CS 6238: Secure Computer Systems

Project II: Strengthening Login Security with Two-Factor Authentication (2FA)

Learning Objectives:

This project aims to enhance password security by incorporating a second authentication factor (2FA). We'll explore this concept by modifying the Linux login implementation. While the specific implementation may be simplified for educational purposes, the underlying principles are similar to password hardening techniques discussed in class. Here's what you'll learn:

1. **Password-Based Authentication:** Understand how traditional login systems with usernames and passwords work.
2. **Strength in Numbers:** Explore the benefits of using multiple factors for stronger authentication compared to passwords alone.
3. **Building a Stronger Login:** Modify the existing login code to incorporate an additional authentication factor.
4. **Security Analysis:** Evaluate the effectiveness of the implemented 2FA system and identify potential limitations.

Focus:

This project concentrates on improving login security for local desktops/laptops. However, the concepts can be extended to secure remote logins as well.

Project Setup:

We'll be using a pre-configured Ubuntu virtual machine (VM) compatible with Oracle VM VirtualBox 7.0. You can directly import this VM into VirtualBox for your work.

The VM has a pre-created user account named "cs6238" with standard user privileges. To access files requiring root access, open a terminal and type:

- `sudo su`

Enter the password for "cs6238" (**note:** the password itself is not shown here for security reasons). Once you have root access, locate the desired file.

Default Credentials:

- Username: `cs6238`
- Password: `cs6238`

Accessing Project Files:

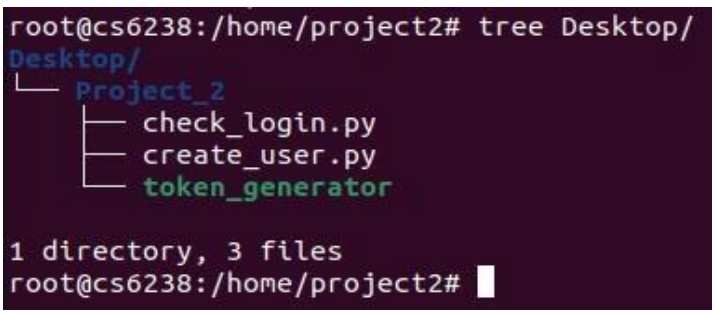
1. Log in to the "`cs6238`" account.
2. Navigate to your desktop directory:
    - `cd /home/cs6238/Desktop`
3. Clone the project repository from GitHub:
    - `git clone https://github.com/TA-gatech/Project2.git`

Project Files:

The cloned repository contains:

- Two Python code files: `check_login.py` and `create_user.py`
- One executable: `token_generator`

Figure 1. Desktop folder Content



Important Note:

- It's recommended to take snapshots of your VM before starting and while progressing through the project to prevent accidental data loss.

Understanding Linux Login System (Pre-requisite):

Before proceeding, familiarize yourself with the following aspects of the Linux login system:

1. How are users created in Linux?
2. What algorithms are used for password encryption/hashing?
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 salt used?
6. Who has access to the file containing password-related information?

There are plenty of online resources for these topics. The "GETTING STARTED ON LINUX LOGIN/PASSWORDS" section in the Appendix can serve as a starting point.

We've provided two Python code files to help you understand how the system works for creating and logging in users:

1. **create_user.py:** This script creates a new user. Analyze its code to understand:
    - What it does
    - Requirements for successful execution
    - Changes made to the /etc/shadow file after creating a user
2. **check_login.py:** This script verifies user login credentials (username and password). By analyzing its code, learn:
    - How user validation works after providing the correct password
    - How it retrieves the hash from the shadow file and compares it with the one generated from the user-provided password

Once you understand these scripts, you're ready for the main project task.

Task 1: Implementing 2FA (80% of grade)

This task focuses on implementing a 2FA system using a provided token_generator (TG) executable. While typical 2FA systems use a unique device per user (like a phone), this project uses a single TG for all user accounts. Each user will have two accounts: one in the 2FA system and one with the TG (registered with a PIN).

Token Generator (TG):

You are provided with the compiled `token_generator` executable. You *do not* need to implement it; you only need to understand its interface and use it as a black box.

The TG offers three options:

- **'1' (Register User):**
    - Prompts for a user-id and a six-digit PIN.
    - Generates an initial token (IT) and creates an encrypted file named after the user-id.
    - Deleting this file effectively deletes the user's TG account.
- **'2' (Generate Tokens):**
    - Requires the user-id and correct PIN.
    - Returns the current token (CT) and the next token (NT).
- **'3' (Delete User):**
    - Requires the user-id and correct PIN.
    - Returns the current token (CT) and deletes the user's account and associated file.

Important TG Behavior:

After each TG operation, the TG will prompt for confirmation. If the corresponding 2FA operation is successful, enter 'y' or 'Y'. Otherwise, entering any other character will cause the TG to revert to its previous state.

The 2FA Method:

The 2FA method involves four main operations: creating a user, logging in, updating, and deleting a user.

1. **Create User:**
    - Requires: username (U), password (P, confirmed), salt, and the initial token (IT) from the TG.

- The PIN for the TG and the password for the 2FA system *can be different*. The username for the 2FA system and the user-id for the TG *must be the same*.
- The hardened password is created by concatenating the password (P) and the initial token (IT): P+IT. This hardened password is then hashed and stored in the `/etc/shadow` file.

2. **Login:**
   - Requires: username (U), password (P), current token (CT), and next token (NT) from the TG.
   - The system checks if the user exists.
   - The hardened password is created by concatenating the password (P) and the current token (CT): P+CT.
   - This hardened password is used to verify against the stored hash in `/etc/shadow`.
   - If successful, a new hardened password is created by concatenating the password (P) and the next token (NT): P+NT. This new hardened password is hashed, and the `/etc/shadow` file is updated with this new hash.
   - The TG confirmation mechanism is used to finalize or revert the changes.
3. **Update:**
   - **Update:** Requires username (U), password (P), new password (NP, confirmed), new salt (NS), current token (CT), and next token (NT).
4. **Delete:** Requires username (U), password (P), and current token (CT).
   - These operations should be implemented based on the same principles as the login functionality.

Implementation of the 2FA method:

Important: Adhere to the specified prompt order. Regrade requests based on incorrect prompt order will not be accepted.

You must create a standalone Python program (2FA.py) based on the provided Python code. Your program must implement the following:

1. **User Prompting (10 pts):**
   - Prompt the user to select an action:
     - Select an action:

1. Create a user
2. Login
3. Update password
4. Delete user account

- Do not loop into the action request. Once one action is selected the python script exits.

2. **Creating Users (20 pts):**
    - Prompt for Username, Password, Confirm Password, Salt, and Initial Token (IT).
        1. Username: `Alice`
        2. Password: `Alice123`
        3. Confirm Password: `Alice123`
        4. Salt: `salt0123`
        5. Initial Token: `eYKCaN0kLB7T0.3Q.vPs40`
    - If the user already exists, display "FAILURE: user <username> already exists" and exit. *Prompt all the information before checking if the user exists*.
    - If the user does not exist, create the user. This involves updating the `/etc/shadow` and `/etc/passwd` files and creating a home directory.
    - On success, print "SUCCESS: <user-id of the user> created".
    - The salt remains the same unless the user updates the password or deletes and recreates the account.
3. **Login (20 pts):**
    - Request Username, Password, Current Token (CT), and Next Token (NT).
        1. Username: `Alice`
        2. Password: `Alice123`
        3. Current Token: `eYKCaN0kLB7T0.3Q.vPs40`
        4. Next Token: `iGxl329/ugOeSnhOzYE1B/`
    - Execute the 2FA login process.
    - On success, display "SUCCESS: Login Successful".

- If the user does not exist, display "FAILURE: user <username> does not exist". *Prompt all the information before checking if the user exists*.
- If the password or token is incorrect, display "FAILURE: either passwd or token incorrect."

4. **Password Update (15 pts):**
    - Request Username, Password, New Password, Confirm New Password, New Salt, Current Token (CT), and Next Token (NT).
        1. Username: `Alice`
        2. Password: `Alice123`
        3. New Password: `New-Password`
        4. Confirm New Password: `New-Password`
        5. New Salt: `salt3210`
        6. Current Token: `eYKCaN0kLB7T0.3Q.vPs40`
        7. Next Token: `iGxl329/ugOeSnhOzYE1B/`
    - On success, display "SUCCESS: user <username> updated".
    - Implement similar error handling as the login functionality.
5. **Deleting a User (15 pts):**
    - Request username, password, and current token.
        1. Username: `Alice`
        2. Password: `Alice123`
        3. Current Token: `Gxl329/ugOeSnhOzYE1B/`
    - If the values are correct, remove all user entries and the home directory.
    - On success, display "SUCCESS: user <username> Deleted".
    - Implement similar error handling as the login functionality.

Important Considerations:

1. Token transfer between your 2FA implementation and the TG is manual (copy-paste).
2. Use the appropriate TG option ('1', '2', or '3') when interacting with the TG.

Task 2: Security Analysis of 2FA (20% of grade)

Perform a security analysis of the implemented 2FA method, addressing the following points:

- **Advantages (2):** Discuss two advantages of the implemented 2FA scheme compared to traditional password-only authentication.
- **Disadvantages (2):** Discuss two disadvantages or limitations of the implemented 2FA scheme.
- **Possible Attacks (2):** Describe two potential attacks that could be carried out against the implemented 2FA scheme.
- **Improvement for Real-World Implementation:** Assuming this 2FA system were to be deployed in a real-world environment, recommend one significant improvement to enhance its security.
- **Server-Client Implementation and Secure Token Transfer:** Explain how this 2FA scheme could be implemented in a server-client architecture. Detail how you would secure the transfer of tokens between the client and the server.

Project deliverables for your 2FA implementation:

1. Python Script (2FA.py):

This file should contain your Python code implementing the 2FA functionality based on the provided description. It should handle user prompts, and perform the necessary security checks during user creation, login, update, and deletion operations.

2. Security Analysis Report (gtid_2FA.pdf):

This PDF report, named according to your GT ID (e.g., jrodriguez_2FA.pdf), should detail the security analysis of the implemented 2FA system. It should address:

- **Task 2: Security Analysis of 2FA**
    - **Advantages (2):**
    - **Disadvantages (2):**
    - **Possible Attacks (2):**
    - **Improvement for Real-World Implementation:**
    - **Server-Client Implementation and Secure Token Transfer:**

By covering these points in your report, you'll demonstrate a comprehensive understanding of the 2FA system's security implications and potential improvements.

Important Notes:

- Remember to adhere to the prompt order while implementing the 2FA functionality in your Python script.
- Ensure your report clearly addresses the security analysis points mentioned above.

Appendix

1. VirtualBox VMs

- If you're unfamiliar with importing VMs, refer to the provided Oracle VM VirtualBox documentation: https://docs.oracle.com/cd/E26217_01/E26796/html/qsimport-vm.html

2. Figures

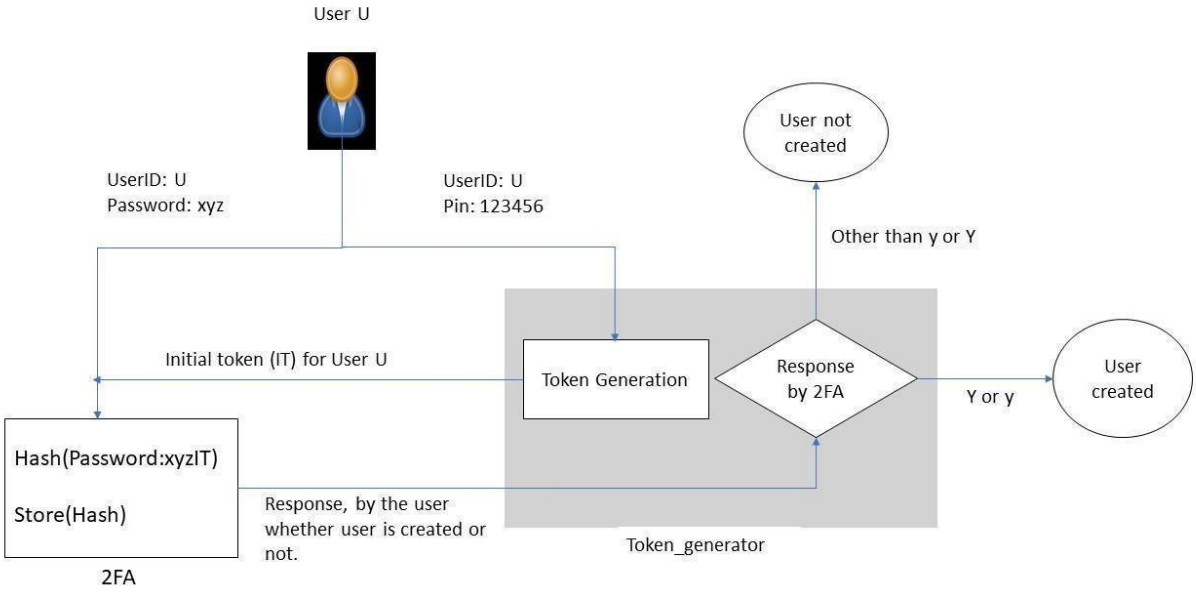


Figure 1: Creating an Account

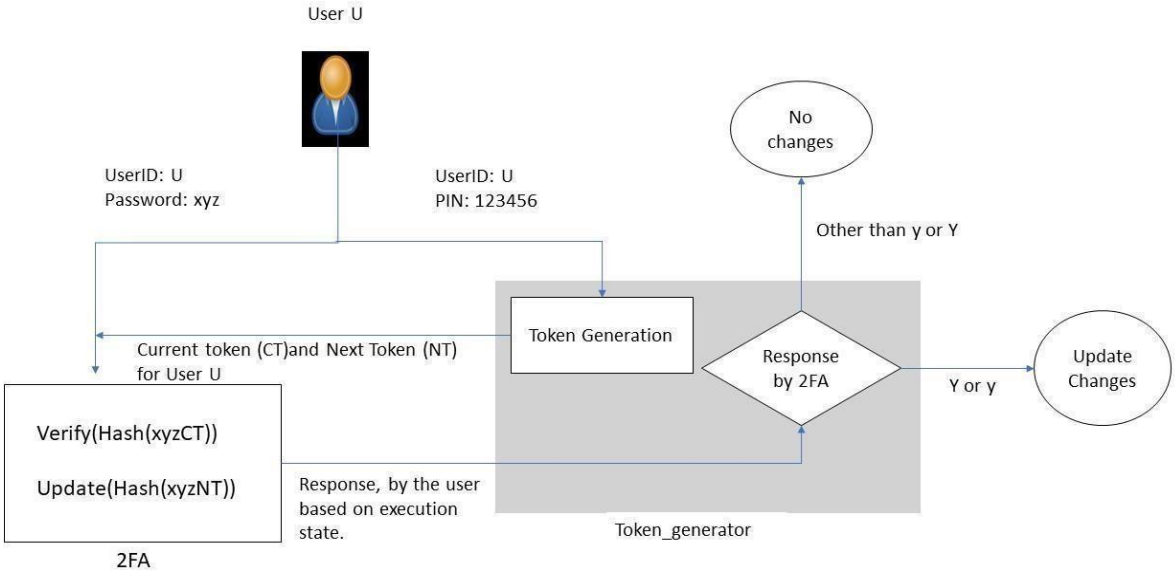


Figure 2: Logging into an Account

3. Getting started on Linux Login and Passwords.

Linux User Creation and Password Storage:

When creating a new user, the Linux system prompts for a password. Depending on the Linux distribution and its configuration, one of several hashing algorithms is used for password encryption. The system generates a random salt, which is then used to create a one-way hash of the password. This hash, along with other user details, is stored in the /etc/shadow file.

Example User Entry in /etc/shadow:

```
cs6238:$6$Cl7HxrVPp7LvCHDb$km3WARvkSdd7toH5lS/OoU5mlSk4.F9ImoQ8H
5Cy5ii10klGO5TCTy9tOZCZFRko6EGM1uIEtwn2f6MN8MLA8/:19589:0:99999:
7:::
```

Structure of a User Entry:

Each user entry in /etc/shadow consists of nine fields separated by colons (:).

1. **Username:** The user's login name (e.g., cs6238).
2. **Password Hash:** The encrypted password. This field itself contains three sub-fields separated by dollar signs ($):
    - **Algorithm Identifier:** Indicates the hashing algorithm used. In the example, 6 denotes SHA-512.
    - **Salt:** The random salt value used during hashing (Cl7HxrVPp7LvCHDb in the example).
    - **Hashed Password:** The actual hash of the password combined with the salt (km3WARvkSdd7toH5lS/OoU5mlSk4.F9ImoQ8H5Cy5ii10klGO5TCTy9tOZCZFRko6EGM1uIEtwn2f6MN8MLA8 in the example).
3. **Last Password Change Date:** Number of days since the epoch (January 1, 1970 UTC) when the password was last changed.
4. **Minimum Password Age:** Minimum number of days before the password can be changed again.
5. **Maximum Password Age:** Maximum number of days before the password must be changed.
6. **Warning Period:** Number of days before password expiration when the user is warned.
7. **Inactive Period:** Number of days after password expiration before the account is disabled.

8. **Account Expiration Date:** Absolute date when the account expires.
9. **Reserved:** Reserved for future use.

User Account Information in /etc/passwd:

After storing the password hash in /etc/shadow, the system creates a home directory for the new user and adds an entry to the /etc/passwd file. This file stores essential user account information needed during login. Each line in /etc/passwd represents a user account.

Example User Entry in /etc/passwd:

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

Structure of a User Entry:

Each entry in /etc/passwd consists of seven fields separated by colons (:):

1. **Username:** The user's login name (e.g., cs6238).
2. **Password:** Historically, this field contained the encrypted password. However, for security reasons, it now almost always contains an "x," indicating that the actual password hash is stored in the /etc/shadow file.
3. **User ID (UID):** A unique numerical identifier for the user (e.g., 1000).
4. **Group ID (GID):** A numerical identifier for the user's primary group (e.g., 1000).
5. **User Information (GECOS):** Optional descriptive information about the user (e.g., the user's full name, office location, etc.). In the example, it's the same as the username.
6. **Home Directory:** The absolute path to the user's home directory (e.g., /home/cs6238).
7. **Login Shell:** The absolute path to the user's default command shell (e.g., /bin/bash).

Project Relevance:

For this project, it is sufficient to understand the basic structure and purpose of the /etc/passwd file, especially the username and home directory fields. Further exploration of the details of the /etc/passwd file is encouraged but not strictly required for completing the project.

User Creation Completion:

After updating the entry in the /etc/passwd file and creating the user's home directory, the user creation process is complete.

Table of Contents