Password Generated Script Based Password Policy (PGSBPP)

Isaiah Friday, M.

Western Governors University

Table of Contents

Summary

The problem was a vulnerable security posture for an enterprise environment due to weak password security. The proposed solution to this problem was implementing a new password policy that required employees to use automated passwords generated from a script. This process was executed by writing a password policy document and developing three bash scripts (to run on Linux systems). The policy was written according to industry recommendations for strong passwords. The scripts were developed using the software development life cycle (SDLC), on a Linux machine. The outcomes of the project were the finished password policy and scripts. The password generating script (PGS) takes a number as an input (of passwords to be created) and outputs that number of strong passwords. Both, the account creation script (ACS) and account modification script (AMS) take a list of usernames as an input, and either create the account with a PGS password, or update their current password with another PGS password. This functionality designates automation as an outcome of the project as well.

Review of Other Work

In addition to the works defined in the proposal, other works also aided the process of developing the policy and scripts required for the projects implementation.

Microsoft's "Password policy recommendations for Microsoft 365 passwords" web page influenced the creation of the proposal's password policy. First it summarizes three practices involved in good password creation. Following this, it lists specific requirements for administrator passwords. Lastly, the page ends with some common negative and positive approaches to password policies, explaining the reasoning behind each one. These recommendations advised the contents of the proposed password policy. During its development, comparisons were often drawn to Microsoft's web page to ensure that none of the "common approaches and their negative impacts" showed in the proposed password policy (Microsoft, 2024). Some direct requirements outlined in the policy also originated from this page, such as a fourteen-character minimum length requirement and no periodic password resets.

The "Sample Password Policy Template" provided a template that assisted with the actual writing of the proposed Password Policy (Firch, 2024). It starts with an overview section that explains why a password policy is necessary, and who the policy applies to. It then provides policy detail specifics, outlining requirements for regular user passwords, system-level passwords (account with ownership), and specific methods to ensure passwords remain protected. This proposed policy was created using this as a template. Details were added, removed, and altered according to proposal specifics, but the template offered a skeleton which improved the efficiency of the creation process. It also provided guidance for what details needed to be included in the policy.

This project also required some research on specific bash scripting methods to correctly implement automation. The "Bash Scripting – How to read a file line by line" article offered a guide explaining how to read files line by line into bash scripts. It provides two methods, one using a "read command and while loop" and the other "using cat and [a] for loop" (Meetgor, 2021). It explains each method in-depth and displays examples that show both methods utilized in an actual bash script, and how they function. This article was essential in the development of two of the three scripts for the proposal. The ACS and AMS both take a list of usernames as an input (as a text file), so it was necessary to read files within a bash script and access the elements within them. The article provided methods to add this functionality, as well as the specific syntax (language) for writing the code, both of which were implemented in the script development process.

Changes to the Project Environment

The project was proposed to resolve a common vulnerability in enterprise environments, a weak security posture due to insecure passwords. In short, the project altered this environment by increasing the strength of employee passwords, therefore increasing the overall security posture as well. There were several, more specific changes that made this a reality.

The project refined a company's password policy which altered the company's strategy and culture. The strategy was changed by updating the method for password creation, allocation, and modification. The original strategy had users create their own passwords, using temporary credentials to allow them initial access. The new policy creates passwords automatically, assigning them to user accounts at creation and then securely transmitting credentials. This shift increases security because it eliminates the human risk factor in password selection. It also increases efficiency by reducing the time and energy spent creating user accounts and passwords.

This improved efficiency is an inherent outcome of automation, which also accompanied a change in culture. In the original environment, the network administrator must manually create user accounts, and the user had to manually create secure passwords in accordance with the companies' requirements. These burdens also came with the inherent risk of an employee using an insecure password. The project shifted the company culture by increasing employee morale, automating these functions and eliminating the stress and worry associated with them. This freed time and energy, allowing employees to be more happy and more productive in the work environment.

Methodology

The project was based on the development of software, so it was completed utilizing the software development life cycle (SDLC). The phases of this methodology were completed as follows.

Planning:

The planning phase is where the purpose and scope were defined. The purpose of this project was to ensure password resiliency across a company by forcing employees to use strong passwords that are automatically generated.

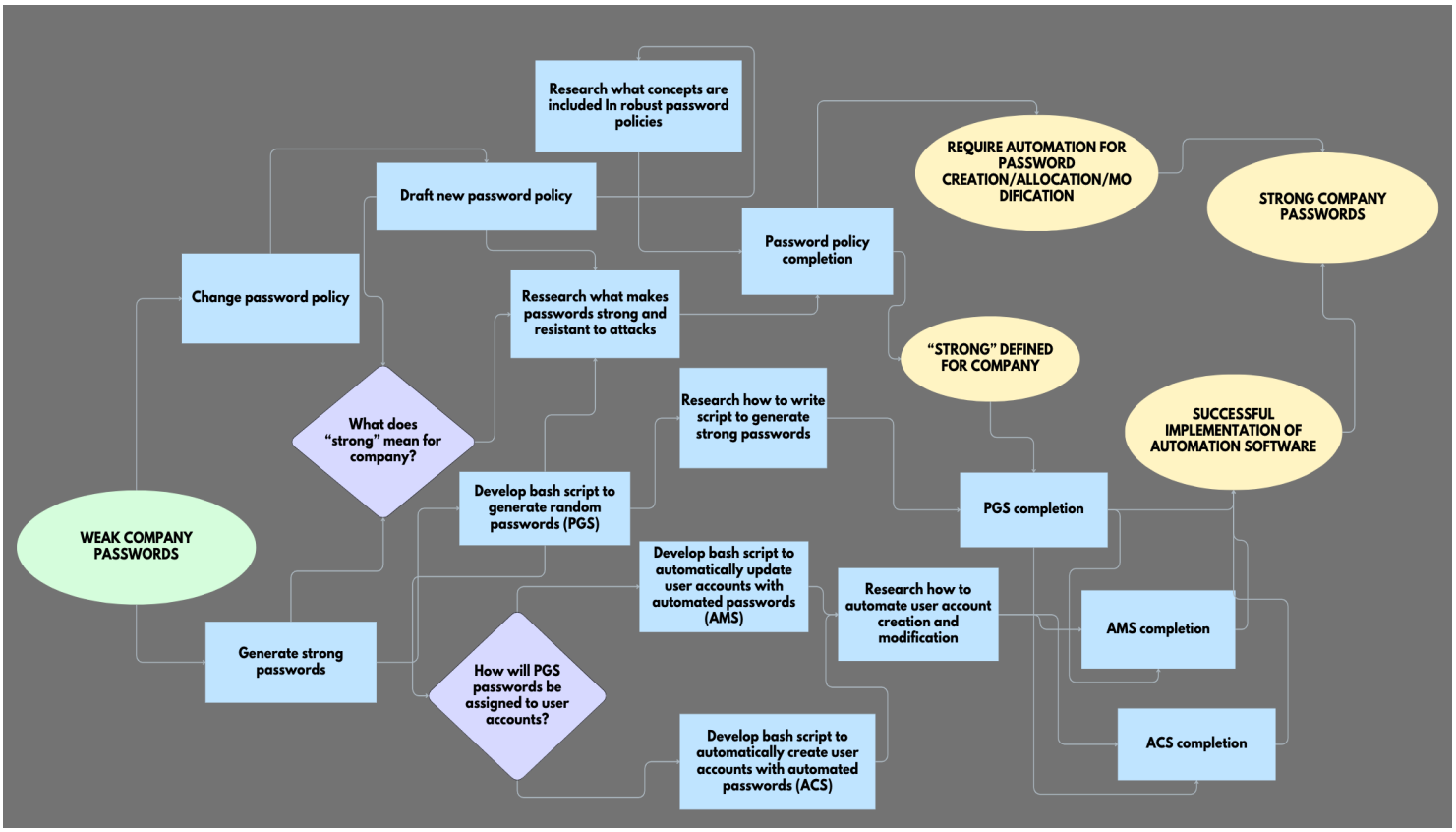
Requirements:

In the requirements phase the resources needed for the project's completion were defined. These included:

- A text editor to write the new password policy
- A Linux LDAP server to host employee account information
- Three bash scripts to generate passwords (PGS), automate account creation (ACS), and automate password updates (AMS).
- A text file of usernames to pass into ACS and AMS.
- A text file "credmap.txt" to monitor employee credentials while testing the function of each script.
- Gmail (or some other mail server that allows encrypted message sending) to securely transmit credentials.

Design:

In the design phase a prototype model of the project was developed. This model explained how the proposal was going to be realized. The flowchart illustrated below displays how the design phase took place in this specific project:



Development:

The development phase is where the password policy was written, and where the scripts were developed. The development process was not completely linear, with some parameters and policies being defined and implemented simultaneously. That being said, the creation of each deliverable occurred as followed:

- Password Policy Development:
    - Researched what constitutes a strong password
    - Researched sample password policies
    - Found sample password policy template
    - Drafted password policy using sample policy template
    - Added, removed, and modified details according to requirements for project
    - Revised and completed document
- PGS Development:
    - Researched how to generate random passwords
    - Wrote bash script
    - Incorporated policy specific requirements
    - Tested code
    - Code completion
- ACS Development:
    - Researched how to automate account creation
    - Created sample “c_users.txt” and “cremap.txt” for code testing

  - Wrote bash script
  - Implemented PGS upon its completion
  - Tested bash script
  - Script completion
- AMS Development:
  - Researched how to automate account modification
  - Created sample “m_users.txt” for code testing
  - Wrote bash script
  - Implemented PGS upon its completion
  - Tested bash script
  - Script completion

Testing:

In the testing phase, tests were run to ensure that the software performed as expected. For this project, testing was performed throughout development to test specific functions throughout each script’s creation. Tests were performed by creating sample username files and using these text files as inputs to the scripts to monitor behavior in certain cases. Results were analyzed through monitoring script output to the command-line, as well as monitoring changes the “credmap.txt” file that contained a list of usernames there corresponding passwords. The scripts were tested for appropriate behavior against the following cases:

- Creating new accounts for usernames that were already in the system or storage file
- Modifying passwords for usernames that were not in the system or storage file

- Creating or modifying accounts with a username file that contained duplicate usernames
- Creating or modifying accounts with no usernames in the system or storage file
- Errors in creating accounts with a given username
- Errors in assigning passwords to accounts
- Running scripts without root privileges
- Running scripts without proper arguments
- Running the ACS or AMS without the PGS in the current directory

Deployment:

In the deployment phase the proposal was prepared and put into use. The first step taken was the transference of employee account information to the LDAP server(s) (if necessary). Employee usernames were then stored in a text file, in the proper format for coherence with the necessary scripts. Then the three bash scripts were written on the LDAP server(s) and assigned the proper permissions. While this was occurring, a mass email was sent to all employees informing them of the password policy change. The new policy was also contained in this email. After this, the network administrator(s) ran the AMS with all current employees and sent encrypted emails to all users.

Maintenance:

In the maintenance phase the proposal was supported and continuously improved. For this project, this entailed periodic monitoring and testing of the scripts, searching for and fixing error cases and vulnerabilities. For example, after implementation a minor error was discovered where the script would stop running halfway through a username file upon detection of an unacceptable case (such as the creation of a username that already has an account). This elicited inefficiencies in the automation process, so the script was modified to mitigate this by checking for the unacceptable cases before creating and modifying any users. This maintenance stage also involved the development of alternate scripts that work with bash on macOS. This increased the usefulness of the project by expanding the environments in which the proposal could work.

Project Goals and Objectives

The main goal of the project was to strengthen a company's security posture by increasing password security. To achieve this, certain objectives had to be met: the password policy had to be refined, and the password automation, allocation, and modification process had to be completely automated. The deliverables that enabled the completion of these objectives were the creation and implementation of the new password policy document as well as the ACS, AMS, and PGS.

These files were successfully created and (the scripts) were added to a Linux system, requiring root permissions to run. The PGS script takes a number and accurately generates that number of passwords in accordance with the new password policy. The ACS takes a text file of usernames, an accurately creates each user account in the system (if username is not already in system) with a PGS generated password and adds them to the storage file ("credmap.txt"). The AMS takes a text file of usernames, and accurately changes the user's password (if username is in system) and updates the storage file ("credmap.txt"). The new password policy declares the requirements for a strong password, and compliance with this was ensured in the script generation. The policy also mandates the use of these scripts for password allocation, creation, and modification, which was to ensure that all company accounts were with strong passwords.

Project Timeline

The projected timeline had the proposal implementation taking place over the span of three days, from December 2nd to December 5th. The actual implementation was faster, taking only two days from November 26th to November 27th. The timeline moved forward due to the proposal's early acceptance which was unanticipated.

Day 1 (November 26th):

In the first day the following milestones were projected to take place: the new password policy creation, the development and testing of all three scripts, and (if necessary) the transfer of all employee credentials to the LDAP server. Because a LDAP sever was already in use with employee credentials, this milestone was ignored leaving more room other tasks. All other milestones were completed around the projected duration lengths of three to five hours. This was possible because they occurred simultaneously. With every added stipulation to the password policy, the corresponding script was updated and tested, which reduced the time spent in this stage of the process. The password policy was completed in around three hours rather than five, which allowed the employees to be notified of the change on this first day rather than the projected second day. This expedited the entire timeline.

Day 2 (November 27th):

In the second day the only projected milestone was to notify the employes of the password policy (and subsequently their individual passwords) changing. This was to give employees a day of preparation, so they were not surprised when the changes occurred. As explained above, this process occurred on the first day, so all the milestones projected for the third day were completed on this day instead. These milestones were: successfully updating current account credentials, and securely transmitting credentials through encrypted emails. The scripts ran according to plan, so these actions were completed within the projected duration of a day.

Unanticipated Scope Creep

An instance of scope creep occurred during the design phase of the project proposal. The original scope of the project was to increase password security by automatically generating and assigning passwords. Research for this uncovered many different onboarding automation and password management software with many different features. Inspired by the usefulness of these features, the project started to stray from its intended scope. The design plan shifted to a cross-platform app that securely creates, stores, and manages user accounts. The cost, time, resources, and expertise required to complete this greatly exceeded those required to fulfill the project's original purpose. This took away from the value and the practicality of the project's implementation, which proved to be a big problem for the proposal.

This problem was solved through the analysis of Norton's Free Password Management Software. They leveraged their products simplicity to create value for consumers, advertising only their main functions of secure password generation and management (Norton, n.d.). Adhering to this principle of simplicity focused the scope of the project, realigning it with the original purpose. This shifted the proposed design from a cross-platform software to three simple scripts hosted only on the directory server(s). This achieved the same purpose of enhancing password security while preventing a drastic increase in the overall scope of the proposal.

Conclusion

The actual outcome of the project was a new password policy and three bash scripts for creating, allocating, and modifying user account passwords for Linux systems.

The positive impact this had were:

- Increasing company security by eliminating the risk of weak passwords and ensuring the use of strong passwords. This is also achieved by eliminating the risks associated with the use of default passwords.
- Increasing company security by ensuring that passwords are resistant to brute-force and dictionary attacks.
- Increasing efficiency by automating account and password creation.
- Increasing employee morale by removing the burden of secure password creation and storage. This is achieved this by increasing employee confidence in company's security posture with the insurance of strong passwords.

The negative impacts this had were:

- Introduced new attack vectors for threat actors. They can exploit script vulnerabilities or attack the encrypted email transmissions to steal account credentials
- Introduced a single point of failure for password infrastructure. If the scripts or the server hosting them stops working, the company's whole authentication system is compromised
- Reduced current employee morale by requiring they all change their current passwords. Also achieved by requiring the secure storage of complex, randomized, twelve-character passwords that are hard to remember

Project success can be measured according to the checklists outlined in the proposal. (PGS is embedded within ACS and AMS, so its success is implicit within their checklists). Each checklist may include minor modifications according to research conducted since then.

Password Policy Success:

- Does it require passwords are automated via PGS?
    - Yes
- Does it require that passwords be at least 12 characters?
    - Yes
- Does it require at least three of following four: upper case, lower case, numbers, and special characters?
    - Yes
- Does it handle account creation and password allocation with ACS?
    - n/a
- Does it handle password resets with AMS?
    - Yes
- Does it include an account lockout policy?
    - Yes

ACS Success:

- Given a text file containing a list of usernames, can the ACS detect and stop running upon finding duplicate usernames (usernames that are already in the credmap.txt file)?
    - Yes
- Given a text file containing a list of usernames, can the ACS assign a unique password to each username and correctly update the credmap.txt file?
    - No, there is no function included to ensure that passwords are unique to each user.
- Given the success of the above criteria, after running the ACS do all passwords in the credmap.txt file meet the requirements defined in the new password policy?
    - Yes

AMS Success:

- Given a text file containing a list of usernames, can the ACS detect and stop running upon finding usernames that are not in the system (not in the credmap.txt file)?
    - Yes
- Given a text file containing a list of usernames, can the AMS assign a new strong password to each username and correctly update the credmap.txt file?
    - No, there is no function included to ensure that passwords are unique to each user.
- Given the success of the above criteria, after running the AMS do all passwords in the credmap.txt file meet the requirements defined in the new password policy?
    - Yes

The project was not completely successful according to the success criteria previously outlined in the proposal. It passed every test but one, the verification that passwords were unique to each user. Even with this, for the most part the project can still be considered successful. This is because of the lack of importance for this specific criterion. With the other measures included within the PGS and the overall password policy, the chance and impact of two employees with same password is low enough to be accepted. The strength of company passwords was still increased, eliminating and reducing the risk associated with weak password security. There were still risks according to the success criteria, but they did not exceed the company's risk appetite, so the project can still be considered a success.

There are numerous potential impacts that can occur because of the projects' implementation, such as the following:

- The functionality of the scripts be expanded upon to increase the efficiency of the account creation and account modification processes. For example, the scripts currently take only a list of usernames as an input but could be expanded upon to take group names as an additional input and assign the user to a specific group with the designated folder and permissions upon creation. If the workload decreases to a certain level, the number of current network administrators might become unnecessary, and the company can increase revenue by decreasing the overall salary they have pay to employees.

- A negative potential effect is the inability to create or modify user accounts due to issues with the scripts. This could lead to availability issues for current employees, keeping them locked out of their accounts for extended periods of time, and/or it could prevention of new users from creating accounts. This could have negative effects in terms of the finances, operation, and productivity of a company. This could also increase employee dissatisfaction.

References

Firch, Jason. (2024, March 16). *Sample Password Policy Template*. PurpleSec. https://purplesec.us/resources/cyber-security-policy-templates/password-security/

Iheanacho, Amarachi. (2024 July 4). Automating user creation with Bash Scripting. Dev. https://dev.to/amaraiheanacho/user-creation-in-bash-script-1617

Meetgor. (2021, November 21). *Bash Scripting – How to read a file line by line*. GeeksforGeeks. https://www.geeksforgeeks.org/bash-scripting-how-to-read-a-file-line-by-line/

Microsoft. (2024, May 28). *Password policy recommendations for Microsoft 365 passwords*. https://learn.microsoft.com/en-us/microsoft-365/admin/misc/password-policy-recommendations?view=o365-worldwide

Norton. (n.d.). *Norton Password Manager*. https://my.norton.com/extspa/passwordmanager?path=pwd-gen

Appendix A

New Password Policy:

Password Policy

Overview

Passwords are the front line of protection for user accounts. A weak password can result in the compromise of the entire corporate network. This policy specifies the processes and procedures that employees must follow in order to ensure strong passwords throughout the organization.

For this reason, anyone with an account on the company's network or systems is responsible for operating in agreement with the policies outlined below.

Policy Detail

Mandatory Usage

All employees must use automatically generated passwords, originating from the company password automation systems. Employees are not allowed to create their own passwords. Exceptions to this policy are made in the following limited cases:

- The password automation systems are compromised or made unavailable.
- An employee needs immediate access to company systems, and for whatever reason does not have access to their account. This access has to be explicitly authorized by at least three (3) other senior-level employees, with at least one holding a C-Suite position.
- The password reset functionality is compromised or made unavailable, and an employee is authorized immediate access to company systems.

In the event of one of these cases, please notify the IT team immediately, and the network administrator(s) will be responsible for creating and allocating passwords that meet the strength requirements defined later in this policy.

Responsibilities of Password Automation Systems

All automatically generated passwords must be strong, random, and in accordance with the requirements outlined later in the policy. The software and hardware that handle the automation, processing, and storage of account credentials must adhere to the following safeguards:

- The server(s) hosting account credentials and/or the automation scripts must be in a physically secure location, only accessible to authorized individuals.
- In case of hardware malfunction, the devices responsible for account creation and management must have backups in a secure storage location.
- Passwords must never be stored in plaintext. Reasonably strong hashing methods or encryption must always be used.
- The scripts that enable password and account automation must be regularly monitored, and tested to ensure the principles of integrity and availability.
- The scripts that enable password and account automation must adhere to the principle of least privilege. Read, write, and execute access should only be allocated as needed for those who need to run them, audit them, and patch them, and update them (e.g., if an employee is responsible for testing the scripts they only need execute access, not read or write access).

Individual Responsibilities

Individuals are responsible for keeping passwords secure and confidential. Therefore, they must adhere to the following principles:

- Passwords must never be shared with other individuals or organizations.
- Passwords must never be stored in a location that is easily accessible to others without encryption.
- Individuals must never remain logged into shared systems without monitoring.
- Passwords must be transmitted with the proper safeguards(i.e., shared via a secure password manager or sent via an encrypted email message)
- If account compromise is suspected or confirmed, the incident must be reported to IT immediately.
- For administrators, additional precautions must be taken to ensure password security:
    - Administrators must not circumvent the Password Policy for the sake of ease of use.

Password Requirements

All passwords must be in accordance with the following guidelines:

- At least twelve characters (14)
- Must contain three of the four: upper case, lower case, numbers, and special characters.
- Must not be obvious keyboard sequences (i.e., qwerty, asdfgh, q1w2e3r4t5)
- Must not comprised of words that can be found in a dictionary

Password Protection

User accounts are subject to the following guidelines:

- **Account Lockout**

  In order to limit attempts at guessing passwords or compromising accounts, an account lockout policy is in effect for all systems.

  - Accounts will lockout after twelve (12) invalid password attempts in fifteen (15) minutes

- **Account Termination**

  In the event that a user is no longer employed by the company, unless otherwise stated, their account should be immediately deleted from company systems.

Password Reset Options

In the event that a user needs to reset their password, the IT team should be contacted immediately. A network administrator will update the account with another automatically generated password, and securely transmit credentials to the user.

Enforcement

This policy will be enforced by the IT Manager and/or Executive Team. Violations may result in disciplinary action such as: suspension, restriction of access, or more severe penalties up to and including termination of employment. Suspicion of illegal activities may result in notification to the proper authorities.

This is the proposed policy that ensures password security for an organization. This specific policy does this by eliminating the option for employees to create passwords requiring the usage of automated passwords (shown in the Mandatory Usage section). These passwords are subject to stringent requirements which are also outlined in the policy. This policy is one of the deliverables of the project proposal.

Appendix B

Bash Scripts

Password Generating Script (PGS):

```
 GNU nano 7.2                                                                pgs.sh
#!/bin/bash

#Check if the current user is sudo, exit if it's not the superuser
if [ "$EUID" -ne 0 ]; then
    echo "Please run as root"
    exit 1
fi
#Function to generate a random password
generate_password() {
    # Define characters for each category
    lowercase="abcdefghijklmnopqrstuvwxyz"
    uppercase="ABCDEFGHIJKLMNOPQRSTUVWXYZ"
    numbers="0123456789"
    special_chars="@#_+=-"
    password=""

    while [ ${#password} -lt 12 ]; do
        password=""
        cat_count=0
        # Ensure at least one lowercase letter
        password+=$(echo $lowercase | fold -w1 | shuf -n1) #Linux
        cat_count=$((cat_count + 1))
        # Ensure at least one uppercase letter
        password+=$(echo $uppercase | fold -w1 | shuf -n1) #Linux
        cat_count=$((cat_count + 1))
        # Ensure at least one number
        password+=$(echo $numbers | fold -w1 | shuf -n1) #Linux
        cat_count=$((cat_count + 1))
        # Ensure at least one special character
        password+=$(echo $special_chars | fold -w1 | shuf -n1) #Linux
        cat_count=$((cat_count + 1))
        # Fill remaining password length with random characters from all categories
        all_chars="${lowercase}${uppercase}${numbers}${special_chars}"
        remaining_length=$((12 - ${#password}))
```

```
        remaining_length=$((12 - ${#password}))
        password+=$(echo $all_chars | fold -w1 | shuf -n$remaining_length | tr -d '\n') #Linux
        # Shuffle the password to mix the characters
        password=$(echo $password | fold -w1 | shuf | tr -d '\n') #Linux
        # Ensure there are at least 3 categories used
        if [ $cat_count -ge 3 ]; then
            break
        fi
    done
    echo "$password"
}
generate_password
```

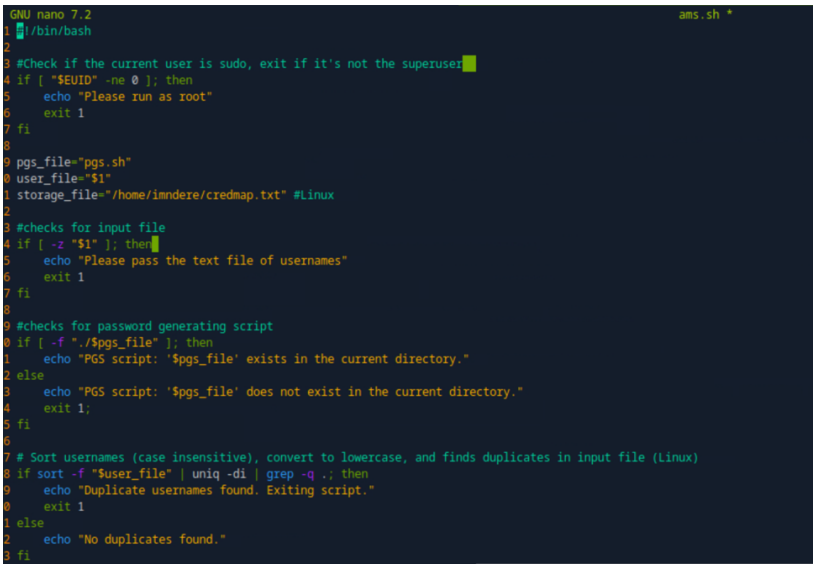
Account Creating Script (ACS):

```
 GNU nano 7.2                                                                                                  acs.sh
#!/bin/bash

#Check if the current user is sudo, exit if it's not the superuser
if [ "$EUID" -ne 0 ]; then
  echo "Please run as root"
  exit 1
fi

pgs_file="pgs.sh"
user_file="$1"
storage_file="/home/imndere/credmap.txt" #Linux

#checks for list of usernames
if [ -z "$1" ]; then
  echo "Please pass the text file of usernames"
  exit 1
fi

#checks for password generating script
if [ -f "./$pgs_file" ]; then
    echo "PGS script: '$pgs_file' exists in the current directory."
else
    echo "PGS script: '$pgs_file' does not exist in the current directory."
    exit 1;
fi

# Sort usernames (case insensitive), convert to lowercase, and finds duplicates in input file (Linux)
if sort -f "$user_file" | uniq -di | grep -q .; then
    echo "Duplicate usernames found. Exiting script."
    exit 1
else
    echo "No duplicates found."
fi
```

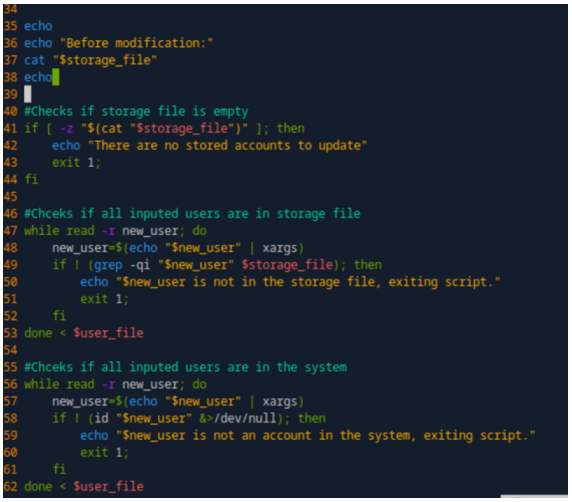
```
fi

echo
#Checks if storage file is empty
if [ -z "$(cat "$storage_file")" ]; then
    echo "Storage file empty"
    #If file is empty, chceks if any inputted usernames are in the system
    while read -r new_user; do
        new_user=$(echo "$new_user" | xargs)
        if id "$new_user" &>/dev/null; then
            echo "$new_user is already an account in the system, exiting script."
            exit 1;
        fi
    done < $user_file
else
    #If storage file isnt empty, checks if any inputted usernanes are in storage file
    while read -r new_user; do
        new_user=$(echo "$new_user" | xargs)
        if grep -qi "$new_user" $storage_file; then
            echo "$new_user is already in the storage file, exiting script."
            exit 1;
        fi
    done < $user_file
    #If storage file isnt empty, also checks if any inputted usernanes are in the system
    while read -r new_user; do
        new_user=$(echo "$new_user" | xargs)
        if id "$new_user" &>/dev/null; then
            echo "$new_user is already an account in the system, exiting script."
            exit 1;
        fi
    done < $user_file
```

```
    done < $user_file
fi

#If input file passes all checks, create username with password
while read -r new_user; do
    new_user=$(echo "$new_user" | xargs)
    #create password
    password=$(./pgs.sh)
    #Create a new user with home directory
    useradd -m "$new_user"
    #error check
    if [ $? -eq 0 ]; then
        #Set the password for the user
        echo "$new_user:$password" | chpasswd
            if [ $? -eq 0 ]; then
                echo "User: $new_user successfully created with password: $password"
                echo "username:$new_user; password:$password" >> $storage_file
            else
                echo "Failed to set password for $new_user. Exiting script."
                exit 1
            fi
    else
        echo "Failed to create user $new_user. Exititng script."
        exit 1
    fi
done < $user_file
```

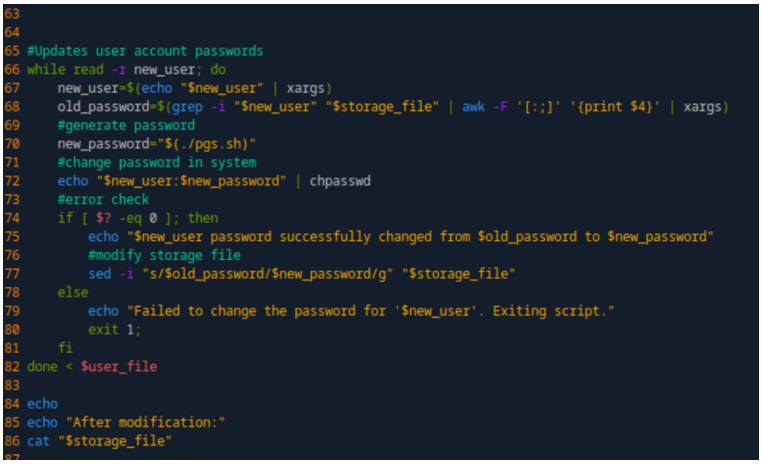
Account Modifying Script (AMS):



```
GNU nano 7.2                                                                    ams.sh *
#!/bin/bash

#Check if the current user is sudo, exit if it's not the superuser
if [ "$EUID" -ne 0 ]; then
    echo "Please run as root"
    exit 1
fi

pgs_file="pgs.sh"
user_file="$1"
storage_file="/home/imndere/credmap.txt" #Linux

#checks for input file
if [ -z "$1" ]; then
    echo "Please pass the text file of usernames"
    exit 1
fi

#checks for password generating script
if [ -f "./$pgs_file" ]; then
    echo "PGS script: '$pgs_file' exists in the current directory."
else
    echo "PGS script: '$pgs_file' does not exist in the current directory."
    exit 1;
fi

# Sort usernames (case insensitive), convert to lowercase, and finds duplicates in input file (Linux)
if sort -f "$user_file" | uniq -di | grep -q .; then
    echo "Duplicate usernames found. Exiting script."
    exit 1
else
    echo "No duplicates found."
fi
```



```
echo
echo "Before modification:"
cat "$storage_file"
echo

#Checks if storage file is empty
if [ -z "$(cat "$storage_file")" ]; then
    echo "There are no stored accounts to update"
    exit 1;
fi

#Chceks if all inputed users are in storage file
while read -r new_user; do
    new_user=$(echo "$new_user" | xargs)
    if ! (grep -qi "$new_user" $storage_file); then
        echo "$new_user is not in the storage file, exiting script."
        exit 1;
    fi
done < $user_file

#Chceks if all inputed users are in the system
while read -r new_user; do
    new_user=$(echo "$new_user" | xargs)
    if ! (id "$new_user" &>/dev/null); then
        echo "$new_user is not an account in the system, exiting script."
        exit 1;
    fi
done < $user_file
```



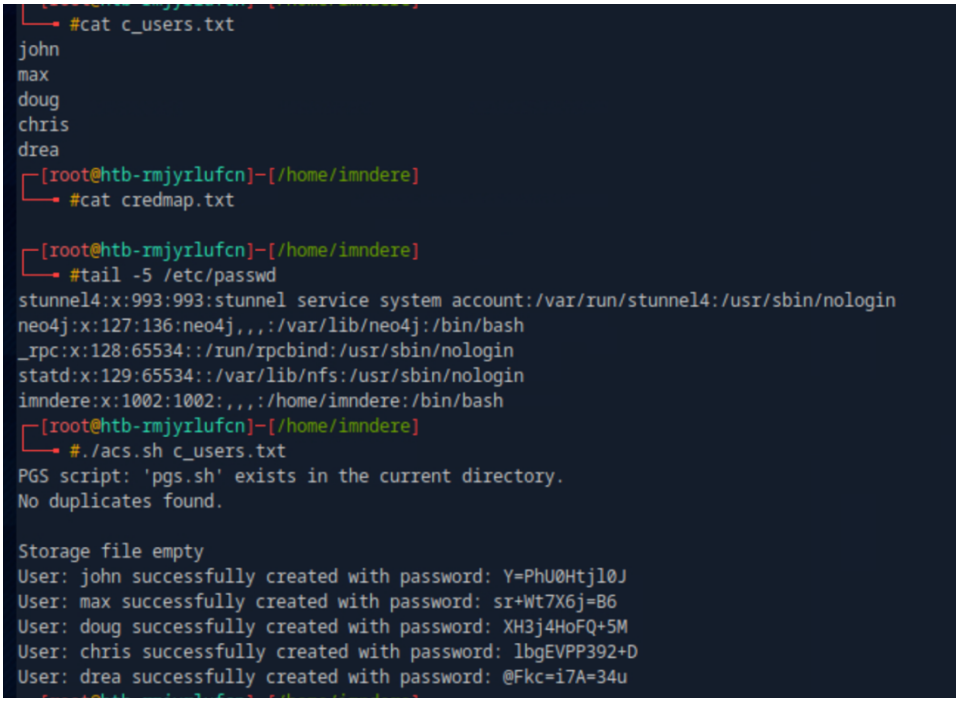
```


#Updates user account passwords
while read -r new_user; do
    new_user=$(echo "$new_user" | xargs)
    old_password=$(grep -i "$new_user" "$storage_file" | awk -F '[:;]' '{print $4}' | xargs)
    #generate password
    new_password="$(./pgs.sh)"
    #change password in system
    echo "$new_user:$new_password" | chpasswd
    #error check
    if [ $? -eq 0 ]; then
        echo "$new_user password successfully changed from $old_password to $new_password"
        #modify storage file
        sed -i "s/$old_password/$new_password/g" "$storage_file"
    else
        echo "Failed to change the password for '$new_user'. Exiting script."
        exit 1;
    fi
done < $user_file

echo
echo "After modification:"
cat "$storage_file"
```

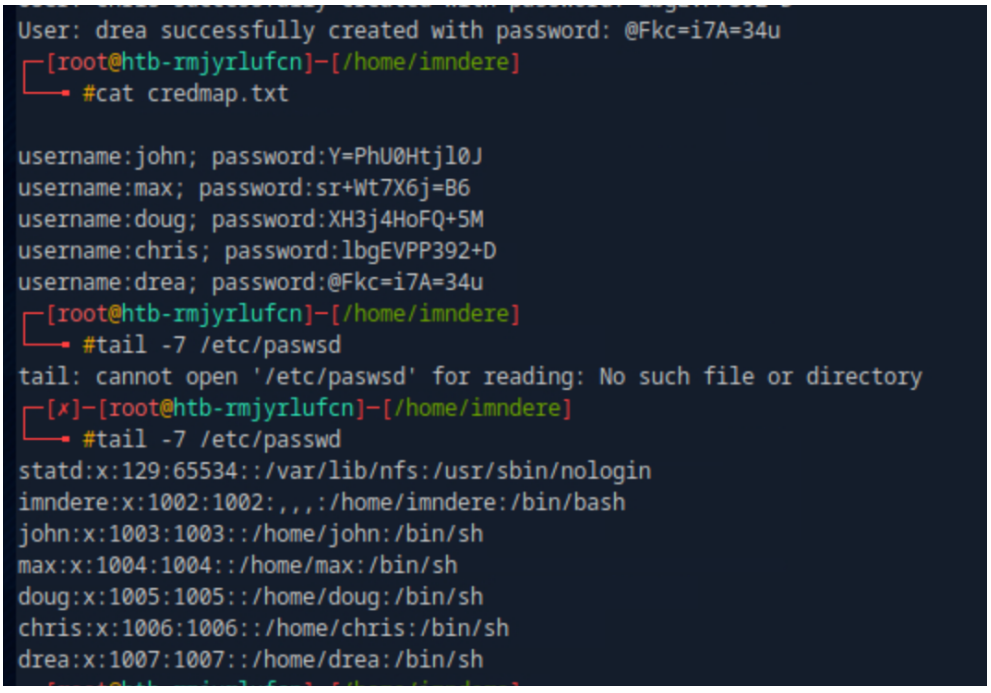
These are the bash scripts that allow for the automation of password creation, allocation, and modification. These scripts provide the basis for the password policy displayed above. The ACS and AMS both take a list of usernames as an input, and create user accounts with PGS generated passwords, or update accounts with PGS generated passwords. These scripts were also deliverables of the project proposal.

Appendix C

Bash Script Test Results

ACS Test:





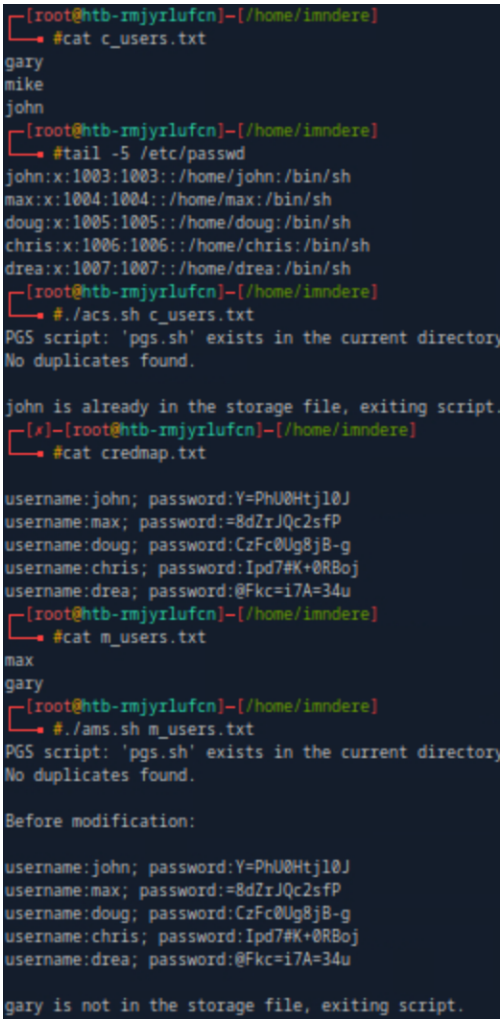
This displays the successful automation of password generation and allocation. Shows that no users were originally contained in the storage file or in the system. After running the ACS with a text file of usernames, the users were created in the system and stored in the storage file with their corresponding password. Also, passwords are all strong and in accordance with policy guidelines as well.

AMS Test:

```
└─• #cat m_users.txt
max
doug
chris
┌─[root@htb-rmjyrlufcn]─[/home/imndere]
└─• #./ams.sh m_users.txt
PGS script: 'pgs.sh' exists in the current directory.
No duplicates found.

Before modification:

username:john; password:Y=PhU0Htjl0J
username:max; password:sr+Wt7X6j=B6
username:doug; password:XH3j4HoFQ+5M
username:chris; password:lbgEVPP392+D
username:drea; password:@Fkc=i7A=34u

max password successfully changed from sr+Wt7X6j=B6 to =8dZrJQc2sfP
doug password successfully changed from XH3j4HoFQ+5M to CzFc0Ug8jB-g
chris password successfully changed from lbgEVPP392+D to Ipd7#K+0RBoj

After modification:

username:john; password:Y=PhU0Htjl0J
username:max; password:=8dZrJQc2sfP
username:doug; password:CzFc0Ug8jB-g
username:chris; password:Ipd7#K+0RBoj
username:drea; password:@Fkc=i7A=34u
```

```
└─• #cat credmap.txt

username:john; password:Y=PhU0Htjl0J
username:max; password:=8dZrJQc2sfP
username:doug; password:CzFc0Ug8jB-g
username:chris; password:Ipd7#K+0RBoj
username:drea; password:@Fkc=i7A=34u
```

The above screenshots display the successful automation of password modification. Ran the AMS with a text file containing three users who were already in the system and storage file (displayed in the previous screenshots). Output contained contents of the original storage file, each individual password change, and the contents of the updated storage file. The contents of the final storage file are also displayed to show that they align with the contents displayed in the script. All passwords are still strong and in accordance with the password policy.



This screenshot includes some test cases that displays how the scripts handle errors. The display shows the scripts trying to create users that were already in the system and trying to modify users that were not in the system. The ACS and AMS both withstood this test. They exited the script and explained the issue that prevented the script from being run successfully. This functionality shows the practicality of the project, displaying its usefulness in real world scenarios where errors are inevitable.