650.445/600.454: Practical Cryptographic Systems November 16, 2016

Assignment 3

Instructor: Matthew Green *Due: 11:59pm, December 5*

Name: _____

The assignment should be completed individually. You are permitted to use the Internet and any printed references.

Please submit the completed assignment via Blackboard.

Problem 1: Attacking a Messaging Client (100 points).

You have been asked to analyze an implementation of the JMessage encrypted instant messaging service. The detailed operation of JMessage is as in Assignment 2. You have been provided with the following components:

1. A complete specification for the JMessage system (from Assignment 2)
2. A working JMessage server, provided as a Docker image
3. Two pre-configured JMessage clients "alice" and "bob", both included on the same Docker image as the server, and programmed to interact[1]
4. Source code for a reference client

You may find the server and reference client at the following locations. As in Assignment 2, you will need to use Git and install a Docker client on your machine,[2] along with some standard tools. Both resources include instructions for building and running the components:

1. Server: `https://github.com/matthewdgreen/jmessage-server` (clone the Git repository and follow the instructions in README.md)

2. Binary reference client and specification: `https://github.com/matthewdgreen/jmessage-client` (this contains a Jar file you can run on the command line using the JRE)

3. Source code for the reference client: `https://github.com/matthewdgreen/jmessage`

Remark: If you had previous installations of the JMessage docker image on your system, you will want to run the **clean.sh** script, and/or reset your Docker install prior to installing the new server image.

[1] There is no guarantee that these reference clients exactly implement the specification. Feel free to look for bugs!

[2] https://www.docker.com

Attacking JMessage The overall goal of your analysis is determine whether JMessage is vulnerable to message modification and decryption. To assist you in this we have provided you with a Docker image containing the server and two built-in clients "alice" and "bob", which are programmed to send messages to each other via the server. There are three separate parts to this assignment:

Part 1: Intercept an *encrypted* message sent between Alice and Bob. As a warmup, your first task in the assignment is to intercept a ciphertext traveling between Alice and Bob. Hint: this should be relatively simple, given that the JMessage server does not implement any mechanism to enforce password access control. However you may need to do some work in order to "catch" the message before it is received by the other client. **Note that for Part 1 you do not need to decrypt this message!** (Note as well that since JMessage implements "read receipts" you may wish to distinguish between ciphertexts that contain receipts vs. those that contain meaningful messages.)

Deliverable: Your deliverable in this portion of the assignment is a program that interacts with the server until it is able to retrieve and output an encrypted message sent between Alice and Bob.[3]

Part 2: Replay and modify a message sent between Alice and Bob. Recall that JMessage implements signatures in order to prevent modifications to messages traveling over the wire. In this part of the assignment your first goal is to take an encrypted message intercepted as in Part 1, and *modify* (maul) the ciphertext into a new valid ciphertext, replaying it such that the intended recipient decrypts a slightly different – but related – message. **Hint:** the modified message does not need to arrive from the same party that sent the original.

Remark: This attack is relatively easy if you are able to register your own public keys for either Alice or Bob. *For this assignment you are not allowed to do this!* Similarly, you can't just figure out what the message said, and send a slightly different message. You must actually modify (in transit) the message that is transmitted by the sender.

Deliverable: Your deliverable in this portion of the assignment is a program that takes as input an intercepted ciphertext sent between Alice and Bob, and then transmits a new ciphertext to the original recipient such that, on decryption by the intended recipient, this ciphertext decrypts successfully to a mauled version of the original plaintext.[4]

Part 3: Retrospectively decrypt a message. Given the ability to maul a message sent between Alice and Bob, implement an attack that *retrospectively* decrypts a message that you intercepted using Part 1 of the assignment. By "retrospective" decryption, we mean that the attack should *not* work by changing the key registered to Alice or Bob prior to either party sending their message. Your attack should take a message as input, and should then issue queries to the server, and monitor the results. **Hint:** look closely at the behavior of Alice and Bob on receiving new messages. You may find that they operate slightly differently from the JMessage specification. This could simplify your attack.

[3]Note: it is okay to deliver a single program that implements all parts of this assignment.

[4]Note: to test that your attack is successful, you may want to run your own clients in place of Alice and Bob.

Deliverable: Your deliverable in this portion of the assignment is a program that takes as input an intercepted ciphertext traveling from one party to another, and automatically executes an attack that decrypts the message. Your program should run the attack interactively with the server, and should output the decrypted message (or a portion thereof) to the screen at the end of the execution.

A note on combining the programs: You do not need to submit three separate programs. It is acceptable (indeed, desirable) to turn in a single program that performs all three parts listed above: (1) intercepting the ciphertext, (2) mauling the ciphertext and sending it to the recipient, and (3) decrypting the ciphertext and printing the result to the screen. Your program should produce a visible output to the user after it has performed each of the above three steps.

Submission and grading: As a deliverable, you must provide the source code for your client. Your code must compile on one of the standard graduate lab machines (MSSI, UGrad, or Masters lab). If you are using a language that is not present on these machines, and you feel this is absolutely necessary, you may submit a VM image that contains all necessary tools. You may include a single script entitled **build** (*e.g.,* **build.sh**) that performs all actions necessary to compile your software. You must also include a simple README that explains how to run and use your code. *If your code does not meet these requirements you will get a* 0 *on this assignment.*