flow Documentation (experimental)

Manfred Morgner

9. Januar 2011

Inhaltsverzeichnis

Data must *flow*!

Kapitel 1

flow

flow will move your data in protected mode

1.1 What *flow* is

flow is a client server application system which may move your data secure over insecure networks using untrusted service providers. The level of security you get out of *flow* is up to you! Meaning you have nearly unlimited control over your data while moving them. At least, if you're the recipient too, which is not as crazy as you might believe. *flow* will also support you in establishing privacy whiule communicating with trustworthy partners.

YOUR data is data you own, which is data you send and data you receive. Receiving data may imply third entities. Other people for example, or data gathering devices.

As far as you're in control of all the security credential at both end of a transportation chain, you are in control of your data. Other people, friends for example, will undermine your control over your data but only in limited ways. So we need to discuss communication security soon.

1.1.1 What *flow* does

flow secures your communication with dedicated partners as far as possible and does not make nonsense down the road. This way it maximizes the privacy you can get while communicating with remote partners and reenforces you to become your own master of your privacy.

1.1.2 Why *flow* is developed

flow is developed for some different reasons

- A young man and a women I know didn't believe I'm able to program in C++. I needed to put this to test.
- There are no useful communiction applications ensuring your privacy.
- I was in need of some tools dealing with OpenSSL, which is in no way easy.
- I couldn't believe that OpenSSL is as complicated as it is. Now I'm a believer.

1.1.3 What *flow* is good for

flow solves the problem you enter if you need to communicate and wish to communication to be protected.

1.2 Some unusual security basics

1.2.1 Two simple definitions

Assuming that data you send to others (autonomous entities, not controlled by you) are not truly your data, we have to split the analyse in two parts:

YOUR data: Means data over which you have full control and which will not change their ownership.

THEIR data: Means data you share and therefore need to be given some control in other hands.

1.2.2 Trustworthyness

Nearly no one knows about Software Certificates. Good willing people, business man, manager and lot of other people tried to send the message about Software Certificates (for our purpose simple 'Certificates') to the world, but the world was not listening. This is because the matter is much too complicated.

Because of this, I will not try to do the next boring attempt to explain. If you'r not familiar with the matter and - as usual - not interested at all, forget the whole chapter about security. You don't need to care about it. Other ones care for you. *The last one is a joke!*

A key question in all matters of security is: Who is trustworthy?

The answer seems to be simple but it is not. You may take a most restrictive point of view and state, that the only person you are will to trust is you. But given the point of our discussion, an open network and (per definition) not trustworthy service providers building this network, your believe has to be wrong.

You'r only trustworthy against yourself if you understand the basics of data security. Even if you read this document, which means you'r interested in Data Security, probably you should not trust yourself - at least for the beginning.

1.2.3 To trust or not to trust

Should you trust yourself? Why? What is it you have or know or believe, that should make you trust yourself? Watch yourself for a while. Do you use cellphones in public? Did you ever wrote an SMS while using public transport? Did you talk about business matter in public? No? Ok, you may be trustworthy against yourself. Yes? You may think about information flow!

1.2.4 Identity in open networks

Who are you?

Sorry, I'm not interested, but believe me, someone is. A lot of people wish to know who you are. Sales man are the least frightening ones. You may believe that thiefs and other criminals are not interested on your identity because you don't know any, but you're wrong. Internet criminals are interested on everyone whom they are able to uncover. You leave traces and lots of individuals, organisations and governments are investing in tracing people, uncovering them and get as much social and other kind of information as possible.

The use of such information is countless. It would fill some books for its own.

But sometimes you run into greater problems. You need to prove your identity against a communication partner but hide it from everyone else. An some times not only this, some times you only wish you communication partner know that you are the right person but not really who you are.

This may be an extreme case, but implementing secure, reliable private communication needs to put the possible requirements to the limit. If you've lesser requirements, dealing with them will become easier than doing it all and may result in some protection you would not get if the limit is only what you currently imagine.

1.2.5 How to trust

If we assume, trusting each other is recognizing each other as what one promised to be for the other one, it is on no way easy. One of the easier solutions is, using an well know communication channel to exchange the necessary enhanced authentication credentials.

For example, you are emailing with a partner for a while, so you may send him your Certificate the same way, by email. This looks simple and indeed it is. But the receiver cant be certain that this Certificate belongs to you indeed. Maybe, someone else interfered. Thats why you may exchange other prove using other ways of communication.

For example, you may print your Certificate and send it by snail mail or hand it over directly. There are different methods. But do not trust any 'web of trust'! In a so called 'web of trust' everyone believes every other one to be careful. A big mistake if you try to establish private communication.

You may trust a web of trust to be reliable if you need to communicate with partners with your and their identity openly known. But if you wish to be anonymous for the public but also sure to know with whom you're communicating with, you fail to fulfill two basic rules for a 'web of trust'. At first, your Certificate does not confirm you by its public content. And second, letting others know who you're for later prove, you discover your identity against others and so (eventually) discover it to others who wish to know who you are.

The best way to ensure others don't know who you are is to ensure, others don't know who you are.

1.2.6 Why should you care?

You don't know what will happen.

Friends become foes. So, possibly you wish to deny that you're the communication partner after a former friend went against you.

Communication partners may be accused to be involved in some criminal act you don't wish to become associated to.

You don't want to spread you social net through the internet.

You may have some of millions of other reason to care and if you have none, do it anyway. As time comes you will find out why this was a good decision.

1.2.7 Where to go from here

We need to have a communication system that enables you to become anonymous but at the same time able to prove who you are and further to not connect your communication with one partner with the communication to another partner. Which means you need to appear as someone other in each communication thread.

Kapitel 2

The project

2.1 Concept

The concept of flow is, to move data securely through insecure networks using untrusted providers. To do this, *flow* has to fullfill three requirements

1. Kick out the 'man in the middle'
2. Ensure even the message server is unable know who is who
3. Ensure only the receiver is able to read the data

2.1.1 The 'man in the middle'

The man in the middle is the easier part. He does not love Transport Layer Security - TLS (former SSL) - and usually will focus on other means to break down your defences if he discovers such crud methods of securing your doings. Currently and in the near future there is enough unencrypted, unprotected traffic to read. So its unlikely to become overheard if using TLS.

2.1.2 The message server

But how to provide the communication server with no information of who you are and to whom to deliver the package without telling him who you are and to whom to deliver the package?

Well, this is not only easy, but eases the rest of the boring transportation requirements too. This is, because you neither have to tell the server who your are nor who the receiver might be. A simple two step trick ensures this:

1. You will be (anonymously) identified by the Certificate from your TLS connection.

2. You address your message by using the identification code of the Certificate of your partner.

For the server (and anyone interfering at the server) this will look like this:

```
FROM:FB0FB87C6F783B35F5F21B69F79F4DCC4EDF1963
TO..:BA95B648CE28F80842CECDC6A4BA8826B6B9FE32
```

Its easy to believe that such addressing makes things obscure for an uninvited observer.

2.1.3 Data protection

Ensuring that only the receiver will be able to understand your message is easy too. Because asymmetric data encryption is 'state of the art', even if no normal user understands how it works, it ensures exactly this.

The credentials we already need for the previous steps are enough to encrypt the messages for the receiver. These credentials are:

- Our own Certificate, used to establish an encrypted connection to the server.
- The partner Certificate, used to decrypt received messages and to address sent messages.

2.2 Secondary targets

2.2.1 Training

2.2.2 Education

2.2.3 Knowledge

2.3 Source Code

2.3.1 class Pulex

The Pulex class represents a neutral object containing client data to become moved to other clients using a server able to move Container objects. To be able to do so, a Container is able to move data wiht minimum amount of knowledge of what these data are. This works similar to a packet moved by a post office. The sender ist known, the receiver ist known but the content is not.

Regarding a Pulex, the sender ist mainly unknown, the receiver will become partially known if and when he calls is Container. The server will give the packet to every receiver who successfuly claims to be the receiver of it. To make things with a Pulex not as easy as with a Container, a Pulex sends the identity of its sender and its receiver in the form of a 40 character ascii representation of the fingerprint of their certificates.

The parent class to be used as 'inherited::methode()' for abstract usage of inherited methods. Mimimizes logical redundancy

```
    typedef CContainer inherited;
```

May be this is not a good idea at all, but enables compact code, which also may not be good idea. This operator appends data to the object as if it were a data sink which it is ;-)

```
            const std::string& operator << ( const std::string& rsData );
```

Friendly operators to pipe data to different types of stream

```
    friend std::ostream& operator << ( std::ostream&, CPulex& );
    friend      CSocket& operator << (      CSocket&, CPulex& );
```

A template to be used by output pipe operators to send the puley off

```
    template<typename T>
      T& Send( T& roStream );
```

```
#ifndef _PULEX_H
#define _PULEX_H

#include
#include

#include <list>
#include <iostream>


extern bool g_bVerbose;

class CPulex : public CContainer
  {
  private:
    typedef CContainer inherited;

    static const std::string s_sClassName;

  public:
              CPulex();
    virtual ~CPulex();
```

```
    virtual const std::string& ClassNameGet() const;

            const std::string& operator << ( const std::string& rsData );

    friend std::ostream& operator << ( std::ostream&, CPulex& );
    friend      CSocket& operator << (      CSocket&, CPulex& );
  protected:
    template<typename T>
      T& Send( T& roStream );

  protected:
    long ClientSideIDGet();

  }; // class CPulex


#endif // _PULEX_H
```

2.3.2 class Crypto

```
/***************************************************************************
 crypto.h  - description
 -------------------------
 begin                : Thu Dec 02 2010
 copyright            : Copyright (C) 2010 by Manfred Morgner
 email                : manfred@morgner.com
 ***************************************************************************/

#ifndef _CRYPTO_H
#define _CRYPTO_H

#include

#include <string>
#include <vector>

#include <openssl/rand.h> // RSA
#include <openssl/evp.h>  // EVP_MAX_KEY_LENGTH, ...


extern bool g_bVerbose;

#define RANDOM_BUFFER_SIZE 1024

typedef std::vector<unsigned char> CUCBuffer;

class CCrypto
  {
  protected:
```

```
  CUCBuffer m_oData;

  X509*     m_pX509;      // m_oEvpPkey frees this pointer
  CEvpPkey  m_oEvpPkey;
  CRsa      m_oRsa;

  const EVP_CIPHER* m_pfCipher;
  const EVP_MD*     m_pfDigest;

  unsigned char m_aucKey [EVP_MAX_KEY_LENGTH]; // 32
  unsigned char m_aucIv  [EVP_MAX_IV_LENGTH];  // 16
  unsigned char m_aucSalt[PKCS5_SALT_LEN];     //  8

  static const std::string s_sDelimiter;

          CCrypto(){}
public:
          CCrypto( const std::string& rsInput );
  virtual ~CCrypto();

  void operator =    ( RSA* pRsa ) { m_oRsa = pRsa; }
       operator RSA* ()            { return (RSA*)m_oRsa; }

  void RsaKeyLoadPublic ( const std::string& rsFileRsaKey );
  void RsaKeyLoadPrivate( const std::string& rsFileRsaKey );
  void RsaKeyLoadFromCertificate( const std::string& rsFileCertificate );

  std::string      EncryptToBase64  ();
  const CUCBuffer& DecryptFromBase64( const std::string& rsBase64 );

protected:
  void RandomSeed();
  void RsaKeyGenerate();

  void RandomGet( CUCBuffer& roBuffer );
  void RandomGet( unsigned char* pucBuffer, size_t nBufferSize );

  std::string      ConvertToBase64  ();
  const CUCBuffer& ConvertFromBase64( const std::string& rsBase64 );

  std::string ConvertToBase64( const unsigned char* pucData, size_t nSize );
  std::string EncryptToBase64( const unsigned char* pucData, size_t nSize );

  bool EncryptRsaPublic ( CUCBuffer& roBuffer );
  bool DecryptRsaPrivate( CUCBuffer& roBuffer );

  std::string SymetricKeyRsaPublicEncrypt();
  bool SymetricKeyRsaPrivateDecrypt( const std::string& rsEncrypted );

  int  SymetricKeyMake( const EVP_CIPHER* pfCipher = EVP_des_ede3_cfb(),
                        const EVP_MD*     pfDigest = EVP_sha1() );
```

```
};  // class CCrypto

#endif // _CRYPTO_H
```

OpenSSL

GPL