IKEV2 TESTING

DAVIDE DE ZUANE & RAHMI EL MECHRI

CONTENTS

ABSTRACT

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

1 INTRODUCTION

2 SETUP

Andiamo a vedere nel dettaglio l'ambiente e la configurazione che abbiamo utilizzato per realizzare i test. Per verificare le capacità di IKE abbiamo previsto:

- 3 modalità di autenticazione;
- 2 chiper suite differenti da utlizzare.

Nella fase di sperimentazione abbiamo utilizzato le seguenti convenzioni:

- **Initiator**: l'host che invia la richiesta di stabilire una SA;
- **Responder**: l'host che risponde alle richieste.

2.1 Environment

Per simulare i due host della comunicazione abbiamo creato due macchine virtuali tramite l'utilizzo di qemu/kvm, questo per avere delle performance il più possibile simili a quelle reali. Le due macchine virtuali sono state create in modalità bridge, questo per evitare problemi con la modalità NAT.
Le macchine virtuali utilizzato hanno le seguenti specifiche:

- *Processore*: 2 core (flag `-smp`)
- *Memoria*: 2048MB (flag `-m`)
- *OS*: Debian 11
- *Network*: Bridge

Le macchine virtuali sono state create utilizzando `qemu/kvm` tramite i seguenti comandi è possibile creare la macchina virtuale.

Per prima cosa è necessario creare un disco immagine.

```
$ qemu-img create -f qcow2 disk.img 10G
```

Ora avviamo la macchina virtuale utilizzando il seguente comando.

```
$ qemu-system-x86_64 -smp 2 -m 2G -hda disk.img -cdrom <debian_iso> \
    -net bridge,br=virbr0 -enable-kvm & disown
```

Un procedimento simile si applica per l'altra macchina virtuale. Se non si vuole proseguire in questo modo si può utilizzare l'interfaccia grafica fornita da `virt-manager`.

2.2 Configuration

I file e le directory coinvolte nel processo di configurazione sono i seguenti. Dato che una delle principali modifiche di IKEv2 rispetto alla versione precedente è la possibilità di autenticazione tramite certificati.

```
/etc
├── ipsec.conf
├── ipsec.secrets
└── ipsec.d
    ├── cacerts
    ├── certs
    └── private
```

- Il file `ipsec.conf`[1] specifica la maggior parte delle configurazioni e le informazioni di controllo per il sottosistema IPsec (ulteriori specifiche e sintassi sono disponibili al seguente link).
- Il file `ipsec.secrets`[1] continene i segreti che poi verrranno utilizzati nella fase di autenticazione (ulteriori specifiche al seguenti link).

Certificati

Una delle principali novità che introduce `IKEv2` è la possibilità di eseguire l'autenticazione tra certificati X.509. In fase di testing abbiamo preso in considerazione due tipi di certificati:

- Certificati RSA
- Certificati ECDSA

A partire da una chiave pubblica è necessario realizzare un certificato di chiave pubblica e questo richiede la chiave privata di una CA. Nel nostro caso ci siamo creati dei certificati da CA e li abbiamo ditribuiti manualmente tra i due host.

Per la generazione abbiamo utilizzato il tool `pki`

CA Certificate

Partiamo con la generazione dei certificati da Certification Authority, di seguito sono riportati i due comandi da utilizzare. Ne occorrono due poichè per firmare i certificati ECDSA occorre una chiave con lo stesso schema.

```
$ pki --gen --type rsa --size 2048 --outform pem > 'ca.rsa.key.pem'
$ pki --gen --type ecdsa --size 256 --outform pem > 'ca.ecdsa.key.pem'
```

Ora utilizziamo la chiave privata per firmare il certificato di chiave pubblica.

```
$ pki --self --ca --lifetime 3650 --in 'ca.<type>.key.pem' --type <type> \
    --dn "CN=CA" --outform pem > ca.<type>.cert.pem
```

Occorre poi distribuire questi due certificati ai due host, vanno messi all'intenro della directory `cacerts`.

Host Certificate

Passiamo ora a generare i certificati che gli host andranno ad utilizzare nella fase di autenticazione, occorre generare la coppia chiave privata, chiave pubblica.

```
$ pki --gen --type ecdsa --size 256 --outform pem > 'host.ecdsa.key'
$ pki --gen --type rsa --size 2048 --outform pem > 'host.rsa.key'
```

E' buona norma salvare le chiavi all'interno della directory `private`. Ora andiamo ad estrarre la chiave pubblica da quella appena genrata e la firmiamo con la chiave delle CA del passo precedente.

```
$ pki --pub --in 'host.rsa.key' --type rsa | pki --issue --lifetime 1825 \
   --cacert 'ca.rsa.cert.pem' --cakey 'ca.rsa.key.pem'                   \
   --dn "CN=<Host_IP>" --san @<Host_IP> --san <Host_IP>                  \
   -- flag serverAuth --outform pem > 'host.rsa.cert.pem'
```

1 Le configurazioni utilizzate si trovano in appendice.

Si procede in maniera analoga con le opportune modifiche anche per il certificato ECDSA. Questi vanno poi posiizonati all'intenro della directory `certs`.

2.2.1 Mschap

Il riassunto della configurazione è mostrato in tabella, per l'initiator e il responder sono riportate le modalità della loro autenticazione.

Configuration	
Initiator	EAP-Mschapv2
Responder	RSA Certificate 2048
Chiper Suite	AES_CBC_128_HMAC_SHA2_256_128_DH_ECP_256

Esaminando gli scambi di IKE AUTH osserviamo che questa modalità richiede in totale 4 exchange.

2.2.2 RSA

Configuration	
Initiator	RSA Certificate 2048
Responder	RSA Certificate 2048
Chiper Suite	AES_CBC_128_HMAC_SHA2_256_128_DH_ECP_256

Utilizzando certificati RSA si osserva che la dimensione di un certificato eccede la dimensione massima di un pacchetto IP per tali motivi si ha la frammentazione: ovvero il contenuto, poichè eccede la dimenisone massima del campo *data* viene spezzato in più paccetti.

Anando ad esaminare il certificato, si osserva che ha una dimensione pari a 1032 byte, di cui abbiamo:

- 256 byte per la rappresentazoine del modulo;
- 1 byte per la rappresentazione dell'esponente di cifratura
- 384 byte per la firma
- i restanti byte sono esaminati in appendice.

Idealmente gli scambi durante IKE AUTH dovrebbero essere 2 ovvero i due si scambiano reciprocamente i certificati. Tuttavia, data la dimensioni di quest'ultimi, gli scambi effettivi risultano essere in totale 4.

2.2.3 ECDSA

Configuration	
Initiator	ECDSA Certificate 256
Responder	ECDSA Certificate 256
Chiper Suite	AES_CBC_128_HMAC_SHA2_256_128_DH_ECP_256

Si osserva che i certificati ECDSA hanno una dimensione ridotta rispetto a quella dei certifcati RSA, infatti quello utilizzato nel nostro caso ha una dimensione pari a 619 byte. Questo fa sì che non si ecceda la dimensione del payload del pacchetto IP, in questo modo la fase di IKE AUTH effettua solamente uno scambio.

3 TESTING

Per misurare i cicli macchina abbiamo utlizzato perf

per installarlo apt-get intstall linux-perf se da problemi con workload failed è a causa dei permessi e per risolverlo basta sovrascrivere il contenuto di

/proc/sys/kernel/perf$_{e}$vent$_{p}$aranoid per fare il report di tutto l'ambiente utilizzare pef report

3.1 Time

3.2 Performance

3.3 Results

4 CONSLUSIONI

A CONFIGURATION FILE

Di seguito riportiamo i file di configurazione `ipsec.conf` e `ipsec.secrets` rispettivamente di initiator e di responder. Una possibile modifica ai file potrebbe essere quella di rendere il tutto simmetrico, allo stato attuale i due non possono scambiarsi di ruolo. Alcune note:

- la connessione **default** definisce la configurazione comune a tutte le altre.
- la connessione **secure** è quella con cui specifichiamo la chiper_suite sicura.
- **also** permette di realizzare l'erditarietà multipla tra le connessioni.
- il parametro **auto** specifica quale operazione effettuare con la connessioni all'avvio di IPsec; il valore *add* la aggiunge alle possibile conessioni ma non cerca di stabilirla

A.1 Initiator

ipsec.conf

```
########################################################
# ipsec.conf - strongSwan IPsec configuration file
########################################################
conn %default
    leftsourceip=%config
    right=<ip_responder>
    rightsubnet=0.0.0.0/0
    auto=add

conn secure
    ike=aes256-sha384-ecp384!

conn base-mschap
    leftauth=eap-mschapv2
    eap_identity="<identity>"
    rightauth=pubkey

conn base-rsa
    rightauth=pubkey-rsa-2048
    leftauth=pubkey-rsa-2048
    leftcert=<path_to_cert>

conn base-ecdsa
    rightauth=pubkey-ecdsa-256
    leftauth=pubkey-ecdsa-2048
    leftcert=<path_to_cert>

conn secure-rsa
    also=base-rsa
    also=secure

conn secure-ecdsa
    also=base-ecdsa
    also=secure

conn ipsec-ike
    also=secure
    also=base-mschap
```

ipsec.secrets

```
##########################################################
# ipsec.secrets - strongSwan IPsec configuration file
##########################################################
<identity> : EAP "<password>"

: ECDSA "/etc/ipsec.d/private/<key>.pem"
: RSA "/etc/ipsec.d/private/<key>.pem"
```

A.2 Responder

ipsec.conf

```
##########################################################
# ipsec.conf - strongSwan IPsec configuration file
##########################################################
conn %default
    keyexchange=ikev2
    left=<ip_host>
    leftsubnet=0.0.0.0/0
    forceencaps=yes
    compress=no
    type=tunnel
    fragmentation=yes
    rekey=no
    right=<ip_initiator>
    rightid=%any
    rightsourceip=0.0.0.0/0
    rightdns=8.8.8.8,4.4.4.4
    auto=add

conn mschap
    rightauth=eap-mschapv2
    eap_identity=%identity
    leftcert=<path_to_cert>
    leftsendcert=always

conn rsa
    leftcert=<path_to_cert>
    leftauth=pubkey-rsa-2048
    rightauth=pubkey-rsa-2048

conn ecdsa
    leftcert=<path_to_cert>
    leftauth=ecdsa-256
    rightauth=ecdsa-256
```

ipsec.secrets

```
<identity> : EAP "<password>"

: RSA "/etc/ipsec.d/private/<key>.pem"
: ECDSA "/etc/ipsec.d/private/<key>.pem"
```

B TOOLS

Per instaurare la connessione IPsec si utilizza il seguente comando.

```
$ ipsec up <conn_name>
```

Per verificare che la SA sia stata correttamente instaurata è possibile utilizzare il seguente tool `ip xfrm`, il quale consente di effettuare la trasformazione dei pacchetti. Questo fornisce un interfaccia ai due database:

- SAD: Security Association Database, tramite l'oggetto `state`.
- SPD: Security Policy Database, tramite l'oggetto `policy`.

L'esecuzione del seguente comando fornisce una vista delle entry presenti nel SAD, possiamo poi utilizzare queste informazioni in wireahrk per poter vedere il traffico tra i due host in chiaro.

```
$ ip xfrm state list
src <initiator> dst <responder>
    proto esp spi 0xc49d3a6d reqid 1 mode tunnel
    replay-window 0 flag af-unspec
    auth-trunc hmac(sha256) <skey> 128
    enc cbc(aes) <skey>
    anti-replay context: seq 0x0, oseq 0xc, bitmap 0x00000000
src <responder> dst <initiator>
    proto esp spi 0xca382e6d reqid 1 mode tunnel
    replay-window 32 flag af-unspec
    auth-trunc hmac(sha256) <skey> 128
    enc cbc(aes) <skey>
    anti-replay context: seq 0x0, oseq 0x0, bitmap 0x00000000
```

Wireshark

Per vedere il traffico sniffato in chiaro occorre configurare il protocollo ISAKMP all'interno di wireshark, andiamo a specificare quelle che sono le chiavi negoziate per l'autenticazione di messaggio e di cifratura.

- Andare su `Edit->Preferences->Protocols->ISAKMP`.
- Aggiungere all'interno della tabella le varie entry riportate tramite `ip xfrm`

C CERTIFICATI

Andiamo a vedere a cosa è dovuta la dimensione dei certificati, per vedere il contenuto del certificato sotto forma di output testuale utilizzare il seguente comando.

```
$ openssl x509 --in <cert> -text
```

Andiamo a vedere nello specifico il conteuto delle due tipologie di certifiati utilizzate per la sperimentazione:

- ECDSA
- RSA

La differenza principali tra i due sta nella dimensione della chiave che nel nostro caso è di fondamentale importanza, in quanto evita la frammentazione del pacchetto. Anche se fa uso di chiavi da 256 bit ECDSA garantisce un livello di sicurezza pari a 2^{256}.

C.1 RSA Certificate

```
Certificate:
Data:
Version: 3 (0x2)
Serial Number: 3952640834610742420 (0x36da99e5a7ad4494)
Signature Algorithm: sha384WithRSAEncryption
Issuer: CN = <info>
Validity
    Not Before: May 29 08:42:06 2023 GMT
    Not After : May 27 08:42:06 2028 GMT
Subject: CN = <info>
Subject Public Key Info:
    Public Key Algorithm: rsaEncryption
        RSA Public-Key: (4096 bit)
        Modulus:
            00:c5:7d:50:95:2c:c3:42:32:b1:b8:1f:55:00:94:
            ---
        Exponent: 65537 (0x10001)
X509v3 extensions:
    X509v3 Authority Key Identifier:
        keyid:99:C3:D7:54:F4:40:EC:DE:9C:7C:60:DC:ED:29:60:BF:75:B6:94:30

    X509v3 Subject Alternative Name:
        DNS:192.168.122.145, IP Address:192.168.122.145
    X509v3 Extended Key Usage:
        TLS Web Server Authentication, 1.3.6.1.5.5.8.2.2
Signature Algorithm: sha384WithRSAEncryption
 18:e9:7c:2b:ea:2f:2c:2b:a6:d4:bd:6c:94:63:41:29:f9:45:
 ---
```

Come possiamo osservare dall'output sono conenute numerose informazioni che dunque aumentano notevolmente la dimensione del certificato e quindi che portano alla frammentazione di quest'ultimo durante la fase di IKE AUTH.

C.2 ECDSA

```
Certificate:
Data:
    Version: 3 (0x2)
    Serial Number: 6875679331162392113 (0x5f6b51ec3b6e0631)
    Signature Algorithm: ecdsa-with-SHA256
    Issuer: CN = CA ECDSA
    Validity
        Not Before: May 29 14:17:10 2023 GMT
        Not After : May 27 14:17:10 2028 GMT
    Subject: CN = 192.168.122.171 ECDSA
    Subject Public Key Info:
        Public Key Algorithm: id-ecPublicKey
            Public-Key: (256 bit)
            pub:
                04:21:d7:c7:a0:6f:fd:13:1a:1e:f4:c6:5b:5c:88:
                5c:99:3e:bf:92:89:7c:b2:0d:44:d0:9a:c7:aa:c3:
                0b:fe:4a:75:3a:ca:7b:91:ee:1b:69:e7:4f:40:06:
                e1:27:ee:62:72:eb:f7:06:30:c6:47:ae:db:01:e4:
                36:62:12:3e:92
            ASN1 OID: prime256v1
            NIST CURVE: P-256
    X509v3 extensions:
        X509v3 Authority Key Identifier:
            keyid:1A:12:82:AD:18:CF:85:0A:24:03:32:DC:D7:10:26:92:15:14:00:F9

        X509v3 Subject Alternative Name:
            DNS:192.168.122.171, IP Address:192.168.122.171
        X509v3 Extended Key Usage:
            TLS Web Server Authentication
Signature Algorithm: ecdsa-with-SHA256
     30:44:02:20:62:aa:81:67:fe:b7:2e:2f:13:f9:69:d4:6c:72:
     7e:a9:62:6a:db:7a:1b:af:35:b7:42:dc:42:fc:11:95:fa:d7:
     02:20:33:6f:7f:6b:a8:c4:c1:33:0e:04:7b:2f:99:14:85:ff:
     93:78:9c:ed:5d:84:58:61:76:d8:4d:b7:24:07:bd:b2
```

REFERENCES