Angels (OpenSSL) and D(a)emons

Athula Balachandran
Wolfgang Richter

PJ1 Final Submission

- SSL server-side implementation
- CGI
- Daemonize

SSL – Stuff you already know!

- Standard behind secure communication on the Internet.
- Data encrypted before it leaves your computer and decrypted only at the computer.
- Hope is it is impossible to crack and eavesdrop!
- Can be used with HTTP, POP3, Telnet etc.

OpenSSL

- Can do a lot more than SSL
  - Message digests
  - Encryption and decryption of files
  - Digital certificates
  - Digital signatures
  - Random number generation

Setup domain name

- Create a DNS hostname for yourself with a free account at DynDNS (or already have a domain name...)
- Don't buy anything, they offer free subdomains and scripts/programs to auto-update the DNS mapping for you.

Set up CA and get certificate

- Add the 15-441 Carnegie Mellon University Root CA to your browser (import certificate, usually somewhere in preferences)
- Obtain your own private key and public certificate from the 15-441 CMU CA.

Implementation

- Use the OpenSSL library, here is a link to their documentation.
- Create a second server socket in addition to the first one, use the passed in SSL port from the commandline arguments.
- Add this socket to the select() loop just like your normal HTTP server socket.
- Whenever you accept connections, wrap them with the SSL wrapping functions.
- Use the special read() and write() SSL functions to read and write to these special connected clients
- In the select() loop, you need to know if a socket you are dealing with is SSL wrapped or not
- Use appropriate IO depending on the 'type' of socket---although use select() for all fd's
- Use your private key and certificate file that you obtained earlier.

Open SSL headers

```
/* OpenSSL headers */
#include <openssl/bio.h>
#include <openssl/ssl.h>
#include <openssl/err.h>
```

Global Initilization

- `SSL_library_init()`
  - registers the available SSL/TLS ciphers and digests.
- `SSL_load_error_strings()`
  - Provide readable error messages.

SSL_METHOD

- To describe protocol versions
- SSLv1, SSLv2 and TLSv1

```
SSL_METHOD* meth = TLSv1_method();
```

SSL_CTX

- Context object
- Store context information (keying material)
- Reused for all connections

```
SSL_CTX* ctx = SSL_CTX_new(meth);
```

SSL_CTX_use_certificate_file()

- Loads the first certificate stored in file into ctx.
- The formatting type of the certificate must be specified from the known types
  - SSL_FILETYPE_PEM
  - SSL_FILETYPE_ASN1.
  - Our CA generates files of PEM format

```
int SSL_CTX_use_certificate_file(SSL_CTX *ctx,
const char *file, int type);
```

SSL_CTX_use_PrivateKey_file()

- Adds the first private key found in file to ctx.
- The formatting type of the certificate must be specified from the known types:
  - SSL_FILETYPE_PEM
  - SSL_FILETYPE_ASN1.
  - Our CA generates files of PEM format

```
int SSL_CTX_use_PrivateKey_file(SSL_CTX *ctx, const
char *file, int type);
```

Initialization Steps

- Global System Initialize
  - `SSL_library_init()`
  - `SSL_load_error_strings()`
- Initialize SSL_METHOD and SSL_CTX
  - `meth=SSLv23_method();`
  - `ctx=SSL_CTX_new(meth);`
- Loading keys
  - `SSL_CTX_use_certificate_file(...)`
  - `SSL_CTX_use_PrivateKey_file(...)`

SSL_new()

- Creates a new SSL structure
- Inherits the settings of the underlying context.

```
SSL* ssl = SSL_new(ctx);
```

SSL_set_fd()

- Connect the SSL object with a file descriptor

```
int SSL_set_fd(SSL *ssl, int fd);
```

SSL_accept

- SSL_accept - wait for a TLS/SSL client to initiate a TLS/SSL handshake

```
int SSL_accept(SSL *ssl)
```

SSL_read and SSL_write

- SSL_read to read bytes from a TLS/SSL connection

```
int SSL_read(SSL *ssl, void *buf, int num);
```

- SSL_write to write bytes to a TLS/SSL connection

```
int SSL_write(SSL *ssl, const void *buf, int num);
```

- NOTE:
  - The data are received in records (with a maximum record size of 16kB for SSLv3/TLSv1).
  - Only when a record has been completely received, it can be processed (decryption and check of integrity)

SSL_shutdown

- Shuts down an active TLS/SSL connection.
- Sends the ``close notify'' shutdown alert to the peer.

```
int SSL_shutdown(SSL *ssl);
```

SSL Wrapping, send and recv data

- Create new SSL structure using SSL_new()
- Connect it to the socket using SSL_set_fd()
- Perform handshake using SSL_accept()
- Read and write using SSL_read() and SSL_write()
- Perform shutdown at the end, also need to clear state and close underlying I/O socket etc.
- As always, check for return value and handle errors appropriately!

BIO - Optional

- I/O abstraction provided by OpenSSL
- Hides the underlying I/O and can set up connection with any I/O (socket, buffer, ssl etc)
- BIOs can be stacked on top of each other using push and pop!
- NOTE: You don't have to necessarily use BIO for this project! The next few slides describe creating BIO and working with it.

BIO_new()

- Returns a new BIO using method type.
- Check `BIO_s_socket()`, `BIO_f_buffer()`, `BIO_f_ssl()`
- Check `BIO_new_socket()`

```
 BIO *  BIO_new(BIO_s_socket());
BIO_set_fd(sbio, sock, BIO_NOCLOSE);
```

SSL_set_bio()

- Connects the BIOs rbio and wbio for the read and write operations of the TLS/SSL (encrypted) side of ssl

```
void SSL_set_bio(SSL *ssl, BIO *rbio, BIO *wbio)
```

Example of Stacking BIOs

```
 buf_io = BIO_new(BIO_f_buffer());
 /* create a buffer BIO */
 ssl_bio = BIO_new(BIO_f_ssl());
/* create an ssl BIO */
 BIO_set_ssl(ssl_bio, ssl, BIO_CLOSE);
/* assign the ssl BIO to SSL */
 BIO_push(buf_io, ssl_bio);
```

BIO_read() and BIO_write()

- Attempts to read len bytes from BIO b and places the data in buf.

  ```
  int BIO_read(BIO *b, void *buf, int len);
  ```

- Attempts to write len bytes from buf to BIO b.

  ```
  int BIO_write(BIO *b, const void *buf, int len);
  ```

Foreground Processes
and
Background processes (daemons)

How to daemonize?

Orphaning

- Fork the process to create a copy (child)
- Let parent exit!
- The child will become child of init process
  - Start operating in the background

```
int i, lfp, pid = fork();
if (pid < 0) exit(EXIT_FAILURE); /* fork error */
if (pid > 0) exit(EXIT_SUCCESS); /* parent exits */
/* child (daemon) continues */
```

How to daemonize?

Process Independency

- Process inherits parent's controlling tty
- Server should not receive signals
- Detach from its controlling tty
- Operate independently from other processes

```
setsid() /*obtain a new process group*/
```

How to daemonize?

Inherited Descriptors and Std I/0 Descriptors

- Close all open descriptors inherited
- Standard I/O descriptors (stdin 0, stdout 1, stderr 2)
- Open and connect to /dev/null

```
for (i = getdtablesize(); i >= 0; --i) close(i);
               /* close all descriptors*/
i = open(“/dev/null”,O_RDWR); /*open stdin */
dup(i)
dup(i)
```

How to daemonize?

File Creation Mask

- Servers run as super-user
- Need to protect the files they create
- Filecreation mode is 750 (complement of 027)

```
umask(027);
```

How to daemonize?

Running directory

- Server should run in a known directory

```
chdir(“/servers/”)
```

How to daemonize?

Mutual Exclusion

- We want only one copy of the server (file locking)
- Record pid of the running instance!
- '`cat lisod.lock`' instead of '`ps -ef | grep lisod`'

```
lfp = open(lock_file, O_RDWR|O_CREAT|O_EXCL, 0640);
if (lfp < 0)
     exit(EXIT_FAILURE); /* can not open */
 if (lockf(lfp, F_TLOCK, 0) < 0)
     exit(EXIT_SUCCESS); /* can not lock */
 /* only first instance continues */
 sprintf(str, "%d\n", getpid());
 write(lfp, str, strlen(str)); /*record pid to lockfile */
```

How to daemonize?

Catching signals

- Process may receive signal from a user or a process
- Catch those signals and behave accordingly.
- Signal_Handler function in the sample code

How to daemonize?

- Logging
    - Assignment – you need to log to file!

Questions?