Project: Phase 5

Conor Lamb, Nick Mullen, Riley Marzka

Introduction:

At this point, we have secured our system against seven potential threats. All threats thus far have been outlined for us. Through implementation of our system and our security measures, we have discovered additional vulnerabilities which could pose threats to our system.

The overall security of our system hinges on the GroupServer's ability to authenticate users. Our GroupServer is able to authenticate users on login by way of a password system. The GroupServer maintains a list of <username, f(password)> bindings. Our f(password) implementation is a SHA-3 hash of the password. Upon a login request, the user is prompted to enter their username, followed by their password. The GroupServer then hashes the password which the user entered and compares the result to the value which it has stored for that user. If the values match, then the GroupServer will allow the user into the system and issue the user his token. If the value does not match, then the user is prompted to enter their password again. After five failed password attempts, the server disconnects from the client.

By disconnecting from the client after five failed password attempts, our system takes a good first step at slowing down an attacker who is attempting an online password attack. The attacker, however can simply reboot the system and try five more times. As such, our system only slows down the attacker by the amount of time it takes for the system to boot, which is not very substantial. In order to further hinder an online attack, we will implement computational puzzles. After five failed password attempts, a user will be required to compute the solution to a puzzle before he is allowed to attempt another password. After another set of five failed password attempts, the puzzle will be made progressively more difficult.

By storing only complementary password information, our system ensures that if the GroupServer becomes compromised, then an attacker would not find himself with a plaintext list of the passwords for every user on the system. By compromising the GroupServer, the attacker would only obtain a list of hashed passwords. This does, however, open the door for the attacker to launch an offline dictionary attack on our system. A reasonably motivated and well-funded attacker could write a brute-force algorithm to (pseudo)randomly generate passwords within a predefined search space. The attacker could then hash the passwords using SHA-3 and search through the list of hashed passwords which he obtained from the GroupServer. If the hash matches any of the passwords in the list, then the attacker has cracked the password for the username to which that password hash corresponds. In order to make this type of offline password attack infeasible, we will implement salting of our password list as well as define a minimum length for valid passwords.This will disable the attacker from brute forcing the whole password list concurrently and only enable him to bruteforce a single password and salt at a time.

Our threat model assumes the same trust model as for Phase 4 of the project.

Threat Models:

T8 Online Password Attacker:

A client may enter an incorrect password up to five times before the system automatically exits. However, there is nothing to stop a client from restarting the system and attempting five new passwords and repeating until they have successfully cracked a password. As such, we must make an online password attack infeasible for an attacker.

T9 Offline Password Attacker:

An attacker has gained access to our password list. They are now able to launch a dictionary attack to eventually uncover the passwords for the users in the system. As such, we must make both an offline attack password attack infeasible for an attacker.

Additionally, since there is no minimum length for valid passwords, passwords can be cracked in very little time if they have few characters.

Attacks:

Online Attack:

Assumption: For our purposes, we will assume this attacker to be a user attempting to crack the password of another user (say an administrator). As such, we will assume that the attacker knows the username for the account he is trying to hack into.

Description: The attacker may write a program that repeatedly launches clients and guesses the password for a certain user. The password-guessing algorithm will be a brute force algorithm. After disconnection upon five failed password attempts, the program will launch another client and guess five more times. This will repeat until the algorithm yields the correct password for the given username. This brute force attack may take some time, depending on the quality of the user's password. Given a reasonable amount of time, however, the program will eventually guess correctly.

Offline Attack:

Assumption: For our purposes, we will assume that this attacker has already compromised the GroupServer and has stolen the list of <username, f(password)> bindings

Description: The attacker may write program that runs a brute force password guessing algorithm. As the algorithm generates passwords, they will be hashed using SHA-3. The program will then feed the hashed guesses into a search algorithm over the list of <username, f(password)> bindings. If the list contains the hash of a given guessed password, then that password will be added to a lookup table of correct guesses, indexed by corresponding username. Depending on the goals of the attacker, he may only need the password of one arbitrary user, in which case his attack is done. If the attacker wants the password for as many users as possible, then the program will continue to run until the threshold has been reached.

Defense Mechanisms:

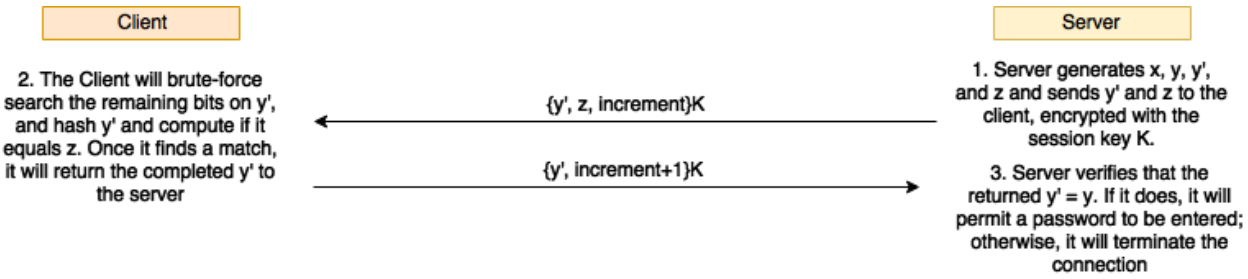
T8 Mechanism:

To protect against the threat of an online password attack, we will introduce hash inversion puzzles to mitigate repeated password guessing attempts. Currently, after 5 failed password attempts the Client closes and disconnects. We will modify this so that the client will still close after 5 failed attempts, but after the first 5 failed attempts of accessing an account, the client will be required to solve a hash inversion puzzle (described further below). The first 5 failed attempts do not require a puzzle, the next 5 attempts will require the client to solve a puzzle with 20 missing bits, which we have found takes approximately 2 seconds, and for all subsequent attempts until a correct password is input, the client will be required to solve a puzzle with 24 missing bits, which takes approximately 10-20 seconds to solve. By increasing the amount of time and work that the client must do between password attempts, we will prevent a DoS attack on our server, as well as make it more difficult for someone to brute force guess a password. This also allows legitimate users to regain access to their account even if someone is attempting to break in.

Hash Inversion Puzzle

Our hash inversion puzzles will work as follows: The server will generate a random 128-bit BigInteger value, x, which will then be hashed using the SHA-3 algorithm. That value (y = H(x)) will then be hashed again (z = H(y)), and will be sent to the user, along with y, minus the k-bits depending on the hardness of the challenge. After receiving the y with missing bits and z, the client must then brute force search the remaining bits, and check if the hash of y with the remaining bits equals z. Once it has found a solution, it will return the completed y to the server for verification.

x = 128-bit BigInteger
y = H(x) = hash of the 128-bit BigInteger
z = H(y) = H(H(x)) = hash of y
y’ = y minus the k-bits (depending on level of hardness) to send to the client



T9 Mechanism:

To protect against the threat of an offline password attack, we will implement salting of the password list stored on the GroupServer. Each password will be concatenated with a random unique 16 byte salt prior to hashing. By salting our password list, we require that an attacker brute-force attack every single password on the system individually. An attacker will no longer be able to brute-force attack the entire password list at once. This mechanism is sufficient to make a brute force offline dictionary attack infeasible for the attacker

Additionally, to further hinder the possible success of any brute-force attack, we will require that a valid password be a minimum of eight characters in length. This protects against the possibility of a user having a password of length one, say, which would take no time at all to guess.

Conclusion:

The mechanisms which we have added to our system for this phase of the project successfully protect against the threats which we have outlined. Our solutions also maintain the simplicity and efficiency of our system. In order to introduce temporary account lockouts into our system, we simply build on a mechanism which our system already implements. Additional requirements consist of simply generating a random 128-bit value and hashing it twice, then a boolean check on whether the value returned by the client matches the original value. All three of these operations are relatively cheap in terms of runtime. The cheapness of the operations on the server side is essential to reducing the resource disparity and preventing a DoS attack on the server. This mechanism also succeeds at rendering a brute force online password attack infeasible. As for the implementation of a salted password list; hashing is cheap. By generating a small 16 byte value, adding it to a user's password, and hashing it, we require very little additional work on the part of our GroupServer. All the while, this measure succeeds at rendering an offline brute force password attack infeasible.