Code and spec/Security Analysis

GitHub URL: https://github.com/SidMad/CS355Project
Commit Number: bc3dc2b
(bc3dc2b1a22c07a8b71f3c54615d014a15a37822)

Libraries: JAAS, MessageDigest, javax.crypto
Language: Java

Runtime Instructions:
Run with the following command (for both Alice and Bob)
Java JavaSecureChannel [filepath] [filepath] [filepath]

Where “[filepath]” is the path to your password file

Security Goals:

- Both Contractors, by running this protocol, are given the ability to authenticate to the other that they have the same password file, and give the other contractor the option to do the same. However, no adversaries, passive or active, gain any of the following information about the Password file:
    - The name of the file.
    - The contents of the file (i.e. the passwords).
    - The number of passwords in the file.
    - The size of the file.

- Alice and Bob will be notified of one the following 2 scenarios:
    - The connected Parties have the same Password file.
    - The connected Parties have different Password files.

Our Protocol:

Preamble:
- Read in the path to the file containing the passwords
- Attempt to connect to the server.
    - if no such server exists then the party is designated as “Alice” and implements the behaviour specified in Alice.

    - If the server exists, then the Client is connected to the server and designated as “Bob” and implements the behaviour specified in Bob.

Alice:

- Start the server.
- When a client (which we will be calling Bob) connects, receive their salt:
- Generate (randomly) and send a salt to Bob.
- Wait for Bob to send his message (containing his Hash)
- On receiving Bob’s message:
    - Generate a hash of {Password file + salt}.
    - send it to Bob.
- Decode Bob’s message to get Bob’s Hash.
- Compare Bob’s Hash to the our generated Hash:
    - If the Hashes match, you are notified that they have the same files.
    - If the Hashes do not match, both are notified that they have different Hashes.
- Program Terminates.

Bob:

- Connect to the server.
- Generate (randomly) and send a salt to Alice.
- Wait and receive Alice’s salt

Create a message consisting of the hash of {Password file + salt} send it to Alice.

- Receiving a message from Alice’s and decode it to get Alice’s Hash.
- Compare Alice’s Hash to our generated Hash:
    - If the Hashes match, both are notified that they have the same files.
    - If the Hashes do not match, both are notified that they have different Hashes.
- Program Terminates.

Note that while a single successful protocol implies the other contractor was connected and has the file, a “different file” message could be the result of someone else impersonating the other party. Thus our recommended protocol is to run our code (coordinated with the other contractor through whatever other messaging platform is available, eg. slack), some constant number of times (Fail) or until the first success(success)

Hashing Algorithm:

We use SHA-512 to generate the hash of the Passwords file. Since the password file is 4GB big, we take a hash of the hash everytime we read in at least 16384 bytes.

Security Argument:
Our protocol is correct, this protocol allows either party verify for the other party whether or not they have the file, that is for either contractor Hash(My Password File + sent salt) = Received hash = Hash(their password file + sent salt), if and only if My password file = Their Password file, Note that no information is leaked with non negligible probability as for an adversary (or alice or bob) to use the exchanged hash (plus their choice of salt) to learn something about the true value of the password file would violate the preimage resistance of our hash function.