CS255: Cryptography and Computer Security **Winter 2002**

Programming Project #1

Due: Monday, February 18, 2002 11:59pm

Overview

In this project, you will be implementing a Web-based authentication system similar to Microsoft Passport or Stanford's WebLogin. You will design and implement a system to allow users to securely log in to merchant stores using a username and password sent only to a trusted authentication service. You will be using the C programming language and the OpenSSL security library.

You may work in groups of up to two on this project. No late days may be used. **Projects submitted after the due date will not be accepted** without advance permission from the instructor.

For project 1, the required security features are:

- Secure authentication via trusted Web login.
- Scalable merchant key infrastructure.
- Password-based encryption of keys on disk.
- Protection against network-based attacks.

We will examine each of these features in detail below.

Secure Authentication

We will implement a secure authentication scheme, where your password is known only to a central authentication service, but a user can authenticate to any number of untrusted host—in our case, online merchants—without giving out their password to anyone but the trusted server.

Our system will make use of redirects, which allow one Web server to transparently tell a Web browser to connect to a different server, passing information from one server to the other. The basic login procedure will work as follows:

1. The user visits the Web site of a merchant.
2. The merchant redirects the user to the authentication service, passing along the name and location of the merchant.
3. The authentication server presents the user with a secure Web form for the user to fill in their username, password and credit card number.

4. The user fills in the information and submits the form back to the authentication server.
5. The authentication service verifies the user's account and password, and if valid, redirects the user back to the merchant's Web site, passing along a secure representation of the user's identity and credit card number.

The goal is twofold: First, for the merchant's Web server to be able to correctly identify a registered user of the authentication service without ever knowing the user's password. Second, the merchant will receive the user's credit card number, through their unsecured (non-SSL) Web server without possibility of an attacker being able to recover it by eavesdropping on the network.

Authentication Scheme

Once the user is properly authenticated to the authentication server by entering their password, we want to transmit proof of that authentication, along with their credit card number, to the merchant server in a secure way.

To do this, we will have a unique secret key K_i, shared between a particular merchant i and the authentication service—each merchant will have their own key so they will not be able to decrypt messages sent to another merchant. The payload of the redirect from the authentication server back to the merchant after authentication will contain key-based information.

For example, a simple payload might look like this, where MAC is a message authentication code algorithm, and E is a block cipher:

$$\mathit{username} \| \mathrm{MAC}_{K_i}(\mathit{username}) \| \mathrm{E}_{K_i}(\mathit{credit\_card\_number})$$

Encrypting the credit card prevents anyone but the merchant from reading it (however, it does not provide integrity for the *credit_card_number*). Likewise, since the MAC of the username can only be generated by the authentication service, the merchant can treat verification of the MAC as a sign that the user logged on correctly. Since the user's password is not used in this message at all, the merchant will not be able to ever recover it.

Note that this scheme, although fairly secure, does not provide the required protection against all the attacks listed below. Therefore you will want to modify it appropriately.

Password Authentication

Since we have not yet discussed in class how to correctly implement password authentication, we will not require secure password authentication for this project. Instead, do the following:

- Ensure that the user types their password only over an SSL-encrypted Web session (i.e., to the authentication service). This will make sure it is safe from eavesdroppers.
- On the authentication server, you may simply store the usernames and passwords in cleartext in a file, and compare the password entered on the Web page to the one in the file. For this project, you may assume that this file is secure and will not be attacked. We will close this security hole in the next project.

Key Infrastructure

As discussed above, there needs to be a secret key K_i shared between the authentication service and each merchant i. As the number of merchants becomes large, there becomes an unwieldy number of keys for the authentication service to manage. Thus we want the following: The authentication server will store only one master key, K_{MASTER}, from which each K_i is derived.

You will need to design a method by which the merchant key can be derived from the master key given the name of the merchant (which is included in the initial redirect). This method should satisfy the following properties:

- The merchant should not be able to recover K_{MASTER} from their individual key.
- Your method should be resistant to collusion; that is, multiple merchants should not be able to combine information to expose any information about K_{MASTER} or expose any information sent to a merchant not in the coalition.
- The scheme should be resistant to attack: If a malicious attacker presents the authentication server with a merchant name for which they do not have the matching key, they should not be able to recover any information about K_{MASTER} or the user's credit card number. Learning the user's username is acceptable.
- The authentication server should also be resistant to an attack that presents the server with the name of a merchant that does not exist. Whether or not to present an error or continue as if the merchant did exist is up to you, but an attacker should not be able to recover information about K_{MASTER} or a user's credit card in this way.

To coordinate the generation of keys, you will need to write a tool to generate a password encrypted file containing K_{MASTER} and a password encrypted file containing individual merchant keys. These files can then be distributed to the authentication server and the merchants, appropriately.

Password-based Encryption

Since your Web server's keys (K_{MASTER} and merchant keys) will be stored on an insecure file system, you will want to protect them with passwords. You should design a method for storing the keys in files, each encrypted using a password that will be entered into the key generation tool when creating the keys, and to the Web servers when loading them.

Important: Since passwords are relatively insecure, you should not use them to encrypt anything used in your network protocol. You should generate full-strength random keys for K_{MASTER} and the various K_i, and use passwords only for encrypting those keys to be stored on disk. Remember to include an integrity check on encrypted keys stored on disk.

Protection Against Attacks

In addition to satisfying the properties listed above, your authentication system should be made secure against all possible attacks you can think of. You should consider especially the following

- An unregistered user should not be able to authenticate to the system, since they do not have a valid password.
- A malicious user should not be able to learn anything about other users' passwords or credit card numbers or recover any information about the merchant or authentication keys.
- A malicious merchant should not be able to recover a user's password, assuming the user will enter their password only to the authentication server.
- An eavesdropper should not be able to recover a user's password or credit card number by listening to any network traffic.
- An eavesdropper should not be able to determine if two users have the same credit card number.
- An active attacker should not be able to tamper with a user's authentication session so that they can make the merchant think the user entered a different credit card number than they did, or that they are have a different username than they do.
- An attack on the hard drive of any server should not be able to change the key in a way that is not detected when the server is started.
- If an attacker eavesdrops on a user's session with a merchant, he cannot "replay" the authentication session back to the merchant and use the victim's credit card.

You do not, however, need to worry about the following

- As stated above, you do not need to worry about the security of the username/password file. We will revisit this in the next project.
- Since the user-to-merchant connection is via a non-secured Web connection, an eavesdropper will know who the user is, what they are shopping for, and what they have bought. However, this should not reveal the user's password or credit card number, nor should it allow the eavesdropper to buy other items as the user.
- Obviously, an active attacker can alter network traffic or stop it entirely, and prevent a user from successfully authenticating or stopping. You cannot stop this, and do not need to worry about it for now. However, if an attacker makes any changes to the traffic on the network, or to encrypted files on disk, the services should detect this and reject the modified data.

You should think carefully about your scheme, and whether it is secure. Be sure and include your thoughts and explanations in your writeup.

Programming Environment

This project is to be implemented in the C programming language. We will be using the OpenSSL library to provide the cryptographic primitives (e.g., block ciphers, hash functions, secure sockets, etc...) We will provide starter code that implements provides the Web server infrastructure and an insecure version of the authentication system for you to start from. The starter code is available on the Leland Systems computers in the following directory, which you will need to copy to your account:

```
    /usr/class/cs255/proj1
```

The starter code is designed to work on the Leland Systems computers running Solaris (e.g., `elaine`, `epic`, `myth`) and we recommend you work on one of those computers. If you choose to work in a different environment, be sure to test your solution on one of these computers, since that is where you will be graded; see the submission instructions at the end of this document for more information.

The starter code provides two Web servers, an authentication server and a merchant (store) server, as well as a library to provide secure or insecure Web service and skeleton code for a tool to generate the shared keys.

Web Library

The `weblib` directory contains the bulk of the starter code, implementing a simple Web server that can support running either an insecure or secure (using SSL) Web server. We also provide a number of useful utility functions for generating Web content, parsing form input, reading and writing files to disk, and others. You should not have to modify any of the code in this directory to complete the assignment, but you should read over the `weblib.h` header file carefully to familiarize yourself with what is available.

Authentication Server

The authentication server is located in the `auth` directory, and is set up to run a secure (SSL) server, so that user's passwords are not transmitted in the clear. For this project, we have provided certificate and private key files (`cert.pem` and `key.pem`) that will allow your browser to communicate with the server. You do not need to modify these or be concerned about them in this project, although we may examine certificates more thoroughly in the second project.

To compile the authentication server, change to the `auth` directory and run `make`. This will generate the `authserver` program which runs the Web server. It takes as an argument the port number to run on, which must be greater than 1024 and not already used on the computer. We suggest you pick a unique port number for your group to avoid conflicts. For example, to run the authentication server on port 8255, you would use the following command:

```
    epic18:~/proj1/auth> ./authserver 8255
```

You can now access this Web server via any Web browser that supports SSL. For example, to access the Web server in the above example (port 8255 on `epic18`), I would use the URL `<https://epic18.stanford.edu:8255/>`. Note that since the SSL certificates we provide are not signed by a trusted authority, your browser will likely give you security warnings. Feel free to ignore these.

To stop the Web server, press *Control-C*.

Merchant Server

A single merchant Web server is contained in the `store` directory. Although you will certainly want to use more than one merchant server, it is up to you whether you want to run them all out of the same directory with the same binary, or duplicate the `store` directory multiple times.

To compile the merchant server, just run `make` from the `store` directory. To run it, you will need to provide a port number for it to run on and the unique name for the merchant service as well as the hostname and port number of the authentication service to use. For example, to run the merchant server `macys` on port 6255 with the authentication service running on port 8255 of `epic18`, you might use the following command:

```
    myth4:~/proj1/store/> ./storeserver 6255 macys epic18 8255
```

In this example, you could then access the Web server by using the URL
`<http://myth4.stanford.edu:6255/>`.

To stop the Web server, press *Control-C*.

Key Generation Tool

The `keytool` directory provides skeleton code for you to implement a command-line key generation utility to generate password-protected keys for the authentication service and for each merchant.

Help

This handout, along with the starter code, should provide everything you need to start working on the project. We will also provide a handout covering the basics of symmetric encryption using OpenSSL.

- We strongly encourage you do use the class newsgroup, `su.class.cs255`, as your first line of defense for the programming projects. TAs will be monitoring the newsgroup daily, and your question may have already been answered there.
- We will be moving some of our office hours to Sweet Hall to help you with programming questions. Check the Web page for up-to-the-minute office hour locations.
- If all other avenues have been exhausted or if you have questions of a private nature, you can email the course staff at `cs255ta@cs.stanford.edu`.

A Note on Programming Style

Since CS 255 is not a class with a programming-intensive focus, nor do we require as prerequisites extensive programming background, we do not expect everyone to be expert coders. We do expect, though, the code you submit to be easy to read, well-decomposed and well-commented. Points may be deducted if we have trouble following your code.

However, since this is not a programming course, we will be looking only at the cryptographic aspects of your programming technique. For example, we do not require the use of efficient algorithms, and we will not be looking at speed or efficiency, except where it comes to cryptography. Where convenient, feel free to ignore efficient data structures like trees or heaps in favor of simple lists or arrays. You may also assume any data set to be bounded in size (so you can use fixed arrays), so long as you assume a reasonable upper bound, and make this well documented.

If you have any questions about programming style or technique, feel free to contact the course staff, although we would suggest erring on the side of better code.

Submission

Important: You must have a Leland account to submit this project. If you are an SITN or Stanford Online student and do not currently have a Leland account, please contact SCPD as soon as possible to obtain one.

In addition to your solution to the assignment, you should submit a README containing the names, Leland usernames and Stanford ID numbers of the people in your group as well as a description of the design choices you made in implementing each of the required security features. Since there is a great deal of design work for this project, please don't skimp on the README.

You should be sure to include a complete description of your authentication system, as well as your solutions to all of the design problems presented. Also include an analysis of why your system is secure, and what choices you made or algorithms you decided against. Include any information you think might be relevant to us in grading your solution.

When you are ready to submit, make sure you are in your `proj1` directory and type `/usr/class/cs255/bin/submit`.