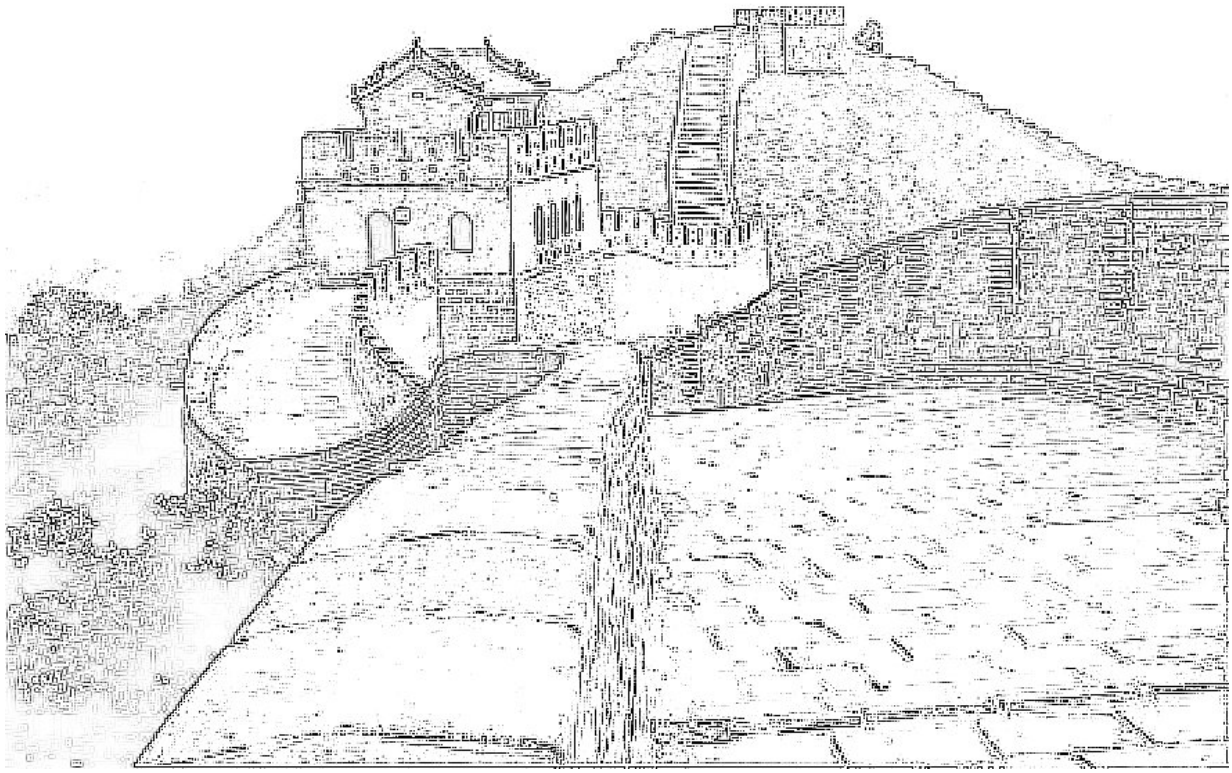
INTERNATO DI LABORATORIO DI TELECOMUNICAZIONI

Appunti sulle

ARCHITETTURE DI RETE SICURE

X509 IPSec/IKE HTTPS



UNIVERSITÀ DEGLI STUDI DI PARMA

ANDREA BARONTINI

Corso di Laurea in Ingegneria delle Telecomunicazioni

INTERNATO DI LABORATORIO DI TELECOMUNICAZIONI

Appunti sulle

ARCHITETTURE DI RETE SICURE

Versione 1.10 - Maggio 2004

Andrea Barontini

(andrea.barontini@ieee.org)

Università degli Studi di Parma

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

Versioni di questo documento:

1.10 (Maggio 2004)	Inserito elenco versioni, indirizzo email aggiornato, modifiche a pagina 9 e 27, PDF versione 1.4
1.00 (Settembre 2003)	Prima versione "ufficiale", PDF versione 1.2
0.90 BETA (Settembre 2003)	Versione per revisione finale
PRELIMINARE 0.30 (Settembre 2003)	Capitolo su HTTPS concluso, figura a pag.3 a toni di grigio
PRELIMINARE 0.20 (Giugno 2003)	Figura a pag.3 a colori, capitolo su HTTPS da concludere, capitolo su IPSec mancante
unnumbered & < 0.20	Per le prime revisioni durante la stesura

Tabella dei contenuti

Introduzione

A proposito di questo documento

Questo documento nasce in relazione all'attività di "Internato di Laboratorio di Telecomunicazioni" svolta sotto la guida del Prof. Luca Veltri nell'ambito del Corso di Laurea in Ingegneria delle Telecomunicazioni presso il Dipartimento di Ingegneria dell'Informazione dell'Università degli Studi di Parma.

Tale attività di laboratorio, della durata di 125 ore e svolta durante i mesi di Aprile, Maggio e Giugno 2003, ha riguardato il rinvenimento, l'installazione, la configurazione e l'analisi di particolari scelte implementative di alcune soluzioni software relative alla messa in sicurezza delle comunicazioni basate sulla suite TCP/IP.
È stato seguito un approccio sistemistico accompagnato dalla presa di coscienza, seppur non esaustiva, delle scelte protocollari alla base delle varie tecnologie prese in esame.
I capitoli che seguono si propongono quindi di essere un po' "relazione conclusiva" di tale attività, un po' "promemoria" delle problematiche affrontate e delle soluzioni trovate, senza ovviamente alcuna pretesa di originalità o completezza. Sono anzi riportati link a siti Internet e riferimenti a pubblicazioni cartacee che si sono rivelate preziose fonti di informazione.
Il lettore "tipo" di questo documento può quindi essere individuato in chi conosce la teoria degli argomenti affrontati e vuole vederne un case-study ragionato, senza la pretesa però di trovarsi davanti a un manuale di configurazione.

Si parla soprattutto di sistemi di crittografia, autenticazione, certificazione; nell'ambito dell'Internato non sono stati presi in considerazione, per es., firewall e IDS che comunque rientrano a pieno titolo tra le tecnologie di sicurezza per le reti dati moderne.
L'ultimo capitolo rende invece conto di "piccolezze" affrontate durante l'installazione e la configurazione della LAN utilizzata. È inoltre riportato un promemoria delle impostazioni di rete dei computer che funzionano da router.

Layout e non solo

L'aspetto grafico di questo documento, specie per quello che riguarda la copertina, è ispirato alle pubblicazioni O'Reilly (www.oreilly.com). La casa editrice californiana è stata una delle prime a far "entrare in tipografia" il mondo OpenSource; mondo tradizionalmente più attento alle problematiche di sicurezza rispetto al software proprietario. Non è infatti un caso che praticamente tutto il software utilizzato durante l'Internato sia costituito da pacchetti per Linux.

L'immagine in copertina raffigura un tratto della Grande Muraglia Cinese. Costruita sotto l'impero di Qin Shihuangdi (221-210 a.C.), lunga più di 6000 Km, avrebbe dovuto servire come frontiera invalicabile per le numerose tribù nomadi, per lo più di origine mongola, che continuamente minacciavano di invadere la Cina.
Recentemente in occidente è quindi sembrato naturale parlare di "Grande Muraglia" per riferirsi all'opera di filtraggio da parte del governo di Pechino dei contenuti Internet consultabili dai cittadini cinesi.
Censura che, come verificato da Jonathan Zittrain e Benjamin Edelman della Harvard Law School (cyber.law.harvard.edu/filtering/china/), avviene sulle dorsali che collegano la Cina al resto del mondo sfruttando, tra l'altro, meccanismi di address-redirection, firewalling, application-proxying tipici delle architetture di rete sicure.

La foto originale è stata scaricata da una pagina del sito www.difrontealfuturo.net (www.difrontealfuturo.net/2breve%20storia.htm) realizzato dal Liceo Linguistico "Veronese" di Montebelluna (TV). Per il resizing e l'applicazione dei filtri che hanno permesso l'effetto "simil-carboncino" è stato impiegato Jasc Paint Shop Pro.
Per quanto invece riguarda sia la stesura dei testi che l'impaginazione si è utilizzato Microsoft Word.

Convenzioni tipografiche

A meno che di volta in volta non venga diversamente specificato, in questo documento tipicamente valgono le seguenti convenzioni tipografiche:

font standard	utilizzato per i testi comuni
grassetto	utilizzato per i titoli e per sottolineare passaggi del testo particolarmente significativi
corsivo	utilizzato nel testo per fare riferimento alle figure e alle tabelle e nei titoli
`spaziatura fissa`	utilizzato all'interno del testo per riportare i comandi da digitare e come font di default nelle figure che riproducono screenshot della console
`corsivo spaz.fissa`	utilizzato per indicare parametri e/o comandi che non devono essere riportati letteralmente ma sostituiti con valori opportuni

Se necessario specificare sinteticamente comandi e/o parametri opzionali si userà la notazione ormai universalmente accettata che si richiama alle grammatiche formali BNF:

`[ ]` racchiuderanno testo (comandi o parametri a seconda del caso) che può essere omesso

`|` si userà per indicare una casistica di opzioni tra cui scegliere

(Si noti l'uso del corsivo a spaziatura fissa, visto che quelli sopra sono "specificatori grammaticali" e non comandi letterali)

La LAN utilizzata per le implementazioni

La rete locale in cui si è svolta l'attività di Internato è costituita da 9 PC con funzioni di router, server e postazioni di lavoro collegati come indicato in *Figura 1:*

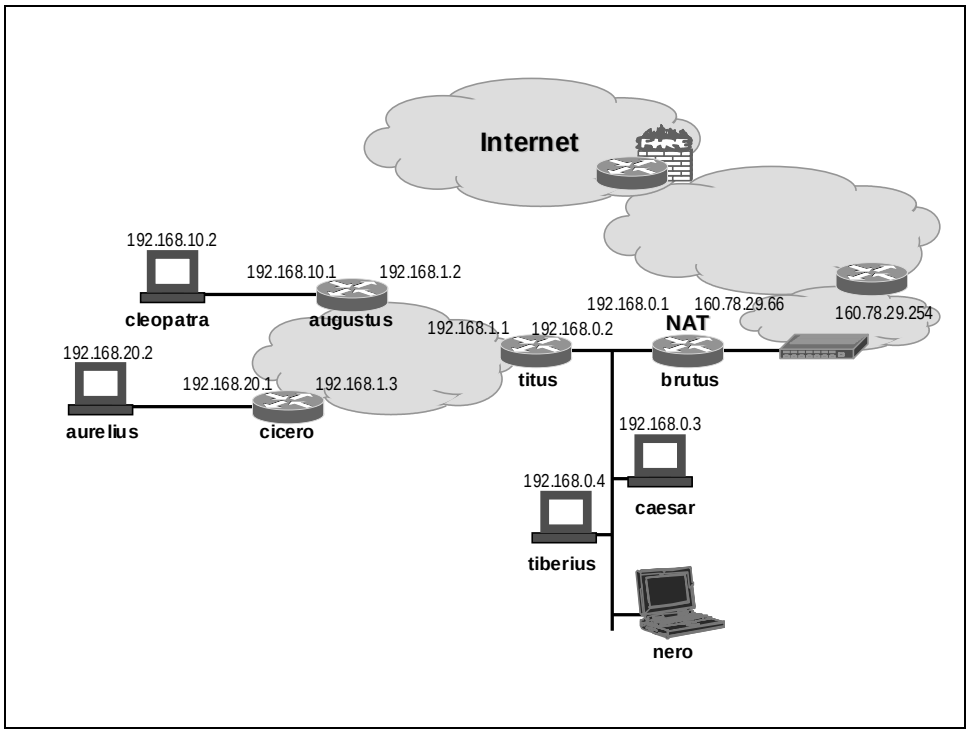


Figura 1: La rete locale utilizzata per le implementazioni

Sistemi operativi installati e ruolo dei computer nell'ambito dell'Internato:

brutus	RedHat Linux 7.1	Firewall esterno, NAT e Router (*)
titus	RedHat Linux 7.1	Firewall interno (correntemente non implementato) e Router (*)
augustus	RedHat Linux 7.3	Router (*) e IPSec Gateway
cleopatra	MS Windows 98 / RedHat Linux 7.3	Terminale utente, terminale VPN, client HTTPS, postazione di lavoro su cui è stata scritta gran parte di questo documento!
cicero	RedHat Linux 7.3	Router (*)
aurelius	MS Windows98 / RedHat Linux 7.3	Terminale utente, terminale VPN, postazione sviluppo Java
caesar	RedHat Linux 7.3	Server DMZ (correntemente Apache)
tiberius	MS Windows 2000	Postazione di lavoro (sviluppo Java e Matlab)
nero	MS Windows 98	Portatile del Prof. Luca Veltri

Tabella 1: Ruolo dei computer e SO installati

Entrambe le distribuzioni RedHat utilizzate utilizzano un kernel della serie 2.4.

(*) Per un promemoria delle configurazioni dei router vedi l'ultimo capitolo.

I Certificati

La "forza" dei certificati

Il ricorso ai certificati rappresenta un metodo di autenticazione estremamente flessibile. Questa superiorità deriva dalla natura intrinseca dei certificati stessi, che rendono superflui eventuali accordi preventivi tra le parti per quanto riguarda la password da utilizzarsi durante la secretazione dei dati (risultato ad onor del vero già raggiunto col sistema di autenticazione a chiave pubblica) e che soprattutto forniscono un metodo integrato di validazione delle informazioni pubbliche fornite (nel caso del sistema a chiave pubblica "semplice" la gestione di questo aspetto non era prevista). Tali vantaggi sono tali da aver reso l'autenticazione tramite certificato uno standard "de facto": ad esempio lo ritroviamo, tanto per voler citare solo le implementazioni più note, nel protocollo HTTPS e nelle soggiacenti librerie SSL/TLS e nella creazione di tunnel IPSec.
Vediamo quindi come sono strutturati i certificati e come creare una semplice Certification Authority.

Lo standard X509

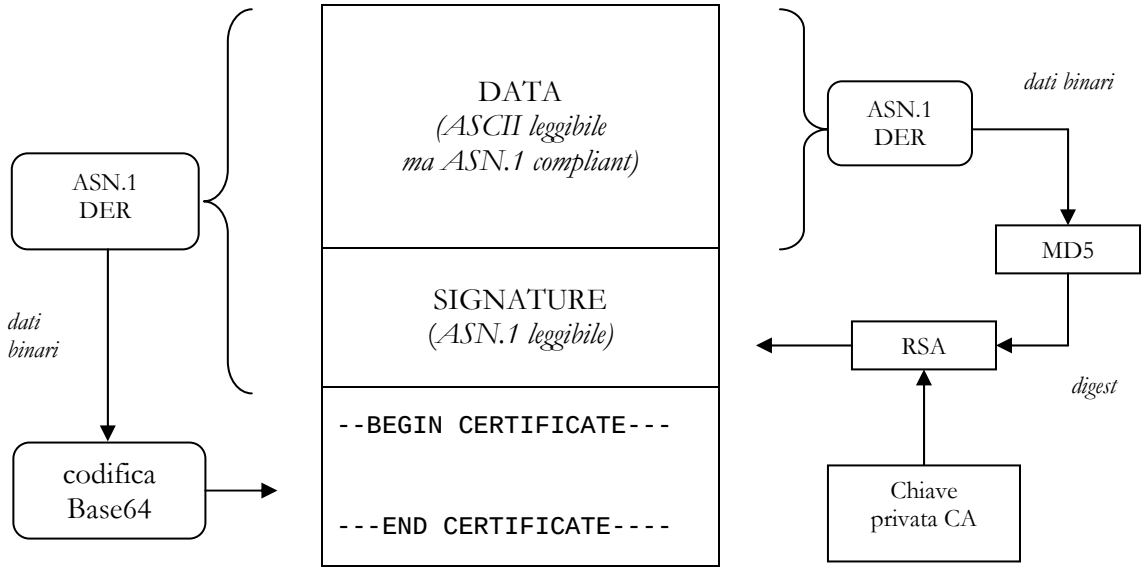


Figura 2: Struttura tipica dei file *.pem

È possibile (specie utilizzando OpenSSL) imbattersi in certificati come quello schematizzato in *Figura 2*. Si tratta di certificati X509, uno standard riconosciuto facente parte di OSI.

I certificati X509 sono definiti utilizzando una notazione chiamata ASN.1 (Abstract Syntax Notation 1) che si occupa di specificare in maniera precisa i tipi dei dati che costituiscono il certificato. ASN.1 può essere codificato in molti modi ma quello di nostro interesse è il DER (Distinguished Encoding Rules), che produce un file binario di ridotte dimensioni. Può inoltre avvenire un'ulteriore codifica Base64, arrivando a un testo ASCII di questo tipo:

```
-----BEGIN CERTIFICATE-----
MIIE5zCCA8+gAwIBAgIBAjANBgkqhkiG9w0BAQQFADCBnTELMAkGA1UEBhMCSVQx
FzAVBgNVBAgTDkVtaWxpYSBSb21hZ25hMQ4wDAYDVQQHEwVQYXJtYTEcMBoGA1UE
ChMTVW5pdmVyc2l0eSBvZiBQYXJtYTEQMA4GA1UECxMHVExDIExhYjEUMBIGA1UE
AxQLQXVndXN0dXNfQ0ExHzAdBgkqhkiG9w0BCQEWEG15LWVtYWlsLWFkZHJlc3Mw
HhcNMDMwNDE2MTQ0MjEyWhcNMDQwNDE1MTQ0MjEyWjCBojELMAkGA1UEBhMCSVQx
FzAVBgNVBAgTDkVtaWxpYSBSb21hZ25hMQ4wDAYDVQQHEwVQYXJtYTEcMBoGA1UE
ChMTVW5pdmVyc2l0eSBvZiBQYXJtYTEQMA4GA1UECxMHVExDIExhYjESMBAGA1UE
AxMJY2xlb3BhdHJhMSYwJAYJKoZIhvcNAQkBFhdjbGVvcGF0cmEtZW1haWwtYWRk
cmVzczCCASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBANM4G6biNljOKnA9
1HRsQmp4YwCFPmnJU2qKgo3IvEXDl2uF8orgKb2wmkJ3/EnkPYXi1OTUwcVAFJky
1luOCrJNIMD30Qf/8VVOs7s2mJsliav7B5qA8zX7SMlb90OmecwUZrYci/o8N1A0
zD1+Mokz4JWNj6dYztO6VfO7Najp75Ib3a7m18cUYGgixCGyt7mlfBQR06CbNKua
EP9EXg20QX1qFsFpdhfxsvB/FjtmFBIvCoCB08/pfvHS3WFPTO46gzGRqelp12qh
jzQnR19lqoK+f3bz04HBTu75dyPG26BTFBPDR08sNsFKdo+XzBjIHLtKoMVmp9BP
ZeiV5TUCAwEAAaOCASkwggElMAkGA1UdEwQCMAAwLAYJYIZIAYb4QgENBB8WHU9w
ZW5TU0wgR2VuZXJhdGVkIENlcnRpZmljYXRlMB0GA1UdDgQWBBQA7EijIVwttl1r
SV8IobO772LxQDCBygYDVR0jBIHCMIG/gBR22bTSAXApmlMoNxV+y1NdW2U4HqGB
o6SBoDCBnTELMAkGA1UEBhMCSVQxFzAVBgNVBAgTDkVtaWxpYSBSb21hZ25hMQ4w
DAYDVQQHEwVQYXJtYTEcMBoGA1UEChMTVW5pdmVyc2l0eSBvZiBQYXJtYTEQMA4G
A1UECxMHVExDIExhYjEUMBIGA1UEAxQLQXVndXN0dXNfQ0ExHzAdBgkqhkiG9w0B
CQEWEG15LWVtYWlsLWFkZHJlc3OCAQAwDQYJKoZIhvcNAQEEBQADggEBAC+ookqq
Mf5IOek8RjYcsSDyRR08queh6SnnHwcsFn29FO5fvty4+yXDHTOeZY+aX0MzBndS
yVc/fgclfmIYPnhHhjx1lCxs0bG58XhHcdqkGdgcsibk098L435+7yu0lUStvhKp
RD+ngqRyLVW/y/3M0Pl/ul245NMU615/pPZ5nK2G8eb6YRlPKJEPZPeJtgSHOI8a
ci6/rIcmqxN3byddwjWkXhz5yDmkha6Yb3KJgLm/A+ywpXhaNzrk7lwqW6uboQoY
ZyVQUozP1Zdg1tvMjELdwcQOiUfwiyPv/WUKs4OSzyVNBPPFA4wmcviQfwS+D7qF
tQ3c8NOrVRxgT24=
-----END CERTIFICATE-----
```

Figura 3: Codifica Base64 di un certificato

La *Figura 1* rende conto della tipica struttura di un file *.pem contenente il certificato, schematizzando le relazioni funzionali tra i vari blocchi dello stesso.
La sezione **DATA** contiene "il carico utile del certificato": da chi è stato emesso (la Certification Authority), qual è il suo periodo di validità (data di inizio e di fine) a chi appartiene e, ovviamente, la chiave pubblica del proprietario del certificato. Si tratta di testo ASCII leggibile, ma comunque riconducibile ad ASN.1.
La **SIGNATURE** contiene la firma, ad opera della CA, del blocco DATA. Tale firma è generata come segue:

- Le informazioni della sezione DATA vengono codificate secondo ASN.1/DER
- Sui dati binari ottenuto viene applicata una funzione hash (tipicamente MD5)
- Il digest (128 bit nel caso di MD5) sono quindi firmati con la chiave privata della Certification Authority utilizzando, per esempio, l'algoritmo RSA
- La firma ottenuta (binaria) viene quindi convertita in forma "leggibile", coerentemente con il blocco DATA

L'**ultima sezione** del certificato è invece ridondante rispetto alle precedenti e quindi utilizzabile in maniera autonoma. È ottenuta codificando in ASN.1/DER i due blocchi precedenti (ricordiamo che anche la firma alla fine viene in qualche maniera codificata in ASCII) e in Base64 i dati binari ottenuti dalla codifica ASN.1/DER.
Nell'ottica di quanto detto può essere interessante riportare il certificato creato per uno dei PC della LAN utilizzata: il computer è **cleopatra** e la certification authority Augustus_CA.

Per comodità di lettura sono evidenziati i confini tra le tre sezioni e sono stati "accorciati" il campo `Modulus`, il campo `Signature Algorithm` e il terzo blocco:

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 2 (0x2)
        Signature Algorithm: md5WithRSAEncryption
        Issuer: C=IT, ST=Emilia Romagna, L=Parma, O=University of Parma, OU=TLC Lab,
CN=Augustus_CA/Email=my-email-address
        Validity
            Not Before: Apr 16 14:42:12 2003 GMT
            Not After : Apr 15 14:42:12 2004 GMT
        Subject: C=IT, ST=Emilia Romagna, L=Parma, O=University of Parma, OU=TLC Lab,
CN=cleopatra/Email=cleopatra-email-address
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
            RSA Public Key: (2048 bit)
                Modulus (2048 bit):
                    00:d3:38:1b:a6:e2:36:58:ce:2a:70:3d:d4:74:6c:
                    42:6a:78:63:00:85:3e:69:c9:53:6a:8a:82:8d:c8:
                    bc:45:c3:97:...
                    ...
                                                        ...:23:c6:
                    db:a0:53:14:13:c3:47:4f:2c:36:c1:4a:76:8f:97:
                    cc:18:c8:1c:bb:4a:a0:c5:66:a7:d0:4f:65:e8:95:
                    e5:35
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Basic Constraints:
                CA:FALSE
            Netscape Comment:
                OpenSSL Generated Certificate
            X509v3 Subject Key Identifier:
                00:EC:48:A3:21:5C:2D:B6:5D:6B:49:5F:08:A1:B3:BB:EF:62:F1:40
            X509v3 Authority Key Identifier:
                keyid:76:D9:B4:D2:01:70:29:9A:53:28:37:15:7E:CB:53:5D:5B:65:38:1E
                DirName:/C=IT/ST=Emilia Romagna/L=Parma/O=University of Parma/OU=TLC
Lab/CN=Augustus_CA/Email=my-email-address
                serial:00

Signature Algorithm: md5WithRSAEncryption
        2f:a8:a2:4a:aa:31:fe:48:39:e9:3c:46:36:1c:b1:20:f2:45:
        1d:3c:aa:e7:a1:e9:29:e7:1f:07:2c:16:7d:bd:14:ee:5f:be:
        dc:b8:fb:...
        ...
                                                    ...:c4:0e:89:
        47:f0:8b:23:ef:fd:65:0a:b3:83:92:cf:25:4d:04:f3:c5:03:
        8c:26:72:f8:90:7f:04:be:0f:ba:85:b5:0d:dc:f0:d3:ab:55:
        1c:60:4f:6e

-----BEGIN CERTIFICATE-----
MIIE5zCCA8+gAwIBAgIBAjANBgkqhkiG9w0BAQQFADCBnTELMAkGA1UEBhMCSVQx
FzAVBgNVBAgTDkVtaWxpYSBSb21hZ25hMQ4wDAYDVQQHEwVQYXJtYTEcMBoGA1UE
ChMTVW5pdmV...
...
                                          ...b6YRlPKJEPZPeJtgSHOI8a
ci6/rIcmqxN3byddwjWkXhz5yDmkha6Yb3KJgLm/A+ywpXhaNzrk7lwqW6uboQoY
ZyVQUozP1Zdg1tvMjELdwcQOiUfwiyPv/WUKs4OSzyVNBPPFA4wmcviQfwS+D7qF
tQ3c8NOrVRxgT24=
-----END CERTIFICATE-----
```

Figura 4: Il file cleopatra.pem

Si noti che sia a proposito della chiave pubblica del proprietario del certificato, sia a proposito della firma, vengono riportati dati e parametri che permettono di identificare gli algoritmi crittografici utilizzati. Questo perchè ricorrere a MD5 e RSA è sicuramente la scelta più diffusa, ma non l'unica possibile: la sola condizione è che tutte le parti in gioco (ovvero gli estremi della comunicazione e il certificatore) siano in grado di utilizzare gli algoritmi scelti.
Ovviamente potrà variare la "forma" della chiave pubblica: il fatto che ora sia costituita dal campo `Modulus` e dal campo `Exponent` è semplicemente una conseguenza dell'uso di RSA.

Potrebbe stupire l'utilizzo di un esponente piuttosto basso (e inoltre facilmente rinvenibile in molti certificati). Bruce Schneier in "*Applied Criptography, second edition*" (John Wiley & Sons) spiega che questo e pochi altri piccoli valori rappresentano un compromesso ragionevole tra sicurezza fornita e risorse computazionali richieste.

L'ambiente OpenSSL

OpenSSL (www.openssl.org) è un "toolkit" che implementa i protocolli SSL (Secure Sockets Layer) e TLS (Transport Layer Security) e gli standard crittografici da loro richiesti.
Può essere utilizzato in modo batch con comandi del tipo:

```
openssl <commando di openssl> [<parametro/i del comando>]
```

oppure come shell, in modalità interattiva.
Sono disponibili comandi per la completa gestione di certificati, algoritmi di digesting/hashing e algoritmi crittografici. Questo documento non può e non vuole essere un manuale di riferimento di OpenSSL, si riportano quindi solo un paio di comandi utili per fare alcune verifiche su quanto fin'ora detto:

```
openssl base64 -in <file di input> -out <file codificato>
```

codifica *<file di input>* in Base64 generando il file *<file codificato>*

```
openssl base64 -d -in <file codificato> -out <file di output>
```

esegue la decodifica

```
openssl asn1parse -inform DER -in <file di input>

openssl asn1parse -inform PEM -in <file di input>
```

eseguono il parsing ASN.1 del file *<file di input>*. La direttiva `-inform` serve per specificare se il file fornito è codificato in DER o DER+Base64. Per DER+Base64 si utilizza lo specificatore `PEM` per ragioni "storiche" visto che la codifica Base64 viene utilizzata nello standard MIME (RFC 1521) PEM (**P**rivacy **E**nhancement for Internet Electronic **M**ail, RFC 1421 e seguenti).
Se la direttiva `-inform` è assente si assume che l'input sia codificato in Base64.
Il risultato del parsing viene ritornato sullo standard output.

Altri comandi di OpenSSL verranno esaminati nei prossimi paragrafi nel momento in cui si genereranno certificati. Per un prontuario completo il testo di riferimento è "*Network Security with OpenSSL*" di John Viega, Matt Messier e Pravir Chandra, O'Reilly Editore.

Creazione di una CA e generazione dei certificati

Per la creazione della Certification Authority e dei certificati si è seguito il tutorial di Nate Carlson (www.natecarlson.com/linux/ipsec-x509.php).

Si tratta di un documento che affronta tali argomenti nell'ottica della configurazione per questo tipo di autenticazione di FreeS/WAN (un pacchetto che implementa IPSec in ambiente Linux; avremo modo di parlarne approfonditamente).
Per gestire praticamente tutti gli aspetti della questione si utilizza uno script appartenente al toolbox OpenSSL: /usr/share/ssl/misc/CA (per lo meno sotto RedHat, per le altre distribuzioni il nome o il posizionamento nel filesystem potrebbe variare).
Tale script (che genera i file *.pem visti in apertura di capitolo) permette operazioni a un livello di astrazione abbastanza alto secondo le seguenti procedure interattive:

Creazione di una Certification Authority (da eseguirsi una sola volta)

- Generazione automatica della coppia chiave pubblica/chiave privata secondo i parametri indicati nel file di configurazione di OpenSSL (è possibile per esempio stabilire il numero di bit della chiave privata e la durata dei certificati).
- Richiesta di una password a protezione della chiave privata (codificata DER+Base64).
- Richiesta di inserimento dei dati della Certification Authority.
- Generazione del certificato self-signed.

Creazione dei certificati

- Generazione di una nuova richiesta di certificato:
  - Generazione automatica della coppia chiave pubblica/chiave privata secondo i parametri indicati nel file di configurazione dello script (è possibile per esempio stabilire il numero di bit della chiave privata e la durata dei certificati).
  - Richiesta di inserimento di una password a protezione della chiave privata appena generata (**non** la password immessa prima per proteggere la chiave privata della CA!).
  - Richiesta di inserimento dei dati del proprietario del futuro certificato.
- Firma della richiesta di certificato appena creata:
  - Richiesta di immissione della password a protezione della chiave privata della Certification Authority (per permettere di fatto l'operazione di firma).
  - Visione e conferma della correttezza dei dati contenuti nella richiesta di certificato
  - Effettiva generazione del certitifcato.

Tutto funziona "magicamente in automatico", ma è comunque interessante sbirciare all'interno dello script: essendo scritto in bash è infatti facile ritrovare i comandi OpenSSL che "fanno il lavoro sporco" e quindi prendere confidenza con il modus operandi di questo toolbox.

Alla fine della generazione della richiesta di certificato lo script produce un file contente sia la richiesta vera e propria, sia la chiave privata generata. Questo non è assolutamente un problema nel caso di test in cui ci sia una Certification Authority "locale" di prova; in caso invece di reale situazione operativa è ovviamente opportuno ricordare di scorporare le due parti per poter inviare **solo la richiesta** al certificatore (anche se la chiave privata sarebbe comunque protetta dalla password immessa).

Nel caso della creazione della Certification Authority si è parlato di certificato self-signed. Questo perché, essendo il certificatore la "radice" del sistema di validazione, non ha alcuna entità "alle spalle" che possa garantirne l'identità. Convenzionalmente si è quindi deciso che tali certificati debbano essere "autofirmati" (utilizzando il comando `openssl asn1parse` è possibile verificare che ovviamente in tal caso i dati del proprietario del certificato e dell'autorità di certificazione coincidono). È quindi necessario che il software che verifica la validità del certificato abbia a disposizione l'elenco delle CA riconosciute, in modo da poter concludere positivamente la "risalita" della catena dei certificati: non a caso con i browser vengono fornite le liste delle root-CA per https (quando si parlerà dell'implementazione SSL di Apache si vedrà anche un caso concreto di "catena").

Un approccio alternativo

Per la creazione della CA e dei certificati è possibile utilizzare **direttamente** i comandi OpenSSL senza dover ricorrere allo script su menzionato. Ovviamente si ha un controllo maggiore della situazione, a patto di sapere esattamente cosa si deve fare e quali sono i comandi che permettono di raggiungere lo scopo. Per quest'ultimo aspetto qualche "dritta" si può trovare nell'articolo *"Apache: sicurezza e crittografia"* di Riccardo Corsanici, apparso sul numero 102 di DEV (Dicembre 2002). Vediamo dunque di capire i comandi OpenSSL utili per la creazione di un certificato self-signed e non:

```
openssl req –new –out <richiesta certificato>
```

crea una richiesta di certificato in *<richiesta certificato>*. Durante la procedura vengono chiesti i dati del proprietario del futuro certificato e la password per proteggere la chiave privata generata e salvata nel file privkey.pem. Se si vuole fornire un file di configurazione alternativo a quello standard di OpenSSL (/usr/share/ssl/openssl.cnf in RedHat Linux) si deve indicare il parametro `–config` *<nome file di configurazione>*

```
openssl rsa –in privkey.pem –out <nome file chiave privata in chiaro>
```

dopo aver richiesto la password di protezione della chiave privata, estrae quest'ultima dal file privkey.pem e la salva, questa volta in chiaro, in *<nome file chiave privata in chiaro>*. L'utilità di questo comando è evidente quando la chiave privata deve essere utilizzata da un daemon: infatti avere la password riportata in un file di configurazione ne vanificherebbe l'utilità, doverla reinserire manualmente ad ogni riavvio del servizio potrebbe non essere accettabile. Alternativamente si può specificare la direttiva `–nodes` durante la creazione della richiesta di certificato per evitare che la chiave privata venga criptata.

```
openssl x509 –req –in <richiesta certificato> -out <certificato> -signkey
        <nome file chiave privata in chiaro>
```

questo comando (da digitarsi tutto sulla medesima riga) è quello che genera il certificato X509 vero e proprio. La direttiva `–req` informa OpenSSL che quanto segue `–in` è il nome del file contente la richiesta di certificato e quanto segue `-out` il nome del file in cui salvare il certificato creato.
`-signkey` è il parametro da usare in caso di "autofirma" ed è seguito dal nome del file contente la chiave privata da usare.

```
openssl x509 –req –in <richiesta certificato> -out <certificato> -CA
        <certificato CA> -CAkey <chiave privata CA> -CAcreateserial –
        extensions v3_usr
```

si comporta come il comando precedente ma invece che generare un certificato self-signed permette di specificare la Certification Authority firmataria (tramite i parametri `–CA` e `–CAkey` se ne indicano il certificato e la chiave privata). Quando le CA firmano un certificato utilizzano un numero seriale: questo numero viene conservato in un file e viene incrementato di volta in volta. Tale file può essere specificato tramite il parametro `–Caserial` *<nome file>*. In caso il file non esista ancora (per esempio in caso di prima procedura di firma) si utilizza `–CAcreateserial`. Il parametro `–extensions v3_usr` indica che il certificato generato non rappresenterà un'ulteriore CA (vedi prossimo paragrafo).

```
openssl x509 –in <certificato> -out <certificato DER> -outform DER
```

converte il certificato da codifica DER+Base64 a codifica DER.

Tutti i comandi sopra riportati generano infatti di default certificati, richieste di certificato, chiavi private in codifica DER+Base64.
I certificati saranno quindi costituiti solo da quella parte che all'inizio del capitolo (a proposito dei file *.pem) è stata definita "l'ultima sezione del certificato": non leggibili direttamente ma sicuramente più compatti.

È inoltre comodo ricordare che molte istruzioni OpenSSL, in assenza di direttive sulla linea di comando, utilizzano parametri di default estrapolati dal file di configurazione sopra citato (/usr/share/ssl/openssl.cnf).

Certification Authority di secondo livello

Le Certification Authority di secondo livello (e successivi) sono CA caratterizzate da certificati non self-signed, bensì firmati da altre CA (che vengono dette di livello superiore). Nell'ottica delle dipendenze di validazione quindi una Autorità di Certificazione "non root" viene vista come "un'utente" dell'Autorità di Certificazione immediatamente superiore. Questo ha il vantaggio di rendere omogeneo il trattamento dei vari certificati, ma pone il problema di distinguere un utente finale da una CA non root (altrimenti un utente comune potrebbe a sua discrezione diventare un'Autorità di Certificazione di livello "n" forte delle vere CA alle spalle che lo hanno riconosciuto). I certificati contengono quindi opportuni parametri che permettono di discriminare i due casi. La direttiva `-extensions v3_usr` vista poche righe sopra serve proprio per generare un certificato per un utente finale; se ne avessimo voluto uno per una CA avremo dovuto utilizzare il parametro `-extensions v3_ca`; il comando sarebbe quindi diventato:

```
openssl x509 -req -in <richiesta certificato> -out <certificato> -CA
        <certificato CA> -CAkey <chiave privata CA> -CAcreateserial -
        extensions v3_ca
```

Nel caso si voglia utilizzare la procedura di Nate Carlson (www.natecarlson.com/linux/ipsec-x509.php) per creare CA di secondo livello bisogna adottare la seguente procedura:

- Modificare il parametro `x509_extensions = usr_cert` nella sezione `[ CA_default ]` del file openssl.cnf in `x509_extensions = v3_ca`.
- Generare una nuova coppia certificato-chiave privata (saranno quelli della Certification Authority di secondo livello).
- In una **nuova directory** iniziare tramite il comando `/usr/share/ssl/misc/CA -newca` la realizzazione della CA di secondo livello. Alla prima domanda invece che premere "invio" per creare la Certification Authority ex-novo, digitare il nome del file contenente la chiave privata ottenuta il passo precedente.
- Una volta ritornati al prompt copiare il certificato (quello già generato) dentro la cartella demoCA/ creata durante il passo precedente (l'esigenza di questa copia manuale è probabilmente dovuta a un bug della versione utilizzata dello script `/usr/share/ssl/misc/CA` visto che l'operazione poteva essere accorpata con la copia della chiave privata, come del resto la domanda posta dallo script sembra suggerire).
- A questo punto è la CA di secondo livello è pronta all'uso. È importante ricordare di ripristinare il valore di `x509_extensions` (vedi primo passo) e tenere presente che d'ora in poi i certificati creati col comando `/usr/share/ssl/misc/CA` saranno firmati dalla CA di primo o secondo livello a seconda della directory in cui ci si trova al momento dell'operazione.

IPSec
sicurezza per il network layer

Le VPN e non solo

Quelle che comunemente sono definite VPN (Virtual Private Network) nascono dall'idea di utilizzare Internet come dorsale di collegamento tra sezioni distaccate (anche su distanze geografiche) di una medesima rete locale. Si tratta dunque di garantire sulla tratta di rete pubblica attraversata, standard di riservatezza, autenticità e integrità dei dati paragonabili a quella di una LAN.
Ma a che livello dello stack protocollare introdurre queste potenzialità? Ovviamente in un layer sufficientemente alto da non risentire dell'eterogeneità della Rete, ma anche ragionevolemente basso per assicurare la trasparenza del maggior numero di protocolli preesistenti. Ovviamente la scelta è caduta su quello che da sempre è il massimo comun divisore di Internet e della maggior parte delle Intranet: il protocollo IP.
Una volta concepita l'idea dell'estensione del network layer verso una maggiore sicurezza ci si è trovati in mano uno strumento dalle possibilità superiori alle necessità per cui era stato creato: perché limitarsi ai collegamenti punto-punto statici delle VPN e non permettere, per esempio, il collegamento alla LAN aziendale del portatile di un rappresentante (ovviamente una volta certificatane l'identità) qualsiasi sia il provider utilizzato per l'accesso a Internet?

Le soluzioni proprietarie, IPSec e l'implementazione di FreeS/WAN

A onor del vero già da molto tempo è possibile realizzare VPN. I produttori di apparati di rete quali router, gateway, firewall da sempre propongono le loro soluzioni proprietarie: le prestazioni sono di buon livello visto che generalmente si utilizza hardware dedicato (cifrare tutti i pacchetti in transito non è certo un compito "leggero") ma ovviamente l'interoperabilità è volutamente limitata ai prodotti del vendor, rendendo lo scenario del "portatile del rappresentante" alquanto problematico e i costi di manutenzione difficilmente controllabili.

IPSec è invece una "creatura" IETF (www.ietf.org), il che significa uno standard di pubblico dominio, aperto (quantomeno in fase di definizione preliminare) a suggerimenti, pensato non per tutelare interessi commerciali ma per garantire la maggiore flessibilità possibile nell'ambito delle specifiche di progetto: è quindi normale che sia diventato uno degli standard de facto in questo ambito.
Però non è tutto oro ciò che luccica: siamo abituati a parlare di IPSec in termini di semplice protocollo, in realtà si tratta di una complessa suite che si deve integrare nell'intero stack protocollare, e non sempre questo è possibile in maniera indolore. Solo alcune indicazioni di massima per dare un'idea:

- L'operazione di cifra dei dati può avvenire con due protocolli diversi: **AH** se ci si concentra sugli aspetti di autenticazione, **ESP** per la secretazione (questa "separazione" è necessaria per permettere l'utilizzo di IPSec in paesi in cui la legge permetta solo l'autenticazione).
- Il trasporto dei dati cifrati avviene tramite una processo di incapsulamento che può conservare gli indirizzi IP di sorgente e destinazione originali (**Transport Mode**) o sostituirli con quelli dei gateway IPSec (**Tunnel Mode**).
- La gestione delle chiavi è affidata a un protocollo specifico, **IKE** che racchiude in sé le idee di altri protocolli analoghi quali ISAKMP, Oakley, SKEME permettendo la negoziazione degli algoritmi di cifra, lo scambio di chiavi, la gestione dei vari parametri e timeout di sessione.
- Per loro natura le modifiche apportate ai pacchetti da IPSec mal si sposano con firewall e NAT box non opportunamente configurate.

A verifica di quanto detto si può constatare il numero dei documenti IETF relativi a questi argomenti (una lista aggiornata è reperibile su www.ietf.org/html.charters/ipsec-charter.html):

- RFC 2406 - IP Encapsulating Security Payload (ESP)
- RFC 2402 - IP Authentication Header
- RFC 2401 - Security Architecture for the Internet Protocol
- RFC 2409 - The Internet Key Exchange (IKE)
- RFC 2412 - The OAKLEY Key Determination Protocol
- RFC 2408 - Internet Security Association and Key Management Protocol (ISAKMP)
- RFC 2407 - The Internet IP Security Domain of Interpretation for ISAKMP
- ….

per non citare i molti relativi all'utilizzo dei vari algoritmi crittografici.

Tale livello di complessità ha portato alla nascita di diverse implementazioni di IPSec, purtroppo non tutte compatibili tra loro.
Durante l'attività di laboratorio si è utilizzato FreeS/WAN, un pacchetto per Linux reperibile sul sito www.freeswan.ca anche in formato .rpm e quindi di semplice installazione in ambiente RedHat. La versione utilizzata è la 1.99 patchata per permetterne il funzionamento con i certificati X509 (è disponibile anche una versione che utilizza quelli di PGP) scaricabile dall'URL download.freeswan.ca/freeswan-x509/RedHat-RPMs/; la scelta dei file da scaricare dipende dalla versione del kernel utilizzata, vista la necessaria interazione con lo stack protocollare che sotto Linux è implementato proprio nei sorgenti del kernel.

FreeS/WAN è costituito da tre componenti principali:

- **Pluto**: un daemon IKE che risponde sulla porta UDP 500 e che serve per stabilire la connessione sicura negoziando i parametri di collegamento e scambiando le chiavi di cifratura.
- **KLIPS** (KerneL IPSec): il componente in kernel-space che implementa i protocolli ESP (IP header Protocol ID 50) e AH (protocol ID 51) e gestisce le operazioni di cifratura e decifrazione utilizzando le chiavi concordate dalle parti in gioco tramite IKE. I futuri kernel della serie 2.6 implementeranno queste funzionalità nativamente, senza bisogno di add-on.
- Un insieme di facility per la configurazione (vedremo in dettaglio il file ipsec.conf) e la gestione di IPSec: sotto RedHat Linux vengono, per esempio, forniti gli script per l'avvio, l'arresto, il reset del servizio nei vari runlevel di funzionamento.

Questo in sintesi lo scenario in cui andremo ad operare. Per un'introduzione meno stringata è possibile consultare gli articoli scritti da Marco Ivaldi pubblicati sui numeri 19 e 21 di "*Linux & C.*", in cui si trova anche una definizione formale del ricorrente concetto di "Security Association".

FreeS/WAN Internals

Si è più volte detto che per sua natura IPSec interagisce pesantemente con la gestione della rete del sistema operativo. Vediamo dunque come KLIPS si integra nel kernel di Linux, nello specifico come agisce sui pacchetti in transito di propria competenza.

Pacchetti in partenza

Per rendere possibile l'invio di pacchetti IPSec viene creata l'interfaccia virtuale `ipsec0` (ulteriori interfacce si chiamerebbero `ipsec1`, `ipsec2` e così via) e inserita una entry nella tabella di routing che associa all'IP di destinazione (quello dell'altro estremo IPSec) l'instradamento attraverso l'interfaccia appena creata.
KLIPS riconosce i pacchetti di sua competenza proprio perché inviati sull'interfaccia virtuale, li incapsula come da configurazione e li instrada attraverso l'opportuna scheda di rete.
Unica eccezione il traffico IKE (porta UDP 500) che, essendo il protocollo di controllo di IPSec, non viene incapsulato per evitare il problema "dell'uovo e della gallina".

```
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
192.168.10.2    *               255.255.255.255 U     0      0        0 ipsec0
192.168.1.0     *               255.255.255.0   U     0      0        0 eth0
192.168.10.0    *               255.255.255.0   U     0      0        0 eth1
127.0.0.0       *               255.0.0.0       U     0      0        0 lo
default         192.168.1.1     0.0.0.0         UG    0      0        0 eth0
```

Figura 5: Routing table di augustus: VPN con cleopatra

Questa è la procedura seguita sia che si stia creando una VPN statica, sia che si stia instaurando un tunnel dinamico (quella che comunemente viene chiamata configurazione Road-Warrior): in questo secondo caso ci si ritrova quindi ad avere un aggiornamento dinamico della tabella di routing basato sulle necessità dettate dagli handshake di IKE.

Pacchetti in arrivo

La gestione dei pacchetti in arrivo su una macchina FreeS/WAN segue lo schema funzionale qua riportato:

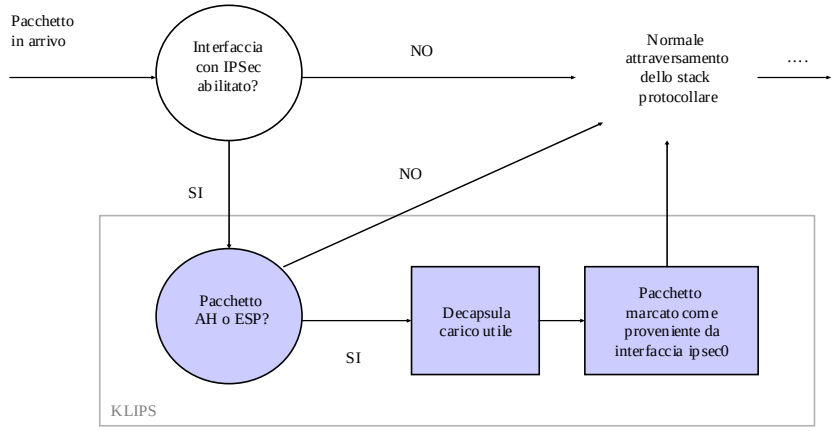


Figura 6: Arrivo pacchetti con IPSec

KLIPS processa solo i pacchetti AH o ESP in entrata attraverso le interfacce su cui FreeS/WAN è configurato: i pacchetti decapsulati vengono reinseriti nello stack protocollare come tutti gli altri, ma risultano provenienti da una interfaccia virtuale, per esempio `ipsec0`. Questo è utile per il riconoscimento, anche in fasi successive, dei pacchetti provenienti dalla connessione IPSec, ad esempio per permette al sottosistema di firewalling di Linux (`iptables`) di accettare solo dati provenienti da tunnel cifrati.

L'utilizzo delle interfacce virtuali impone di disabilitare la funzionalità di *Reverse Path Filtering* (`rp_filter`) di Linux. Si tratta di un meccanismo anti-spoofing: un pacchetto in arrivo viene ignorato se l'interfaccia di entrata non risulta essere anche quella di uscita per l'eventuale pacchetto di risposta (in pratica impedisce il routing asimmetrico). Le connessioni IPSec vengono ovviamente bloccate: abbiamo infatti visto che "escono" attraverso le interfacce virtuali, per es. `ipsec0`, ma non possono che "entrare" da un'interfaccia fisica, per es. `eth0` (è vero che i pacchetti decapsulati vengono marcati come provenienti dall'interfaccia virtuale, ma `rp_filter` interviene prima).

In ambiente RedHat per disabilitare completamente `rp_filter` è necessario impostare a 0 l'omonima voce nel file /etc/sysctl.conf; altrimenti su qualsiasi distribuzione si può intervenire direttamente sui parametri del kernel accessibili attraverso la gerarchia /proc (tipicamente /proc/sys/net/ipv4/conf/*interfaccia*/rp_filter), magari per disabilitare la funzionalità solo su specifiche interfacce. Una interessante fonte di informazioni in proposito è il "*Linux Advanced Routing & Traffic Control HOWTO*", documento scaricabile dal sito del Linux Documentation Project (www.tldp.org), anche in formato PDF (www.ibiblio.org/pub/Linux/docs/HOWTO/other-formats/pdf/Adv-Routing-HOWTO.pdf).

La configurazione di FreeS/WAN

L'attività di laboratorio relativa a FreeS/WAN ha coinvolto tre computer: **augustus**, **cleopatra**, **aurelius** (vedi paragrafo sulla LAN utilizzata).

Inizialmente si è creata una VPN tra **augustus** e **cleopatra**, utilizzando prima l'autenticazione tramite preshared secret (leggi a chiave simmetrica), poi quella a chiave asimmetrica. Per la configurazione si sono seguite passo-passo le indicazioni contenute nel capitolo 25 di "*Securing and Optimizing Linux Red Hat Edition – A Hands on Guide ver. 1.3*" (www.tldp.org/LDP/solrhe/Securizing-Optimizing-Linux-RH-Edition-v1.3/fSWAn.html). L'intera guida è ora disponibile anche in una versione più recente dal nome "*Securing & Optimizing Linux: The Ultimate Solution ver. 2.0*" (relativamente a FreeS/WAN non sono però state introdotte nuove informazioni significative): entrambe sono reperibili in diversi formati, insieme ad altri documenti LDP, su www.tldp.org/guides.html.
Per evitare di "reinventare l'acqua calda" si riporta solo un promemoria sul ruolo e sulla struttura dei file di configurazione; quelli di nostro interesse sono due ed entrambi contenuti nella directory /etc/: ipsec.secrets e ipsec.conf:

- **ipsec.secrets** contiene il preshared secret oppure la chiave privata (ovviamente quella del terminale IPSec cui appartiene!) e quindi sarà opportuno impostare i criteri di protezione sul file in maniera "conservativa".
- **ipsec.conf** serve per impostare i parametri di funzionamento delle connessioni protette: è diviso in una sezione generale (che comprende, per esempio, la scelta delle interfacce da utilizzare e delle modalità di funzionamento dei vari daemon) e in sezioni specifiche per ciascuna connessione si voglia instaurare, con indicazioni sugli indirizzi IP coinvolti e le modalità di autenticazione scelte (in caso di chiave asimmetrica, quelle pubblica è riportata in questa sezione). È anche presente una sezione relativa alla "connessione di default" che contiene i parametri validi per TUTTE le connessioni.

```
# Parametri di funzionamento generali
config setup
…
```

```
# Connessione “di default”
conn %default
…
```

```
# Connesione
conn ConnessioneCleopatra
…
```

```
# Connessione
conn ConnessioneAurelius
…
```

```
…
```

Figura 7: Sezioni del file ipsec.conf

Nelle sezioni relative alle connessioni, gli indirizzi IP dei terminali IPSec sono indicati dalle keyword `right` e `left`. Sono anche presenti informazioni utilizzate per il routing dei pacchetti: `rightsubnet` e `rightnexthop` indicano rispettivamente la rete dietro il terminale di “destra” e il primo router che i pacchetti provenienti dal terminale di “destra” dovranno attraversare per arrivare al terminale di “sinistra” (vedi la *Figura 8* che riporta un esempio di file ipsec.conf, anche se in una configurazione più complessa di cui parleremo nel prossimo paragrafo).
Si noti che si è parlato di terminale di “destra” e di “sinistra” senza distinguerli specificatamente: questo riflette il ruolo paritario dei computer durante la connessione e quindi la struttura simmetrica del file ipsec.conf: è possibile scriverlo su un computer e poi copiarlo sull’altro, modificando solo eventuali dettagli (ovvero l’interfaccia di funzionamento e la chiave pubblica se presente). Sarà poi compito del motore IPSec, che gira su ciascun terminale, capire se le informazioni di propria competenza sono quelle “di destra” o quelle “di sinistra”: una sorta di meccanismo plug and play per l’interpretazione del file di configurazione che permette all’amministratore di rete di concentrarsi sul canale protetto piuttosto che sui ruoli delle due estremità.

Utilizzo dei Certificati X509

Una volta presa confidenza con le specificità di FreeS/WAN si è passati a una modalità di funzionamento un poco più raffinata: si è reso **augustus** un IPSec gateway che accetta connessioni protette da più terminali IPSec, nel nostro caso **cleopatra** e **aurelius**, utilizzando autenticazione tramite certificati X509. Questo rende possibile lo scenario tratteggiato all’inizio del capitolo: quello di un portatile che si può collegare alla rete aziendale indifferentemente dal suo punto di accesso a Internet: è la cosiddetta configurazione a Road-Warrior.
In questo caso il documento di riferimento è il già citato tutorial di Nate Carlson (www.natecarlson.com/linux/ipsec-x509.php), raggiungibile anche dal sito ufficiale di FreeS/WAN attraverso la sezione “support” (www.freeswan.ca/docs/freeswan-1.99/doc/interop.html).
Si devono generare i certificati con OpenSSL e mettere in grado FreeS/WAN (opportunamente patchato) di utilizzarli. Questo comporta:

- copiare i certificati nelle cartelle opportune all’interno della gerarchia /etc/ipsec.d/.
- impostare nel file ipsec.conf l’utilizzo dei certificati piuttosto che della chiave pubblica.
- indicare in ipsec.secrets la password con cui è protetta la chiave privata.

È interessante osservare come il computer che inizia la connessione abbia bisogno anche del certificato dell'altra estremità: nel documento di Nate Carlson questo fatto forse non viene sufficientemente sottolineato, ma in effetti alla prova pratica si è rivelato indispensabile. Ciò rappresenta probabilmente una particolarità di FreeS/WAN (si ricordi che all'inizio del capitolo si accennava a come le varie implementazioni di IPSec possano differire in maniera anche rilevante tra loro).
Vediamo il file ipsec.conf scritto per **augustus**:

```
# /etc/ipsec.conf - FreeS/WAN IPsec configuration file – augustus IPSec Gateway

# More elaborate and more varied sample configurations can be found
# in FreeS/WAN's doc/examples file, and in the HTML documentation.


# basic configuration
config setup
       # THIS SETTING MUST BE CORRECT or almost nothing will work;
       # %defaultroute is okay for most simple cases.
       interfaces="ipsec0=eth0 ipsec1=eth1"
       # Debug-logging controls:  "none" for (almost) none, "all" for lots.
       klipsdebug=none
       plutodebug=none
       # Use auto= parameters in conn descriptions to control startup actions.
       plutoload=%search
       plutostart=%search
       # Close down old connection when new one using same ID shows up.
       uniqueids=yes


# defaults for subsequent connection descriptions
# (these defaults will soon go away)
conn %default
       keyingtries=0
       disablearrivalcheck=no
       authby=rsasig
       leftrsasigkey=%cert
       rightrsasigkey=%cert


# TLC Lab VPN connections

# cleopatra
conn nearoadwarriors
       # Left security client
       left=%any
       # Right security gateway, subnet behind it, next hop toward left.
       right=192.168.10.1
       rightsubnet=0.0.0.0/0
       rightnexthop=%direct
       rightcert=augustus.pem
       auto=add

# aurelius
conn roadwarriors
       # Left security client
       left=%any
       # Right security gateway, subnet behind it, next hop toward left.
       right=192.168.1.2
       rightsubnet=0.0.0.0/0
       rightnexthop=192.168.1.1
       rightcert=augustus.pem
       auto=add
```

Figura 8: Road-Warrior: il file ipsec.conf di augustus

Non è un caso che esistano due sezioni per le connessioni: `nearoadwarriors` e `roadwarriors`. La prima serve a configurare "connessioni vicine", dove con "vicine" si intendono computer direttamente collegati ad **augustus** (nessun router intermedio); la seconda è invece relativa a computer raggiungibili attraverso il router con IP 192.168.1.1 (vedi parametro `rightnexthop`).
Nel nostro caso come gateway verso **aurelius** è stato indicato **titus** (192.168.1.1) piuttosto che **cicero** (192.168.1.3) per poter eventualmente sfruttare queste impostazioni anche per altre connessioni, ad esempio con **nero** (vedi La LAN utilizzata).

L'unico parametro che cambia significativamente è dunque `rightnexthop` (porlo uguale a `%direct` significa che non ci sono router intermedi e quindi il nexthop è proprio l'IP dell'altra estremità IPSec); `right` è diverso nei due casi solo perché diverse sono le interfacce che si affacciano verso **cleopatra** e verso **titus**.

Si è insomma dovuto ovviare a quello che forse è il maggiore limite di FreeS/WAN, almeno per quanto riguarda la versione utilizzata: il fatto che le informazioni di routing per i pacchetti in uscita debbano essere fornite esplicitamente costringe ad avere nel file ipsec.conf una entry `conn` per ciascun router direttamente connesso alla macchina che stiamo configurando. Situazione che forse è solo noiosa per realtà limitate, ma difficilmente accettabile per situazioni più complesse e dinamiche come la parte "core" di Internet.

Per completezza si riporta anche il file ipsec.conf di **cleopatra** (quello di **aurelius** è analogo):

```
# /etc/ipsec.conf - FreeS/WAN IPsec configuration file - cleopatra

# More elaborate and more varied sample configurations can be found
# in FreeS/WAN's doc/examples file, and in the HTML documentation.



# basic configuration
config setup
       # THIS SETTING MUST BE CORRECT or almost nothing will work;
       # %defaultroute is okay for most simple cases.
       interfaces="ipsec0=eth0"
       # Debug-logging controls:  "none" for (almost) none, "all" for lots.
       klipsdebug=none
       plutodebug=none
       # Use auto= parameters in conn descriptions to control startup actions.
       plutoload=%search
       plutostart=%search
       # Close down old connection when new one using same ID shows up.
       uniqueids=yes



# defaults for subsequent connection descriptions
# (these defaults will soon go away)
conn %default
       keyingtries=0
       disablearrivalcheck=no
       authby=rsasig
       leftrsasigkey=%cert
       rightrsasigkey=%cert



# TLC Lab  VPN connections
conn cleopatraConn
       # Left security gateway, next hop toward right.
       left=192.168.10.2
       leftnexthop=192.168.10.1
       leftcert=cleopatra.pem
       # Right security gateway, subnet behind it.
       right=192.168.10.1
       rightsubnet=0.0.0.0/0
       rightcert=augustus.pem
       auto=start
```

Figura 9: Road-Warrior: il file ipsec.conf di cleopatra

A differenza di quanto detto precedentemente si è scelto di non avere file simili per le due estremità, prediligendo invece la semplificazione introdotta eliminando alcune voci non necessarie: il parametro `leftnexthop`, per esempio, manca nella configurazione di **augustus** ma è presente in quella di **cleopatra** perchè insieme a `rightsubnet` viene utilizzato da FreeS/WAN per creare la entry nella tabella di routing che governerà l'instradamento dei pacchetti in uscita.
Si noti anche il parametro `auto`, posto a `start` solo per il computer che avvia la connessione.

HTTPS
la suite SSL/TLS per il Web

Il web sicuro: https

Ideata da Netscape (SSL – Secure Sockets Layer), standardizzata da IETF (TLS – Transport Layer Security, RFC 2246), la suite SSL/TLS è di fatto costituita da librerie che si posizionano appena sopra lo strato di trasporto. Le differenze tra le versioni più recenti di SSL e la standardizzazione TLS sono molto poche: ricercando con Google nei gruppi di discussione la stringa `"ssl vs tls"` si rinvengono molti documenti in cui si spiega che di fatto TLS è talmente simile a SSL v3 da potersi considerare una sorta di SSL v3.1.
Non a caso si è parlato di librerie e non di strato, visto che anche l'attuale implementazione "sconta" il ruolo della versione originale, concepita come parte integrante di Netscape Navigator/Communicator per poter garantire una comunicazione web sicura: un'applicazione che vuole utilizzare SSL/TLS deve in qualche modo implementarne gli algoritmi all'interno del proprio codice (al più **richiamando** delle librerie esterne); non esiste infatti un meccanismo trasparente alle applicazioni analogo ai servizi forniti da IPSec.
Ancora oggi il web è il fondamentale ambito di utilizzo di SSL/TLS: quando i browser utilizzano https, in realtà lavorano con una connessione http confidenziale e autenticata; arrivati a questo capitolo dovrebbe essere semplice indovinare che tipicamente la confidenzialità è ottenuta mediante l'utilizzo di algoritmi crittografici e l'autenticazione tramite l'utilizzo di certificati.
Vediamo dunque l'implementazione di SSL/TLS offerta da Apache (www.apache.org), il server web più diffuso in ambito Linux, e come le configurazioni si riflettono sul funzionamento dei browser (Opera e Internet Explorer).

Configurare Apache

Il server Apache è stato installato sull'host **caesar** (vedi paragrafo sulla LAN utilizzata). La versione utilizzata (come del resto la quasi totalità di quelle recenti) è già compilata per l'utilizzo di mod_ssl, il componente che si occupa di SSL/TLS. La configurazione si riduce quindi alla modifica dei parametri di interesse contenuti all'interno del file di configurazione standard /etc/httpd/conf/httpd.conf (a differenza del nostro caso, la documentazione consultata prevede in alternativa l'utilizzo di un file ad hoc: ssl.conf).

Per comprendere che cosa si va a modificare è opportuno avere un'idea di come il file httpd.conf è organizzato:

```
### Section 1: Global Environment
…
```

```
### Section 2: 'Main' server configuration
…
```

```
### Section 3: Virtual Hosts
…
```

```
## SSL Global Context
…
```

Figura 10: Sezioni del file httpd.conf

La **sezione 1** definisce parametri generali del web server come la posizione dei file di log, il numero di server da creare, il numero massimo di connessioni accettate, vari timeout, la posizione di vari moduli che implementano le funzionalità più disparate.

La **sezione 2** e la **sezione 3** sono quelle che specificano i parametri relativi ai vari siti web gestiti dal server: nome del dominio gestito, directory contenente le pagine .html, posizione di log specifici, vari permessi di accesso. Le sezioni sono due per motivi "storici", infatti inizialmente Apache poteva gestire un unico dominio, da cui un'unica sezione (quella che ora prende il nome di Main Server Configuration). Nelle versioni successive sono comparsi i cosiddetti domini virtuali (in pratica un meccanismo di multiplexing a livello di protocollo http per poter gestire più domini con un unico server) e quindi anche la sezione relativa.
A livello di curiosità è interessante sapere che Apache può operare il multiplexing dei siti ospitati anche a livello IP (potenzialità però quasi mai utilizzata per risparmiare indirizzi).

L'**ultima sezione** è ovviamente quella di maggior interesse per SSL/TLS. All'interno della direttiva "`<VirtualHost _default_:443>`" contiene le impostazioni predefinite, ereditabili dai virtualhost nei quali non siano state ridefinite (si segnala che l'indicazione della porta 443 nella direttiva `VirtualHost` deve essere coerente con quanto indicato nella direttiva `Listen` nelle sezioni precedenti).
Questa sezione contiene quell'insieme di parametri che prima si dicevano scorporabili nel file separato ssl.conf.

Molte sono le fonti che forniscono informazioni specifiche per la configurazione. A tale scopo si possono ricordare:

- il già citato articolo di Riccardo Corsanici *"Apache: sicurezza e crittografia"* (DEV numero 102, Dicembre 2002) per quanto riguarda un excursus sulle tematiche di sicurezza in Apache.
- la documentazione ufficiale del server http (httpd.apache.org/docs/) per quanto riguarda la creazione e la gestione dei domini (virtuali e non).
- la documentazione sul sito ufficiale di mod_ssl (www.modssl.org/docs/), il modulo SSL/TLS di Apache 1.3 e 2.0.
- i commenti presenti all'interno del file httpd.conf.

Cercheremo adesso di capire come poter configurare Apache per utilizzare i nostri certificati.

Certificati server e certificati personali

A differenza di quelli per IPSec, i certificati che si utilizzano per una sessione https sono diversi a seconda della parte in gioco che stiamo considerando.

Quando si parla di **Server Certificate** ovviamente ci si riferisce a quelli (i più comuni) che certificano il server web. Tipicamente i browser quando li ricevono effettuano tre verifiche:

- ne controllano il periodo di validità (data di inizio e di scadenza)
- riconoscono la Certification Authority grazie a una lista di CA attendibili di cui sono forniti a priori (lista comunque aggiornabile)
- verificano che il certificato sia stato emesso proprio per il dominio richiesto nell'URL

Un riscontro negativo su uno qualsiasi di questi tre punti comporta una segnalazione di allerta all'utente in modo che possa coscientemente decidere di continuare o abortire la sessione.
Può essere utile sapere che, per poter attuare la terza verifica, lo standard vuole che il nome del dominio sia contenuto nella sezione data, campo Common Name (quello che normalmente contiene il nome del proprietario del certificato, vedi il già citato tutorial di Nate Carlson).
L'utilizzo del Server Certificate è sufficiente anche per garantire la secretazione del flusso dati.

Se le esigenze di sicurezza sono maggiori si può integrare un Server Certificate con un **Client Certificate** (o **Personal Certificate**). Tipicamente servono per autenticare gli utilizzatori del sito web in maniera "più forte" rispetto a quanto possibile, per esempio, attraverso form http o il sistema di autenticazione standard di Apache via htpasswd (vedi articolo di Riccardo Corsanici): quindi di solito identificano una persona fisica e non un servizio come i precedenti.

A differenza del server dove la dispersione delle impostazioni di configurazione è tollerata (essendo oltretutto un po' la filosofia dei sistemi Unix-like), quando si parla di macchine client è opportuno che tutto ciò che serve a una particolare facility sia contenuto in un unico file. Ecco quindi che se per Apache troveremo un directory per i certificati, una per le chiavi private e così via, per i client avremo la necessità di fornire il certificato personale e la chiave privata relativa in un unico "pacchetto chiavi in mano". Questo sarà possibile utilizzando uno standard definito dagli RSA Laboratories, il PKCS12 (www.rsasecurity.com/rsalabs/pkcs/).

Modificare httpd.conf

Vediamo i parametri del file /etc/httpd/conf/httpd.conf , sezione "SSL Global Context" di nostro interesse:

```
SSLCertificateFile <nome file certificato>

SSLCertificateKeyFile <nome file chiave privata>
```

specificano i percorsi dei file che contengono il Server Certificate e la chiave privata relativa. Tipicamente tali file sono contenuti in /etc/httpd/conf/ssl.crt/ e ssl.key/. Devono essere codificati in Base64 (l'estensione non deve necessariamente essere .pem). Se la chiave privata è protetta da pass-phrase, questa verrà richiesta ad ogni avvio/riavvio del daemon (ecco quindi motivati i comandi OpenSSL visti in merito nel capitolo precedente).

`SSLCertificateChainFile` *<file catena di validazione certificato>*

specifica il file che contiene la catena delle dipendenze di certificazione per il server web. Quindi se per esempio il certificato del nostro server è garantito dalla CA "Pippo" che a sua volta è garantita dalla CA "Pluto" la quale è garantita da Verisign, allora il file indicato con questa direttiva tipicamente conterrà:

- Il certificato della CA Pippo firmato dalla CA Pluto.
- Il certificato della CA Pluto firmato da Verisign.
- Il certificato autofirmato di Verisign (autofirmato perché costituisce la "radice" della catena di validazione, come spiegato a suo tempo).

I certificati concatenati devono essere codificati in Base64. Come spesso succede per tale codifica, la concatenazione avviene con una semplice comando di appending dei singoli certificati (visto che sono delimitati dalle direttive `BEGIN CERTIFICATE` e `END CERTIFICATE`).
Il file indicato può essere lo stesso utilizzato anche nella direttiva `SSLCertificateFile`: in tal caso la catena dei certificati viene "appesa" al certificato del server.
In entrambi i casi tutta la catena verrà passata al browser per agevolare la verifica del certificato.

`SSLVerifyClient` *<tipo di autenticazione>*

stabilisce il tipo di autenticazione da applicarsi al client. Il parametro *<tipo di autenticazione>* può assumere i seguenti valori:

- **none**: non è richiesto un certificato personale.
- **optional**: il client PUÒ presentare un certificato personale.
- **require**: il client DEVE presentare un certificato personale.
- **optional_no_ca**: il client PUÒ presentare un certificato personale ma questo NON deve essere positivamente verificabile dal server: la CA di più alto livello di cui eventualmente viene fornito il certificato self-signed deve cioè essere sconosciuta al server.

Ovviamente i casi di interesse concreto sono **none** e **require**.

`SSLVerifyDepth` *<profondità di verifica catena di certificati>*

come avviene per i certificati server (vedi `SSLCertificateChainFile`), anche i client certificate possono fornire la loro catena di validazione. Il parametro *<profondità di verifica catena di certificati>* fissa il numero di CA (oltre al certificato vero e proprio) che si possono "risalire" prima che il certificato venga considerato non valido a priori (di fatto si limita la lunghezza della catena di validazione accettabile).

Es.:
`SSLVerifyDepth 0` accetterà solo certificati self-signed (quelli delle CA).
`SSLVerifyDepth 1` accetterà anche certificati che dipendono da una CA.
`SSLVerifyDepth 2` accetterà i certificati che dipendono da un massimo di due CA in cascata.

`SSLCACertificateFile` *<file lista certificate CA riconosciute>*

contiene la lista dei certificati (in Base64) autofirmati delle Certification Authority riconosciute per la validazione dei certificati personali dei client. Nel momento in cui il server richiede un certificato personale, questa lista viene passata al browser in modo che all'utente sia fornita la possibilità, attraverso dialog box, di identificarsi con un certificato firmato da una CA riconosciuta dal server (l'utente potrebbe infatti avere più certificati personali).

```
SSLCARevocationFile <file certificati revocati>
```

Specifica il file contente le liste (ovviamente in Base64!!) dei certificati revocati. Quest'ultima direttiva, come anche la precedente, è sostituibile da una analoga che permette di specificare una directory contentente le varie liste separate, piuttosto che un unico file con tutte le liste concatenate tra loro.

Creare un Personal Certificate

Per creare un Client Certificate il primo passo è ovviamente utilizzare OpenSSL per ottenere un certificato "classico". Nulla è diverso rispetto a quanto ormai visto più volte, bisogna però tenere presente che Internet Explorer volutamente non supporta chiavi private RSA più lunghe di 512 bit, a seguito delle leggi americane sull'esportazione del software crittografico (si noti che questa limitazione non si applica ai Server Certificate perché in quel caso il browser non ha alcuna competenza sulla gestione e l'utilizzo della chiave privata). Per forzare OpenSSL in tal senso durante l'uso dello script /usr/share/ssl/misc/CA bisogna modificare il parametro `default_bits` del file /usr/share/ssl/openssl.cnf.
Per creare il file in formato PKCS12 contenente (almeno) il personal certificate e la chiave privata relativa si usa ancora una volta OpenSSL:

```
openssl pkcs12 –export –in <file certificato> -inkey <chiave privata> -out
        <nome file.p12>
```

dove `-export` indica che si vuole effettivamente generare il file in formato PKCS12 e chiamarlo col nome che segue il parametro `–out`: tipicamente le estensioni per questo formato sono .P12 o .PFX.
Ovviamente se è stata utilizzata una pass-phrase, questa verrà richiesta per poter effettuare la lettura della chiave privata. La procedura inoltre prevede l'immissione di una password di "export" con cui la chiave privata viene protetta all'interno del nuovo "file contenitore" PKCS12 e che verrà richiesta dal browser durante l'installazione del Personal Certificate; i browser potranno poi porre all'utente ulteriori domande che influenzeranno il grado di sicurezza con cui il certificato personale viene gestito (tali domande sono non solo *browser* ma anche *version dependent* e quindi non sono state oggetto di analisi).

Le direttive a disposizione sono ovviamente molte di più di quelle viste: tramite OpenSSL è per esempio possibile includere tutta la catena di certificazione fino alla root-ca o anche inserire campi che rendono più user-friendly la gestione del certificato all'interno del browser. Per una casistica completa si rimanda alla documentazione di OpenSSL (www.openssl.org/docs/) e del formato PKCS12 (www.rsasecurity.com/rsalabs/pkcs/pkcs-12/index.html); una interessante fonte di informazioni è il sito di Stephen Norman Henson (www.drh-consultancy.demon.co.uk/pkcs12faq.html) dove è possibile trovare anche indicazioni specifiche per i vari browser.

Le "reazioni" dei browser

Viste quali sono le possibilità di configurazione in Apache per quanto riguarda i certificati, si sono verificate le "reazioni" di un paio di browser a varie modalità di configurazione dei **Server Certificate**. In ambiente Windows è stato preso in considerazione Internet Explorer, sotto Linux invece la scelta è caduta su Opera. Si è utilizzato il certificato self-signed fornito di default con l'installazione di Apache e poi uno creato e firmato dalla nostra CA, con e senza catena di certificazione a monte.

Tipo di Certificato	Linux + Opera	Windows + Internet Explorer
Self-Signed (Apache default)	Il certificato (di root perché self-signed) non viene riconosciuto. Il browser offre la possibilità di proseguire, di generare un warning tutte le volte in cui si incontra questa CA, di installare il certificato di root.	Il certificato non viene considerato attendibile ma viene offerta la possibilità di installarlo tra le root-certification riconosciute.
CA-Signed SENZA catena di certificazione	Il browser parla di "server certification chain incompleta" e di certificatore (definito "signer" o "issuer" a seconda dei casi) non riconosciuto. Non viene offerta alcuna possibilità di registrazione per il Server Certificate.	Il browser avverte che non è possibile verificare il Server Certificate a causa di "informazioni insufficienti" (la catena di certificazione!); viene comunque offerta la possibilità di installare il certificato nella sezione "Altri utenti".
CA-Signed CON catena di certificazione (*)	Il comportamento è analogo a quello ottenuto in presenza del solo certificato self-signed (vedi la prima riga di questa tabella). Il Server Certificate viene rilevato ma non è permessa la sua registrazione, lasciando questa prerogativa alla root-CA.	L'utente è informato che non è possibile verificare il certificato fino a un'autorità di certificazione attendibile: la catena di certificazione viene infatti risalita ma il browser non conosce la nostra Certification Authority. Il Server Certificate può essere inserito nella sezione "Altri utenti", quello della nostra CA tra le root-CA attendibili.

Tabella 2: I browser e i Server-Certificate

(*) In una prima prova la catena di certificazione è stata accorpata al Server Certificate, in un secondo momento è stata inserita in un file apposito (vedi la descrizione del parametro `SSLCertificateChainFile` del file etc/httpd/conf/httpd.conf). Come era ragionevole aspettarsi né Opera né Internet Explorer hanno modificato il loro comportamento passando da una scelta all'altra.

Il comportamento fondamentale che emerge è relativo all'uso di certification-chain: in sua presenza il client può decidere di registrarne i certificati per permettere navigazioni successive meno interrotte da dialog-box di allerta ma parimenti sicure. Come ciò avvenga è però altamente *browser dependent* e le varie politiche di gestione non sono facilmente e univocamente confrontabili: stando a quanto sopra riportato Intenet Explorer sembra per esempio essere più flessibile di Opera permettendo la memorizzazione separata di certificati cui si affidano ruoli e quindi livelli di confidenza diversi (i root e i server certificate nel nostro caso); tuttavia non bisogna dimenticare che la possibilità di memorizzare certificati non-root potrebbe portare a situazioni critiche nel momento in cui si è abituati a riporre troppo spesso fiducia su dichiarazioni di identità che in effetti non si è in grado di verificare.

Conclusione

Ciò che non si è visto in questo documento

Le cose che per motivi di tempo non sono state trattate sono molte. Altrettanto vero però è che tra queste si può forse distinguerne un subset particolarmente evidente, almeno per chi ha scritto queste pagine:

- Non si è parlato di Certificate Revocation List, perché farlo in modo significativo avrebbe quasi sicuramente "ingigantito" il discorso portando a discutere di protocolli come OCSP (Online Certificate Status Protocol, RFC 2560).
- Data la loro popolarità si sono affrontati solo i certificati X509 (tranne un breve accenno agli standard PKCS sviluppati dagli RSA Lab); in queste righe è quindi doveroso almeno citare PGP che sulla carta sembra offrire un meccanismo di certificazione per certi versi più flessibile di X509.
- Non si è presa in considerazione l'implementazione dei protocolli AH e ESP integrata nello stack protocollare di Linux nei recenti kernel della serie 2.6.
- Non si sono eseguite prove di VPN tra Linux e sistemi Windows.
- SSL/TLS è stato visto solo nell'ambito di https, che probabilmente ne costituisce l'uso più noto. Altri protocolli possono beneficiare però delle sue librerie: SMTP, IMAP, POP3 per citare solo quelli di uso comune.
- La casistica delle reazioni dei browser poteva essere più dettagliata e includere le configurazioni con Personal-Certificate.

Le cose che invece si sono dette in questo documento possono considerarsi ragionevolmente accurate, nel senso che provengono da fonti molteplici e attendibili (spesso citate nel testo) e che sono state oggetto di attività sistemistica sulla rete locale di prova.

Eventuali suggerimenti, consigli, errata corrige (specie relativamente alle "idee" che stanno dietro alle implementazioni) sono graditi all'indirizzo di posta andrea.barontini@ieee.org.

Andrea Barontini

Miscellaneous

Tips & Tricks

Di seguito alcune "dritte" liberamente assortite relative a comuni task di configurazione in cui si è incorso durante l'installazione della rete locale. Si noti che per quanto riguarda Linux si è fatto uso esclusivo di RedHat. È quindi probabile che alcuni di questi suggerimenti non valgano per le altre distribuzioni, specie quando chiamano in causa file della gerarchia /etc.

Tastiera italiana con "/" comodo sotto Linux

Avendo la tastiera italiana e specificando in /etc/sysconfig/keyboard la entry: `KEYTABLE="it2"` (invece che `"it"` o `"it-ibm"`) si fa in modo che il tasto a sinistra dello Shift di destra corrisponda a "`/`" (come nelle tastiere americane). Per poter continuare a usarlo per scrivere "`-`" si deve usare il tasto Alt.

IP-aliasing e "nic bringing up" su RedHat

Lo script /etc/rc.d/init.d/network ricerca nella cartella /etc/sysconfig/network-scripts i file ifcfg* (solo quelli che considera di reali IP di reali schede, quindi per es. niente ifcfg-lo o file di configurazione che nel nome contengono "`:`", indice di ip-aliasing).
Di fatto si procura perciò una lista delle schede installate (eth0, eth1, ...): per ciascuna richiama lo script /etc/sysconfig/network-scripts/ifup che accetta come parametro il nome del device da "alzare". Sarà questo script quindi che di fatto attiverà la nic, compresi tutti gli eventuali ip-alias che andranno specificati in file di configurazione analoghi a quelli degli ip "veri".
Es:

ifcfg-eth0	file di configurazione per scheda ethernet eth0
ifcfg-eth0:0	file di configurazione per il primo alias della scheda ethernet eth0

Hang di lilo: "LI" e poi più nulla...

I vecchi BIOS non riescono a indirizzare oltre un certo limite i dischi di grande dimensione; questo può portare al blocco del processo di boot se l'immagine del kernel è situata (anche) oltre i limiti imposti dal BIOS. (Una volta avvenuto il boot non c'è più alcun problema perchè Linux non usa il BIOS per accedere al disco).
Per ovviare al problema bisogna essere sicuri che l'immagine del kernel risieda tutta entro il limite. Sotto RedHat Linux tutto ciò che serve è nella cartella /boot quindi è possibile risolvere la situazione creando in fase di installazione una piccola partizione nella "zona bassa" del disco e poi montarla come "/boot".

Mount di cartelle

Per montare una cartella in effetti già accessibile si deve usare il comando:

```
mount --bind <cartella già visibile> <nuovo mount-point>
```

disponibile dal kernel 2.4 in poi. Utile per esempio per mettere in FTP cartelle che sono "al di sopra" della root del server FTP.

Ricerca di pattern

Usando more, vi, le man pages è possibile ricercare un pattern digitando (in modalità comandi per vi)

```
/<testo da ricercare>
```

e quindi INVIO.
Si salterà alla prima occorrenza di *<testo da ricercare>*. Per passare alle successive si deve digitare la lettera "`n`" (sempre in modalità comandi per vi).

Comandi multipli sulla linea di comando

Per concatenare sulla medesima riga della shell più comandi da eseguire in successione si usa il "`;`" come separatore. Es: `ifconfig; route` stamperà in successione l'output dei comandi `ifconfig` e `route` (vedi paragrafo sulla configurazione dei router).

Abilitazione forwarding su RedHat

Il parametro è `net.ipv4.conf.ip_forward`, va posto uguale a 1 e si trova in /etc/sysctl.conf
Non fa altro che andare a settare "l'omonima voce" nel filesystem virtuale /proc

Routes statiche su RedHat

Il formato del file /etc/sysconfig/static-routes (in cui vanno specificate) è il seguente:

```
any net x.x.x.x netmask y.y.y.y .....
```

infatti lo script /etc/rc.d/init.d/network all'interno del file considera solo le linee che iniziano per "`any`" e concatena tutto il resto al comando:

```
/sbin/route add -
```

(per questo motivo quando si specificano le route bisogna scrivere `net` e non `-net`, oppure `host` e non `-host` e questi devono essere i primi parametri, pena la non concatenazione con "`-`") Per il resto qualsiasi configurazione accettata da route a linea di comando dovrebbe essere possibile.

MS Windows: cartelle condivise in reti diverse (attraversamento router)

I PC apparteneti a uno stesso workgroup ma in reti diverse non sono direttamente "browse-abili" attraverso le Risorse di Rete, ma è comunque possibile avere cartelle condivise a patto di "connettere

unità di rete" (tasto dx mouse su Risorse di Rete) associate alle cartelle che si vogliono condividere. Per creare tale associazioni è sufficiente indicare il PC su cui risiede la cartella col suo IP piuttosto che col suo nome NETBIOS.
Es:
Unità: `X:`
Percorso: `\\192.168.20.2\software`

Avvio/Stop dei servizi su RedHat Linux

Avvio: `service <nome servizio> start`
Arresto: `service <nome servizio> stop`

(di fatto vengono richiamati i file contenuti in /etc/rc.d/init.d/)
Attenzione alle dipendenze tra i servizi!
Es: riavviando il servizio network senza aver prima stoppato il servizio ipsec e averlo riavviato dopo, renderà non funzionante la VPN.

Default-route su RedHat Linux

Il default route su Redhat Linux viene impostato inserendo nel file /etc/syconfig/network la seguente direttiva:

```
GATEWAY=xxx.xxx.xxx.xxx
```

All'avvio del servizio network (per es. tramite il comando "`service network restart`") la routing table conterrà la default-route verso il gateway indicato.

Gestione servizi nei runlevel di RedHat Linux

Per impostare quali servizi avviare o arrestare nei vari run level di Redhat Linux è comodo usare `chkconfig`
Alcuni esempi:

```
chkconfig --level 2345 httpd on      # avvia il servizio httpd nei runlevel 2,3,4,5
chkconfig --level 016 httpd off      # arresta il servizio httpd nei runlevel 0,1,6
chkconfig --list [<nome servizio>]   # Visualizza stato dei/del servizi(o) nei vari runlevel
```

RedHat 7.3 e vecchi PC

Uno dei punti in cui RedHat 7.3 si blocca durante l'installazione su vecchi PC è durante il probing del'usb (ovviamente non presente); per evitarlo passare alla riga di comando del kernel, durante il boot, il comando: `linux nousb`

Tipi di file in Linux

Per avere informazioni a proposito di un file (se per es. è un binario o un eseguibile) senza doverlo aprire si può utilizzare il comando:

```
file <nome file>
```

interessante perché parte delle informazioni vengono ricavate tramite tecniche euristiche.

Concatenazione di file sotto Linux

Per concatenare più file (tipicamente di testo) in un unico file si può usare il comando:

```
cat <nome file 1> <nome file 2> … <nome file n> > <nome file>
```

che sfrutta la capacità di `cat` di mandare in output più file uno dietro l'altro associata alla redirezione dello standard-output.

MS Windows 2000: permessi per gli utenti "semplici"

Può presentarsi il dubbio se gli account per utenti "semplici" sotto Windows 2000 vadano inseriti nel gruppo "User" o nel gruppo "PowerUser". Di fatto la scelta dipende soprattutto dal software che tali utenti andranno ad impiegare.
Il gruppo "User" infatti possiede i permessi minimi, ma Microsoft garantisce il corretto funzionamento solo delle applicazioni certificate per Windows.
Il gruppo "PowerUser" di fatto garantisce la "compatibilità all'indietro" per applicazioni non certificate, dando maggiori privilegi rispetto agli "User" (ma ovviamente meno rispetto agli "Administrator"!)
Tendenzialmente ciò che sotto NT 4.0 poteva essere eseguito come "User", in 2000 deve essere eseguito come "PowerUser". Ciò ovviamente comporta un indebolimento del livello di sicurezza medio di 2000.

Configurazioni dei router

```
eth0      Link encap:Ethernet  HWaddr 02:60:8C:2A:08:8B
          inet addr:160.78.29.69  Bcast:160.78.29.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:948482 errors:2 dropped:0 overruns:0 frame:155
          TX packets:2374 errors:0 dropped:0 overruns:0 carrier:0
          collisions:79 txqueuelen:100
          Interrupt:5 Base address:0x250 Memory:d8000-da000

eth1      Link encap:Ethernet  HWaddr 02:60:8C:3B:B5:1D
          inet addr:192.168.0.1  Bcast:192.168.0.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:943741 errors:0 dropped:0 overruns:0 frame:184
          TX packets:3552 errors:0 dropped:0 overruns:0 carrier:0
          collisions:128 txqueuelen:100
          Interrupt:9 Base address:0x280 Memory:cc000-ce000

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:14 errors:0 dropped:0 overruns:0 frame:0
          TX packets:14 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0


Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
160.78.29.0     *               255.255.255.0   U     0      0        0 eth0
192.168.0.0     *               255.255.255.0   U     0      0        0 eth1
192.168.0.0     192.168.0.2     255.255.0.0     UG    0      0        0 eth1
127.0.0.0       *               255.0.0.0       U     0      0        0 lo
default         160.78.29.254   0.0.0.0         UG    0      0        0 eth0
```

Figura 11: Configurazione di brutus

```
eth0      Link encap:Ethernet  HWaddr 00:20:AF:72:78:56
          inet addr:192.168.0.2  Bcast:192.168.0.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:1351053 errors:0 dropped:0 overruns:0 frame:0
          TX packets:5507 errors:0 dropped:0 overruns:0 carrier:0
          collisions:140 txqueuelen:100
          Interrupt:5 Base address:0x220

eth0:0    Link encap:Ethernet  HWaddr 00:20:AF:72:78:56
          inet addr:192.168.1.1  Bcast:192.168.1.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          Interrupt:5 Base address:0x220

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:6 errors:0 dropped:0 overruns:0 frame:0
          TX packets:6 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0


Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
192.168.20.0    192.168.1.3     255.255.255.0   UG    0      0        0 eth0
192.168.1.0     *               255.255.255.0   U     0      0        0 eth0
192.168.0.0     *               255.255.255.0   U     0      0        0 eth0
192.168.10.0    192.168.1.2     255.255.255.0   UG    0      0        0 eth0
127.0.0.0       *               255.0.0.0       U     0      0        0 lo
default         192.168.0.1     0.0.0.0         UG    0      0        0 eth0
```

Figura 12: Configurazione di titus

(Notare l'IP Aliasing sull'interfaccia eth0 che rivela la condivisione del dominio di collisione Ethernet tra reti IP diverse; scelta operata, visto il fine didattico dell'infrastruttura, per ottimizzare le risorse a disposizione).

```
eth0      Link encap:Ethernet  HWaddr 00:50:FC:6D:0E:1E
          inet addr:192.168.1.2  Bcast:192.168.1.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:830 errors:0 dropped:0 overruns:0 frame:0
          TX packets:4 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:100
          RX bytes:115747 (113.0 Kb)  TX bytes:240 (240.0 b)
          Interrupt:10 Base address:0x3000

eth1      Link encap:Ethernet  HWaddr 00:01:02:B7:74:14
          inet addr:192.168.10.1  Bcast:192.168.10.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:47 errors:0 dropped:0 overruns:1 frame:0
          TX packets:10 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:100
          RX bytes:7155 (6.9 Kb)  TX bytes:600 (600.0 b)
          Interrupt:9 Base address:0x3400

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:10 errors:0 dropped:0 overruns:0 frame:0
          TX packets:10 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:700 (700.0 b)  TX bytes:700 (700.0 b)


Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
192.168.1.0     *               255.255.255.0   U     0      0        0 eth0
192.168.10.0    *               255.255.255.0   U     0      0        0 eth1
127.0.0.0       *               255.0.0.0       U     0      0        0 lo
default         192.168.1.1     0.0.0.0         UG    0      0        0 eth0
```

Figura 13: Configurazione di augustus

```
eth0      Link encap:Ethernet  HWaddr 00:50:DA:28:31:4D
          inet addr:192.168.1.3  Bcast:192.168.1.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:7369 errors:0 dropped:0 overruns:0 frame:0
          TX packets:11 errors:0 dropped:0 overruns:0 carrier:3
          collisions:0 txqueuelen:100
          RX bytes:1021515 (997.5 Kb)  TX bytes:734 (734.0 b)
          Interrupt:10 Base address:0x3000

eth1      Link encap:Ethernet  HWaddr 00:01:02:B7:74:13
          inet addr:192.168.20.1  Bcast:192.168.20.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:9 errors:0 dropped:0 overruns:0 frame:0
          TX packets:9 errors:0 dropped:0 overruns:0 carrier:4
          collisions:0 txqueuelen:100
          RX bytes:590 (590.0 b)  TX bytes:764 (764.0 b)
          Interrupt:9 Base address:0x3400

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:12 errors:0 dropped:0 overruns:0 frame:0
          TX packets:12 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:844 (844.0 b)  TX bytes:844 (844.0 b)


Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
192.168.20.0    *               255.255.255.0   U     0      0        0 eth1
192.168.1.0     *               255.255.255.0   U     0      0        0 eth0
127.0.0.0       *               255.0.0.0       U     0      0        0 lo
default         192.168.1.1     0.0.0.0         UG    0      0        0 eth0
```

Figura 14: Configurazione di cicero

Indice delle figure e delle tabelle