Project: Secure File Transfer

Objective

The aim of this project is to familiarize you with cryptographic tools by implementing a secure file transfer protocol, similar to the widely-used AirDrop feature in Apple devices. Throughout this project, you will gain practical experience in various cryptographic and cybersecurity concepts, including symmetric and asymmetric cryptography, digital certificates, public key infrastructure, mutual authentication, non-repudiation, confidentiality, integrity protection, and password security.

Project Team & Milestones

This project is designed to challenge and enhance your critical reasoning skills. To streamline the implementation process, we have three key recommendations:

Programming Language

We strongly recommend using Python 3 as the programming language for this project. Python is chosen for its ease of learning and extensive support for popular cryptographic libraries like `pycryptodome` and `cryptography`. While you are free to choose another programming language (with agreement from all team members), remember that this project's complexity requires a language that enables rapid prototyping and real-time feedback.

Team Size

We advise forming teams of three students. Maintaining a consistent team size facilitates fair grading. Once you've decided on your team, please send a single email with all team member names. Ensure that all team members are copied on the email. Keep in mind that team composition cannot be changed during the semester, so choose your teammates thoughtfully.

Project Milestones

This project may seem daunting, especially if you haven't undertaken such a large project before. To address this, we've divided the project into milestones. Each milestone can be considered a smaller and more manageable sub-project. Completing these milestones in sequence should make the overall implementation much more approachable. For each milestone, your grade will be divided equally between secure design and correct implementation.

Overview

Secure file transfer has various real-world applications, such as uploading source code, transmitting configuration files, or sharing sensitive banking information. In this project, we will focus on a specific scenario: securely transferring a file to another person's computer within our contact list and on the same local network (wired or wireless).

This project will take shape as a command-line utility, referred to as `SecureDrop`, featuring an executable named `secure_drop`. Our primary emphasis will be on the Linux environment. As the semester progresses, the instructor will furnish Docker containers and networks to facilitate code deployment. Below, we provide a sample project run, accompanied by explanatory comments:

```
# The secure_drop command serves as the entry point for the application. If no user exists,
# the registration module will activate, with the email address serving as user identifier.
$ secure_drop
 No users are registered with this client.
```

```
 Do you want to register a new user (y/n)? y

 Enter Full Name: John Doe
 Enter Email Address: john.doe@gmail.com
 Enter Password: *********
 Re-enter Password: *********

 Passwords Match.
 User Registered.
 Exiting SecureDrop.
$

# The user registration is a one-time process. Once a user is registered on a client, the
# login module is subsequently activated. After a successful login, a "secure_drop>" shell
# is initiated.
$ secure_drop
 Enter Email Address: john.doe@gmail.com
 Enter Password: ****
 Email and Password Combination Invalid.

 Enter Email Address: john.doe@gmail.com
 Enter Password: *********
 Welcome to SecureDrop.
 Type "help" For Commands.

 secure_drop> help
   "add"  -> Add a new contact
   "list" -> List all online contacts
   "send" -> Transfer file to contact
   "exit" -> Exit SecureDrop
 secure_drop>

# The "add" command adds a new contact for the user. If a contact already exists, it
# overwrites the existing details. Note that the email address is used as user identifier.
 secure_drop> add
   Enter Full Name: Alice
   Enter Email Address: alice@gmail.com
   Contact Added.
 secure_drop> add
   Enter Full Name: Bob
   Enter Email Address: bob@gmail.com
   Contact Added.
 secure_drop>

# The "list" command displays only those contacts that meet specific criteria:
# 1. The contact email is in this user's contacts.
# 2. The contact has also added this user's email to their contacts.
# 3. The contact is online on the same network.
  secure_drop> list
    The following contacts are online:
    * Bob <bob@gmail.com>
  secure_drop>

# The "send" command transfers a file to the contact. The contact must receive an
```

```
# alert and approve the transfer. You can save the file to a directory of your choice,
# using the same file name as the transmitted file.
#   Contact 'John Doe <john.doe@gmail.com>' is sending a file. Accept (y/n)?
  secure_drop> send bob@gmail.com /home/john/Documents/bob.zip
    Contact has accepted the transfer request.
    File has been successfully transferred.
  secure_drop> exit
$
```

Milestone 1 - User Registration

While implementing this module, consider the following:

1. Database implementation is not required; simple files, YAML, or JSON structures suffice.
2. Initially, implement the module without security controls. After verifying its correctness, introduce security measures to protect against traditional password cracking attacks.
3. Emphasize code reusability. The password protections can be reused in the User Login milestone.
4. Utilize third-party APIs; the Python `crypt` module is valuable for implementing salted hashes for password security.
5. Generate and store information during registration for mutual authentication in future milestones. Assuming a CA (Certificate Authority) is present and trusted on all clients can facilitate the use of digital certificates.

Milestone 2 - User Login

This milestone can be implemented swiftly. Key considerations include:

1. Leverage code reuse, as much of the code will resemble the User Registration milestone.
2. Examine how data from login credentials can bolster security in upcoming milestones. Keep this information in memory while ensuring that password security remains intact. It's essential to forget all data from memory when the program exits to maintain security.

Milestone 3 - Adding Contacts

This module is relatively simple and can be implemented quickly. Key points to consider include:

1. Database implementation is unnecessary; simple files, YAML, or JSON structures suffice. Assume that each user will have a limited number of contacts.
2. Explore ways to use information generated in Milestone 2 to ensure the confidentiality and integrity of contact information. Prevent unauthorized access or tampering by malicious actors.

Milestone 4 - Listing Contacts

This milestone presents a complex challenge, demanding meticulous design and implementation. It marks the initial phase where the application communicates over a network, expanding its attack surface. When working on this module, consider the following key points:

1. Display contact information only if certain conditions are met: the user has added the contact, the contact has reciprocated, and the contact is online on the same network.
2. Initially, implement the module without security controls, and later add security measures. Utilize TCP or UDP for communication, and consider using Python's `socket` module for transport layer protocols.

3. Strive to create reusable code, especially for cryptographic functions. Leveraging Python's `pycryptodome` and `cryptography` modules can greatly assist in the implementation of diverse cryptographic tasks. These implementations can be valuable for later use in the Secure File Transfer milestone.
4. Encrypt identities of communicating entities to safeguard against packet sniffing. Compromise of contact information can lead to spam and targeted attacks.
5. Mitigate impersonation attacks; implement a protocol for mutual authentication between entities.
6. Consider exchanging unique and protected information between entities to enhance security without requiring reauthentication in subsequent steps. Ensure this information remains confidential.

Milestone 5 - Secure File Transfer

The complexity of this milestone depends on previous milestones. Key considerations include:

1. Efficiently transmit large files while maintaining file confidentiality and integrity. The received file must match the transmitted one before notifying the user of a successful transfer.
2. Mitigate replay attacks by using sequence numbers, starting with a random seed on each client.

Grading

Your project will be graded based on the following criteria:

- 20 points - Code Quality (Readability, Modularity, Documentation, Error Handling)
- 20 points - Successful implementation of Milestone 1 (10 points for security, 10 for implementation)
- 10 points - Successful implementation of Milestone 2 (5 points for security, 5 for implementation)
- 10 points - Successful implementation of Milestone 3 (5 points for security, 5 for implementation)
- 20 points - Successful implementation of Milestone 4 (10 points for security, 10 for implementation)
- 20 points - Successful implementation of Milestone 5 (10 points for security, 10 for implementation)