Implementation of Password Hardening Based on Keystroke Dynamics

Harshit Chawla
Chawla.harshit@gatech.edu
https://github.com/hchawla/CS6238-Password_Hardening

Introduction

The Project is based on the implementation of Password Hardening scheme based on [1]. The implementation can be used to increase the entropy or make it harder for an attacker to gain access to a system implementing this scheme. Password hardening uses the feature values of a particular user to harden the password. The feature values include but are not limited to Keystroke timings, Key-hop timings, etc. In this implementation, the password has been set to 'Password' and there are 15 features of the password. The features can also be calculated using (2n-1) where n is the length of the password. The rest of the sections cover the Class and the method implementation of the project.

Background

In the normal cases of user login, a user logs into a system and one of the three cases takes place:

1. Password entered is compared with a password stored in the system.
2. Password is hashed and then matched with a stored hash in the system.
3. Password is used as a key to decrypt a well-known cipher text stored in the system.

In the password hardening scheme, the system takes as input the password entered by the user and the feature values which together form a feature vector. The feature values are then used to decide whether the user is faster or slower than a certain pre-decided threshold. If the user is slower than the threshold, then using Shamir's secret sharing algorithm, a particular share is chosen from the instruction table which is built as the system gains knowledge about the user's typing patterns or features. The shares are chosen accordingly and a Keyed-hash function is subtracted from the chosen value. The keyed-hash function takes as input the key or the password entered by the user. The subtracted value is then used to compute a polynomial or basically the zeroth coefficient of the polynomial as that is the hardened password for the system. The hardened password thus calculated is used to decrypt a history file which has a certain known plaintext which is then checked to see whether the decryption happened correctly or not. If the decryption is successful, the user is allowed to login and the history file is updated with a new set of feature vectors. The system then chooses a new set of coefficients for the polynomials and computes the new values of shares for future logins.

Project

In the implementation, this project contains several java files, namely

1. EncryptandDecrypt.java
2. hardeningMainClass.java
3. LoginAttempt.java
4. Initialize.java

Apart from these java files, the Project Folder also contains another folder called loginAndThresholdFeatureValues and a package called CSV Reader.

Initialize.java

Class Variables:

- BigInteger q
- BigInteger[] c
- Int setSize= 1024
- BigInteger[] a
- BigInteger[] b
- Int bitlength = 160

Class Methods:

- Public Initialize()
- Public void Polynomials(BigInteger)
- Public void Instruction_Table()
- Public BigInteger Polynomial_Calculation(int)
- Public BigInteger generateHMac(String, String, String)
- Public void History_File()

hardeningMainClass.java

Class Variables:

- BigInteger[] featureValuesReceived
- BigInteger[] thresholdValues
- String pwd
- BigInteger[] alpha
- BigInteger[] beta
- BigInteger[] x_coordinate
- BigInteger[] y_coordinate

LoginAttempt.java

Class Variables:

- BigInteger[] featureValuesReceived
- BigInteger mean
- BigInteger dev
- BigInteger[] thresholdValues

Class Methods:

- public BigInteger[] loginAttempt()
- public BigInteger[] fetchThresholdValues()
- public BigInteger calculateMean(int)
- public BigInteger calculateDev(int)
- public BigDecimal calculateHpwdnew(BigInteger[] ,BigInteger[], BigInteger)
- public void History_File_Update(in,BigInteger[])

encryptAndDecrypt.java

Class Methods:

- public void His_Encrypt(BigInteger)
- public void His_Decrypt(BigInteger)
- public void encryptOrDecrypt(String, int, FileInputStream, FileOutputStream)

The loginAndThresholdValues folder stores two files:

- loginFeatureValues.txt
- thresholdValues.txt

The system generates a random hardened password for the first attempt, and computes random coefficients for the polynomial of degree (no. of features -1). On computation of the coefficients, the system creates a random 160-bit value of q. The system then creates a ‘History File’ and adds a text ‘This is the history File’ which is used as a reference after the decryption to check whether the decryption happened successfully or not. The system encrypts the history File generated with the coefficient of x[0] , which is the hardened password. The system computes the value of α and β stored in the instruction table using the following equation:

$$\alpha = y^{0}_{ai} + G_{pwd,a}(2*i)\text{modq}$$

$$\beta = y^{1}_{ai} + G_{pwd,a}(2*i+1)\text{modq}$$

The user tries to login using the text password and the feature values. The feature values are stored in a text file which is stored on the disk. The system uses the thresholdValues.txt file stored on the disk to determine whether that particular feature is fast or slow for the user. The system picks up the appropriate share choosing α if the user is fast or β if the user is slow. The system then generates the polynomials based on the value of x and y, which are generated by inversing the previous mentioned equation:

$$y^{0}_{ai} = \alpha - G_{pwd,a}(2*i)\text{modq}$$

$$y^{1}_{ai} = \beta - G_{pwd,a}(2*i+1)\text{modq}$$

Upon calculation of the values of x and y, the system calculates the value of polynomial at f(0) which is the hardened password using the following equation:

$$f(0) = \sum_{i=1}^{m} y[i].\lambda[i].\text{modq}$$

$$\lambda[i] = \pi_{1<=j<=m} x[j]/x[j]-x[i]$$

The value of f(0) is used to decrypt the History File, and if the decryption is successful, the user is allowed to enter into the system. The History file is then updated with the new feature vector. A new set of coefficients are generated along with the new values for α and β. The process goes on till the time the system has enough knowledge to determine whether the next user who logs in is the actual user or some adversary. For this, the system calculates the means and standard deviation at the end of the 8th login. On the basis of the mean and the standard deviation, the following equation is used to put the correct value in the proper share and randomize or put garbage value in the other share.

$$\mu_{ai\,+\,k*}\sigma_{ai\,<}\ t_i \ \text{: The correct share is } \alpha$$

$$\mu_{ai\,-\,k*}\sigma_{ai\,<}\ t_i \ \text{: The correct share is } \beta$$

Now, the 9th time when a user logs in, the system is capable enough of determining whether the user is the actual user or an adversary.
The value of threshold depends on system, keyboard and various other factors. I have varied the value for threshold for each feature and they generally lie in the range of 50-60.
To implement this project in a server-client like model, the communication of values needs to be secure. The server can retrieve these values from the client using a secure mode of communication and run this project on the server side. The server can then send a response back to the client about the authentication response. However, this adds up another layer of complexity, which increases the attack surface for an attacker.

Summary

The above mentioned scheme has been implemented in java using the methods and the variables mentioned earlier. The value of threshold was chosen arbitrarily at first and then tweaked so as to make the features distinguishing enough, hence increasing the entropy. The range of threshold as mentioned in the paper is in the range of 60-140. The value of k is constant and is chosen to be 0.4, which is strong enough for a small set of users as pointed out in the paper.

Conclusion

The project only works for a single user at the moment, but can be implemented to store features for various users. Moreover, the system takes 8 user logins to calculate the mean and standard deviation. It can easily be reduced to a smaller number, but for the sake of low false negatives, the more time the system is given to learn about keystroke patterns, the better the system is.

References

[1] Fabian Monrose, Michael K. Reiter, Susanne Wetzel:Password Hardening Based on Keystroke Dynamics
[2] https://github.com/davidju/Password_Hardening_Keystroke_Dynamics