SSL and CGI and Everything else

15-441: Computer Networks

Yours Truly

Based on Slides By "Generations of TAs"
P1  Final Submission

(1) SSL
(2) CGI
(3) Daemonize
SSL

Adding the “S” in HTTPS
Lets talk about Security

CMU Student: Alice

Network Link (Ethernet, Wifi etc)

“RE: World Domination”

CMU Student: Bob

Bad Guy 1

Bad Guy 2

Bad Guy 3
Lets talk about Security

CMU Student: Alice

“RE: World Domination”

CMU Student: Bob

“RE: World Domination”

Network Link (Ethernet, Wifi etc)

“WAFE#”

Bad Guy 1

Bad Guy 2

Bad Guy 3

Decode

Encode
Public-Private Key Encryption

1. Generate two keys – **Private Key** and **Public Key**

2. Messages can be **encrypted** using the **Public Key**

3. Messages can be **decrypted** using the **Private Key**

4. Everyone knows my public key – that’s why it’s “public”

5. Only I know my private key - that’s why it’s “private”
Here’s my public key everyone!

Public-Private Key Encryption

Bob

Secret Plan

ENCRYPT

Public Key

Secret Plan

ASDF#

Liso

Secret Plan

DECRYPT

Private Key

ASDF#
Implementing an SSL Server
What is SSL?

• Standard behind secure communication on the Internet.
• Provides confidentiality & integrity
• Sits between transport & application
Implementing SSL:
Getting the files

1. Get free domain name from www.noip.com

2. Get the public certificate file and private key file from https://project1.myheartisinthenetwork.com

Extra slides at the end will have more detailed info on how to go about this…
Implementing SSL: Coding

1. Look at the provided SSL Example code and learn how to wrap a connection with SSL
2. Modify your server to take in a HTTPS port
3. Bind socket to this port and add it to your select read_fds
4. When you get a new connection on this port, accept connection, wrap it in SSL
5. Do the rest as usual, but read and write using SSL_read and SSL_write functions
CGI
What is CGI?

• A standard method used to generate dynamic content on Web pages and Web applications.

• Provides an interface between the Web server and programs that generate the Web content.

• Usually written in a scripting language.
Serving Dynamic Content

• A Web server that supports CGI can be configured to interpret a URL that it serves as a reference to a CGI script.
• A common convention is to have acgi/directory containing the CGI scripts.

GET /cgi/horoscope.py HTTP/1.1
• The server **forks** a child process and runs the program identified by the **URI** in that process.

• The server captures the content of the child and forwards it without modification to the client.
How does the client pass arguments to the server?

- **GET**: The arguments are appended to the URI can be encoded directly in a URL typed to a browser or a URL in an HTML link.
  - A question mark appended to the URL, followed by param=value pairs.
  - e.g. http://name.com/cgi/find?first=justine&last=sherry

- **POST**: The arguments are passed in the request body.
  - e.g. name=“mark”
How does the server pass arguments to the cgi program?

- Environment Variables
  - set before execution.
  - passed through `execve`.
  - list of required environmental variables is available on the writeup

- Request body
  - request body passed to the cgi program’s stdin using `dup2` and `pipe`
Implementing CGI: Coding

1. Check if URI starts with “/cgi/”
2. Parse the args in the URI
3. Fork a child
4. Set environment variables
5. Execute script
6. Pass in request body from parent through pipe
7. Add child -> parent pipe to select loop
8. Pass on everything you get from the child, back to the client...you are now a proxy (cue 213 flashbacks)
Daemonizing
What is a daemon?

- A background process that is supposed to run “forever”
- Does not exit when you exit the terminal
- Does not receive any input from or write to stdin/out
- Hard to observe or accidentally kill
- We want liso\texttt{d} to be a daemon, that’s what the “d” was for all along
- We provide most of the code for daemon-izing so don’t worry
Look at handout for SSL examples, CGI code, and daemonize.c
Getting a...

Domain Name
Create a Domain Name

- Get a **free** domain name from https://www.noip.com/

- Use your **Andrew ID** as the **hostname**
Get the Update Client

- You don't have root, so...
  - Just build (make), don't install (make install)
  - Run manually when your IP changes
Create No-IP Conf File

./noip2 -C -c noip.conf

[stariq@unix3 ~/noip-2.1.9-1]$ ./noip2 -C -c noip.conf
Auto configuration for Linux client of no-ip.com.
Please enter the login/email string for no-ip.com
Please enter the password for user '<username>'

Only one host [stariq.ddns.net] is registered to this account. It will be used.

Please enter an update interval:[30]
Do you wish to run something at successful update?[N] (y/N)
New configuration file 'noip.conf' created.
Update Your IP Address

./noip2  -c  noip.conf  -i  108.17.82.243

[stariq@unix3 ~/noip-2.1.9-1]$ ./noip2 -c noip.conf -i 108.17.82.243  IP address detected on command line.

Running in single use mode.
Getting ... 

Keys
Get your public certificate and private key

https://project1.myheartisinthenetwork.com
Get your public certificate and private key

15-441 Carnegie Mellon University CA

Don't forget to add the signing CA to your browser.

Here is your private key. Don't let anyone get that...although we aren't really secure here, are we? Oh, and your public certificate.
SSL

Extra
OpenSSL Toolkit

• Command line tools, SSL library, and crypto library

• Can do a lot more than SSL
  • Message digests
  • Encryption and decryption of files
  • Digital certificates (more later)
  • Digital signatures
  • Random number generation
Open SSL headers

/* OpenSSL headers */
#include <openssl/bio.h>
#include <openssl/ssl.h>
#include <openssl/err.h>
SSL Server Basics

/*step 1: Initialize Library */
SSL_load_error_strings();
SSL_library_init();

/* Step 2: Initialize SSLContext to v1 */
ssl_context = SSL_CTX_new(TLSv1_server_method()))

/* Step 3: Add your private key to the context */
SSL_CTX_use_PrivateKey_file(ssl_context, "my.key",
                              SSL_FILETYPE_PEM)

/* Step 4: Add your public key (certificate) to the context */
SSL_CTX_use_certificate_file(ssl_context, "my.crt",
                             SSL_FILETYPE_PEM)

/* Step 5: Make a listening socket and wait for a connection */
/* Step 6: Accept an incoming connection */
client_sock = accept(sock, (struct sockaddr *) &cli_addr,
                    &cli_size))

/* Step 7: Create a new instance of the context for the client */
client_context = SSL_new(ssl_context)

/* Step 8: Wrap the client socket with TLS */
SSL_set_fd(client_context, client_sock)

/* Step 9: Finalize the SSLConnection */
SSL_accept(client_context)

/* Step 10: Add to the select loop like any other socket but
remember that this socket uses SSL*/

/* Step 11: Use SSL_read and SSL_write to receive and send data
SSL_read(client_context, buf, BUF_SIZE)
SSL Server Basics

/* Step 7: Create a new instance of the context for the client */
client_context = SSL_new(ssl_context)

/* Step 8: Wrap the client socket with TLS */
SSL_set_fd(client_context, client_sock)

/* Step 9: Finalize the SSL Connection */
SSL_accept(client_context)

/* Step 10: Add to the select loop like any other socket but remember that this socket uses SSL*/

/* Step 11: Use SSL_read and SSL_write to receive and send data */
SSL_read(client_context, buf, BUF_SIZE)

/* Step 12: Clean Up State */
SSL_shutdown(client_context);
SSL_free(client_context);
close_socket(client_sock);
close_socket(sock);
SSL_CTX_free(ssl_context);
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(...)`
Global Initialization

- 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

- Data structure to store keying material
- Reused for all connections; make ONE for your server

```c
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

```c
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

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

- 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!
SSL_new()

- Creates a new SSL structure
- Create one per connection
- Inherits the settings of the underlying context.

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

- Tell the SSL object which socket it will wrap

    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)

- (Do this after a standard accept().)
**SSL_read and SSL_write**

- **SSL_read** to read bytes from a TLS/SSL connection
  
  ```c
  int SSL_read(SSL *ssl, void *buf, int num);
  ```

- **SSL_write** to write bytes to a TLS/SSL connection
  
  ```c
  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 integrity check)
SSL_shutdown

- Shuts down an active TLS/SSL connection.

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

- (Then do a standard **close**().)
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

```c
void SSL_set_bio(SSL *ssl, BIO *rbio, BIO *wbio)
```
Example of Stacking BIOs

```c
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_CLOS

/* 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);
Daemonizing

Extra
How to make a daemon?

• Start by making an orphan
• Fork the process to create a copy (child)
  Let parent exit!
• The child will become child of init process
  • Start operating in the background

```c
int pid = fork();
if (pid < 0) exit(EXIT_FAILURE);  /* fork error */
if (pid > 0) exit(EXIT_SUCCESS);  /* parent exits */
/* child (daemon) continues */
```
Process Independence

- Process inherits parent's controlling tty; need to detach
- Server should not receive signals from the process that started it
- Operate independently from other processes

setsid() /*obtain a new process group*/
Close File Descriptors

• Close all open descriptors inherited
  
  int i;
  for (i = getdtablesiz(); i >= 0; --i)
    close(i);

• Connect standard I/O descriptors (stdin 0, stdout 1, stderr 2) to /dev/null
  
  i = open("/dev/null",O_RDWR);     /* open stdin */
  dup(i) /* stdout */
  dup(i) /* stderr */
File Creation Mask

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

umask(027);
Running Directory

- Server should run in a known directory

  chdir("/servers/");
Mutual Exclusion

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

```c
lfp = open(lock_file, O_RDWR|O_CREAT, 0640); if (lfp < 0)
    exit(EXIT_FAILURE); /* cannot open */ if (lockf(lfp, F_TLOCK, 0) < 0)
    exit(EXIT_SUCCESS); /* cannot lock */

sprintf(str, "%d\n", getpid());
write(lfp, str, strlen(str)); /*record pid to lockfile */
```
Logging

• You sent stdout and stderr to /dev/null, so you need to log to a file!