CS6238 Secure Computer Systems
Fall 2014
Project 2: Secure Distributed Data Repository (SDDR)
Deadline: 12/3/2014 (11:55pm)

Goals & Assumptions

The goal of this project is to explore how distributed services can be secured. You will design and develop a simple *Secure Distributed Document Repository* (SDDR) service that stores copies of documents created by its multiple users who access the documents at their local machines. Thus, the system consists of SDDR clients and a server that stores the documents. The server also facilitates sharing of documents between clients as allowed by the owners of the documents. A document's owner is the client that creates the document.

To secure communication between clients and the server, and authenticate sources of requests, we will assume that certificates are available for this purpose. Thus, to implement such a system, you will need to set up a Certificate Authority (CA) that generates certificates for clients and the server. All nodes trust the CA and have its public key. You can make use of a library such as **OpenSSL** (see resources section) for setting up the CA.

SDDR Implementation

After an SDDR server starts, a client can make the following calls:

1. **start-ssession(*hostname*)**: A new secure session is started with the server running at host hostname. Mutual authentication is performed, and a secure communication channel is established between the client executing this call and the server.
2. **get(Document UID)**: After a session is established, a client can request a document over the secure channel from the server using this call. A request coming over a session is honored only if it is for a document owned by the client that set up the session, or this client has a valid delegation (see delegate call). If successful, a local copy of the document data is made available to the client. The server must maintain information about documents (e.g., meta-data) that allows it to locate the requested document, decrypt it and send the data to the requestor.
3. **put(Document UID, SecurityFlag)**: A document is sent to the server over the secure channel that was established when the session was initiated. If the document already exists on the server, you may overwrite it along with its meta-data. If a new document is put, this client becomes the owner of the document. If the client is able to update because of a delegation, the owner does not change. You need to use some scheme to ensure that documents created by different clients have unique UIDs. The **SecurityFlag** specifies how document data should be stored on the server.
4. **delegate(Document UID, client C, time T, PropagationFlag)**: A delegation credential (e.g., signed token) is generated that allows an owner client to delegate rights (put, get or both) for a document to another client C for a time duration of T. If C is ALL, the delegation is made to any client in the system for this document. If you use a protocol that needs to securely exchange messages between clients for implementing delegation, you should use secure channels. **PropagationFlag** is a Boolean that specifies if C can propagate the rights delegated to it. A true value permits delegation and false disallows it.
5. **end-ssession()**: Terminates the current session. If any documents that were received from the server are updated, their new copies must be sent to the server before session termination completes.

The SDDR server stores data in encrypted form when **SecurityFlag** is **CONFIDENTIAL**. It generates a random AES document encryption key. To decrypt the data at a later time, this key is also encrypted using the server's public key and stored with document meta-data. When a request comes for such a document, the server locates its key and data, decrypts the data and sends it back over the secure channel. If **SecurityFlag** is **INTEGIRY**, the server stores a copy signed with the document key, and it also checks the signature before sending a copy to a client. Neither encryption nor signing is necessary when the **SecurityFlag** is **NONE**.

This project should be implemented on Linux/Unix using C/C++/Java for programming and you can work in groups of two. Submissions should be made via T-square by the team-leader/point of contact. Since this is a security class, you should use secure coding practices. You are also expected to use static code analysis tools like flawfinder, and splint (Findbugs, PMD for Java) and minimize the use of unsafe function calls (Justify any such calls you make by providing inline comments). The report should list tools you used to check your code is vulnerability free.

For bonus points (up to 20 points), you should measure the overhead of security. You can do this by performing simple performance evaluation of your system. For example, times for session setup and transfer of various size documents should be included in your report for various values of **SecurityFlag**. Explain the details of your performance measurement experiments in your report, including the network environment where the experiments are conducted.

Project Deliverables

The project deliverables include SDDR code and a report. In the report, state the protocols that you utilize to implement the interactions between a client and the server. In particular, define the message formats for requests-responses and the steps taken to validate requests and delegation tokens. Required deliverables include:

- The source code along with a Makefile/Ant script.
- A README file explaining how to run the program, along with instructions for compiling and testing your application.
- A PDF report for your application that includes protocol details and performance evaluation of SDDR (if you choose to complete it). Also, if any features are not implemented or tested, highlight them in the report. The report should also briefly describe individual contributions of each member – write about who did what (function/feature implementation, integration, design, evaluation etc.)

All files should be submitted in tarball or zip archive.

Resources / Links:

1. OpenSSL: http://www.openssl.org/
2. Secure programming with the OpenSSL API: http://www.ibm.com/developerworks/linux/library/l-openssl.html
3. Java SE Security page: http://www.oracle.com/technetwork/java/javase/tech/index-jsp-136007.html