Project II

CS 6238: Secure Computer Systems

Password Hardening with 2FA

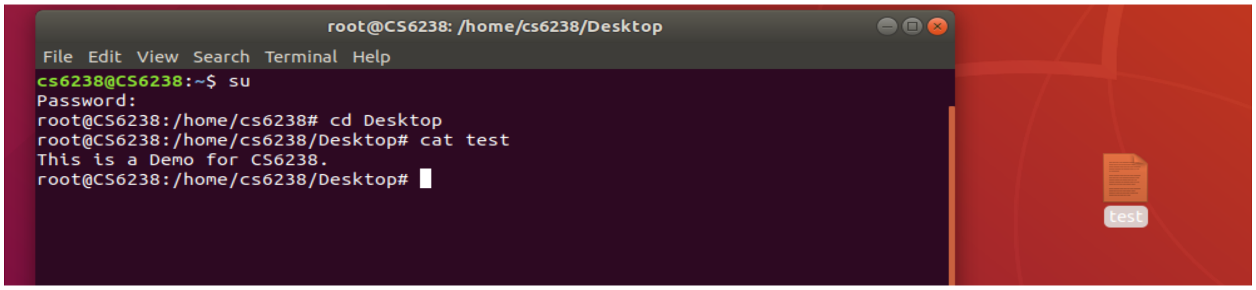
Due Date: October 1, 2020 11:59pm

Learning Objectives: The goal of this project is to harden password-based authentication by including information obtained from a second factor (two-factor Authentication or 2FA). We will use the Linux login command implementation to explore this. Although somewhat contrived, the motivation for the 2FA scheme explored in this project is similar to the password hardening paper discussed in course lectures. More specifically, information maintained by the system to check the validity of a login request is updated after each login request to limit the effectiveness of offline guessing attacks. The following are the learning objectives of this project.

1. Understand how password-based authentication is implemented.
2. Understand the benefits of multiple factors for stronger authentication.
3. Augment a password-based login command code to include an additional factor obtained from a second source.
4. Analyze security benefits (or lack of them) of the 2FA implementation.

To keep it simple, this project focuses only on hardening the basic login scheme for a desktop/laptop system. However, this scheme can be extended to provide password hardening for remote logins.

Project Setup: For this project, you are provided an Ubuntu-based Virtual Machine (VM), the link to which will be posted on Piazza and Canvas. Lookout for the Project2 post for the VM link. This VM was tested on Oracle VM Virtual box 5.2.16 and can be directly imported to it[1]. This VM has a default account "CS6238" setup with root privilege. You may also need to access some files through a root terminal. To access a file as root, open a terminal, type "su", enter the password for the root user, and then access the file you desire to open. As you can see from Fig 1., the file test can be accessed (located on Desktop) as a root user.


root@CS6238: /home/cs6238/Desktop
File Edit View Search Terminal Help
test

```
cs6238@CS6238:~$ su
Password:
root@CS6238:/home/cs6238# cd Desktop
root@CS6238:/home/cs6238/Desktop# cat test
This is a Demo for CS6238.
root@CS6238:/home/cs6238/Desktop#
```

Fig 1. Reading file 'test' from root

Password for both the "root" and "CS6238" accounts is "CS6238_2FA". You should not include "" while entering password. When you log into **account "CS6238", you will find 2FA named folder on Desktop**.

[1] By now, you should know how to import a VM. If not, please visit: https://docs.oracle.com/cd/E26217_01/E26796/html/qs-import-vm.html.

This folder contains:

- 2 Python code files, *check_login.py* and *create_user.py*,
- and one executable, *token_generator*.

Additional details of these code files and the executable will be described later.

IMPORTANT NOTE: We have observed that students tend to erase or delete the */etc/passwd* and */etc/shadow* file while working on this project and lose access for login into the VM. It would be safe to take snapshots of your VM before starting the project and while progressing through the project[2].

We have also created a copy of the files namely */etc/passwd.bkp* and */etc/shadow.bkp* on the VM in case you delete or erase or modify them. Since, you have root access it would be safe to have your own copy and also exercise caution while updating these files.

Prior to starting on the project, you should familiarize yourself with the working of the login command in Linux. In particular, you should be able to answer the following questions.

1. How are users created in Linux?
2. What algorithms are used in the process of encryption/hashing of passwords?
3. Where is information derived from passwords stored?
4. How does the system check if a correct password is provided by a login request?
5. Why and how is salting used?
6. Who has access to the file containing password related information?

There are plenty of online resources for finding answers to these questions. To help you get started, see the section "GETTING STARTED ON LINUX LOGIN/PASSWORDS" in the Appendix Section.

1. Project Details

Getting Started: To help you get started, we have provided two Python code files to help you better understand the inner working of the system while creating and logging in users.

1. *create_user.py:* to create a new user. You should study its code to understand what it does, and what are the requirements that should be met to run it successfully. Check what changes are made to the */etc/shadow* file after creating a user with *create_user.py*. A good understanding of these changes will help you later in this project.
2. *check_login.py:* which checks whether the user can login using a pair of user-id and password values. By analyzing the working of its code, you can learn how a user is validated after providing the correct password. In particular, explore how it retrieves the hash from the shadow file and checks it against the hash generated by the password provided by the user.

After understanding the code of *create_user.py* and *check_login.py*, you are ready to work on this project.

[2] How to take a VM snapshot on Virtual Box: https://www.virtualbox.org/manual/ch01.html#snapshots.

2. Task1: Implementing 2FA (80% of grade):

In this task, you need to implement 2FA using the provided **token generator** (TG) executable, which serves as a second factor. The 2FA method uses the tokens generated by TG to harden the login mechanism used in Linux.

Typically, a unique second factor source/device is associated with a user (i.e., your phone for Duo 2FA used by Georgia Tech) but in this project, the same TG will serve as a second factor source for all accounts you create. For this to be possible, a user must be registered with TG along with a PIN.

Thus, each user has two accounts:

- One account on the 2FA system
- One account on TG

The Token Generator and **The 2FA Method** are described first. Details regarding what you have to implement are given in **Implementation of 2FA.**

Token Generator (TG)

Before moving to the 2FA method, it is important to know the working of TG. It gives a user three options – '1' for registering a new user, '2' for generating token for the current/registered user and '3' for deleting an existing user account.

NOTE: You have been provided the Token Generator(TG) executable. You only have to understand how it works so that you can use it as a black box in your project. You **do not have to implement** the Token Generator.

If user enters:

'1' then

- TG will prompt the user for user-id and a six-digit PIN. After this,
- It will generate an **initial token, IT**, and create an encrypted text file with entered user-id in its name.
- Deleting this generated file is equivalent to deleting the account of the user from TG. We will see how this initial token IT will help us when we discuss the 2FA method.

'2' then

- user will have to provide user-id and correct PIN for user U (think of this as unlocking the screen on your phone with a PIN or pattern when the phone is the 2FA device).
- If correct information is provided, the token generator will return **current token CT** and **next token NT.**
- These tokens will be used in the 2FA method.

'3' then

- user will have to provide the same information as in option '2'(user-id and correct PIN)
- the TG will only produce current token CT and will delete the user account and associated file.

NOTE(Very Important!!!): ***After execution of each option in TG, the user will be prompted for confirmation of the requested task in the 2FA method. If the task in 2FA method, for which tokens are generated using TG is completed successfully, the user has to enter 'y' or 'Y'. If the user enters some other character, then the TG will revert itself to the previously known state for the user.***

The 2FA Method

1. Create a User Account:

First, when user U tries to create an account, 2FA login method requires a user to provide three things, **username U**, **password P** and the **initial token IT** generated by the TG when registering a user account for user.

NOTE: The PIN for the TG and the password for 2FA system should be different. But the **username for the 2FA system** and the **user-id for the TG** should be the same.

The 2FA login method will take this token IT, concatenate it with the provided password P and this will be the hardened password (P+IT) that goes into the password hashing algorithm to generate an entry in the shadow file. With this entry, the new user will be successfully created. The whole process can be visualized as shown in Figure 2[Appendix Section]:

2. Logging into created User Account

After a user U is created, he/she can log into his/her account. For login, a user has to provide **username U**, **password P**, **current token CT** from the token generator and **next token NT** from the token generator[3].

2FA method first checks if the user U exists. If yes, it will then concatenate the password P and current token CT to construct the hardened password(P+CT), as done above in user creation. This hardened password will we used for validating against the hash value in the user entry in */etc/shadow*. After successful validation, the user entered password P will be concatenated with the next token NT to create the new hardened password (P+NT). This new password is then hashed and this new hashed value is used to update the corresponding field in the */etc/shadow* file. Based on successful or failed execution of above request, the user will enter the response in the TG, which will decide whether the changes will be saved or discarded. The full functionality of 2FA is visualized in Figure 3[Appendix Section].

3. Update and Delete

2FA should be able to update user's password. This deals with the situation when a user's password is compromised or certain amount of time has elapsed since the password was created. For update, a user has to provide **username U**, **password P**, **current token CT** from the token generator, **new password NP** and **next token NT** from the token generator.

2FA should also be able to delete user's account. For delete, user has to provide **username U**, **password P and current token CT.**

Update and delete functionality should be extrapolated from login functionality.

[3] Ideally, these tokens should be obtained without any user effort but to simplify the project, you will provide these tokens manually (e.g., cut and paste).

Implementation of 2FA

After becoming familiar with the working of the 2FA method and TG, you must create a standalone program based on the functionality of 2FA method. Please start from the python code that we have provided.

NOTE: (Information Regarding Salt): For all scenarios like creating a user and updating user information when salt is required, **we will provide you with the salt for testing and evaluation**. In a real scenario, salt is randomly generated by system. For Ubuntu, the salt is generally 8 character long. For more information regarding salts look at the appendix section GETTING STARTED ON LINUX PASSWORDS.

Your program should be capable of handling the following steps:

1. **Prompting user for different requests** (10 pts.)**:** '1' for creating new user, '2' for login, '3' for updating password and '4' for deleting user account. Also, prompting the user for appropriate inputs such as username, password, salt and tokens is also needed.
2. **Creating Users** (20 pts.)**:** If '1' for creating user is selected from the prompt, the program should do the following
    a. Prompt for username, password, salt, initial token IT.
    b. If a user already exists, the program should display **"FAILURE: user <username> already exists"** and exit (this should be considered as a failed attempt at creating user).
    c. If not, create the user. When you create a user, you should update the */etc/shadow* and */etc/passwd* file for the user.
    d. A home directory for that user should be created and there should be an entry of home directory in the passwd file.
    e. If a user is created, your code should print **"SUCCESS: <user-id of the user> created".**

    NOTE: The salt will be same for a user account unless the user changes the password or deletes it and then create it again.
3. **Login** (20 pts.)**:** If a user enters "2" for login, program should do the following
    a. the user should be prompted for the username first.
    b. if the user does exist with that username then he/she should be prompted for the password, current token, and next token in that order.
    c. At this point, the complete login process described in 2FA method should be executed.
    d. On successful completion it should display **"SUCCESS: Login Successful".**
    e. If user does not exist, then it should display **"FAILURE: user <username> does not exist"**
    f. If password or token is incorrect, then it should display **"FAILURE: either passwd or token incorrect"**
4. **Password Update** (15 pts.)**:** If user enter "3" for update, the program should:
    a. ask the user for username, password, new password, new salt, current token and next token in that order and update the account
    b. On successful completion it should display **"SUCCESS: user <username> updated".**
    c. Error handling should be done similar to login functionality above.

5. **Deleting a user** (15 pts.)**:** If user enters “4” for deleting an account, then program should
    a. ask the user for its username, password and current token in that order.
    b. If correct values are supplied for all of these, all entries and home directory for the deleted user-id should be cleaned.
    c. On successful completion it should display **“SUCCESS: user <username> Deleted”.**
    d. Again error handling has to be done similar to login functionality.
6. **Evaluation:** For evaluation, we will give you some test cases, and you should submit screenshots before each test case and after finishing all test cases. Do not alter shadow or any other files in between test cases. As one test case can be dependent on other, if you delete some entries after test case 1, this may lead to a different output than the expected for test case 2. The test cases and the instructions to run them will be available in the file named **Project2_Testcases.pdf**.

Please keep the following in mind as you work on the project.

1. After every successful request, user should type ‘y’ or ‘Y’ when asked by the TG. You should not enter ‘Y’ or ‘y’ before completion of the task by your script.
2. Remember, the interaction of 2FA with the TG is manual and tokens have to be copy pasted across from TG onto your 2FA implementation
3. You should type following commands in TG when invoking one of the four functions implemented by you. If you are creating a new user, you must enter ‘1’ when prompted for the user input by TG. Similarly, ‘2’ in case of updating or login, and ’3’ in case of user deletion.

3. Task2: Security Analysis of 2FA (20% of grade):

Complete a security analysis for the implemented 2FA method. More specifically,

- Discuss the advantages(any 2), disadvantages(any 2) and the possible attacks(any 2) on the above-given method.
- Let us say 2FA has to be implemented in a realistic environment. Recommend one improvement for the current 2FA scheme.
- How can one implement the 2FA scheme in a server-client setting, and how will you secure the token transfer between the separate systems?

4. Project Deliverables:

- Your python script for the 2FA implementation. The file should be name ‘2FA.py’
- A pdf report named ‘<gtid>_2FA.pdf’. For example, for me, it is ‘hnagarajan7_2FA.pdf’. The report should contain:
    - Your screen shot responses for the testcases for Task1 under section “Task1”.
    - Your answers for Task2 under section “Task2”

5. Appendix

FIGURES

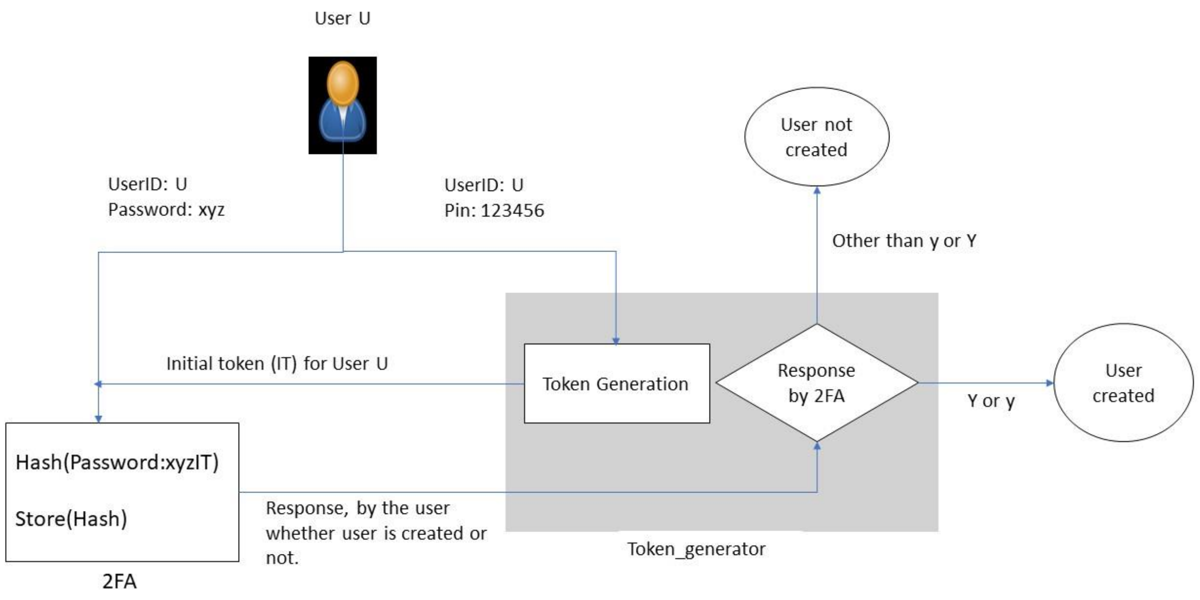


Figure 2: Creating an Account

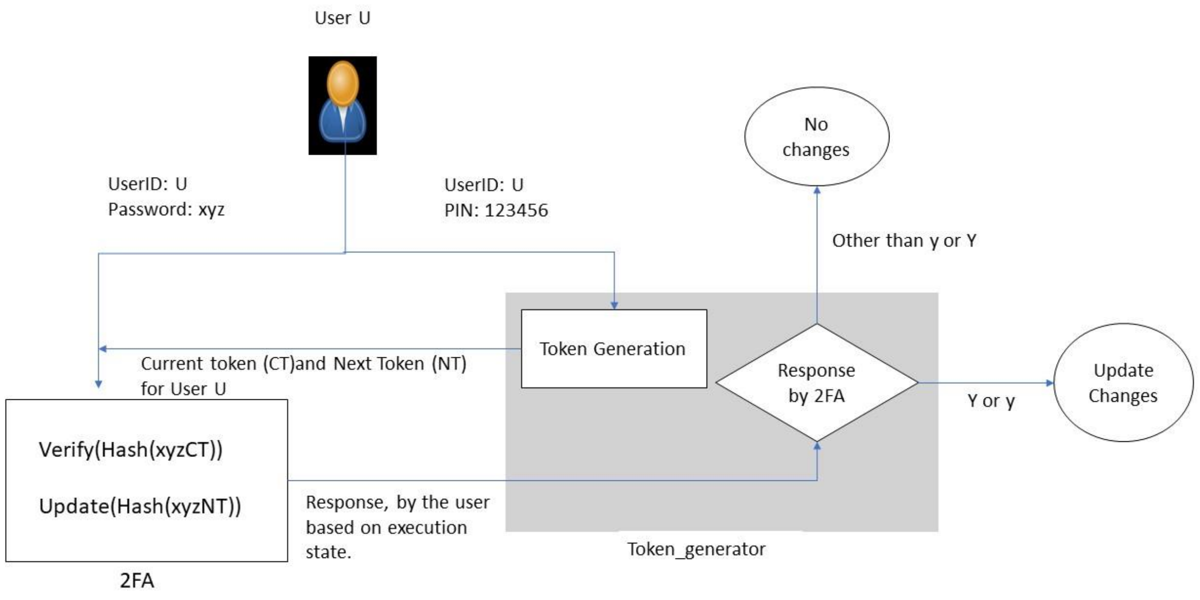
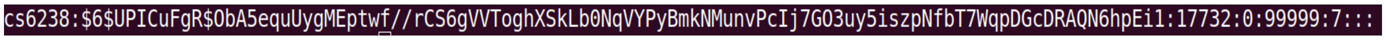


Figure 3: Logging into an Account

GETTING STARTED ON LINUX LOGIN/PASSWORDS

When the Linux system creates a user, it prompts the user for a password. Then, based on the version of Linux, one of six algorithms are chosen for password encryption. The system generates a random salt and uses that salt to generate a one-way hash and store that hash with user details in */etc/shadow* file. The user entry looks like the below-given example:



```
cs6238:$6$UPICuFgR$ObA5equUygMEptwf//rCS6gVVToghXSkLb0NqVYPyBmkNMunvPcIj7GO3uy5iszpNfbT7WqpDGcDRAQN6hpEi1:17732:0:99999:7:::
```

As you can see the user entry consists of 9 fields, each separated by the ":". The first field is the username and the second is hash. The hash contains 3 other fields separated with the dollar sign ("$"). The first field tells us about the hashing algorithm used, in this case, "6" denotes SHA-512. The second field is the salt value used to make hash value unique. The last field is the hash of the combination of your salt and password. You can easily verify the generation of hash by using the perl one-line script on your Ubuntu terminal.

```
perl -e 'print crypt("<PASSWORD>","\$<HASH-ALGO>\$<SALT-VALUE>\$") . "\n"'
```

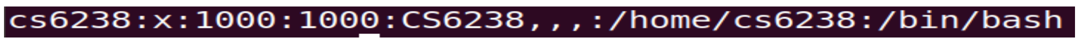
Here, <PASSWORD>=CS6238_2FA

<HASH-ALGO>=6

<SALT-VALUE>=UPICuFgR

Note: Explore all other fields as you will need to know them for project.

After storing the hash entry in file */etc/shadow*, the system will create a home directory and a user entry in */etc/passwd* file which stores essential information required during login, i.e., user account information. This file contains one entry per line for each user. An entry in */etc/passwd* for user cs6238 is looks like:



```
cs6238:x:1000:1000:CS6238,,,:/home/cs6238:/bin/bash
```

Each entry in */etc/passwd* has seven fields, each separated with ":". The first field contains the username. The second field contains the password for the user. "x" denotes that the hashed password entry is in the shadow file. Next two entries are of uid and guid. Last two entries are home directory of the user and the absolute path of the command shell. We are not going to discuss passwd file contents in detail as for the project it is sufficient to know what in the passwd file. However, you are welcome to further explore details of passwd files. After updating the entry in passwd file, user creation completes.