Programming with OpenSSL and libcrypto in examples

BurgasLab, Burgas
April, 2014
secured communications

- the need for secured communications
- world war II Enigma cipher machine
- bank transfers
- private data (drunk pictures from that party, etc)
- crypto-what?
- what is SSL/TLS
- OpenSSL and libcrypto
alternatives

- Apple's libsecurity_ssl
- PolarSSL  (used by OpenVPN)
- full list
  - http+ssh:// ?
  - LibreSSL - OpenBSD's OpenSSL fork
concepts in cryptography

- plaintext/ciphertext
- block ciphers vs stream ciphers
- symmetric cryptography
- public key cryptography
- hash function
- digital signature
- message authentication code
- digital certificates
security algorithms

- hash functions - MD5, SHA1
- authentication codes - HMAC
- cryptographic algorithms
- symetric - Blowfish, DES, AES
- public key - DSA/RSA
- key agreement algorithms - Diffie-Hellman
- public key infrastructure
Contents of a typical digital certificate

See also: X.509 § Structure of a certificate

- **Serial Number**: Used to uniquely identify the certificate.
- **Subject**: The person, or entity identified.
- **Signature Algorithm**: The algorithm used to create the signature.
- **Signature**: The actual signature to verify that it came from the issuer.
- **Issuer**: The entity that verified the information and issued the certificate.
- **Valid-From**: The date the certificate is first valid from.
- **Valid-To**: The expiration date.
- **Key-Usage**: Purpose of the public key (e.g. encipherment, signature, certificate signing...).
- **Public Key**: The public key.
- **Thumbprint Algorithm**: The algorithm used to hash the public key certificate.
- **Thumbprint** (also known as fingerprint): The hash itself, used as an abbreviated form of the public key certificate.
what is SSL/TLS

- cryptographic protocols, designed to provide communication security over unsecured network

- provide connection security by
  - privacy - encrypt connection
  - authentication - prove identity through certificates
  - reliability - maintenance of secure connection through message integrity checking
how SSL works

- four protocol layers
- record layer - formats messages, incl. Generated HMAC at the end
- ChangeCipherSpec protocol layer - one message that signals the beginning of secure communication
- alert protocol - sends errors, problems or warnings about the connection
- handshake protocol - establish a handshake that begins secure connection
how SSL works (2)

SSL Handshake Protocol
SSL Change Cipher Spec Protocol
SSL Alert Protocol
SSL Record layer protocol
HTTP
Application layer
TCP
Transport layer
IP
Internet layer
Network
Network layer
SSL handshake
SSL handshake, 2-way authentication

1. Client hello (crypto information, rand. number)
2. Server hello (crypto information, server rand. number)
3. Server certificate, incl. Public key, client certificate request (optional)
4. Client key exchange
   - Send secret key information (encrypted with server public key)
5. Send client certificate
6. Verify client certificate, if required
7. Client finished
8. Server finished
9. Exchange messages (encrypted with shared secret key)
before we start programming

Learn to code C properly !!!
good programming practices

- clear design
- coding style (indentation matters too!)
- compiler warnings
- code versioning systems
- code reviews
- static code analyzers
- unit testing
- fuzz testing
- automation testing
- documentation
good C coding practices

- input validation
- bounds checking
- string manipulation
- initialize data
- sanitize output
- proper cleanup
- error checking
- principle of least privilege and privilege separation
- keep it simple
good C coding practices (2)

- Build a habit of applying those!
- All of them!
- Always!
Apple's gotofail bug

Apple's gotofail bug (2)
OpenSSL's heartbleed
OpenSSL's heartbleed (2)

- http://git.openssl.org/gitweb/?p=openssl.git;a=commitdiff;h=4817504
OpenSSL's heartbleed (3)

- “First, I have yet to see a SSL library where the source code is not a nightmare.” Poul-Henning Kamp, 2011-02-15
- “It is, bar none, the worst library I have ever worked with. I can not believe that the internet is running on such a ridiculous complex and gratuitously stupid piece of code.” Marco Peereboom, 2009
- “"Catastrophic" is the right word. On the scale of 1 to 10, this is an 11.” Bruce Schneier, 2014-04-09
- “OpenSSL is not developed by a responsible team.” Theo de Raadt, 2014-04-08
"I'm writing this on the third day after the "Heartbleed" bug in OpenSSL devasted internet security, and while I have been very critical of the OpenSSL source code since I first saw it, I have nothing but admiration for the OpenSSL crew and their effort.

In particular considering what they're paid for it.

...  

But software is written by people, real people with kids, cars, mortgages, leaky roofs, sick pets, infirm parents and all other kinds of perfectly normal worries of an adult human being.” Poul-Henning Kamp, 2014-04-11
"Every time I think “this change is so simple, it doesn't need any tests,” it breaks in some horrible, unpredictable way. EVERY. TIME." Mislav Marohnić, 21-12-2013
Debian Random generator bug, 2008

- Know what your code is doing

```c
int getRandomNumber() {
    return 4; // chosen by fair dice roll.
    // guaranteed to be random.
}
```
OpenSSL architecture
OpenSSL command-line interface

```
OpenSSL> ?
onopenssl: Error: '?' is an invalid command.

Standard commands
asn1parse  ca  ciphers  cms
  crl  crl2pkcs7  dgst  dh
  dhparam  dsa  dsaparam  ec
ecparam  enc  engine  errstr
  genconf  gensa  genpkey  gensect
  msg  ocsp  passwd  pkcs12
  pkcs7  pkcs8  pkey  pkkeyparam
  pkeyutl  prime  rand  req
  rsa  rsautl  s_client  s_server
  s_time  sess_id  smime  speed
  spkac  srp  ts  verify
  version  x509

Message Digest commands (see the `dgst` command for more details)
md4  md5  mdc2  rmd160
sha  shal

Cipher commands (see the `enc` command for more details)
aes-128-cbc  aes-128-ecb  aes-192-cbc  aes-192-ecb
aes-256-cbc  aes-256-ecb  aes-cbc  aes-ecb
bf-cbc  bf-cfb  bf-cbc  bf-ofb
camellia-128-cbc  camellia-128-ecb  camellia-192-cbc  camellia-192-ecb
camellia-256-cbc  camellia-256-ecb  camellia-cbc  camellia-ecb
cast5-cbc  cast5-cfb  cast5-cbc  cast5-ofb
des  des-cbc  des-cfb  des-cbc
desede  desede-cbc  desede-cfb  desede-cbc
desede3  desede3-cbc  desede3-cfb  desede3-cbc
desofb  des3  des  idea
des-ede-cbc  idea-cfb  idea-ecb  idea-ofb
rc2  rc2-40-cbc  rc2-ecb  rc2-ofb
rc2-cfb  rc2-cfb  rc2-cfb  rc2-cfb
rc4  rc4
rc4-40  rc4-40  rc4-40  rc4
rc5  rc5  rc5  rc5
rc5-cbc  rc5-cbc  rc5-cbc
seed-cfb  seed-cfb  seed-cfb
seed  seed  seed
```

OpenSSL> ```
generating message digest/HMAC

syrinx:demetra:/openssl dgst -md5 openssl-verify-certs.png
MD5(openssl-verify-certs.png)= 6d3d806d8b178d1a753ed6786fe51ffd

syrinx:demetra:/openssl dgst -sha1 openssl-verify-certs.png
SHA1(openssl-verify-certs.png)= dbf8ff0ea8f6b41b9022d31b0eb3ce68709b325f

syrinx:demetra:/openssl dgst -sha1 -hmac 'burgaslab' openssl-verify-certs.png
HMAC-SHA1(openssl-verify-certs.png)= 6eb5396d098a68022d47e18f0a3c153d53847dd2
syrinx:demetra:/
encryption/decryption

```
syrinx:demetra:/echo "This is plaintext!" > plaintext.txt

syrinx:demetra:/openssl enc -e -aes-256-cbc -in plaintext.txt -out plaintext.bin
enter aes-256-cbc encryption password:
Verifying - enter aes-256-cbc encryption password:

syrinx:demetra:/openssl enc -d -aes-256-cbc -in plaintext.bin -out plaintext2.txt
enter aes-256-cbc decryption password:
syrinx:demetra:/cat plaintext2.txt
This is plaintext!

syrinx:demetra:/openssl enc -d -aes-256-cbc -in plaintext.bin -out plaintext2.txt
enter aes-256-cbc decryption password:
bad decrypt
syrinx:demetra: /

syrinx:demetra:/openssl base64 -e -aes-256-cbc -in plaintext.bin -out plaintext.asc
enter aes-256-cbc encryption password:
Verifying - enter aes-256-cbc encryption password:
syrinx:demetra:/cat plaintext.asc
U2Fs-dGVkX1I/Eg+RX++d7VhWEAI8HgyP7WpR341i0nxadwV1Szsvzy4ef2XKydpzU8SWpieTULOELz1z0I8icz1q1h+H6pgK/95KsDPdUkU=
```
EVP_CIPHER_CTX ctx;

memcpy(iv, keyb, ENC_AES_IV_SIZ);
if (decrypt == 0) {
    if (EVP_EncryptInit(&ctx, EVP_aes_128_cfb128(), keyb, iv) != 1) {
        error = EX_DATAERR;
        goto cleanup;
    }
    if (EVP_EncryptUpdate(&ctx, outb, &outl, inb, inl) != 1 ||
        EVP_EncryptFinal(&ctx, outb + outl, &outl) != 1)
        error = EX_DATAERR;
} else {
    if (EVP_DecryptInit(&ctx, EVP_aes_128_cfb128(), keyb, iv) != 1 ||
        EVP_CIPHER_CTX_set_padding(&ctx, 0) != 1) {
        error = EX_DATAERR;
        goto cleanup;
    }
    if (EVP_DecryptUpdate(&ctx, outb, &outl, inb, inl) != 1 ||
        EVP_DecryptFinal(&ctx, outb + outl, &outl) != 1)
        error = EX_DATAERR;
}

EVP_CIPHER_CTX_cleanup(&ctx);
OpenSSL programming - create keys

- create CA cert, server & client certificate request/keys, sign csr

```bash
syrinx@demetra:/mkdir -p ca/private
syrinx@demetra:/chmod 700 ca/private
syrinx@demetra:/openssl req -x509 -days 3650 -newkey rsa:1024 -keyout ca/private/ca.key -out ca/ca.crt
Generating a 1024 bit RSA private key
.................+++++
........................................................................+++++
writing new private key to 'ca/private/ca.key'
Enter PEM pass phrase:

syrinx@demetra:/mkdir -p server/private
syrinx@demetra:/chmod 700 server/private
syrinx@demetra:/openssl genrsa -out server/private/server.key 1024
Generating RSA private key, 1024 bit long modulus
.................+++++
........................................................................+++++
e is 65537 (0x10001)
syrinx@demetra:/openssl req -new -key server/private/server.key -out server/server.csr
You are about to be asked to enter information that will be incorporated into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter ' . ', the field will be left blank.
-----
Country Name (2 letter code) [AU]:
```
OpenSSL - create keys(2)

```
syrinx@demetra:/mkdir -p client/private
syrinx@demetra:/chmod 700 client/private
syrinx@demetra:/openssl genrsa -out client/private/client.key 1024
Generating RSA private key, 1024 bit long modulus

...........................................
...........................................
e is 65537 (0x10001)

syrinx@demetra:/openssl req -new -key client/private/client.key -out client/private/client.csr

You are about to enter information that will be incorporated into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter ".", the field will be left blank.

----- Country Name (2 letter code) [AU]:
```
syrinx@demetra:/openssl x509 -req -days 1460 -in server/server.csr -CA ca/ca.crt -CAkey ca/private/ca.key -CAcreateserial -out server/server.crt
Signature ok
subject=/C=BG/ST=Burgas/L=Burgas/O=sotirova/CN=sotirova/emailAddress=shteryana@yahoo.com
Getting CA Private Key
Enter pass phrase for ca/private/ca.key:
syrinx@demetra:/openssl x509 -req -days 1460 -in client/client.csr -CA ca/ca.crt -CAkey ca/private/ca.key -CAserial ca/ca.srl -out client/client.crt
Signature ok
subject=/C=BG/ST=Burgas/L=Burgas/O=shopova/CN=shopova/emailAddress=syrinx@freebsd.org
Getting CA Private Key
Enter pass phrase for ca/private/ca.key:
OpenSSL - test certificates

subject=/C=BG/ST=Burgas/L=Burgas/O=shopova/CN=shopova/emailAddress=syrinx@freebsd.org
issuer=/C=BG/ST=Burgas/L=Burgas/C=shteryana/CN=shteryana/emailAddress=shteryana@gmail.com


CIPHER is ECDHE-RSA-AES256-GCM-SHA384

Secure Renegotiation is supported
ERROR
shutting down SSL
CONNECTION CLOSED
ACCEPT

\[[A^C

syrinx@demetra:/openssl s_server -CAfile ca/ca.crt -cert server/server.crt -key server/private/server.key -Verify 1
verify depth is 1, must return a certificate
Using default temp DH parameters
Using default temp ECDH parameters
ACCEPT

Start Time: 1398421735
Timeout : 300 (sec)
Verify return code: 0 (ok)

\[A^C

syrinx@demetra:/openssl s_client -CAfile ca/ca.crt -cert client/client.crt -key client/private/client.key
BIO * bio;
int x;

if ((bio = BIO_new_connect("hostname:port")) == NULL ||
    BIO_do_connect(bio) <= 0) {
    /* Handle failed connection */
}

if ((x = BIO_read(bio, buf, len)) <= 0) {
    /* Handle error/closed connection */
}

BIO_reset(bio); /* reuse the connection */
BIO_free_all(bio); /* cleanup */
setting up a secured connection

```c
SSL_CTX * ctx;
SSL * ssl;

if ((ssl = SSL_CTX_new(SSLv23_client_method())) == NULL)
    err(1, "SSL_CTX_new()");

if (SSL_CTX_load_verify_locations(ctx, "/path/to/TrustStore.pem", NULL) != 0) {
    /* Handle failed load here */
    SSL_CTX_free(ctx);
}

if ((bio = BIO_new_ssl_connect(ctx)) == NULL) {
    SSL_CTX_free(ctx);
    err(1, "BIO_new_ssl_connect()");
}

BIO_get_ssl(bio, & ssl);
SSL_set_mode(ssl, SSL_MODE_AUTO_RETRY);

/* Attempt to connect */
BIO_set_conn_hostname(bio, "hostname:port");

/* Verify the connection opened and perform the handshake */
if (BIO_do_connect(bio) <= 0 || SSL_get_verify_result(ssl) != X509_V_OK) {
    BIO_free_all(bio);
    SSL_CTX_free(ctx);
    err(1, "BIO_do_connect()/SSL_get_verify_result()");
}

BIO_free_all(bio);
SSL_CTX_free(ctx);
```
printf("Error: %s\n", 
ERR_reason_error_string(ERR_get_error())); 

ERR_print_errors_fp(FILE *); 

ERR_print_errors(BIO *); 

CRYPTO_mem_ctrl(CRYPTO_MEM_CHECK_ON); /* XXX: really needed? */ 

(void)SSL_library_init(); 

SSL_load_error_strings(); 

printf("Error: %s\n", 
ERR_error_string(SSL_get_error((ssl),(err)), NULL);
```c
SSL_load_error_strings();
OpenSSL_add_ssl_algorithms();

if ((ctx = SSL_CTX_new(SSLv23_server_method())) == NULL)
    fatalx("ctx");
if (!SSL_CTX_load_verify_locations(ctx, SSL_CA_CRT, NULL))
    fatalx("verify");
SSL_CTX_set_client_CA_list(ctx, SSL_load_client_CA_file(SSL_CA_CRT));
if (!SSL_CTX_use_certificate_file(ctx, SSL_SERVER_CRT, SSL_FILETYPE_PEM))
    fatalx("cert");
if (!SSL_CTX_use_PrivateKey_file(ctx, SSL_SERVER_KEY, SSL_FILETYPE_PEM))
    fatalx("key");
if (!SSL_CTX_check_private_key(ctx))
    fatalx("cert/key");
SSL_CTX_set_mode(ctx, SSL_MODE_AUTO_RETRY);
SSL_CTX_set_verify(ctx, SSL_VERIFY_PEER | SSL_VERIFY_FAIL_IF_NO_PEER_CERT, NULL);
SSL_CTX_set_verify_depth(ctx, 1);

/* setup socket – socket()/bind()/listen() */

for (; work != 0;) {
    if ((s = accept(sock, 0, 0)) == -1)
        err(EX_OSERR, "accept");
    sbio = BIO_new_socket(s, BIO_NOCLOSE);
    ssl = SSL_new(ctx);
    SSL_set_bio(ssl, sbio, sbio);
    if ((r = SSL_accept(ssl)) == -1)
        warn("SSL_accept");
}
OpenSSL - client example

```c
SSL_load_error_strings();
OpenSSL_add_ssl_algorithms();
if ((ctx = SSL_CTX_new(SSLv23_client_method())) == NULL)
    fatalx("ctx");
if (!SSL_CTX_load_verify_locations(ctx, SSL_CA_CRT, NULL))
    fatalx("verify");
if (!SSL_CTX_use_certificate_file(ctx, SSL_CLIENT_CRT, SSL_FILETYPE_PEM))
    fatalx("cert");
if (!SSL_CTX_use_PrivateKey_file(ctx, SSL_CLIENT_KEY, SSL_FILETYPE_PEM))
    fatalx("key");
if (!SSL_CTX_check_private_key(ctx))
    fatalx("cert/key");
SSL_CTX_set_mode(ctx, SSL_MODE_AUTO_RETRY);
SSL_CTX_set_verify(ctx, SSL_VERIFY_PEER, NULL);
SSL_CTX_set_verify_depth(ctx, 1);
/* setup connection */
if ((hp = gethostbyname("localhost")) == NULL)
    err(EX_OSERR, "gethostbyname");
/* init socket - socket()/connect() */
/* go do ssl magic */
ssl = SSL_new(ctx);
sbio = BIO_new_socket(sock, BIO_NOCLOSE);
SSL_set_bio(ssl, sbio, sbio);
if (SSL_connect(ssl) <= 0)
    fatalx("SSL_connect");
if (SSL_get_verify_result(ssl) != X509_V_OK)
    fatalx("cert");
printf("connected to server!\n");
SSL_free(ssl);
BIO_free_all(sbio);
SSL_CTX_free(ctx);
```
compiling and running the code

- http://people.freebsd.org/~syrinx/presentations/openssl/
- download, untar & make
- needs libbsd for Linux/Ubuntu
references

https://www.openssl.org/
http://www.libressl.org/
http://www.ietf.org/rfc/rfc2246.txt
http://www.ietf.org/rfc/rfc3546.txt
http://www.open-std.org/jtc1/sc22/wg14/www/docs/n1255.pdf
http://cacr.uwaterloo.ca/hac/
https://www.owasp.org/index.php/Guide_to_Cryptography
https://www.ssllabs.com/
https://www.howsmyssl.com/
https://we.riseup.net/riseuplabs+paow/openpgp-best-practices#openpgp-key-checks
http://www.secureconsulting.net/2008/03/the_key_management_lifecycle_1.html
A Guide to Understanding Flow Charts

Presented in Flow Chart Form

START

Do you understand flow charts?

- YES
  - Good
  - YES
    - Let's go drink.
    - 6 drinks
      - Hey, I should try installing FreeBSD!
  - GOOD
    - NO
      - Screw it.
      - THAT WASN'T A QUESTION.

- NO
  - Okay. You see the line labeled "yes"?
    - YES
      - ...and you can see the ones labeled "no"?
        - NO
          - BUT YOU JUST FOLLOWED THEM TWICE!
        - YES
          - Wait, what?
    - NO
      - BUT YOU SEE THE ONES LABELED "NO"?
        - YES
          - Wait, what?
        - NO
          - Listen.

- NO
  - I HATE YOU.
thank you!