|  |
| --- |
|  |
| Password Management Systems and Password Crackers |
| A Brief Overview |
|  |
| **CIS 423 Class Project** |
| **12/17/2014** |

|  |
| --- |
|  |

Contents

[Part I 4](#_Toc406361070)

[Password Management Systems Overview by Yi Hernandez 4](#_Toc406361071)

[Requirements of a Password Management System by Ryan Boaz 5](#_Toc406361072)

[Generic Architecture for Password Management Systems by Mark Darr 6](#_Toc406361073)

[Modern Password Management System Architecture by David Chang 8](#_Toc406361074)

[State-of-the-Art Password Management Systems’ Advantages and Limitations by Dwaine Wintz 10](#_Toc406361075)

[Structure of the Developed Password Management System by David Chang 13](#_Toc406361077)

[Functionality of the Developed Password Management System by Conner Ingram 15](#_Toc406361076)

[Behavior of the Developed Password Management System by Bo Nugent and Ryan Titzer 16](#_Toc406361078)

[Part II 18](#_Toc406361079)

[Definition of Password Cracking by Dwaine Wintz and Ryan Boaz 18](#_Toc406361080)

[Generic Architecture of Password Cracking Tools by Mark Darr and Ryan Titzer 18](#_Toc406361081)

[Classification of Password Cracking Tools by David Chang and Conner Ingram 19](#_Toc406361082)

[Experimentation with Password Crackers 20](#_Toc406361083)

[Rainbow Crack by Yi Hernandez 20](#_Toc406361084)

[Burp Suite by John D’Antonio 21](#_Toc406361085)

[John the Ripper by John D’Antonio 22](#_Toc406361086)

[SHA-256 Hash Kracker by John D’Antonio 22](#_Toc406361087)

[References 23](#_Toc406361088)

# Part I

## Password Management Systems Overview

A password management system (PMS) is software used by individuals to organize and encrypt many personal passwords using a single login; this often times involves encryption keys. These programs keep these passwords stored on a database or file, which a master key is normally required to access it. The core purpose of a PMS is to securely store large collections of passwords, but it also functions as a form of filling and password generation.

The definition of a PMS can be divided into multiple categories. For Identification, it is a "unique identifier that a user types to sign into a system or application is that user's login ID onto that system." As a way to authenticate a person, it "is a process by which a user provides his identity to the system," which is often times at the log in. The password manager must be able to authenticate the user by having the user prove to the system that he/she is legitimate. It can be a combination of something the user knows (passwords), a physical object (smart card or ID card), or something about the user, like a physical trait scanned by biometrics. These things also must be hard to replicate in order to prevent spoofing. Most systems now a day's use a combination of theses said authentication techniques in order to authorize and authenticate legitimate users.

The passwords that the PMS is responsible for also have to protect the information stored onto the files. In order to prevent spoofing of the username and password, which is often time able to be guessed, the system hashes the two components and stores that hash on a different file. When doing the check, the username, password, and the salt value are hashed together to get a hash value. This value is then checked when the user inputs his/her information in order to get into the system. If it matches, then the user is authorized access, and is denied access when the information is incorrect. The salt value can also be part of the code stored on a token, or the image of a biometric that has been converted into a value. Another way to prevent unauthorized intrusion is during the login in phase. When an unauthorized user does not have the proper password, he/she will use programs to make attempts onto the system. The system can deny access to that account if it fails a certain number of attempts within a time frame. The monitors and administrators can then track the source of the intrusions and block that source from further attempts. Granted if the source is spoofed, it will take more time to find where it originated.

Aside from the security aspect of password management systems are also responsible for the maintenance of the users. This includes the ability to reset passwords if need be and synchronize passwords across multiple systems. These services need to be available in case there is a breach in the system and this would allow users to change their passwords to make the system secure again. In order to ensure that the users do not reset account passwords to any random user, challenge-response mechanisms are set in place to authenticate users. On top of having the account and password to gain access into the system, the PMS is also designed to deny accessibility to those who are trying to intrude. On the login, people who are unable to log in within the certain number of tries cause the account to be locked. In the recovery, the authorized user would be able to regain access by answering security questions and authenticate that they are who they say they are. Also, any changes hat do happen must be applied across the system in order to ensure proper synchronization. If not, the older passwords would be able to gain access to the account on a different server using the older credentials. The PMS is a tool that is needed that acts not only as security device, but also an enabler for authorized users.

With this even said, the PMS is also vulnerable. Usernames and passwords are often stored on a text file or database, and those can be accessed by attackers. A way to help mitigate these attacks is by having these files encrypted. Also, by having the username and passwords on separate files, this also helps prevent attackers from accessing all the information at once. Current hashing methods are able to be broken into, granted it varies depending on the program. Brute force methods take time to break into due to it testing every possible outcome until the correct value is found. Users must balance their passwords to be long enough so password crackers will take longer to decipher the password, but simple enough so the user can remember them. Password crackers currently are capable of breaking into systems that use lowercase, uppercase, and numbers with relative ease. When combining special characters the amount of passwords increase drastically causing more time to be used on the cracker. At a point, the amount of time it would take to break a password would not be worth the effort. The class project focuses on developing a PMS and the understanding the "in" and "out" of the system.

## Requirements of a Password Management System

In order to create a robust password management system (PWD-MS) several key features must be accounted for to include enrollment, verification, and identification procedures, secure storage of passwords and files, proactive password protection, and password replacement methods.

The enrollment process should be the beginning focus for PWD-MS. A straightforward user interface (UI) for enrollment into the PWD-MS is necessary to ensure users are able to properly utilize the system (a PWD-MS that cannot be used by a user is essentially useless). The database must also be able to create and access files that store usernames and passwords in a secure manner. In order to attain secure enrollment processes, the enrollment process must include a hashing method to store a user’s password, ensure usernames are not used more than once concurrently, and utilize proactive password checking such as creating password length requirements or requiring a combination of special ASCII characters, uppercase and lowercase characters, and numbers.

In the event a user’s password becomes compromised or lost, password replacement or recovery methods must be set in place by the PWD-MS. Users should be expected to forget their password at some point in time. To remedy this problem a proper PWD-MS should include a method to reset the user’s password and send the automatically generated password to a linked email account or other form of verification, or involve a series of secret recovery questions associated with the account that, if answered correctly, will prompt the user to enter and verify a new password for enrollment into the system.

## Generic Architecture for Password Management Systems

Password management systems are important because they enable administrators to provide confidentiality, authentication, and access control. As a basis for our password management system, we decided to examine the components that Solaris 10 OS uses to give users system access. The focus of this section is to identify how Solaris stores its user account information, and authenticates users.

Before a user can be authenticated, a user must first be registered to a configuration file within the operating systems files. Solaris 10 OS stores user account and group entry information in the following system files:

* /etc/passwd
* /etc/shadow
* /etc/group

Solaris 10 OS stores usernames and the user’s primary group in /etc/passwd. The /etc/shadow file contains the following fields:

* loginID: The user’s login name.
* password: Solaris 10 OS contains a password policy associated with this field.
* lastchg: the password modification date.
* min: the minimum number of days required between password changes
* max: the maximum number of days the password is valid before the user is prompted to enter new password at login.
* Warn: the number of days the user is warned before the password expires.
* Inactive: the number of inactive days for the user before the user’s account is locked.
* Expire: date the user account expires.

/etc/shadow file contains the encrypted passwords associated with the usernames. Last is the /etc/group file that defines the default system group entries, which also allows a user to belong to secondary groups1.

In sum, Solaris 10 OS issues a login prompt, accepts login names and passwords, and authenticated the users. The system authenticates it’s users by looking through the system files, and compares the stored credentials with the entered credentials2. If the credentials do not match the credentials stored on file then, the system rejects the user’s access.

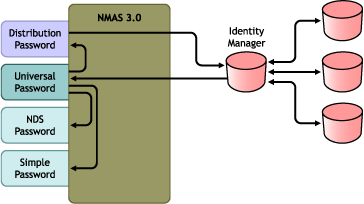
Solaris 10 OS contains functions that are similar to generic password management systems across the board. Generic password management systems contains configuration files in where credentials are stored, possibly a password checker, a password policy, a way to recover the password if the user forgets their password, user groups to restrict/grant permissions, and a way to authenticate authorized users while simultaneously denying access to unauthorized users.

A generic password management system either stores their passwords and usernames in a single file, or stores the user’s credentials into two separate files much like Solaris. System administrators may or may not implement a password policy for acceptable passwords within their password management systems. Depending on the level of the account, a system administrator may allow any password to be accepted from a general user, but may require an administrative account to have uppercase, lowercase, numbers, and special characters in their passwords to prevent dictionary or other forms of password guessing attacks. Other mechanisms system administer may impose to prevent password guessing is to add a counter for the number of failed attempts a user attempts after a certain amount of time. Users forget their passwords from time to time, so a system administrator may implement a way to reset a user password by a reset method. An example of a password reset method is to require a user to answer secret questions at the time of registration, and then verifying that the user is authorized by asking the user to answer the secret questions correctly during the request for the password reset. User groups that would restrict/grant permissions typically include administrative accounts, regular user accounts, or user accounts with elevated privileges. The last part of a generic password management system is to authenticate users. The password management system ultimately fails if authorized user’s accounts are not available because of a system’s malfunctioning authentication system.

All in all, Solaris 10 OS contains many components that generic password management systems incorporate. The important components of generic password management systems are a method of storing and encrypting credentials, and authenticating users. Things to consider as a system administrator of a password management system is the value of the user account. If a user does not have any restricted permissions it would be wise to have a strong password policy on that account, but if the user has very limited permissions, then allow weak/“crackable” passwords. Future consideration in relation to our password management system would include possibly adding user groups to our password management system. Therefore, denying regular users access to some function; while allowing users with escalated privileges to perform some task that regular users do not have access to do.

## Modern Password Management System Architecture

NetIQ Identity Manager uses an identity vault where one can synchronize different types of passwords and then have it managed by the Identity Manager. This is displayed in the diagram below.



1. Passwords come in through Identity Manager.
2. Identity Manager goes through NMAS to directly update the Universal password.
3. NMAS synchronizes the Universal password with the Distribution password and other passwords according to the NMAS password policy settings.
4. Identity Manager retrieves the Distribution password to distribute to connected systems that are set to accept passwords.

This universal password requires the password management system to have the following.

1. Password Policy Configuration
2. Password Synchronization Settings
3. Driver Configuration

These requirements are specific to this software Identity Manager and are not a generic guideline to follow for all PMS but it exemplifies how modern PMS software is modeled.

**Table A-2**Synchronizing by Using Universal Password

|  |  |
| --- | --- |
| **Advantages** | **Disadvantages** |
| Allows synchronization of passwords to and from the Identity Vault and the connected system.  Allows passwords to be validated against the NMAS password policy.  Allows e-mail notifications for failed password operations, such as when a password coming from a connected system does not comply with Password.  Supports the Check Password Status task in iManager, if the Universal password is being synchronized with the Distribution password and if the connected system supports checking passwords.  NMAS enforces the Advanced Password Rules in your password policies, if you have the rules enabled. If a password coming from a connected system does not comply, an error is generated, and an e-mail notification is sent if you have specified that option.  If you don't want password policy rules enforced, you can deselect*Enable Advanced Password Rules* in the NMAS password policy. | By design, resetting passwords in the connected system is not supported with this method because the Distribution password and Universal passwords might not be the same, depending on your settings in the password policies. |

The advantages are clearly displayed by NetIQ in that this PMS software allows for a lot of customization in how an administrator can enforce policies, validate passwords, and notify users without much manual interaction.

The limitation of this modern PMS software is that one cannot reset the passwords in a connected system because the passwords that are already distributed and the universal passwords might not correlate. This seems to be a limitation problem that exists in basic PMS to modern state of the art PMS.

## State-of-the-Art Password Management Systems’ Advantages and Limitations

PMS architecture has been developed over the years for one main reason. Overtime the plethora of passwords that users create on a network is too many to handle, to create a usable system these passwords need to have their own management system. These management systems all have certain things in common, for instance they all encrypt the passwords using a strong symmetric encryption algorithm such as AES or Blowfish. They all mask the password from the view of prying eyes and shoulder surfers. They support a method of entering the password at login prompts without the need to type them. This prevents shoulder surfing as well and also thwarts key loggers. If the clipboard is used to copy a password from the application it automatically clears the clipboard afterword. The best of them keep passwords encrypted even in the memory of the running application. This prevents the contents of the computer's memory or page file from divulging plaintext passwords.

PMS systems all use different algorithms to make their passwords encrypted but any user who is really concerned about a PMS system knows to look for something like MD5 to encrypt their password. These are hash functions that encrypt the password when it is entered into a database. For example, PHP5 has the built in MD5 that can use the given hash function SHA-256:

$password\_hash = hash("sha256", "iamawesome");

// 4aa4029d0d0265c566c934de4f5e0a36496c59c54b6df8a72d9c52bdf0c1a0e8

The second line is the encryption, if someone wanted to hack a user they would have to decode that entire line of code created by the PH5 PMS. This way, only the user ever knows the real password. If someone were to look at the database of stored hashes (whether it is a dishonest employee, or because it was stolen) they'll only ever be able to see the hash, and won't be able to go around getting into people's email accounts.

An even better approach to a PMS algorithm is to use a salted hash. A salted hash random bits that are used as an input to a key derivation function, basically just another word for a nonce. Normally to create a hash, a person provides one thing as input (the original password) and gets the hash as an output. A salt/nonce is a random string of characters used as another input into the hash function in order to get the output.

So now when storing a user's password, instead of just hashing the password, one can concatenate the password and the salt/nonce and hash that instead. So in PHP it would look something like this,

$salted\_hash = hash($password . $random\_salt);

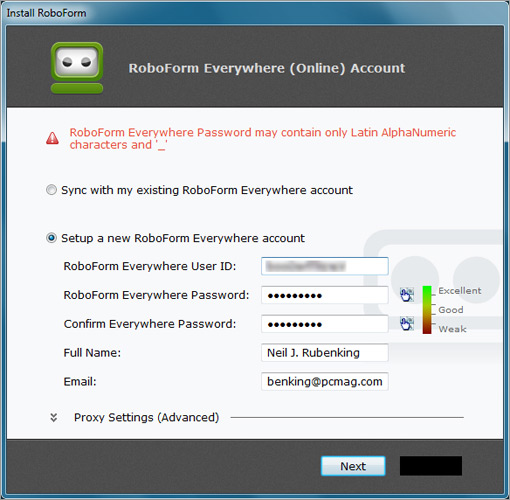
The salt should be a string of random characters, ideally it should be long (more than 20 characters) and not just alphanumeric, it should have special characters too.

Functions are highly customizable as long as the random salt and password are both used in order to construct the stored hash. The method used does not change the effectiveness of the password storage since one is not really any more secure than the other. Relying on the design of how one hashes the password and salt together to provide security is called security through obscurity and should be avoided, since in reality the method used does not affect the security.

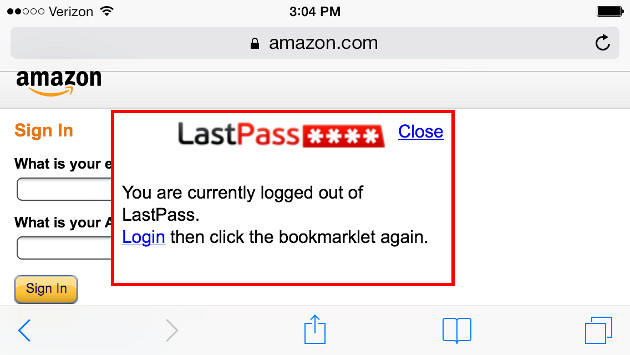
$salted\_hash = hash(hash($password) . $random\_salt);

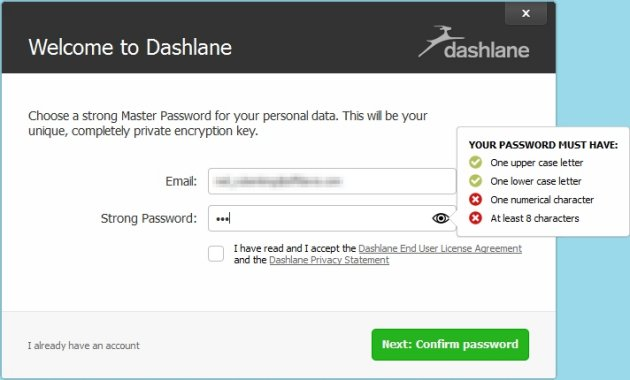
$salted\_hash = hash(hash($password) . hash($random\_salt));

There are many password management systems that are used today, but the ones that have been rated the overall best in 2014 are “RoboForm Everywhere”, “LastPass Premium”, and “Dashlane”. It is important to understand why these password management systems are regarded so highly, and why they seem to be above the rest due to user reviews.

 The all new RoboForm Everywhere license gives unlimited access to all RoboForm data on multiple computers, USBs, tablets, and mobile devices. RoboForm Everywhere seamlessly keeps the user’s passwords and other data in sync, making password saving extremely easy and usable. Obviously this is a great thing to keep in mind for the user who would want something to save and backup passwords, and not have to keep re installing when switching devices. One of the main benefits of RoboForm Everywhere is that the user will always have a backup copy of their passwords. Since the user has a RoboForm Everywhere account, a copy of their Passcard files is securely stored on their server. The user’s stored data is accessible online, and bookmarks allow automated login from anyweb browser. Compared to other password management systems it is relatively cheap (10). There are some cons as well though, like with any big time password management system there are flaws. For one the remote access does not allow editing form-fill data. Some features, such as plug-ins, are available for only some browsers (chrome and firefox). Bookmarks and smartphone apps cannot fill forms, and some smartphones can't sync automatically which can be annoying for the user. Some users worry about their saved passwords even though it is encrypted (because its saved on all devices) if someone was able to gain access to the password they would have access to all devices with the saved password.

The second highly regarded password management system is “LastPass Premium” which is also regarded as a powerful and flexible PMS. It exhibits effective web form filling, is usable on a broad spectrum of mobile devices, and handles application passwords. It can easily work with secondary authentication devices like fingerprint scanners and USB keys. A user can also organize their vault entries into categories, and then assign categories to certain identities, limiting how many passwords are visible at a time. For someone who is extremely security minded this is something that is highly sought after, but for a user who just wants their phone secured the vault features can be easily overlooked.

There are some concerns with Last Pass, for instance it cannot manage all application passwords, so anywhere no plugin solutions didn't work correctly in testing it won't work in a real life application scenario. Unlike RoboForm it does not support all devices, or web browsers. Main concentration as of 2014 is with iphones and their safari web browsing platform. The price is very high for it not doing many of the features that basic users would want which makes its rating a little lower than that of Roboform. 

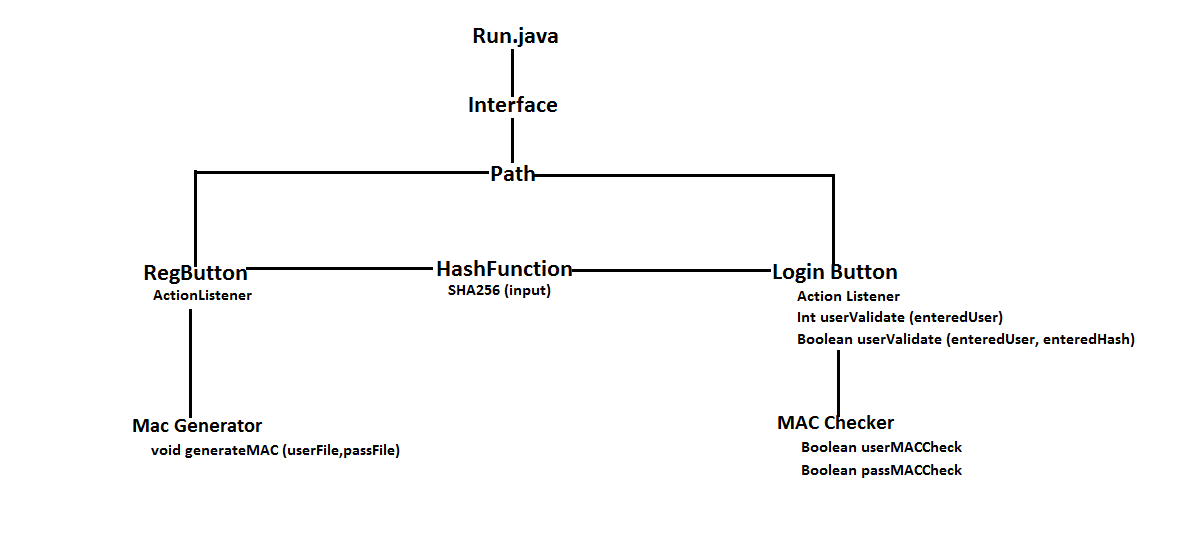
 The final password management system users find to be the best is “Dashlane”. Dashlane has a unique user interface experience, which can make the password management system fun to interact with. It syncs across PC, MAC, and Android iOS. What sets it apart is that if a user uses it on only one device the service is completely free. But if a user uses it on more than one device it costs even more than LastPass. It has an emergency password recovery system so if a user cannot seem to recall their password for whatever reason it will help them reopen their account. Dashlane also tracks online shoppingto authenticate a user’s account and rates the user’s password so that the user can know how strong the encryption will be and how hard it would be to break the password.

On the downside there are some pages that dashlane can't handle. Dashlane will throw an error screen for some users this never happens, but for others it can be a common occurrence.

## Structure of the Developed Password Management System

We will analyze the structure of our PMS by looking at the breakdown of our components and how they interact with each other.

The diagram above depicts how the components are linked and structured in our PMS. The MAC Generator is with the Register Button and the MAC Checker is with the Login Button. Then the SHA256 Hash Function is with both the register and login. The components are structured this way intentionally. The MAC needs to be generated for the newly created username and password when someone registers, then that MAC needs to be checked when the user is logging in. The hash function needs to be working for both register and login in order to create and check the hash.



The actual classes are broken down as follows:

1. Run.java – main method
2. Interface.java – GUI interfaces
3. LoginButton.java – action listener to log in
4. RegButton.java – action listener to register
5. Hashfunction.java – SHA256
6. MACGenerator.java – generate our MAC
7. MACChecker.java – integrity check of files using MAC
8. Path.java – to run the code on any computer

We divided up our code into 8 different classes to separate our GUI interface and action listeners, MAC integrity check, and hash function. The way we separated our coding components helped us maintain separation of concerns in our PMS structure. It also allowed us to enable or disable the MAC up to the administrator’s discretion. Our code was originally in 2-3 classes but this lacked organization and separation of concerns. Our current organization for our developed PMS allows us to make the proper changes and improvements in our code according to the specific concern we must address. This allows developers to patch and update the PMS with ease rather than re-structuring one class of the whole code.

We store the hashed usernames and passwords into two separate files as another measure of security. If an intruders gains access to one or the other then there is still a sense of security in that they do not know the missing part. This is our simple form of distribution that is implemented at a complex level with secure database storage systems. Oracle Database XE is composed of logical structures, physical structures, and recovery-related structures. The logical structure of our PMS would be the hash tables used by SHA256. The physical structure is how we separate the usernames and passwords into two separate files. Our PMS does not have a recovery-related structure but we do have a MAC code we use to check the integrity of our stored files. Our MAC code will deny access to everyone once the integrity of the files has been altered. Although this eliminates availability, it is our measure of maintaining integrity of the files.

If we had more time and ability then we would try to add a recovery component to our PMS. This would allow us to recover to a specific state of the PMS from unintentional mistakes the administrator makes or from tampering of the files from intruders.

We chose SHA256 over the original Java hashCode() for several reasons. This is a secure hash algorithm which is computed with 32bits and 64bit words. According to NIST, the message to be hashed is padded with its length so that the result is a multiple bit of 512 bits and then parsed into 512 bit message blocks. SHA256 is a fast hashing function and easy to implement in the hardware. We used this because this was more secure than the Java hashCode() function we were originally planning to use which takes the data stored in an instance of a class into a single 32 bit hash value. SHA256 is well suited for security sensitive applications as ours where access to usernames and passwords can be critical and lead to other valuable information.

## Functionality of the Developed Password Management System

The functionality of our PMS has been greatly improved from previous versions that were created individually. We were able to integrate many different features that allow for improved efficiency, organization, and ease in locating bugs or implementing new features. As mentioned above, there are 8 classes that are all working seamlessly together to create one working PMS. To ensure that our PMS would work on any user’s computer we made sure that the paths to where files were created and stored were not hardcoded. To be able to call the file paths there are four methods created that return the file path in Path.java.

Our interface is a very simple interface that consists of two text fields (username/password) and two button objects (register/login buttons). Inside the RegButton and LoginButton contains the two separate actions listeners. The RegButton ActionListener is able to take the username and password from the text fields to write them to a corresponding user.txt and pass.txt file. When writing these two values to their files, the password is ran through the Sha256() method that outputs a hash value that represents that password. The username is inserted in the user.txt file in plaintext because it is used as an index when searching for username availability and login validation. The ActionListener for RegButton also uses the generateMAC() method to create a new MAC for both user.txt and pass.txt to then store in their own corresponding files (userMAC.txt/passMAC.txt). These codes are also computed based on the data in the user.txt and pass.txt files then hashed.

The LoginButton has three main components that include the ActionListener and two user validation methods. The ActionListener takes the user entered username and password and verifies its validity with the corresponding user.txt and pass.txt files. To do so it first takes the username and uses the userValidate() method which returns the index of where the identical username is in the user array, within user.txt. If the method returns “-1” then the username entered is incorrect. Once obtaining the correct index for the username, the entered password is hashed and compared with the corresponding hash in the pass.txt based on the index obtained from the username. There is a lockout feature to the ActionListener that locks out a user who has failed at attempting to login three times. This feature is able to track multiple usernames so it is not just tracking the number of failed logins for a single user.

## Behavior of the Developed Password Management System

The Password Management System has two large overarching behaviors.  These are registering and logging in.  Within these two behaviors we find specialized functions, such as hashing, MAC utilization, validation of username, and storage of usernames and passwords in text files.

Here is a step-by-step walkthrough of a user’s registration process.  There will be a prompt for the user to enter a desired username.  The entered username will be checked against previously registered usernames found in the text file to ensure that it is unique.  Once the username has been accepted as unique, the user will be prompted for a password.  The user’s password must comply with certain length and character requirements in order to be accepted by the program’s policies.  Once the password is accepted, it is hashed using the SHA-256 hashing algorithm.  The accepted username and hashed password combination are then linked by an index and stored in separate text files.  The use of separate files adds a level of security in case one of the files is compromised, yet the index allows the username and password to remain associated.

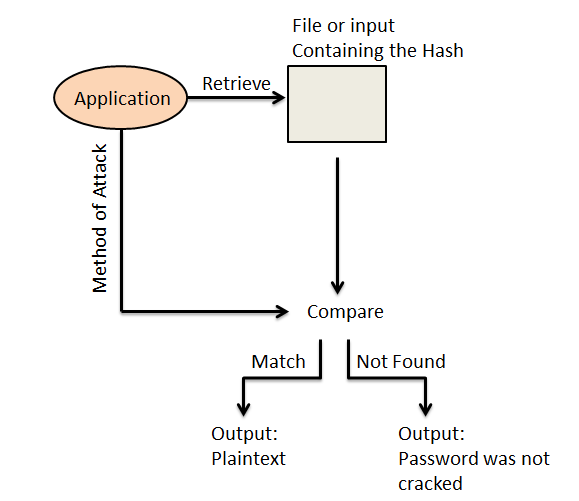
Here is a walkthrough for logging in to the system.  The user is prompted to enter his or her username and password credentials.  The username is checked against the text file holding the list of usernames to see if there is a match or not.  If there is not a match, the user is blocked from accessing the system because he/she is not authorized to log in.  If there is a match, the index of that matched username is temporarily stored.  The entered password is then hashed and compared to the hashed password found at the stored index value within the password text file.  If the two hashed passwords are the same, then the user is authenticated and granted access into the system.  We have also implemented a lockout system.  This means that if the user enters the incorrect credentials three times consecutively, he or she will be locked out of the system for a certain amount of time before more attempts may be made to login.

We also have a MAC built into our password management system.  The MAC creates two hashed codes, one for each of the text files (usernames in one file, hashed passwords in the other), based on the data contained in these two text files.  These two codes are stored in a third file in order to keep the username and password files from being cluttered.  Each time a new user is registered new MAC codes are generated based on the updated files and replace the previous ones.  Anytime a user attempts to log in, MAC codes are generated based on the current state of the two text files and compared to the ones stored.  If the MAC codes match, then the files are intact.  If the MAC codes do not match, this indicates that one or both of the files was compromised and the system may not be secure.  If this happens, no one is permitted to register or log in to the system until the issue is resolved.

# Part II

## Definition of Password Cracking

Password cracking is when a hacker attempts to guess or crack passwords to gain access to a computer system or network. Good crackers will use a variety of tools, scripts, or software to crack a system password. The goal of the cracker is to obtain the password for system administrator uses. Password cracks work by comparing every encrypted dictionary word against the entries in system password file until a match is found. Password cracking is often synonymous with black hat hackers and theft, but this thought is too limited; white hat hackers utilize password crackers in penetration testing and password recovery.

**** Password crackers are often conceptualized as all-in-one tools that extract passwords and spit out passwords to the user. Password cracking attacks, however, are much more complex. First a cracker or hacker steals a copy of the encrypted password file from the target. They then run a password cracking program on the password file against a dictionary and employ knowledge of human psychology to generate password hash until a match is found between the generated password and the passwords in the file.

## Generic Architecture of Password Cracking Tools

The basis for developing an abstract for a state of the art password cracker is to determine the basic components. Once the components are identified, the structure is then simplified and can be replicated by a group of individuals that would want to develop their own password cracker. As a comparative case study, we’ve decided to compare the Cain and Able and SHA Kracker cracking applications. Cain and Able cracks encrypted passwords using Dictionary, Brute-force and Cryptanalysis. SHA Kracker is a user friendly GUI application that implements Rainbow tables and performs dictionary attacks. Based on how these two applications crack passwords, we’ve generated an abstract on how one can structure a program to perform the functions of these password crackers. Generally, a user interacts with either a GUI-Based application or command line based application. The application then retrieves the hash from either a file containing the hash, or from user input. After, the hash is received the application chooses the form of attack whether it is a cryptanalytic, brute force, dictionary, or rainbow table attack. Depending on the complexity of the password, will determine how long the password cracker will take to crack the password. If the password is not too complex and able is relatively easy to crack then it will generate the password in plaintext. On the other hand, if the time required by the password cracking method is too long, or the password is not contained in the dictionary then it will be deemed as a password that cannot be cracked.

## Classification of Password Cracking Tools

Password cracking can be classified by their methods of obtaining the passwords through dictionary attack, brute-force attack, rainbow attack, and social engineering. Dictionary attacks compare the passwords to a database of words such as a dictionary. Dictionary attacks are not always fruitful and are effective for easily guessed passwords. Brute force attacks can crack any password with enough time because it tries every combination of number, letter, characters, and special characters. Brute force attacks can be customized to specify testing criteria but this attack is not always practical because it can take forever to exhaust the possibilities. Keystroke loggers could also be used in this instance to obtain the plaintext form of a password. If one were able to either install key logger software or key logger hardware then they could obtain the passwords of a victim very easily. In rainbow attacks the hashes are pre-generated so this increases the speed greatly. The downside to rainbow attacks is that it takes significant time to make the rainbow tables of pre-calculated hashes. This is because not only does it have to create every possible hash but it also has to create every possible hash for every salt value. Due to this, the process of creating rainbow tables can be time consuming. Packet analyzing can also be conducted to attempt finding any passwords in packets that are being transferred. Generally with this technique the passwords are already encrypted when being sent, but this is also where rainbow tables could be used to find a password match for the captured hash. Social engineering is simply where an intruder acts to be someone else to obtain information. This can be done through email phishing, calling as an IT security officer over the phone, or befriending someone who works at the company to obtain information from daily conversations.

Now let’s look at some of the popular password cracking tools by their classification:

Dictionary Attack

* Cain and Abel – only available for Windows platforms
* John the Ripper – free open source. Can detect weak passwords

Brute Force Attack

* Wfuzz – web application password cracking tool
* THC Hydra – fast network logon password cracking tool
* Medusa – Similar to Hydra. Command line tool so need to learn the commands to use it
* Aircrack NG – WiFi password cracking tool that can crack WEP or WPA passwords

Rainbow Attack

* RainbowCrack – Comes with developed LM, NTLM, MD5, Sha1 rainbow tables
* OphCrack – Most popular Windows password cracking tool
* LOphtCrack – Similar to OphCrack but also uses dictionary and brute force methods to generate passwords

Social Engineering

* Phishing – acting to be someone else through an email
* Shoulder Surfing – Being observant around the office and coming across valuable information such as a post it note with user login and password
* Spidering – Realizing that administration passwords usually relate to the corporation and studying up on the corporation

## Experimentation with Password Crackers

### Rainbow Crack

This portion of the project was to test a password cracker on the PMS system that we had. The program that I used was rainbow crack for my portion of the testing. This part of the report will explain what rainbow crack is and how it affected our password management system.

Rainbow crack is a password cracking system that uses hash tables to crack the hash in the plain text file. The way that rainbow cracker works is that it first scans the text file and finds the hash function. I then select which hash function that I want to crack. Rainbow-crack then takes that function and then tries to find that function within the table of generated hashes, downloaded on line from the main website, and finds the ones that match the hash of the plain text file. The ones that match are then decrypted and presented to the user for him or her to plug into the login interface.

In our program, I had to run it twice, once for the username and once for the password for one set of passwords. When performing these tests, I was going against an 8 bit password with the ASCII table because it was the smallest table to download and it covered the most characters. I also specified which hashing method we used, which was sha1. Once I had the hash table, I then booted up the GUI executable, and then selected which of the plain text files to crack. I tested 10 hashes, one hash at a time, and each one took about 150 seconds. Then testing all of them together drastically increased the time to 3500 seconds, peaking at about 4000 seconds. When testing this particular table against ones that had a longer length, Rainbow-crack was unable to find the password after several hours of searching.

As a result , this program quicker than my previous experiences with brute force programs because it does not have to check every single hash combination, rather just finding it on the table and cross reference it with a stored password in the rainbow table. This makes for significant performance increase. The only down side to this program is that the rainbow tables take a long time to generate or download in VMI's network.

### Burp Suite

Burp Suite is an integrated platform that has the purpose of testing the security of web applications. It is java based which allows for portability to all operating systems. By adding an extensive dictionary you can run Burp Suite against a system in an attempt to gain a password of choice. However, as most systems go, Burp Suite is not the most efficient system since it is not made to specifically conduct dictionary attacks. Rather it has the primary purpose of security testing through other means. This was most evident when it did not allow more than a few hundred passwords to be attempted at a time. This is extremely inefficient because the amount of combinations that can be used in passwords amount in the millions.

### John the Ripper

John the Ripper (hereafter known as John) was originally developed for Unix and now runs on fifteen different platforms. These include Windows, DOS and Mac OS. It offers several functions including auto-detecting the hash and dictionary attack and brute force attack modes. John uses the terminal of whatever platform it is on instead of using a GUI which makes things more difficult for people who are not used to using the terminal. However there are many tutorials on YouTube and other websites. I found John to be more efficient than Burp Suite and a lot faster. John allows the use of downloaded dictionaries to conduct the attacks. I was using a 4 million word dictionary with this tool and while it contained most dictionary words used as passwords, it did not have many of the different combinations involving letter case and special characters.

### SHA-256 Hash Kracker

This tool (hereafter known as the Kracker) was developed specifically to be used against the SHA256 hash. Unlike John the Ripper, the Kracker is only able to be used with the Windows Operating Systems. I conducted most of my testing with this tool. The reason I did so is that SHA256 is the hash function that our program uses and I felt it was the tool that was most pertinent to this project. It is very easy and simple to use because of its user friendly GUI. It allows salts to be taken into consideration, something that John the Ripper does not account for. It also comes with a simple five thousand word dictionary containing a lot of basic, often used passwords. However for the purpose of testing it against more complicated passwords I downloaded a 9 million word dictionary. It is able to run around 2000 passwords a second. This is not very fast with a dictionary as large as the one I downloaded (some more complicated passwords took upwards of 30 to 45 minutes) however it is very thorough and is able crack many of the more complicated passwords due to the size of the dictionary. In fact you can download even larger dictionaries which will be slower but will allow you to have a better chance of cracking the password that you are aiming for.

# References

http://csrc.nist.gov/publications/drafts/800-118/draft-sp800-118.pdf

https://hosam.wordpress.com/learn-solaris10-os/

http://www.cs.uic.edu/~jbell/CourseNotes/OperatingSystems/2\_Structures.html

http://docs.oracle.com/javase/1.5.0/docs/api/java/lang/Object.html#hashCode%28%29

https://docs.oracle.com/cd/E17781\_01/server.112/e18804/storage.htm#ADMQS183

http://csrc.nist.gov/groups/STM/cavp/documents/shs/sha256-384-512.pdf

http://security.fnal.gov/UserGuide/how.htm

http://resources.infosecinstitute.com/10-popular-password-cracking-tools/

http://windowsitpro.com/security/types-password-attacks

http://www.dummies.com/how-to/content/types-of-hightech-password-cracking.html

http://www.infosecisland.com/blogview/18538-Top-Ten-Password-Cracking-Methods.html

https://securelist.com/analysis/36138/keyloggers-how-they-work-and-how-to-detect-them-part-1/

https://www.netiq.com/documentation/idm401/idm\_password\_management/data/bs1f5op.html

http://csrc.nist.gov/publications/drafts/800-118/draft-sp800-118.pdf

http://securityxploded.com/sha256-salted-hash-kracker.php

http://resources.infosecinstitute.com/dictionary-attack-using-burp-suite/

http://www.openwall.com/john/

http://www.oracle.com/technetwork/server-storage/solaris10/solaris-10-security-167783.pdf

http://www.pinkas.net/PAPERS/pwdweb.pdf

"Cain & Abel Hacking Tool - ALIENPRO." *Cain & Abel Hacking Tool - ALIENPRO*. N.p., n.d.

Web. 14 Dec. 2014.

"Password Cracking Using Cain & Abel - InfoSec Institute." *InfoSec Institute*. N.p., 25 Jan.

2013. Web. 14 Dec. 2014.

"Salted Hash Kracker : All-in-one Free Tool to Recover Password from Salted Hash of Type MD5/SHA1/SHA256/SHA512." *Salted Hash Kracker : All-in-one Free Tool to Recover Password from Salted Hash of Type MD5/SHA1/SHA256/SHA512*. N.p., n.d. Web. 14 Dec. 2014.