Foundations of Network and Computer Security

John Black

Lecture #15
Oct 19th 2004

CSCI 6268/TLEN 5831, Fall 2004

Announcements

- Project #1 Due Today
  - Please hand in to me
  - Distance students may email directly to Mazdak as usual

- Project #2 Assigned today

- No Office Hours This Week!
  - Traveling
  - Can still meet with me today or Friday by appt
  - Guest lecture on Thursday; you are responsible for its content

- No Class Tues, Nov 30th (if that affects your Thanksgiving plans)

- Return and discuss
  - Project #0
  - Midterm
    - This will take up most of the lecture today

But First, A Movie

- Network Security, a mainstream concern?
  - Cisco has run ads along these lines
  - Oracle, to a lesser extent
    - “Unbreakable”
  - I thought this AT&T ad was really well-done
  - Disclaimer: I don’t really watch TV, so maybe I’m just out of it

Project #2: Secure Email System

Our goal is to provide a secure email system to each member of the class (including your grader).

We are going to use both symmetric-key and public-key techniques in this project, thus tying together several of the concepts discussed in lecture. As usual, we'll use OpenSSL as our toolkit, either via the command-line interface (easiest) or via system calls (you'll need the OpenSSL book for this!)

The program you write will have three main functions:

1. A mini-database utility to keep track of certs you have acquired from the web site

   `http://ucsub.colorado.edu/~hashemi/certs.html`
2. A method to send encrypted and signed email
3. A method to verify and decrypt received email

Format of the Message

- We’ll start by describing what a message will look like. Then we’ll back-fill the details about how to generate and digest messages in this format.   Messages will look like this:

```
-----BEGIN CSCI 6268 MESSAGE-----
<session pwd encrypted under target’s public key>
<blank line>
<message encrypted under session pwd above>
<blank line>
<signature of above content>
-----END CSCI 6268 MESSAGE-----
```

Message Format

- First `-----BEGIN CSCI 6268 MESSAGE-----` must appear exactly as shown; this is the indicator that the message begins immediately after this line. (This allows the message to be embedded in a bunch of other text without confusing the recipient's parser.)
- The next line is the session password encrypted under the target's public key. This password is a random string of 32 characters using A-Z, a-z, and 0-9 generated by the sender; the sender then encrypts his message with AES in CBC mode using this password.
- There is a blank line, followed by the AES-CBC encrypted message in base64 format. This is followed by another blank line.
- Next comes the signature of the sender which is generated using the sender's private key. This signature will be the RSA sig of the SHA-1 hash of every line above from the first line after the BEGIN marker to the line just before the blank line ending the message.
- Finally, `-----END CSCI 6268 MESSAGE-----` concludes the encrypted message.

The Cert Database

Your program should maintain a simple catalog of certs which you have collected from the web site. You may store them in whatever format you prefer (a flat file is the simplest, but if you prefer to use MySQL or something fancier, be my guest).

A cert should always be verified using the CA's public key before being inserted into the database.

A cert should always be verified using the CA's public key after being extracted from the database (to ensure someone hasn't tampered with it while you weren't watching).

You need not store the person's email address in your database since this is embedded in the cert, but it might be easier to go ahead and store the email addresses as an index field in the file. Of course, you must not rely on these index names as the validated email addresses; always make sure the email in the cert matches!

Sending Secure Mail

Your program should accept a plain-text message along with a destination email address and output an encrypted and signed message as we described a moment ago. Here is the algorithm:

1. Get the cert of the target from the database, using the email address as the index; if the email is not there, you must extract it from the web page.
2. Verify the signature on this cert for your email target.
3. Generate a 32-character passphrase. Normally we would use a very strong random-number generator for this, but feel free to use random() or the rand function of OpenSSL if you like.
4. Encrypt the message with AES in CBC mode with the session key and a random IV (OpenSSL does this for you). Use base64 encoding, and save the output.
5. Encrypt the session password with the target's public key.
6. Sign the stuff generated so far as described previously, using SHA-1 and your private key (you will need to type in your passphrase to do this).
7. Format and send.

Receiving Secure Mail

This is how you will process incoming secure email:

1. Obtain sender’s email address from mail header
2. Find sender’s cert in your database, or obtain from the class website. Verify sender’s cert is signed by CA; output sender name from the cert (not from the email header!)
3. Verify signature on received message using SHA-1 and public key of sender. If invalid, reject the message. Else, continue.
4. Decrypt session key with your private key (you will need to type in your passphrase for this).
5. Use session key to decrypt message; print out resulting message.

Hints for Success

- You already know many of the OpenSSL commands you will need for this project; using the command-line interface is probably the easiest way to get this task done.
- You can call the command-line interface from C or C++, or you can write your whole system in Perl, Python, or sh.
- A text-based menu system is fine, but if you want to build a GUI, feel free. As long as Mazdak can get it to run! ☺
- You can test your program by sending messages to yourself. Additionally, Mazdak will provide a test message to each of you that you can use for testing.
- The most useful advice I can give is this: don’t wait until the last minute to start this project! It’s more work than you think, and we have other things yet to come in the class.

Important Information

- Due Date: 11/18 in class
- What to hand in:
  - Complete source for your program in *printed* form (not on a disk or CD)
  - An example run of each of the main functions (list database, send msg, receive msg)
  - Runs on the test messages Mazdak sends to each of you, showing the outputs