusb-laniksj        srgn
cmake-language-server   mdsh               taskopen
cotp                   mmdbinspect         typstfmt
edgevpn                mubeng              vrc-get
erlang_ls               nmail               wstunnel
gimmeccert             npm-check-updates
gnmic                  pass-import
==> New Casks
apidog-europe          font-reddit-mono
arturia-software-center font-reddit-sans-condensed
cahier                  ibkr
```

```
eve@Eves-Air:~ 2:21 | +  
~ (-zsh)  
▶ ~ nmap 192.168.1.1-130  
Starting Nmap 7.94 ( https://nmap.org ) at 2024-03-14 16:14 CDT  
Nmap scan report for SAX1V1K.lan (192.168.1.1)  
Host is up (0.0033s latency).  
Not shown: 846 closed tcp ports (conn-refused), 151 filtered tcp ports (no-response)  
PORT      STATE SERVICE  
53/tcp    open  domain  
80/tcp    open  http  
443/tcp   open  https  
  
Nmap scan report for MacBook-Pro.lan (192.168.1.15)  
Host is up (0.021s latency).  
Not shown: 995 closed tcp ports (conn-refused)  
PORT      STATE SERVICE  
631/tcp   open  ipp  
5000/tcp  open  upnp  
5432/tcp  open  postgresql  
7000/tcp  open  afs3-fileserver  
49165/tcp open  unknown  
  
Nmap done: 130 IP addresses (2 hosts up) scanned in 54.22 seconds  
▶ ~ |
```

## References

- [1] ‘crypt — Function to check Unix passwords’, Python documentation. Accessed: Mar. 14, 2024. [Online]. Available: <https://docs.python.org/3/library/crypt.html>
- [2] W. Stallings, Cryptography and network security: principles and practice, Seventh edition. Boston: Pearson, 2014.
- [3] ‘How Unix Implements Passwords - Practical UNIX and Internet Security, 3rd Edition [Book]’. Accessed: Mar. 14, 2024. [Online]. Available: <https://www.oreilly.com/library/view/practical-unix-and/0596003234/ch04s03.html>
- [4] ‘zipfile — Work with ZIP archives’, Python documentation. Accessed: Mar. 14, 2024. [Online]. Available: <https://docs.python.org/3/library/zipfile.html>
- [5] ‘Python Programming Tutorials’. Accessed: Mar. 14, 2024. [Online]. Available: <https://pythonprogramming.net/python-port-scanner-sockets/>
- [6] ‘Download the Free Nmap Security Scanner for Linux/Mac/Windows’. Accessed: Mar. 14, 2024. [Online]. Available: <https://nmap.org/download#macosx>