CSE 543 Fall 2013 - Assignment 1

September 17, 2013

1 Dates

- **Out**. *3rd Sept 2013*
- **Due**. *24th Sept 2013 (Tuesday), 11:59PM*
- **Checkpoint (optional)**. *17th Sept 2013 (Tuesday), 11:59PM.* In this optional checkpoint, submit the code with comments indicating places where you think changes are necessary to: (1) fetch the real server's certificate and a generate self-signed MITM certificate to return to the client, and (2) log the request and response to and from the client. Comments should include the word `CHECKPOINT` so I can locate them easily. This is a way to check progress and clarify general direction as needed. This submission will **not** be evaluated.

2 Introduction

In this project, you aim is to implement a realistic man-in-the-middle (MITM) attack on SSL. **Note: This project is for educational purposes only, and should *never* be used outside of this class**.

3 Background

Recall that an eavesdropper on an SSL connection has little power because of the encryption being used, but if an attacker is able to trick the user into using the attacker's public key rather than the intended receipient's, this security is lost. In this exercise, we will use three machines (Figure **??**)– C (HTTP/SSL client), A (MITM attacker), and S (HTTP/SSL server) – such that A is on the path between C and S (e.g., a router). A will intercept and manipulate the network packets going to and from C and S. During the initialization phase, C will attempt to connect to S to perform an SSL handshake. However, A will intercept this communication and pretend to be S to C. A will present its own certificate to C. If C accepts this certificate *once*, it is enough to compromise the subsequent communication's confidentiality.

4 Exercise Goals

4.1 Dynamically Generate Certificates

In the provided code, the attacker A uses a single, fixed SSL certificate. However, having A use a single, fixed SSL certificate is not ideal, because modern web browsers check the common name (CN) field of the certificate against the domain name of the remote server. So, to mount a more transparent MITM attack, A will have to generate new server certificates on the fly for each new request from C, based on the fields in S's certificate. This is your first goal (Steps 3 and 4 in the initialization phase of Figure **??**).

4.2 Logging

The attacker's aim in compromising an SSL connection is to obtain sensitive information such as passwords and credit card numbers. Your second goal is to log all decrypted communication (requests and responses) passing through A (Steps 2 and 5 in the communication phase of Figure **??**).

5 Knowledge Gain

In completing this exercise, you will learn:

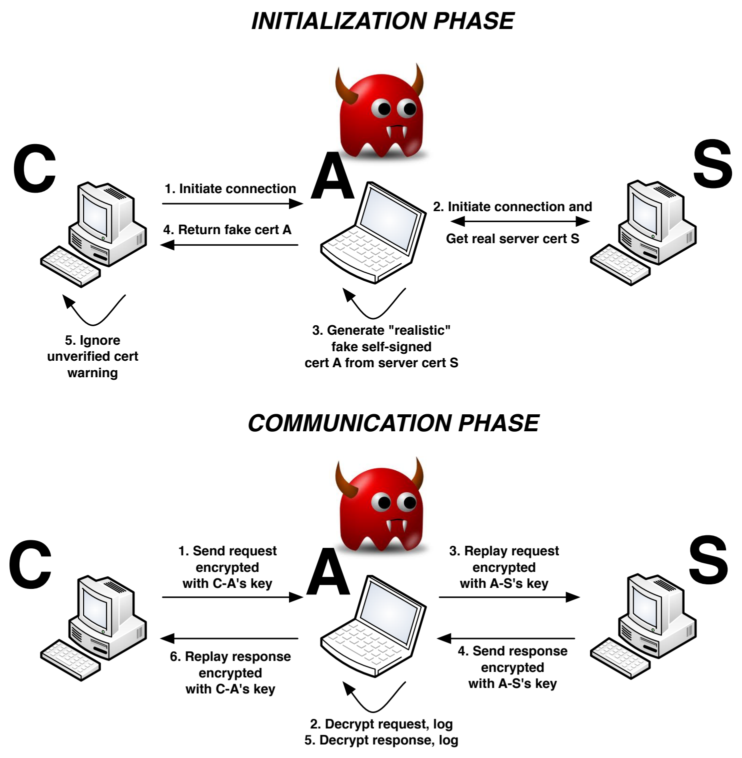
INITIALIZATION PHASE
C
A
S
1. Initiate connection
4. Return fake cert A
2. Initiate connection and
Get real server cert S
3. Generate "realistic" fake self-signed cert A from server cert S
5. Ignore unverified cert warning
COMMUNICATION PHASE
C
A
S
1. Send request encrypted with C-A's key
6. Replay response encrypted with C-A's key
3. Replay request encrypted with A-S's key
4. Send response encrypted with A-S's key
2. Decrypt request, log
5. Decrypt response, log

Figure 1: Showing an SSL MITM attack

- How a real-world public-key infrastructure (PKI) functions, including generation and management of certificates, and how a realistic SSL MITM attack is carried out,
- Content of X509 certificates used on the Internet, and
- How to use `libopenssl` and `libcrypto` `C` APIs for SSL.

6 Requirements

We will provide you with code for the basic SSL MITM attacker, and you will need to do the following.

- Build X509 SSL certificates,
- Modify the SSL proxy to dynamically generate new X509 SSL server certificates, based on the domain name of the requested remote web server.
- Log decrypted communication between client and server.

We will examine each of these features in detail below.

7 Details

7.1 Offline Key Generation

The SSL proxy has a public/private key pair which is generated offline using `keytool`. The keytool is used to generate a keystore for each entity in the system. Before the system is bootstrapped, you will have to generate a public/private key pair for the attacker A. The public key of the proxy is self-signed.

7.2 Generating new server certificates

After connecting to a remote webserver S, A will have to create a new server certificate which has the same common name (CN) and serial number fields as the remote webserver's certificate. This new certificate will then be presented to the client C for use in an SSL session. You will use `libssl` and `libcrypto` to create and sign these new server certificates.

7.3 Logging

You will have to log the intercepted communication at A (both requests from C and responses from S). You will be using `libssl`'s Basic Input/Output Abstraction (BIO) APIs.

8 Implementation

Each of you will be provided with login to a pair of machines C and A. The routing tables are set up such that all traffic from C has to pass through A. You will be modifying code in A and testing its functionality using a browser in C. We have provided you with starter code (all code is in `C`). The location and structure of the code in A will be updated once we assign C and A machines to each of you.

8.1 Running the code

You should spend some time getting familiar with the provided code and reading the comments in the starter code. This will help you understand where you need to place your code, and also how a typical SSL client and server works.

To run the code, find the directory `assignment1` in your assigned machine. Compile the code using the command `make`.

Then, generate your self-signed certificate using the `openssl` command in the PEM format. Finding out how to do this is part of the exercise, but the following general outline is provided. Helpful places to refer are the Internet and the manpage for `openssl`. First, we need to generate an RSA private key for our server – `openssl genrsa`. At the end of this step, you should have the `server.key` file. Second, we need to associate our common name (and other details) with this RSA key, generating a *certificate signing request* – `openssl req`. At the end of this step, you should have the `server.csr` file. Third, usually, this request will be given to a trusted third party (e.g., Verisign) for signing. However, we are going to self-sign it using our own key (produced in step 1) – `openssl x509`. Now, you should have the certificate file – `server.pem`.

Suppose the generated private key is in `server.key` and the certificate is in `server.pem`. Then, run

```
sudo ./mitm -k server.key -c server.pem -p 443
```

You need `root` privileges for binding on any well-known (¡1024) port.

In your browser (`firefox`) in the client machine, go to `https://www.cse.psu.edu`. Equivalently, you can also use the text-only `wget` or `curl`. You should get an invalid certificate warning. Accept it, and you should be able to access the site as normal. Note that the warning will note two problems with the certificate: (1) that it is self-signed, and (2) that the common name (CN) does not match the website you are attempting to connect to. By dynamically generating a certificate with the same common name CN, you will get rid of the second warning. For additional realism, you will also copy the serial number of the server's certificate. Note that access to only the CSE website has been allowed through the network firewall for this exercise. You can access the client's browser using SSH X-forwarding (`ssh -X hostname`), provided your local machine has an X server (linux or mac hosts have this installed by default).

8.2 Library Documentation

You will primarily be using X.509 part of the `libopenssl` and the associated `libcrypto` for this exercise. Unfortunately, `libssl`, although the de-facto library for SSL, is not well documented, so source code is the best, and often, only, documentation.

Refer the links provided below for further details. If you require assistance finding some API, contact me (`hvijay at cse psu`).

- X509 certificates: `http://en.wikipedia.org/wiki/X.509`
- libopenssl source code (from `http://www.openssl.org/source/`)
  - generating a certificate dynamically: `demos/x509/mkcert.c`
  - simple SSL client and server (including fetching remote server's cert): `demos/ssl/`
- IBM articles about openssl (see all parts) `http://www.ibm.com/developerworks/library/l-openssl/index.html`

9 Deliverables

Submit your well-commented MITM code to the assignment as a tarred and gzipped archive file with the following filename: `yourfirstname_yourlastname_cse543_assn1.tar.gz`. The archive should include a `Makefile` and all files (e.g., public keys) required to run the code.

In addition to the code, include in the archive a file named `ANSWERS` with an answers for the question below.

Benign MITM. There are actually legitimate uses for an SSL MITM. Give one example of software that uses SSL MITM, and describe the use case.

Send your archive file to `hvijay at cse psu` and copy `tjaeger at cse psu`. Good luck!

10 Evaluation

- Key and certificate generation using `openssl`: **4 points**.
- Dynamic X509 certificate generation with same common name and serial number: **10 points** (partial credit will be given).
- Logging traffic: **2 points**.
- Answer to question. **4 points**.

11 Credits

The basic idea for the exercise comes from Dan Boneh's CS255 class at Stanford University. Whereas they implemented the attack using an SSL proxy, we use a real MITM. In addition, their code is in Java; we use C.

Hayawardh Vijayakumar