Università degli Studi di Salerno
Facoltà di Scienze Matematiche, Fisiche e Naturali

Monitoraggio attivo utenti

Implementazione in ANSI C di una applicazione utilizzando le OpenSSL

Progetto realizzato da: Cappetta Fabio, Marotta Salvatore, Rispoli Francesco

Prof. Alfredo de Santis

Introduzione

- Dallo scenario tipico dei sistemi multiutente odierni emerge la problematica del monitoraggio "sicuro" delle attività degli utenti

Il controllo della posta

Supponiamo di aver ottenuto, dalle autorità competenti, le necessarie autorizzazioni per il controllo della posta degli utenti

Il controllo della posta: problema

- Comunicazione in chiaro
- Intrusi sulla rete
- Segretezza delle informazioni

Il controllo della posta: proposte

- Crittografia a chiave pubblica (RSA) ?
- 3DES ?
- RC6 ?

Combiniamoli in Digital Envelope

Il controllo della posta: una soluzione

- Utilizzo di un *bridge* "cifrato"
- Utilizzo di un client "autorizzato"
- Scambio di chiavi di sessione

Il nuovo scenario: gli attori

- Il CryptoBridge

  Una applicazione scritta in C che gestisce la comunicazione sicura tra il server pop3 ed i client

- Il suo Client

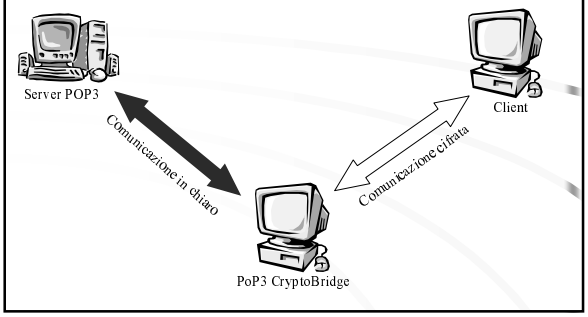
  invia le richieste dell'utente, cifrate, al CryptoBridge e ne decifra le risposte

Il protocollo: scelte implementative

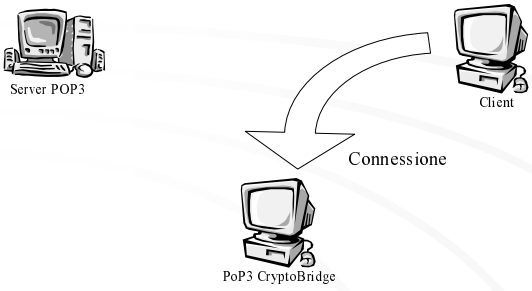
- 3 DES (ECB)
  - Genera casualmente la coppia (k_1, k_2)
  - k_i lunga 56 bit
  - Vita brevissima (solo una sessione)
- RSA
  - Chiavi in formato PEM lunghe 512 bit
  - Chiave pubblica solo nel Crypto Bridge
  - Chiave privata solo nel Client

Il protocollo

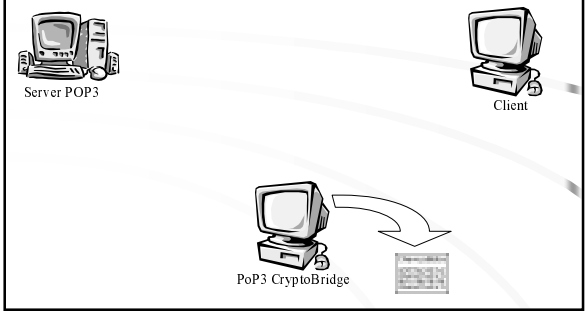
Il protocollo



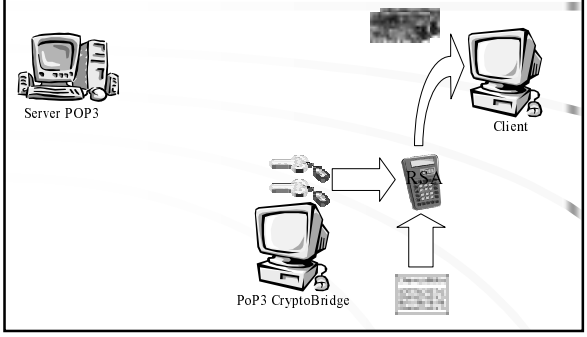
Il protocollo



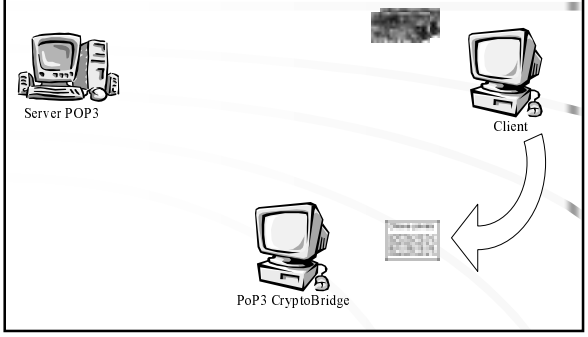
Il protocollo



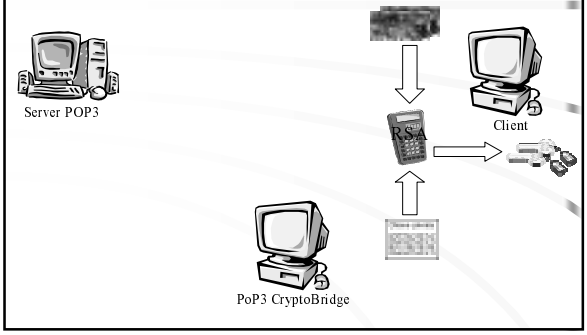
Il protocollo



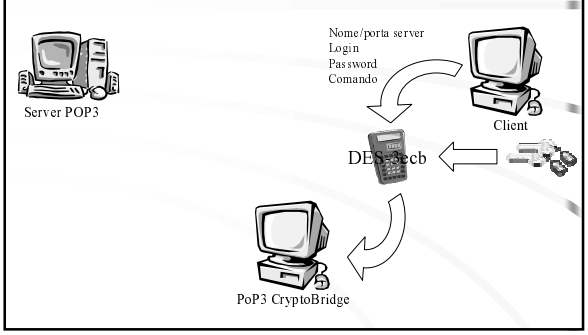
Il protocollo



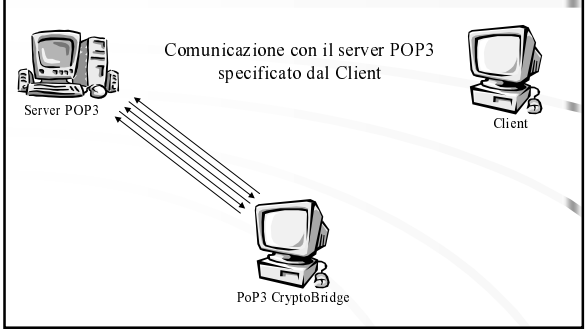
Il protocollo



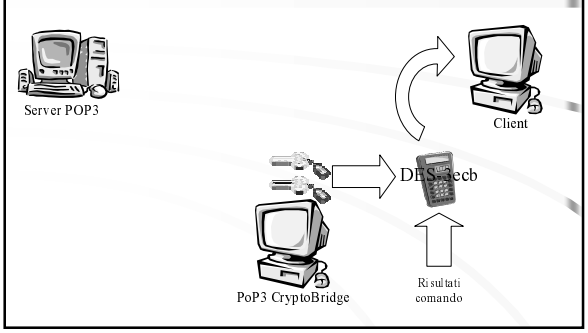
Il protocollo



Il protocollo



Il protocollo



Il protocollo

Il PoP3 Crypto Bridge chiude la connessione

Gli strumenti utilizzati

Gli strumenti utilizzati

- OpenSSL 0.9.5a (http://www.openssl.org)
- ANSI C
- Berkley's API per i socket

OpenSSL

- E' una libreria che implementa utili funzioni di crittografia
- E' GPL (Gnu Public License)
- Sono disponibili i sorgenti
- E' altamente portabile

OpenSSL: primitive DES 1

- Le funzioni utilizzate per il DES:
  - des_random_key: genera una chiave des di 8 byte (56 bit + padding) in formato des_cblock e ne verifica la sicurezza
  - des_set_key: trasforma la chiave des_cblock in una chiave des_schedule, cioè nello scheduling proprio di quella chiave

OpenSSL: primitive DES 2

- Le funzioni utilizzate per il DES:
  - Des_ecb3_encrypt: a seconda dell'ultimo parametro, cifra o decifra (risp. DES_ENCRYPT, DES_DECRYPT) un blocco des_cblock di 8 byte utilizzando le chiavi k1, k2, k1.

OpenSSL: primitive RSA 1

- Le funzioni utilizzate per RSA:
  - Per generare le chiavi in formato PEM è stata utilizzata l'utility openssl in questo modo:

    openssl genrsa –out chiave.pem *KEY_LEN*
  - Per estrarre l'esponente pubblico dalla chiave è stata utilizzata

    openssl –in chiave.pem –pubout –out public.pem

OpenSSL: primitive RSA 2

- Le funzioni utilizzate per RSA:
  - BIO_read_filename: apre un file PEM e lo prepara alla lettura;
  - PEM_read_bio_RSA_PUBKEY: legge la chiave **pubblica** dal file aperto con BIO_read_filename.
  - PEM_read_bio_RSA_PrivateKey: legge la chiave **privata** dal file aperto con BIO_read_filename

OpenSSL: primitive RSA 3

- Le funzioni utilizzate per RSA:
  - RSA_public_encrypt: cifra un blocco di N byte utilizzando la chiave pubblica
  - RSA_private_decrypt: decifra un blocco di N byte utilizzando la chiave privata

OpenSSL: esempio 1

```
int Write (int filedes, const char * buffer,int writetype)
{
  int i=0;
  des_cblock block2crypt,cryptedblock;
  des_key_schedule skey1,skey2;
  int retval=0;

  if(writetype==PLAIN)
    write(filedes,buffer,strlen(buffer));
  else
    {
      des_set_key(&sessionkey1,skey1);
      des_set_key(&sessionkey2,skey2);

      while(retval!=-1)
          {
            retval=substring8(buffer,block2crypt,i);
            des_ecb3_encrypt(&block2crypt,&cryptedblock,
                             skey1,skey2,skey1,DES_ENCRYPT);
            write(filedes,cryptedblock,8);
            i+=8;
          }
    }
}
```

OpenSSL: esempio 2 (parte 1)

```
void SendSessionKey (int sockfd)
{
  des_cblock in,out,outin;
  des_key_schedule myks1,myks2;
  char des2key[16],
    des2key_c[512],
    string2send[512],
    temp[50];
  int num,
    clen = 0,
    plen=16,
    i=0;
  RSA *key;
  BIO *keyin=NULL;

  while (!des_random_key(&sessionkey1));
  while (!des_random_key(&sessionkey2));

  for (i = 0; i <= 7; i++)
    {
      des2key[i]=sessionkey1[i];
      des2key[i+8]=sessionkey2[i];
    }
```

OpenSSL: esempio 2 (parte 2)

```
  keyin=BIO_new(BIO_s_file());
  key=RSA_new();

  BIO_read_filename(keyin,"public.pem");
  key=PEM_read_bio_RSA_PUBKEY(keyin,NULL,NULL,NULL);
  num = RSA_public_encrypt(plen, des2key, des2key_c, key, RSA_PKCS1_PADDING);


  sprintf(temp,"%i ",num);
  strcpy(string2send,temp);
  for(i=strlen(temp);i<num+strlen(temp);i++)
    string2send[i]=des2key_c[i-strlen(temp)];
  string2send[num+strlen(temp)]='\0';
  write(sockfd,string2send,num+strlen(temp));
}
```

OpenSSL: Il formato PEM

- Standard per la scrittura di chiavi e certificati in file ASCII **(rfc-1423)**
  - Usa due Encapsulation Block (EB):
    - -----BEGIN RSA KEY -----
    - -----END RSA KEY -----
  - Tra gli EB viene inserita la chiave codificata in BASE 64 secondo l'algoritmo descritto nell'rfc.