Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE
Corso di Laurea in Informatica

Libvdeplug_agno: Ethernet encryption su reti virtuali

Relatore:
Chiar.mo Prof.
Renzo Davoli

Presentata da:
Michele Nalli

19 dicembre 2018

Per la mia famiglia,
che ha sempre creduto in me.

Introduzione

In questo elaborato viene descritto il processo di sviluppo di libvdeplug_agno: un modulo crittografico per VDE4 che permette di cifrare il traffico di una rete Ethernet virtuale VDE. Il capitolo 1 introduce il problema che libvdeplug_agno mira a risolvere e darà un idea delle attuali tecniche di Ethernet encryption. Nel capitolo 2 si descrivono i requisiti che la soluzione deve possedere per essere soddisfacente e si introducono alcuni strumenti che saranno utilizzati nell'implementazione. Nel capitolo 3 saranno giustificate le scelte implementative fatte con particolare attenzione alle operazioni crittografiche. Nel capitolo 4 si discute la sicurezza complessiva del modulo sviluppato, sono presentati i risultati di alcuni test di performance e vengono suggeriti alcuni futuri sviluppi.

Il codice sviluppato è stato rilasciato nel repository ufficiale [1]. Al momento di scrittura dell'elaborato esso non si trova ancora nel master branch, bensì in altri branch sperimentali.

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Stato dell'arte

1.1 VDE

VDE (Virtual Distributed Ethernet) è un sottoprogetto di **Virtual Square** [2] che permette di creare reti Ethernet virtuali distribuite su più macchine fisiche. Gli elementi di base delle reti virtuali VDE sono i corrispettivi virtuali, ovvero realizzati via software, degli elementi di base delle reti Ethernet fisiche, ovvero switch, connettori e cavi.

VDE è compatibile con Ethernet, questo significa che, oltre ad apparire come una rete Ethernet fisica ai programmi di virtualizzazione collegati, è possibile instradare pacchetti dalla rete virtuale a quella fisica e viceversa, creando così reti ibride.

VDE è supportato da molti software di virtualizzazione popolari come Qemu, KVM, User-Mode Linux e Virualbox, oltre che dagli altri progetti dell'ambito Virtual Square.

1.1.1 VDE2

Inizialmente VDE [3] non era altro che un insieme di programmi, da lanciare tramite interfaccia a linea di comando.

Ne segue un elenco con descrizione di ognuno:

vde_switch Uno switch virtuale configurabile al quale i programmi di virtualizzazione che supportano VDE si possono collegare direttamente.

vde_plug Un connettore virtuale da collegare alle porte dei vde_switch. vde_plug converte tutto il traffico proveniente dallo switch a cui è collegato in uno stream che può essere dato come input a programmi.

VDE wire Non è un programma specifico, bensì qualsiasi applicazione in grado di trasportare dati in maniera bidirezionale che viene usata per virtualizzare il filo di un cavo Ethernet. Per fare ciò si può usare il programma **dpipe** (bi-directional pipe). Usando dpipe insieme a programmi come netcat o ssh è possibile fare in modo che il VDE wire si estenda tra due macchine fisiche.

VDE cable Virtualizzazione di un cavo Ethernet completo. È composto da un VDE wire e due vde_plug. Viene usato per connettere due vde_switch. Per creare una rete Ethernet virtuale distribuita è sufficiente che i due vde_plug si trovino su due macchine diverse e che il VDE wire li metta effettivamente in connessione.

vde_cryptcab È un VDE cable criptato.

wirefilter Tool di testing che permette di simulare problemi, limitazioni ed errori delle reti reali.

slirpvde Programma che permette ai network VDE di interfacciarsi a Slirp.

vde_plug2tap Programma che è usato per legare un vde_switch ad un'interfaccia TAP. Tutto il traffico che proviene dallo switch è ridirezionato sull'interfaccia TAP specificata e viceversa.

vdetelweb Implementa un server web/telnet per il controllo a distanza di un vde_switch.

vde_cryptcab

Per realizzare un VDE cable criptato si potrebbe utilizzare ssh. Tuttavia ssh incapsula il traffico della rete virtuale in pacchetti TCP criptati. Questo significa che, nel caso sulla rete virtuale venga usato TCP, ci sarebbero due protocolli TCP che, lavorando parallelamente, introducono un overhead inutile.

vde_cryptcab incapsula il traffico della rete virtuale in pacchetti UDP cifrati. In questa maniera l'affidabilità della connessione è decisa esclusivamente dai protocolli utilizzati sulla rete virtuale.

1.1.2 VDE4

VDE4 [4] è nato come conseguenza del bisogno di semplificare e modularizzare VDE. VDE2 per ogni funzione prevista esige un programma specifico. In VDE4 certe funzionalità, che prima erano implementate in programmi specifici, possono essere implementate, in modo più elegante e flessibile, in plug-in.

Libvdeplug4

Libvdeplug è la libreria che permette di connettersi ad una rete VDE. In VDE4 questo elemento permette l'utilizzo di plug-in. Uno di questi plug-in permette di collegarsi agli switch come si faceva in VDE2, rendendo il progetto retrocompatibile.

In VDE2 un programma per collegarsi ad uno switch doveva specificare il percorso del socket dello switch. VDE4 espande l'interfaccia presentata ai programmi consentendo la possibilità precedente, ma consentendo anche di specificare un VDE URL che attivi il plug-in corrispondente.

Più precisamente i plug-in sono librerie dinamiche. Per esempio, la libreria libvdeplug_foo.so definisce un nuovo tipo di VDE plug-in "foo". Il VDE URL associato è "foo://...".

Il seguente è un elenco di alcuni plug-in implementati:

libvdeplug_vde Permette di collegarsi ad uno switch come in VDE2. È il plug-in che viene attivato anche nel caso un percorso sia specificato al posto di un VDE URL.

libvdeplug_switch Crea uno switch integrato nel programma che avvia il plug-in. Questo switch non è manageble ed ha meno funzionalità del vde_switch. Assomiglia concettualmente agli switch integrati nei router. Inoltre esso verrà terminato nel momento in cui il programma che ha avviato il plug-in viene terminato.

libvdeplug_tap Permette di connettersi alle interfacce TAP. Va a sostituire vde_plug2tap.

libvdeplug_slirp Sostituisce slirpvde [6].

libvdeplug_udp Incapsula il traffico della rete virtuale in pacchetti UDP spediti da parte della macchina fisica. Si può usare come VDE wire.

libvdeplug_cmd Manda il traffico ad un programma specificato come argomento. Il traffico viene mandato in forma di stream. Questo plug-in può essere usato per creare dei VDE wire (es. con ssh o netcat).

libvdeplug_vxvde Implementa un'evoluzione delle **VXLAN** (Virtual Extended LAN) [7] che sarà approfondita successivamente in questo elaborato.

libvdeplug_vlan Implementa le VLAN (IEEE 802.1Q). Permette di spostare le funzionalità relative alle VLAN, che sulle reti fisiche si trovano implementate negli switch, al di fuori degli switch virtuali.

Al momento il vde_switch di VDE4 non è ancora stato completato. Al suo posto si può usare comunque il vde_switch di VDE2 oppure il plug-in libvdeplug_switch.

La struttura modulare di VDE4 ci permette di rimuovere alcune funzionalità dagli switch, come la gestione delle VLAN, e di sostituirle con plug-in distribuiti sulla rete virtuale. Questo fa sì che gli switch si possano occupare esclusivamente dello smistamento del traffico, aumentando la velocità di smistamento e quindi mitigando l'effetto collo di bottiglia associato.

Inoltre, dal momento che anche vde_plug usa libvdeplug4, i vde_plug possono usare i plug-in. Il vde_plug diventa così più di una semplice virtualizzazione di un connettore.

Per il momento, non esiste nessun plug-in in grado di sostituire vde_cryptcab.

1.1.3 VXVDE

Libvdeplug_vxvde è un plug-in di VDE4 [8]. Implementa un'evoluzione delle VXLAN (Virtual eXtensible LAN) [7] che è utilizzabile contemporaneamente sia per implementare le VXLAN, che per estendere la rete virtuale locale su più macchine fisiche.

Il traffico della rete virtuale viene incapsulato dal plug-in VXVDE in pacchetti UDP inviati da parte della macchina fisica ad un indirizzo IP multicast, il quale è l'unico argomento necessario ad avviare VXVDE.

Macchine virtuali collegate con VXVDE allo stesso indirizzo IP multicast comunicano come se fossero sulla stessa rete locale, anche se si trovano su macchine fisiche differenti.

Sicurezza

Il traffico delle macchine virtuali appare sulla rete locale come traffico UDP-IP. Questo significa che ogni utente di una macchina fisica può lanciare una macchina virtuale collegata tramite VXVDE a qualsiasi IP multicast.

Usando VXVDE, affinché il sistema sia sicuro, bisogna che gli utenti non amministratori abbiano accesso alle macchine virtuali, ma non alle macchine fisiche e che non possano nemmeno riconfigurare la connessione delle loro macchine virtuali. Se così non fosse potrebbero collegarsi a reti VXVDE per cui non hanno i permessi.

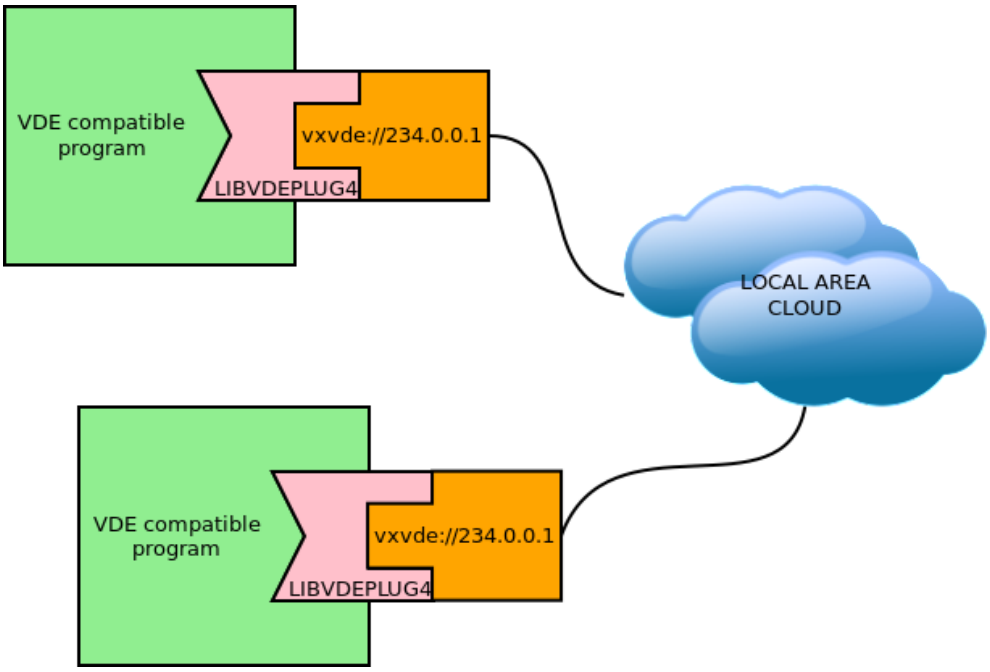

Figura 1.1: VXVDE local area cloud

1.1.4 VXVDEX

VXVDEX [9] implementa un meccanismo di controllo degli accessi sulle reti VXVDE. Una volta installato VXVDEX ed il relativo modulo kernel è possibile collegarsi alla rete solo tramite il plug-in e solo con le restrizioni imposte per controllare gli accessi.

Sicurezza

Con VXVDEX, oltre che l'accesso alle macchine virtuali, è possibile dare agli utenti accesso shell ad un namespace di rete [10]. In questo modo gli utenti non possono avere accesso diretto alla rete reale e, conseguentemente, non possono vedere il traffico scambiato dagli altri utenti sulla stessa macchina.

Usando **libpam_net** [11] è possibile fornire un namespace di rete ad ogni utente al momento del login.

1.1.5 Fornire accesso alla rete reale

In alcuni contesti si potrebbe voler dare accesso diretto alla rete reale da parte degli utenti mantenendo il servizio sicuro. In questi casi VXVDEX non può essere usato.

Per fare ciò il traffico della rete virtuale deve essere cifrato. Se così non fosse un utente potrebbe collegarsi a qualsiasi rete VXVDE oppure catturare direttamente i pacchetti in transito sull'interfaccia di rete della macchina fisica.

Inadeguatezza di vde_cryptcab

vde_cryptcab è utilizzabile unicamente per collegare due switch su macchine remote. Questo significa che non è utilizzabile con VXVDE perché non può essere usato in un contesto multicast.

Si vorrebbe poter avere a disposizione un elemento di VDE4 che sia in grado di sostituire vde_cryptcab per quanto riguarda le connessioni point-to-point e che possa essere utilizzabile in simbiosi con VXVDE per risolvere il problema di cui si è parlato.

1.2 Media Access Control Security

Media Access Control Security (MACsec), per come definito dallo standard IEEE 802.1AE [12], è un protocollo che, tramite l'utilizzo di tecniche crittografiche, permette di dare garanzie di confidenzialità, integrità ed autenticazione sul traffico di livello data-link.

MACsec è utile nei casi in cui una LAN supporta applicazioni di importanza critica, ma non è possibile, o risulta troppo costoso, impedire l'accesso fisico alla rete da parte di potenziali attaccanti.

Con MACsec tutto il traffico è autenticato e, opzionalmente, criptato. Per questo motivo MACsec può mitigare attacchi indirizzati ai protocolli di livello 2.

1.2.1 Struttura hop-by-hop

MACsec è una tecnologia "hop-by-hop". Ad ogni salto i pacchetti uscenti sono cifrati e quelli entranti sono decifrati. Anche se una tecnologia "end-to-end" introdurrebbe un overhead minore per quanto riguarda le operazioni crittografiche, non è detto che essa sia più efficiente; il fatto di decifrare i pacchetti su ogni switch permette per esempio di usare tecniche di load balancing.

Inoltre il fatto di avere unicamente due interlocutori su ogni link facilita lo scambio ed il rinnovo delle chiavi crittografiche.

1.2.2 Formato frame

Il frame MACsec ha più campi di un normale frame Ethernet.

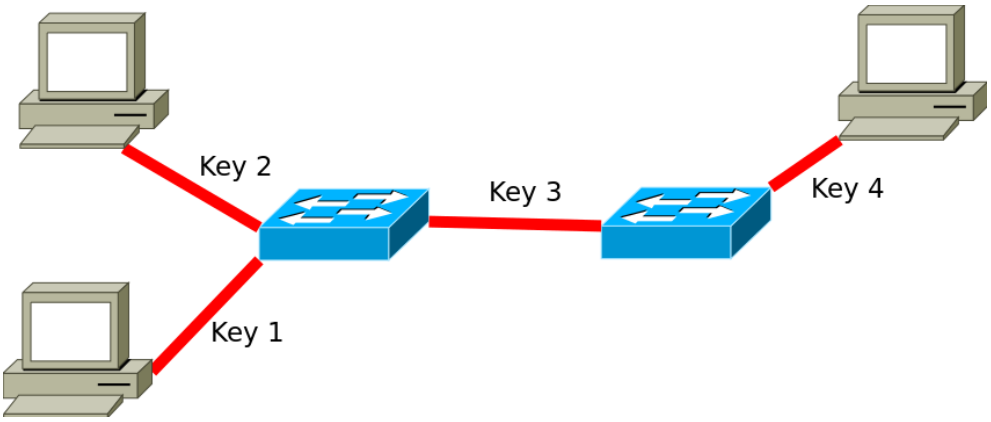

Figura 1.2: MACsec hop-by-hop encryption

1. **Ethernet header**
   L'EtherType è specifico per il protocollo MACsec (0x88E5).

2. **SecTAG** (Security Tag)
   Un header che contiene informazioni sul protocollo in uso, permette di identificare univocamente il mittente e contiene un contatore incrementale di pacchetti detto Packet Number (PN).

3. **Secure Data**
   È formato dall'EtherType del pacchetto originale e dal payload. Può essere criptato.

4. **ICV** (Integrity Check Value)
   Un MAC (Message Authentication Code) aggiunto come coda del pacchetto che ne garantisce l'integrità.

Ethernet header	SecTAG	secure data	ICV

Tabella 1.1: Struttura di un pacchetto MACsec

1.2.3 Servizi di sicurezza supportati

MACsec fornisce i seguenti servizi ai partecipanti autenticati:

1. **Connectionless data integrity**

2. **Data origin authenticity**
   Il pacchetto proviene da un mittente autenticato.

3. **Confidentiality**

4. **Replay protection**
   Un pacchetto non può essere ricevuto più di una volta.

5. **Bounded receive delay**
   Un pacchetto non sarà mai ricevuto dopo un certo limite di tempo fissato. Nel caso di MACsec questo limite è di circa 2 secondi.

1.2.4 Cipher suite

MACsec usa AES in modalità GCM [13] (Galois/Counter Mode) con chiave di 128 bit.

GCM-AES-128 è un algoritmo di authenticated encryption with associated data [14] (AEAD), ovvero fornisce garanzie di confidenzialità, integrità ed autenticità in una sola operazione. L'algoritmo può anche essere usato per fornire solamente garanzie di autenticità ed integrità senza confidenzialità (GMAC).

GCM accetta IV (Initialization Vector) di lunghezza arbitraria, anche se è fortemente raccomandato l'uso di IV di 96 bit principalmente per motivi di efficienza [15].

Nella specifica della modalità GCM [13] viene richiesto che gli IV usati con debbano soddisfare il seguente requisito di unicità:

"La probabilità che la funzione di crittografia autenticata sia invocata con lo stesso IV e la stessa chiave su due (o più) input distinti non deve essere maggiore di 2^{-32}**"**.

Per questo motivo al posto di IV da ora in poi useremo il termine **nonce**. Il precedente è un requisito importante quasi quanto la segretezza della chiave, in quanto il fatto di cifrare due plaintext diversi con stessa chiave e stesso nonce esporrebbe l'implementazione a vari tipi di attacchi.

In MACsec vengono usati nonce di 96 bit costruiti del modo seguente:

1. I 64 bit più significativi contengono informazioni che identificano univocamente il mittente.

2. I 32 bit meno significativi sono il PN.

Come detto in precedenza, queste informazioni sono trasmesse al destinatario all'interno del SecTAG.

Per soddisfare il requisito di unicità del nonce, la chiave viene cambiata all'esaurirsi del PN. In questo modo la probabilità che un nonce sia usato più volte è nulla.

Il meccanismo di gestione e scambio delle chiavi non è parte di MACsec, ed è specificato nello standard IEEE 802.1X-2010 [16].

Nella revisione di MACsec del 2011 [17] è stato aggiunto il supporto per GCM-AES-256.

Capitolo 2

Introduzione al contributo originale

2.1 Ethernet encryption per VDE4

Come delineato nel capitolo precedente, si necessita lo sviluppo di uno strumento per VDE4, che consenta di cifrare il traffico della rete virtuale.

Dato che VDE virtualizza una rete Ethernet e che lo strumento da sviluppare deve essere "trasparente" alle macchine virtuali, è ragionevole pensare di agire sul livello data-link.

2.2 Libvdeplug_agno

In linea con la filosofia di modularità del progetto VDE4, si vuole che questo strumento sia implementato come plug-in per libvdeplug4.

Questo plug-in è stato chiamato agno in quanto le tecniche crittografiche utilizzate sono "agnostiche" rispetto ai protocolli usati dai pacchetti. Tutto il traffico processato da agno viene cifrato qualsiasi sia il protocollo in uso.

Inoltre agno è un "nesting plug-in", ovvero si appoggia sulla connessione gestita da un altro plug-in "nested". Agno si occupa esclusivamente della messa in sicurezza del traffico e non del modo in cui esso viene trasmesso.

2.2.1 Agno come strumento generale

Il fatto che agno si renda necessario come strumento complementare di VX-VDE non esclude che esso possa essere pensato come strumento di tipo generale

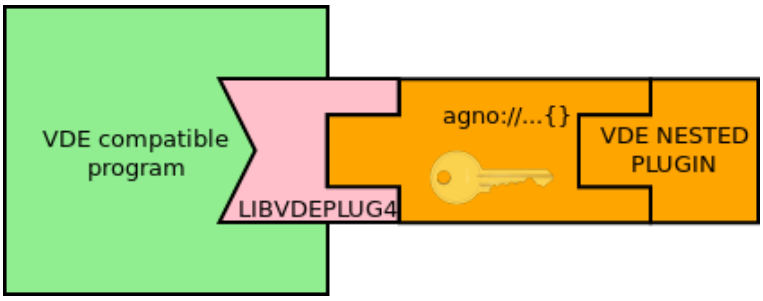

Figura 2.1: Agno nesting plugin

che permetta di sostituire vde_cryptcab ed avere anche utilità su reti ibride.

Agno con VXVDE

In questo caso agno usa come connessione interna quella aperta dal plug-in VXVDE. Per essere efficacemente usabile in questo contesto, agno dovrebbe richiedere una configurazione minima, in modo che esso appaia quasi come un'estensione di VXVDE.

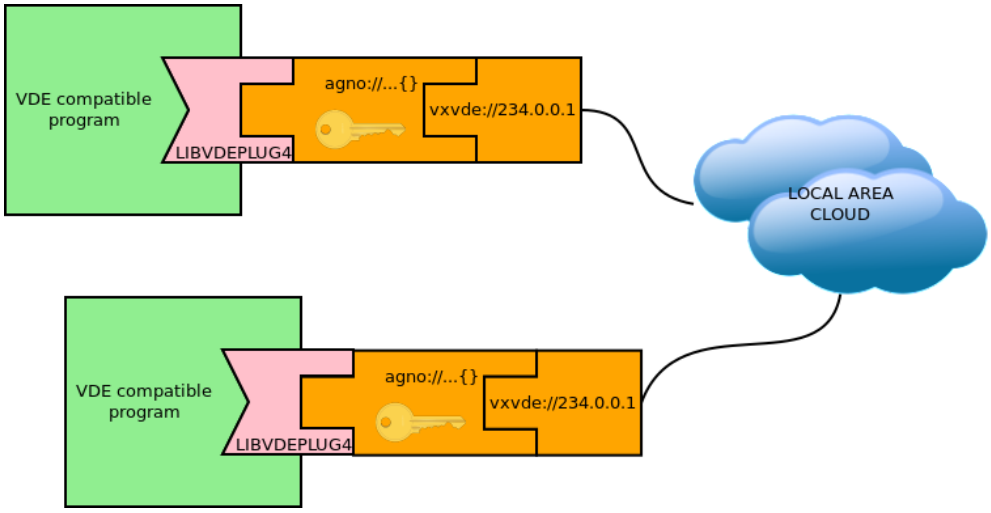

Figura 2.2: Agno con VXVDE

Agno successore di vde_cryptcab

In questo caso agno usa come connessione interna quella di un VDE wire non criptato. Agno dovrebbe essere in grado superare i limiti di vde_cryptcab, pur garantendo lo stesso, se non più alto, grado di sicurezza ed efficienza.

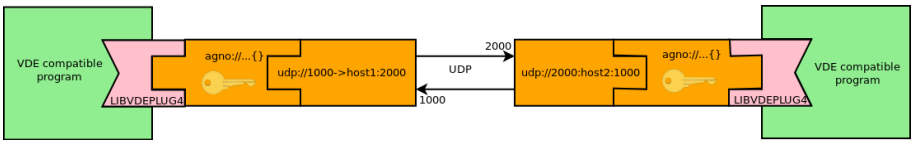

Figura 2.3: Agno con VDE wire

Compatibilità con rete reale

Dato che il traffico di una rete virtuale VDE può essere direzionato su una rete fisica si possono pensare applicazioni di agno che servano a garantire sicurezza della rete fisica.

Il plug-in deve essere pensato in modo tale che gli switch fisici siano in grado di smistare il traffico cifrato correttamente.

In questo modo, agno potrebbe essere usato come alternativa "end-to-end" a MACsec per le macchine virtuali collegate alla rete fisica.

2.2.2 Prototipo di agno

Prima del presente lavoro era già stato realizzato un prototipo di agno come nesting plug-in. Questo prototipo, realizzato usando la libreria crittografica di OpenSSL, oltre a non dare alcuna garanzia di integrità sull'header del pacchetto Ethernet, implementa la crittografia in maniera migliorabile e, probabilmente, non sicura. Al momento questo prototipo si trova sul branch master del repository ufficiale [1].

2.3 Inadeguatezza di MACsec

MACsec presenta alcune caratteristiche che non ne permettono l'utilizzo in questo contesto.

La sua struttura "hop-by-hop" fa sì che su ogni link ci siano unicamente due interlocutori, cosa che rende il rinnovo delle chiavi relativamente facile.

In MACsec le chiavi di ogni link vengono rinnovate quando non è più possibile generare nuovi nonce. Dato che l'unicità dei nonce dipende dal campo Packet Number del SecTAG e che questo campo è di soli 32 bit, una stazione può cifrare massimo 2^{32} pacchetti con la stessa chiave.

In VXVDE lo smistamento dei pacchetti avviene senza switch sfruttando il servizio IP multicast, quindi oltre ad avere generalmente più di due interlocutori non è disponibile un elenco di tutti gli host virtuali collegati alla rete locale.

2.4 Librerie di crittografia

Come libreria di crittografia nelle prime fasi dello sviluppo si era deciso di usare quella di OpenSSL [18]. Tuttavia, dato che si vuole rilasciare il codice con licenza GPL, si è considerato di utilizzare una libreria rilasciata con licenza compatibile, per non dover aggiungere una GPL linking exception.

2.4.1 GnuTLS

GnuTLS [19] non è propriamente una libreria di crittografia, bensì un'implementazione dei protocolli SSL e TLS. La libreria di crittografia che viene usata internamente è Nettle [20].

GnuTLS fornisce, per quanto riguarda le operazioni crittografiche, un'interfaccia di più alto livello rispetto a Nettle, che permette comunque di usare la maggior parte degli algoritmi implementati da Nettle. Nel caso si rendesse necessario l'uso di un algoritmo fornito unicamente dall'API di Nettle si potrebbe utilizzare direttamente quest'ultima libreria.

Oltre a questo, GnuTLS mette a disposizione primitive per la gestione dei certificati X.509, cosa che potrebbe avere utilità nel caso in cui si decida di implementare un meccanismo di negoziazione della chiave di sessione.

Capitolo 3

Analisi del contributo originale

Partendo dal prototipo di plug-in già presente, si è proseguito lo sviluppo con l'obiettivo di trovare un meccanismo sicuro che fosse il più possibile flessibile, efficiente, manutenibile da parte degli sviluppatori e comodo da usare.

3.1 Distribuzione automatica della chiave

Il prototipo di agno presentava una gestione statica delle chiavi. Il plug-in prendeva come argomento il path di un file contenente una chiave codificata in caratteri esadecimali e la usava per cifrare il traffico.

Nelle prime fasi dello sviluppo ci si è domandati se fosse il caso di implementare un meccanismo che permettesse di distribuire la chiave di sessione ai plug-in autorizzati in modo automatico e trasparente agli utenti.

Si pensava che un meccanismo di questo tipo avrebbe semplificato la gestione delle chiavi, ma questa prima fase dello studio ha portato alla conclusione che un meccanismo a chiave statica sarebbe risultato molto più conveniente sotto ogni aspetto.

In ogni caso, per convincere il lettore di ciò, ripercorreremo i ragionamenti fatti nel tentare di implementare un meccanismo di distribuzione della chiave.

Nel realizzare questo meccanismo, a causa della diversità dei contesti di applicazione, non ci si è potuti ispirare allo scambio di chiavi del protocollo MACsec. In agno, infatti, lo scambio di chiavi deve avvenire "end-to-end", interessando unicamente i plug-in.

Per fare ciò si è scelto di utilizzare gli strumenti di crittografia a **chiave pubblica**, firma digitale e gestione dei certificati X.509 forniti da GnuTLS.

L'idea è di dare ad ogni utente una coppia di chiavi da conservare in una location di default. In questo caso l'unico argomento da specificare sarebbe un elenco di utenti autorizzati a partecipare ad una determinata rete sicura e le loro chiavi pubbliche, le quali saranno utilizzate per cifrare la chiave di sessione nel momento dello scambio.

3.1.1 Protocollo

Dal momento che le soluzioni proposte in [22] e [23] non fanno riferimento ad IP multicast, e quindi non sono efficacemente utilizzabili, bisogna pensare ad un protocollo differente.

Per come il meccanismo era stato pensato inizialmente, al momento dell'avvio del plug-in, una richiesta di autenticazione veniva inviata in broadcast sulla rete virtuale al quale agno era collegato. Se la richiesta veniva riconosciuta come valida da qualche ricevente, ovvero il richiedente compariva sulla sua lista, quest'ultimo inviava una risposta firmata contenente la chiave di sessione cifrata con la chiave pubblica del richiedente. Nel caso nessun plug-in riconoscesse la richiesta come valida, o nel caso il richiedente fosse il primo a collegarsi alla rete, nessuna risposta veniva inviata. Allo scadere di un timeout il richiedente procedeva a generare casualmente la chiave di sessione.

Risulta chiaro che un utente che non compare sulle liste degli utenti autenticati non può partecipare alla rete sicura in quanto la sua richiesta non viene riconosciuta come valida. Inoltre, dopo lo scadere del timeout, se il plug-in avviato da questo utente avesse cercato di rispondere a plug-in richiedenti autenticazione, la risposta non sarebbe stata ritenuta valida. Questo perché il ricevente della risposta ne verifica la firma con la chiave pubblica del mittente.

3.1.2 Problemi

Di seguito si elencano vari problemi che si sono incontrati durante l'implementazione di questo meccanismo.

1. VXVDE si basa su **IP multicast**, che fa uso di UDP. Questo significa che, nel caso agno venga usato con VXVDE, la richiesta potrebbe non giungere agli interessati. Per avere garanzie di affidabilità della connessione in un contesto multicast, in cui non si ha un elenco dei riceventi di un messaggio, bisognerebbe centralizzarne la struttura, ovvero usare un processo server [24]. Ogni plug-in agno attivato dovrebbe essere in grado di connettersi in

modo affidabile a questo server che, nel caso il plug-in client sia autorizzato, invierebbe la chiave di sessione cifrata come risposta. Questo metodo potrebbe essere valido, ma risulterebbe molto scomodo da configurare e richiederebbe un programma server. Questo sarebbe poco allineato con la filosofia di modularità di VDE4.

2. Se si sceglie di mantenere una struttura distribuita ogni plug-in agno deve avere a disposizione una lista di utenti autorizzati. Se si volesse aggiungere o togliere un utente alla lista sarebbe necessario modificare tutte le liste.

3. Ogni utente deve essere associato ad un **identificativo univoco**. Bisognerebbe pensare ad un meccanismo per soddifare anche questo requisito.

4. Per usare meccanismo del genere bisogna interfacciarsi ad una **Certification Authority** (CA).
   Si hanno due casi:

   CA esterna Bisogna usare gli strumenti forniti con le loro limitazioni.

   CA interna Richiede uno sforzo di gestione non trascurabile. In pratica bisogna che gli amministratori firmino di persona i certificati con la chiave privata della CA interna.

   In ogni caso l'overhead di gestione delle reti sicure è abbastanza elevato.

5. Si potrebbe pensare di prendere le chiavi pubbliche da **repository di chiavi su internet**. Questo però non permetterebbe l'uso di agno su macchine fisiche a cui è preclusa la connessione ad internet, cosa che in alcuni contesti potrebbe essere una limitazione.

6. Un **nesting plug-in** è pensato per essere un elemento che modifica il traffico da inviare al **nested plug-in**. Implementare protocolli di distribuzione di chiavi in un simile contesto risulta forzato e poco elegante.

7. Dato che i plug-in non hanno accesso alle informazioni delle macchine virtuali ed in generale non è detto che siano collegati a macchine virtuali (possono anche essere lanciati da vde_plug), i pacchetti di protocollo sono inviati in broadcast con MAC address sorgente nullo. Questo può far sì che agno non sia compatibile con le reti fisiche.

8. Le liste di utenti autenticati dovrebbero essere disgiunte. Se così non fosse, essendo l'autenticazione distribuita, si rischia di "propagare" l'autenticazione anche ad utenti non autorizzati da tutti i partecipanti.

9. Una soluzione del genere risulta superflua nel caso in cui agno sia usato in sostituzione di **vde_cryptcab**.

10. Gestire la **revoca dei certificati** può essere complesso.

11. La sicurezza di un sistema complesso è più difficile da dimostrare di quella di un sistema semplice. Se si vuole che agno sia considerato sicuro, la scelta migliore è di **semplificare il meccanismo**.

3.2 Chiave statica

Adottare un meccanismo a chiave statica semplifica notevolmente l'implementazione di agno, rendendolo più manutenibile e riducendo il rischio di introdurre insicurezze nel protocollo di distribuzione della chiave.

Inoltre il plug-in diventa molto più immediato da usare, cosa che si sposa con la filosofia di "zero-configurabilità" di VXVDE.

Il struttura della rete sicura diventa elegante e simmetrica, ed agno appare essere un layer che implementa un servizio di sicurezza senza ulteriori complicazioni. Agno ha le potenzialità per essere una tecnologia "deploy-and-forget".

Garantire la segretezza delle chiavi diventa responsabilità dell'utente. Questo, dal punto di vista delle garanzie di sicurezza che agno vuole fornire, può essere visto come un vantaggio perché permette di rifarsi a strumenti esistenti e dalle caratteristiche di sicurezza verificate.

In concreto l'utente deve provvedere a compiere in modo sicuro le seguenti azioni:

1. **Generare la chiave**
   La generazione della chiave può essere eseguita da qualsiasi strumento in grado di generare sequenze casuali in modo sicuro [21]. È importante che le vecchie chiavi non siano riutilizzate per nuovi scopi. Se **openssl** è disponibile nel sistema si può usare il seguente comando per generare una chiave di 256 bit:

   ```
   openssl rand -out keyfile -hex 32
   ```

2. **Trasmettere la chiave**
   Per trasmettere in maniera sicura le chiavi si può adoperare qualsiasi servizio che fornisca forti garanzie di confidenzialità e contro il "replay". Se si possiede un account sulle macchine interessate e queste macchine hanno

dei server ssh attivi si può usare **scp**. Altri metodi possono essere quelli out-of-band, copy-and-paste da shell ssh, e-mail cifrate o l'uso di **NFS**. Se le chiavi si trovano in una cartella condivisa con NFS bisogna verificare che la versione di NFS utilizzata faccia uso di crittografia.

3. **Conservare la chiave**
   I file contenenti le chiavi devono essere protetti da permessi tali che solo gli utenti autorizzati ne possano leggere il contenuto e solo il proprietario modificarlo.

Risulta evidente che in un contesto del genere sia molto scomodo rinnovare le chiavi. Per fare ciò infatti bisognerebbe aggiornare il file della chiave su tutte le macchine fisiche che ne fanno uso e riavviare tutti i plug-in. Inoltre, riavviare un plug-in significa riavviare il programma su cui esso dipende, cosa che in molti casi si vorrebbe poter evitare.

Preso atto di questa limitazione, se l'obiettivo è quello di creare un sistema davvero flessibile, bisogna strutturare le operazioni crittografiche in modo che non sia necessario pianificare rinnovi di chiavi.

3.3 Garanzie fornite

Con agno si vuole offrire garanzie di sicurezza comparabili a quelle di MACsec. Nella tabella 3.1 si può vedere un confronto tra i servizi di sicurezza offerti da MACsec e quelli offerti da agno.

In agno si è scelto di non implementare nessun meccanismo di protezione dai **replay attack** oltre che la garanzia di bounded receive delay.

In MACsec la struttura "hop-by-hop" rende abbastanza facile implementare meccanismi di replay protection: è sufficiente prendere in considerazione il valore del PN del pacchetto ricevuto e scegliere una finestra di di valori accettabili di ampiezza adeguata rispetto all'ultimo PN valido ricevuto.

In agno per implementare un meccanismo del genere bisognerebbe utilizzare un dizionario che ha come chiavi gli indirizzi MAC dei mittenti e come valori i massimi valori dei PN ricevuti da ogni mittente. Le operazioni di lookup, validazione ed eventualmente aggiornamento andrebbero fatte per ogni messaggio ricevuto. Inoltre, allo scadere della validità dell'ultimo pacchetto ricevuto da un mittente, bisognerebbe rimuovere dal dizionario l'entrata corrispondente. Il meccanismo di rimozione degli elementi introdurrebbe un ulteriore overhead.

Si ritiene che le garanzie di sicurezza fornite siano sufficienti a fornire un servizio sicuro rispetto al contesto in cui agno sarà usato.

Nel caso il protocollo in uso sia **TCP** i pacchetti duplicati saranno scartati dal protocollo. Nel caso il protocollo in uso sia **UDP** il ricevente si aspetta che i pacchetti possano essere ricevuti duplicati. Per quanto riguarda la sicurezza dei **protocolli di livello 2** si può affermare che le garanzie di integrità ed autenticità fornite siano sufficienti.

	MACsec	libvdeplug_agno
Connectionless data integrity	✓	✓
Data origin authenticity	✓	✓
Confidentiality	✓	✓
Replay protection	✓	✗
Bounded receive delay	✓	✓

Tabella 3.1: Confronto tra servizi di sicurezza offerti da MACsec e da agno

3.4 Struttura del pacchetto agno

Ethernet header	agno header	Secure Data	ICV

Tabella 3.2: Struttura di un pacchetto cifrato con agno

Un pacchetto cifrato con agno ha una struttura molto simile a quella di un pacchetto MACsec. Essa si può vedere nelle figure 3.2, 3.3 e 3.4. I due header sono autenticati, mentre il campo secure data è sia cifrato che autenticato. Il **timestamp** contenuto nell'header agno è ciò che permette di dare la garanzia di **bounded receive delay**. I pacchetti dopo 3 secondi dal momento dell'invio non sono più ritenuti validi.

Timestamp	Nonce

Tabella 3.3: Struttura di un header agno

EtherType	Payload

Tabella 3.4: Struttura di secure data

3.5 Operazioni crittografiche

Per fornire le garanzie di sicurezza enunciate si è scelto di prendere in considerazione algoritmi **AEAD**.

3.5.1 GCM-AES-128

Inizialmente si è deciso di utilizzare, come in MACsec, GCM-AES-128. **GnuTLS** permette di usare questo algoritmo tramite un'apposita API di alto livello per gli algoritmi di authenticated encryption che si può vedere di seguito.

```
int gnutls_aead_cipher_init(gnutls_aead_cipher_hd_t* handle,
    gnutls_cipher_algorithm_t cipher, const gnutls_datum_t *key)

int gnutls_aead_cipher_encrypt(gnutls_aead_cipher_hd_t handle, const
    void *nonce, size_t nonce_len, const void *auth, size_t auth_len,
    size_t tag_size, const void *ptext, size_t ptext_len, void *ctext,
    size_t *ctext_len)

int gnutls_aead_cipher_decrypt(gnutls_aead_cipher_hd_t handle, const
    void *nonce, size_t nonce_len, const void *auth, size_t auth_len,
    size_t tag_size, const void *ctext, size_t ctext_len, void *ptext,
    size_t *ptext_len)

void gnutls_aead_cipher_deinit(gnutls_aead_cipher_hd_t handle)
```

ICV

Come si può leggere in [25], la modalità GCM ha alcune debolezze per quanto riguarda l'autenticazione. Per questa ragione deve essere usata solo con ICV sufficientemente lunghi. Per agno, come per MACsec, si è scelto di adottare la massima lunghezza consentita: 128 bit.

3.5.2 Costruzione dei nonce

La specifica della modalità GCM [13] fornisce due framework per costruire i nonce: deterministicamente o casualmente.

Costruzione deterministica

Un nonce costruito deterministicamente è formato da due campi:

Fixed field Identifica univocamente il mittente.

Invocation field Identifica univocamente il pacchetto cifrato nell'ambito di un determinato mittente. Come invocation field viene in genere usato un contatore. La chiave deve essere sostituita prima dell'esaurisi del contatore.

La costruzione consigliata per il nonce standard di 96 bit è 32 bit di fixed field e 64 di invocation field.

In un contesto come quello di agno usare una costruzione di questo tipo risulta difficile. Ci si pone infatti il problema di come costruire il fixed field. Una possibilità potrebbe essere quella di usare il MAC address della macchina virtuale. Tuttavia, valutando attentamente, questa metodologia ci offre veramente poche garanzie di unicità. In programmi come **vdens** questo indirizzo è generato casualmente, cosa che, nel caso la generazione del numero casuale avvenga in modo sicuro, potrebbe essere ritenuta accettabile. In altri programmi come **Qemu** il MAC address può essere specificato dall'utente. Se un utente per leggerezza avviasse due macchine virtuali con stesso MAC address e stessa chiave la sicurezza di tutto il traffico cifrato con quella chiave risulterebbe compromessa. Inoltre, si vorrebbe che agno possa essere avviato non solo dalle macchine virtuali, ma da qualsiasi programma che supporti libvdeplug, come per esempio **vde_plug**. Questo non sarebbe direttamente possibile nel caso si usasse il MAC address come fixed field.

Costruzione casuale

Un nonce costruito casualmente è formato da due campi:

Free field Può essere qualsiasi cosa.

Random field Numero casuale.

Lo standard consiglia che il campo free field sia vuoto in qualsiasi caso in modo che l'intero nonce sia generato casualmente e impone che, in ogni caso, il random field non sia più breve di 96 bit.

3.5.3 Limiti sul numero di invocazioni di funzione

Lo standard impone dei limiti sul numero di invocazioni delle funzioni crittografiche nell'ambito della stessa chiave.

Concretamente viene richiesto che tutte le implementazioni che fanno uso di nonce costruiti deterministicamente con lunghezza inferiore a 96 bit o di nonce costruiti casualmente di qualsiasi lunghezza soddisfino il seguente requisito:

"Il numero totale di invocazioni della funzione di authenticated encryption non deve superare 2^{32}, comprese tutte le lunghezze di IV e tutte le istanze della funzione di authenticated encryption con una data chiave"

Nell'implementazione di agno in ogni caso si vuole supportare solo una costruzione di nonce.

Come sottolineato nello standard questo **requisito globale** può essere soddisfatto ponendo **limiti locali** su ogni stazione. In concreto si deve fissare un numero massimo di pacchetti che ogni stazione può inviare per fare in modo che il requisito sia globalmente valido.

In agno questo requisito non può essere soddisfatto efficacemente per due motivi:

1. Dato che non si conosce il numero massimo di plug-in che useranno una stessa chiave, non è possibile decidere in modo soddisfacente quali dovrebbero essere i limiti da imporre ai singoli plug-in.

2. Come detto in precedenza, si vuole evitare rinnovi di chiave obbligati. Per garantire che il requisito sia soddisfatto bisognerebbe infatti cambiare chiave quando non è più possibile soddisfare i limiti locali per qualche plug-in.

Inoltre notiamo che 2^{32} pacchetti sono relativamente poco traffico totale. Considerando pacchetti di 1500 byte si ha:

$$1500B \times 8 \times 2^{32} \approx 50000Gb$$

Nel caso si abbiano migliaia di macchine collegate ad una rete virtuale con stessa chiave 2^{32} pacchetti totali possono essere trasmessi molto velocemente, anche per il fatto che i pacchetti Ethernet possono essere considerabilmente più brevi di 1500 byte.

3.5.4 Costruzione dei nonce ad hoc

Dato che le modalità di costruzione pubblicizzate nello standard non sono soddisfacenti, bisogna provvedere a costruire i nonce in modo diverso, assicurandoci comunque che il requisito di unicità sia rispettato.

Costruzione casuale con timestamp

Un metodo potrebbe essere quello di adottare la costruzione casuale usando un timestamp come free field. Questo metodo di costruzione in effetti è considerabile come una costruzione casuale "non consigliata".

In concreto il nonce è il risultato di una concatenazione di un timestamp di 32 bit ed un numero casuale di non meno di 96 bit.

In linea teorica, se l'implementazione usa solo questo tipo di costruzione, il requisito precedente dovrebbe essere rilassato, ovvero si dovrebbe richiedere un tetto massimo di 2^{32} pacchetti inviati nell'arco di ogni secondo.

Rientrando però nel caso di costruzione casuale per come definita dallo standard, sorge il dubbio che, per qualche motivo non meglio specificato, il requisito debba valere per come è stato enunciato.

Costruzione deterministica con fixed field casuale

In questa metodologia di costruzione si genera casualmente il fixed field e si usa come invocation field un contatore di 64 bit.

Per dare forti garanzie di unicità del nonce, bisogna fare in modo che la probabilità che si verifichi una collisione nella generazione del fixed field sia molto bassa. Risulta evidente che, per rispettare il requisito di unicità enunciato in precedenza, 32 bit di fixed field siano troppo pochi. Infatti è sufficiente che due plug-in siano stati avviati con stessa chiave perché ci sia una probabilità di 2^{-32} che si sia verificata una collisione.

La probabilità che si verifichi una collisione cresce con l'aumentare del numero totale di plug-in che sono stati avviati con stessa chiave secondo le regole del problema del compleanno. Va presa nota del fatto che si considera il numero totale di plug-in avviati e non di quelli correntemente attivi.

Una costruzione del genere risulta inoltre molto efficiente in quanto il numero casuale, per quanto lungo sia, viene generato una sola volta.

3.5.5 Considerazioni

Risulta evidente che, qualunque sia la costruzione che si decide di adoperare, sia necessario l'uso di nonce più lunghi complessivamente di 96 bit.

Possibilmente questo si sarebbe voluto evitare, in quanto, per la modalità GCM, l'uso di nonce di 96 bit risulta molto più efficiente dell'uso di nonce di altre dimensioni [15].

Cercando di implementare un tale meccanismo con **GnuTLS 3.5.19** (current stable), ci è resi conto di un bug nell'implementazione. Se si prova a

chiamare la funzione gnutls_aead_cipher_encrypt() con un valore diverso da 12 byte come nonce_len si ottiene il seguente errore:

```
gcm.c:429: nettle_gcm_update: Assertion
‘ctx->auth_size % GCM_BLOCK_SIZE == 0’ failed.
```

Come si vede l'errore è dovuto ad un'asserzione fallita all'interno di una funzione parte della libreria Nettle. Indagando si è capito che questo errore è causato da un uso non corretto delle funzioni di update. Nella documentazione [20] viene spiegato che queste funzioni vanno chiamate prima dell'operazione di encrypt (o decrypt) e che tutte le chiamate consecutive a questa funzione, ad eccezione dell'ultima, devono specificare come lunghezza un multiplo del block size dell'algoritmo usato. Quella specifica asserzione non è verificata nel caso la funzione update sia usata due volte consecutivamente specificando come lunghezze due valori non multipli del block size.

Per aggirare il problema si potrebbe usare Nettle direttamente. Nettle tuttavia fornisce un'interfaccia di livello molto basso che risulta poco comoda da usare.

```
void gcm_aes128_set_key(struct gcm_aes128_ctx *ctx, const uint8_t *key)

void gcm_aes128_set_iv(struct gcm_aes128_ctx *ctx, size_t length, const
    uint8_t *iv)

void gcm_aes128_update(struct gcm_aes128_ctx *ctx, size_t length, const
    uint8_t *data)

void gcm_aes128_encrypt(struct gcm_aes128_ctx *ctx, size_t length,
    uint8_t *dst, const uint8_t *src)

void gcm_aes128_decrypt (struct gcm_aes128_ctx *ctx, size_t length,
    uint8_t *dst, const uint8_t *src)

void gcm_aes128_digest(struct gcm_aes128_ctx *ctx, size_t length,
    uint8_t *digest)
```

Come si può vedere, le funzioni non ritornano alcun valore e l'ICV (digest) non viene aggiunto automaticamente. Per verificare l'autenticità di un pacchetto bisogna esplicitamente calcolare l'ICV e confrontarlo con quello presente nel pacchetto ricevuto.

Una soluzione di così basso livello si rivelerebbe senza dubbio molto poco manutenibile. Per questo motivo si è scelto di valutare altre opzioni e di usare Nettle solo in caso non si trovi null'altro.

Oltre che cercare altre librerie, si considera anche di **cambiare algoritmo impiegato**. Questo potrebbe permettere di superare i limiti di efficienza di GCM-AES.

3.5.6 ChaCha20-Poly1305

ChaCha20 è uno stream chipher che risulta molto più veloce di AES per quanto riguarda le implementazioni software. Se combinato con l'autenticatore Poly1305 permette di fare authenticated encryption.

ChaCha20 è un discendente di Salsa20. ChaCha20-Poly1305 fu usato per la prima volta al posto di Salsa20-Poly1305 ed in seguito fu standardizzato dall'IETF con caratteristiche lievemente diverse [26].

XChaCha20 è una variazione di ChaCha20 che ne estende la lunghezza del nonce, applicando il procedimento usato per estendere Salsa20 in XSalsa20 [27]. La lunghezza del nonce di XChaCha20 è di 192 bit.

L'algoritmo non pone limitazioni pratiche sul numero di operazioni crittografiche eseguibili nell'ambito della stessa chiave. È possibile infatti cifrare fino a 2^{64} messaggi.

Nella tabella 3.5 si può vedere un confronto tra le tre versioni dei relativi algoritmi di authenticated encryption.

	ChaCha20-Poly1305	ChaCha20-Poly1305-IETF	XChaCha20-Poly1305
Key size	256 bit	256 bit	256 bit
Nonce size	64 bit	96 bit	192 bit
Block size	512 bit	512 bit	512 bit
MAC size	128 bit	128 bit	128 bit

Tabella 3.5: Confronto tra caratteristiche di ChaCha20-Poly1305

XChaCha20-Poly1305 possiede un nonce abbastanza lungo da poter applicare le costruzioni dei nonce di cui si è parlato precedentemente.

Libsodium

Questo algoritmo è implementato unicamente in versioni di libsodium successive alla 1.0.12. Dal momento che il contesto preso in considerazione non

pone requisiti di interoperabilità, questo non rappresenta un problema.

Libsodium, inoltre, è rilasciata con licenza **ISC**, compatibile con **GPL**.

3.6 Struttura delle operazioni

Per agno si userà quindi **XChaCha20-Poly1305** e il nonce sarà costruito con **costruzione deterministica con fixed field casuale**. Il nonce sarà quindi formato da un fixed field casuale di 128 bit e da un contatore di 64 bit come invocation field.

Capitolo 4

Valutazione e futuri sviluppi

4.1 Sicurezza di agno nel complesso

4.1.1 Probabilità di ripetizione dei nonce

Dato che si è scelto di usare come metodologia di costruzione dei nonce la costruzione deterministica con fixed field casuale, la probabilità che un nonce sia ripetuto equivale alla probabilità che ci sia una **collisione** nella generazione casuale dei fixed field. Questa dipende quindi dal numero totale di plug-in che sono stati avviati con la stessa chiave.

Eseguendo i calcoli basati sul problema del compleanno, si ottiene che per avere una probabità che avvenga una collisione sui fixed field di 2^{-32}, come richiesto dal requisito di unicità enunciato precedentemente, bisognerebbe generarne circa $3,9 \times 10^{14}$.

Questo numero è ampiamente sufficiente.

4.1.2 Lasciare agli utenti controllo sulle chiavi

Dato che la probabilità che si verifichi una collisione nella generazione dei fixed field è trascurabile, la sicurezza di agno dipende completamente dalla segretezza della chiave utilizzata.

Si sono analizzate precedentemente le azioni concrete da dover intraprendere per garantire la segretezza della chiave. Nonostante queste siano molto semplici da applicare, l'imprevedibilità del comportamento degli utenti potrebbe rivelarsi essere il tallone di Achille di agno. L'esposizione delle chiavi di sessione all'utente

è il prezzo da pagare se si vuole avere uno strumento che richieda configurazione minima. Come visto precedentemente, infatti, un meccanismo di distribuzione della chiave distribuito sarebbe scomodo e non risulterebbe comunque essere una soluzione soddisfacente. Se si vuole evitare l'esposizione della chiave agli utenti l'unico modo è quello di fare in modo che agno negozi la chiave di sessione con un server che deve essere configurato ed avviato precedentemente.

Anche se al momento non lo si ritiene necessario, un possibile sviluppo di agno potrebbe essere quello di inserire quest'ultima funzionalità come opzione.

4.1.3 VXLAN hop in VXVDE

In VXVDE l'header della VXLAN del pacchetto UDP incapsulante non ha alcuna garanzia di integrità. Questo significa che è possibile, da parte di un utente con accesso alla rete fisica, fare VLAN hop con le VXLAN. Questo non risulta essere un problema di sicurezza se le VXLAN sono cifrate con **chiavi differenti** in quanto il traffico verrà scartato dal plug-in agno durante la ricezione.

Come detto in precedenza, è importante che le chiavi usate da agno siano usate per un solo scopo.

4.2 Performance

Per valutare la performance sono stati eseguiti dei test su due macchine Dell OptiPlex 760 dotate di CPU Intel Core2 Duo con frequenza 3GHz, 2GB RAM, controller Intel 82567LM Gigabit, con Debian stretch/sid come sistema operativo. Le due macchine sono connesse da uno switch NetGear FS728TP configurato per supportare pacchetti baby giant. vdeplug4, VDE2 e vdens sono stati installati dai repository ufficiali del progetto [4], [3], [5]. Il codice di libvdeplug_agno è stato scaricato dal repository ufficiale [1] e ne sono stati compilati i branch gnutls_static e libsodium_static.

La struttura dei test si può vedere in figura 4.1. Come switch sono stati usati dei vde_switch di VDE2. A questi si collegano due vdens all'interno dei quali vengono eseguiti i programmi di testing (ping ed iperf3[28]). Il VDE cable che unisce i due switch cambia a seconda del test.

La tabella 4.1 riassume i risultati ottenuti.

I vdens hanno interfacce di rete con 1500 come MTU, mentre le macchine fisiche 2000. Questo è stato fatto per evitare che i pacchetti che incapsulano il traffico della rete virtuale fossero frammentati.

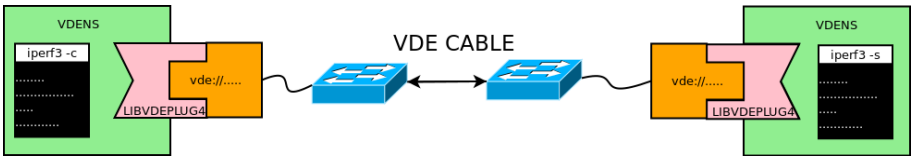

Figura 4.1: Struttura dei test

VDE cable	Bandwidth	RTT
udp	658 Mbps	0.311 ± 0.024 ms
agno (XChaCha) + udp	431 Mbps	0.330 ± 0.024 ms
agno (AES) + udp	308 Mbps	0.407 ± 0.032 ms
ssh	510 Mbps	0.460 ± 0.037 ms
vde_cryptcab	110 Mbps	0.621 ± 0.034 ms

Tabella 4.1: Risultati dei test

udp Il test è eseguito usando libbvdeplug_udp per realizzare il VDE cable. Il VDE cable creato non è cifrato. I risultati sono forniti come termine di paragone per valutare l'overhead introdotto dalle operazioni crittografiche degli altri metodi.

agno (XChaCha) + udp Il traffico viene criptato da agno con l'algoritmo XChaCha20-Poly1305 utilizzando libsodium e trasmesso tramite il plug-in libvdeplug_udp.

agno (AES) + udp Il traffico viene criptato da agno con l'algoritmo GCM-AES-128 utilizzando la libreria GnuTLS. Il nonce viene costruito con costruzione deterministica con fixed field casuale di 32 bit. Questo metodo, come detto in precedenza, non fornisce garanzie di sicurezza adeguate. I risultati sono forniti per confrontare la performance di GCM-AES-128 con quella di XChaCha20-Poly1305.

ssh Il VDE cable è realizzato usando il plug-in libvdeplug_cmd per redirezionare il traffico verso il programma ssh il quale lo trasmette ad un vde_plug remoto collegato allo switch di destinazione.

vde_cryptcab Si usa vde_cryptcab come VDE cable.

Come si può vedere, quella di adoperare XChaCha20-Poly1305 è stata una scelta che ha avuto risultati positivi anche dal punto di vista dell'efficienza. L'overhead introdotto dalle operazioni crittografiche di agno riduce del 35% il bandwidth di udp.

vde_cryptcab risulta di gran lunga inferiore a tutti gli altri strumenti utilizzati. Questo significa che agno può sostituirlo.

L'uso di ssh risulta essere la soluzione migliore in quanto permette di raggiungere un bandwidth maggiore di quello di agno con XChaCha20-Poly1305 di quasi 80 Mbps. Agno tuttavia risulta avere un round trip time ICMP inferiore.

Questi risultati non combaciano completamente con ciò che ci si sarebbe aspettato. In particolare ci si sarebbe aspettato che agno, usato con udp, e vde_cryptcab si sarebbero rivelati entrambi più efficienti di ssh. Questo perché usando una connessione UDP per incapsulare traffico TCP non introducono overhead non necessario. Se si vuole che agno diventi più efficiente di ssh ulteriori approfondimenti sono necessari.

Va notato che agno per funzionare non necessita di un account sulla macchina remota e della presenza di un server ssh nel caso di connessioni point-to-point e che risulta insostituibile se usato con VXVDE.

4.3 Altri futuri sviluppi

4.3.1 Agno

Agno vuole essere l'unico componente crittografico di VDE4. Come tale si può volerlo modificare in base ai nuovi progetti di VDE4, primo fra i quali lo switch. Come detto in precedenza, per il momento un meccanismo a chiave statica sembra sufficiente.

Point-to-point

Siccome agno, se usato con udp, risulta comunque meno efficiente di ssh si potrebbe pensare di rendere disponibili ottimizzazioni per le connessioni point-to-point. Usare ChaCha20-Poly1305-IETF in questi casi (nonce di 12 bit) porterebbe ad un incremento di velocità senza particolari rischi di sicurezza. Il problema è che gli utenti potrebbero usare questa opzione scorrettamente con VXVDE, creando un problema di sicurezza.

Derivare la chiave da una passphrase

Si potrebbe aggiungere un opzione che permetta ad agno di derivare la chiave da usare a partire da una passphrase. Libsodium fornisce delle primitive apposite per questa operazione.

Integrità senza confidenzialità

Nel caso in cui garanzie di confidenzialità sul traffico non siano richieste potrebbe essere utile permettere ad agno di autenticare il traffico senza cifrarlo.

4.3.2 Identificare i vde_plug

Durante lo sviluppo di agno ci si è resi conto che sarebbe utile, non solo per agno, avere un meccanismo che permetta di identificare i processi vde_plug in esecuzione, in modo da esplicitare quale sia la loro funzione in una rete e quali plug-in hanno in funzione.

Conclusioni

In questo lavoro si è introdotto libvdeplug_agno, un plug-in per libvdeplug4 che permette di cifrare il traffico virtuale. È stata presentata un'implementazione che raggiunge gli obiettivi posti per quanto riguarda la sicurezza e la flessibilità. Agno rende obsoleto il vecchio vde_cryptcab e dà la possibilità di usare VXVDE senza togliere agli utenti l'accesso alla rete reale.

L'overhead introdotto dalle operazioni crittografiche è in ogni caso notevole. Per questo motivo bisognerebbe strutturare l'intero sistema in maniera tale da ricorrere all'uso di crittografia il meno possibile.

Appendice A

Appendice

Ulteriori informazioni sulle possibilità e sui casi d'uso di agno si possono reperire nel file README.md presente sul repository ufficiale [1]. Al momento di scrittura dell'elaborato il file più informativo si trova sul branch libsodium_static.

Bibliografia

[1] https://github.com/rd235/vdeplug_agno.

[2] "Virtual Square: all the virtuality you always wanted but you were afraid to ask", http://wiki.v2.cs.unibo.it/wiki/index.php?title=Main_Page.

[3] R. Davoli, "VDEv2: Virtual Distributed Ethernet.", https://github.com/virtualsquare/vde-2

[4] R. Davoli, "Vde: Virtual distributed ethernet. plug your vm directly to the cloud.", https://github.com/rd235/vdeplug4.

[5] R. Davoli, "vdens: Create User Namespaces connected to VDE networks", https://github.com/rd235/vdens.

[6] R. Davoli, "A slirp plugin for vdeplug4", https://github.com/rd235/vdeplug_slirp.

[7] M. Mahalingam, D. Dutt, K. Duda, P. Agarwal, L. Kreeger, T. Sridhar, M. Bursell, and C. Wright, "Virtual eXtensible Local Area Network (VXLAN): A Framework for Overlaying Virtualized Layer 2 Networks over Layer 3 Networks", RFC 7348 (Informational), Internet Engineering Task Force, agosto 2014. Available: http://www.ietf.org/rfc/rfc7348.txt.

[8] R. Davoli and M. Goldweber, "VXVDE: A switch-free VXLAN replacement", in 2015 IEEE Globecom Workshops, San Diego, CA, USA, December 6-10, 2015.

[9] R. Davoli, "Vxvdex: connect distributed private network namespaces", https://github.com/rd235/vxvdex.

[10] R. Davoli, "VXVDEX: Internet of threads and networks of namespaces", in: IEEE International Conference on Communications.

[11] R. Davoli, "LIBPAM-NET: create/join network namespaces at login", https://github.com/rd235/libpam-net.

[12] IEEE Standard for Local and metropolitan area networks, "Media Access Control (MAC) Security", 2006.

[13] Dworkin, M., "NIST Special Publication 800-38D: Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC.", U.S. National Institute of Standards and Technology, November 2007, http://csrc.nist.gov/publications/nistpubs/800-38D/SP-800-38D.pdf.

[14] McGrew, D., "An Interface and Algorithms for Authenticated Encryption", gennaio 2008, https://tools.ietf.org/html/rfc5116.

[15] Rogaway P., "Evaluation of Some Blockcipher Modes of Operation", febbraio 2011, http://web.cs.ucdavis.edu/~rogaway/papers/modes.pdf

[16] IEEE Standard for Local and metropolitan area networks, "Port-Based Network Access Control", 2010, https://standards.ieee.org/standard/802_1X-2010.html.

[17] IEEE Standard for Local and metropolitan area networks, "Media Access Control (MAC) Security Amendment 1: Galois Counter Mode, Advanced Encryption Standard, 256 (GCM-AES-256) Cipher Suite", 2011, https://standards.ieee.org/standard/802_1AEbn-2011.html.

[18] "OpenSSL: Cryptography and SSL/TLS Toolkit", https://www.openssl.org/.

[19] Free Software Foundation, "The GnuTLS Transport Layer Security Library", https://www.gnutls.org/.

[20] Niels Möller, "Nettle: a low-level cryptographic library", https://www.lysator.liu.se/~nisse/nettle/nettle.html

[21] Eastlake D., "Randomness Requirements for Security", RFC 4086, Internet Engineering Task Force, giugno 2005, https://tools.ietf.org/html/rfc4086.

[22] D. Wallner, E. Harder e R. Agee, "Key Management for Multicast: Issues and Architectures", RFC 2627, Internet Engineering Task Force, giugno 1999, https://tools.ietf.org/html/rfc2627

[23] A. Ballardie, "Scalable Multicast Key Distribution", RFC 1949, Internet Engineering Task Force, maggio 1996, https://tools.ietf.org/html/rfc1949.

[24] H. Harney e C. Muckenhirn, "Group Key Management Protocol (GKMP) Architecture", RFC 2094, Internet Engineering Task Force, luglio 1997, https://tools.ietf.org/html/rfc2094.

[25] Ferguson N., "Authentication weaknesses in GCM", 2005, https://csrc.nist.gov/csrc/media/projects/block-cipher-techniques/documents/bcm/comments/cwc-gcm/ferguson2.pdf

[26] Nir Y. and Langley A., "ChaCha20 and Poly1305 for IETF Protocols", RFC 7539, Internet Engineering Task Force, maggio 2005, https://tools.ietf.org/html/rfc7539

[27] Daniel J. Bernstein, "Extending the Salsa20 nonce", https://cr.yp.to/snuffle/xsalsa-20081128.pdf

[28] J. Dugan, S. Elliott, B. A. Mah, J. Poskanzer, and K. Prabhu, "iperf3", http://iperf.fr/.

Ringraziamenti

Ringrazio il Prof. Renzo Davoli per il tempo dedicatomi durante la preparazione della tesi e per avermi fatto precedentemente scoprire la bellezza dell'Informatica.

Ringrazio infinitamente i miei genitori e tutta la mia famiglia, dal momento che sono loro la ragione per cui ho potuto frequentare l'università.

Ringrazio i miei amici e colleghi Patrick Guidetti e Francesco Moneta, perché con la loro determinazione ed il loro coraggio mi hanno sempre ispirato a dare del mio meglio.

Ringrazio tutti quelli che durante il corso della mia vita si sono dimostrati degli amici e spero di aver fatto lo stesso con loro.