

# IPSec and IP トンネリング on Linux

早川 仁 (cz8cb01@linux.or.jp)

宮本 久仁男 (kmiya@coe.nttdata.co.jp)

## 概要

IPSec とはデータの改竄や盗聴を防ぐことができない<sup>1</sup> 通常の IP 通信に、IP パケットの認証や暗号化を行う機能を追加したセキュリティプロトコルである。IPSec は TCP/IP プロトコルにおけるインターネット層で提供されるため、IP および上位レイヤの通信を安全に行うことを可能とする。一方トンネリングは様々なレイヤで提供される性質のものであり<sup>2</sup>、トンネル処理の内容も様々である。本セッションでは IPSec およびトンネリングのアーキテクチャ、Linux における IPSec の実装である FreeS/WAN、IP トンネルの実装の 1 つである Zebedee に関する解説と比較を行なう。なお本セッションで言う IP トンネルとは、アプリケーション層におけるトンネリングソフトウェアのことを指している。プロトコルや実装の制限に関する記述は 2000 年 11 月 06 日現在のものである。

## 1 TCP/IP について

TCP/IP は図 1 のようなレイヤ構造になっている。

このように階層構造にすることにより、各層は上下の層のみとやり取りを行なえばよいために設計が簡単になり、また相手の層とのインターフェースが変更されない限り、各層の設計が変更になってもその影響を受けない。

例えばネットワーク層が TokenRing の環境で作成された、(アプリケーション層で動作する) 一般的なアプリケーションの実装は Ethernet の環境であっても変更する必要はない<sup>3</sup>。

<sup>1</sup>そもそも仕様として考慮されていない

<sup>2</sup>例えばネットワークインターフェース層の L2TP、インターネット層の Tunneling mode の IPSec、アプリケーション層におけるアプリケーションレベルでの実装 (SSH の port forwarding や Zebedee など)

<sup>3</sup>Sniffer のような下層の設計に依存したアプリは除く

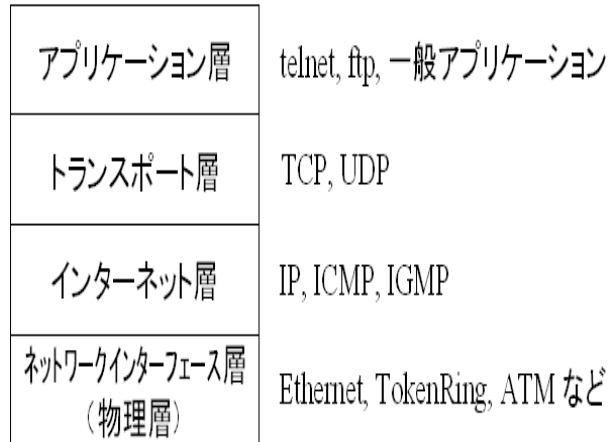


図 1: TCP/IP プロトコルスタック

ここで認証や暗号化の機能を実装する層による影響を考えてみると、次のような特徴がある。

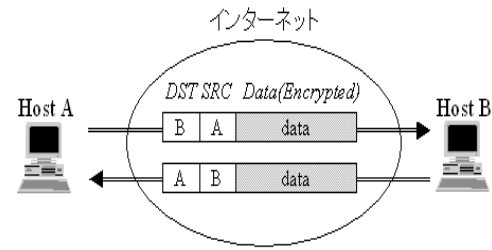
- アプリケーション層
  - 自由度は非常に高いが、セキュリティ機能の実装はアプリケーションごとに別々に実装する必要がある。
- トランスポート層
  - SSL や TLS のような共通のしくみ (ライブラリ) を用意することにより、各アプリケーションで必要な作業はセキュリティ機能を呼び出すだけである。
  - アプリケーションの修正を行なう必要がある。
  - トランスポートプロトコルごと (TCP, UDP) に実装が必要。通常 SSL や TLS は UDP に対応していない。
- インターネット層
  - セキュリティ機能を実装するのに最適

- 複数のセキュリティプロトコルを実装する作業は不必要
- アプリケーションを修正する必要はない。
- 主にノードの識別を行なうことを目的としている層のため、個々のユーザーの認識は性質上困難である。

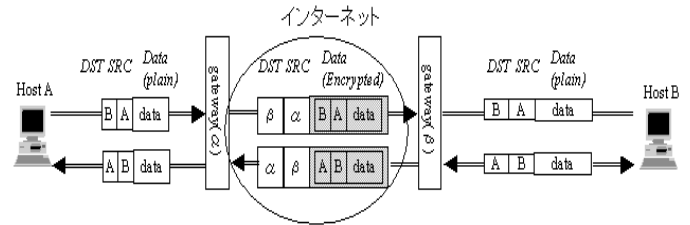
- ネットワークインターフェース層

- Ethernet, TokenRing, ATM, シリアルなど実際の伝送路とリンクした層で、I/O デバイスとそのデバイスドライバを合わせて、この層は構成される。

TCP/IP に関する詳細に関しては [2, 22] などが参考になる。



(1) Transport mode



(2) Tunnel mode

図 3: Transparent mode と Tunnel mode

## 2 IPSec について

IPSec はインターネット層を拡張することで、IP 通信に認証および暗号機能を追加する。明確な境界を表現することは困難だが、図 2 のようにインターネット層とトランスポート層の間に位置する。これは ICMP と IP の関係を想像するとわかりやすいだろう。

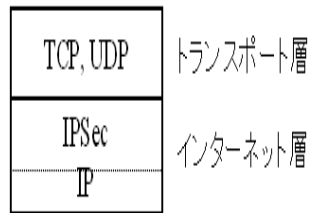


図 2: インターネット層における IPSec と IP の関係

### 2.1 Transparent mode と Tunneling mode

IPSec の規格では、ホスト間通信の認証・暗号化を行う Transparent mode(図 3 の 1) と、ゲートウェイ間通信の認証・暗号化を行う Tunnel mode(図 3 の 2) が規定されている。

図 3 のパケットの図からわかる通り、Transparent mode ではホスト自身で認証・暗号化を行なう

が、Tunnel mode では中間に存在するゲートウェイ自身が処理を行なうため処理が透過的に行われ、ゲートウェイに接続されている機器では特に何もする必要はない。

Host A および Host B 間の送受信パケットは gateway  $\alpha$ ,  $\beta$  を通る時には認証・暗号化処理が行なわれているが、gateway を抜けて相手のネットワークにパケットが届いた時には、パケットは送信時と同じ状態に戻っている。このことを利用することにより、Tunnel mode では図 4 のような仮想のネットワークを構成できる。両者のネットワーク間のパケットはインターネット上を流れることになるが、そのパケットは gateway により認証・暗号化が行なわれるためネットワーク同士が直接、専用の回線で接続されていると見なすことが可能である。

### 2.2 IPSec のプロトコル

IPSec はインターネット層の IP パケットを認証・暗号しているということは既に述べた。この認証および暗号を行なうプロトコルとして AH および ESP を定義している。また鍵の交換として IKE というプロトコルを利用する。

AH と ESP は IP パケットに対して下記の機能を提

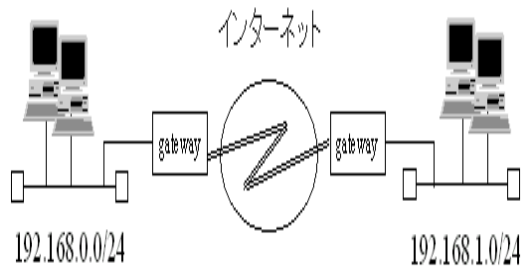


図 4: Tunnel mode による VPN

供するが、どちらか一方、あるいは両方を同時に利用することが可能である。

- AH
  - 認証
  - Replay Attack に対する防御
- ESP
  - 認証
  - Replay Attack に対する防御
  - データの暗号化

### 2.2.1 認証について

認証により、受信した IP パケットが通信の途中で第三者に改竄されていないことを保証できる。また認証の副産物としてデータの信頼性を保証しない UDP の通信においても信頼性を保証することができるが、UDP でパケットを送受信するのは、パケットの信頼性が低くてもあまり支障がない動画のストリーム転送や、アプリケーション層で信頼性を確保している NFS といった通信で使用されるため、多くの場合 IPSec の AH や ESP で UDP に信頼性を保証しても処理のオーバーヘッドが増すだけである<sup>4</sup>。

認証のアルゴリズムは HMAC-MD5(RFC 2403[10]) および HMAC-SHA(RFC 2404[11]) の実装が義務づけられているが、その他のアルゴリズムに関しては AH や ESP の仕様では定義されていない。これは利用しているアルゴリズムが安全ではなくなった、あるいは

<sup>4</sup>NFS のような暗号化を行っていないアプリケーションの場合、ESP によるデータの暗号化は利用価値があるだろう

より強固なアルゴリズムが開発された際に簡単にアルゴリズムを変更できるようにするためである。

IPSec ver.1.0 の仕様では ESP には認証の機能はなかったが、IPSec ver.2.0 では ESP でも認証を行えるようになった。ver.1.0 の ESP に関しては RFC 1827[6] を、ver.2.0 の ESP は RFC 2406[13] を参照の事。

ESP に認証の機能が追加されたことによって、暗号化をせずパケットの認証のみを行う AH の存在意義についての議論もあるが、暗号技術の利用が法律で規制されている国<sup>5</sup>や、ローカルの通信を AH で認証し、そのパケットを ESP でトンネリングする場合など ESP と AH を入れ子で使うことがあるため、両者は今後も共存していくと思われる。

### 2.2.2 Replay Attack

Replay Attack とは、通信の途中のパケットを第三者が (不正な方法で) 盗み、その盗んだパケットを本来の相手に送信することにより相手の本来の処理の妨害、なりすましの攻撃を行うことである。図 5 の例ではホスト A からホスト B への通信パケットをホスト X も受信している (盗んでいる)。ホスト A から B への全てのパケットを記録しておき、あとでホスト X があたかもホスト A のように IP アドレスを詐称してパケットを送信 (つまり replay) することにより、ホスト B にホスト A からのパケット送信と誤認させることが可能になる。

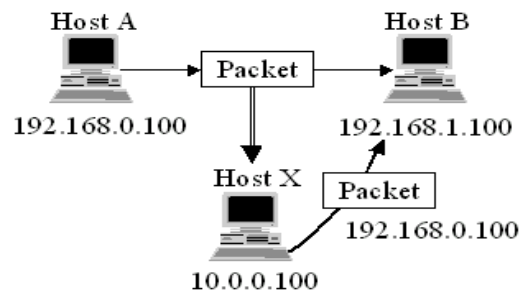


図 5: Replay Attack の例

これに対応するために、AH と ESP ではシーケンス番号とスライド受信ウィンドウを使用して Replay Attack に対する防御を行っている。

<sup>5</sup>今現在のステータスは未調査だが、以前フランスなどでは民間人の暗号使用が制限されていた

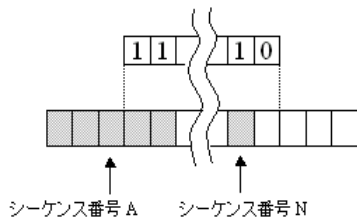


図 6: スライド受信ウィンドウの例

スライド受信ウィンドウは現在受信中のシーケンス番号の範囲を指している。受信したパケットのシーケンス番号がウィンドウの範囲よりも小さい場合、不正なデータとして単純に破棄される。図 6 では、スライド受信ウィンドウはシーケンス番号 A+1 ~ シーケンス番号 N+1 の範囲を指している。この場合シーケンス番号が A 以下のパケットを受信した時には単純に破棄されることになる。逆にウィンドウの範囲よりも大きいシーケンス番号を持ったパケットを受信した時には、そのパケットがウィンドウの範囲に入るようにウィンドウが右へスライドする。

このシーケンス番号と TCP ヘッダのシーケンス番号の間には何の関係もないため、ウィンドウがスライドして歯抜けになっても通信には何の問題もない。またスライド受信ウィンドウを使う場合、AH もしくは ESP によるパケットの認証が必須となる。これは大きいシーケンス番号を持ったパケットを偽造することによりウィンドウが右へスライドしてしまい、有効なはずのシーケンス番号が破棄されてしまうのを防ぐためである。

### 2.2.3 データの暗号化

データの暗号化は AH では提供されていない。ESP では 56bit 長の鍵を使用する DES-CBC(RFC 2405[12]) の実装が義務づけられているが、認証と同様にその他のアルゴリズムに関しては定義されていない。

ESP を利用する際に認証アルゴリズムを指定しないことを可能にする実装も多いが、その時には AH を併用することが必須となる。ESP パケットは図 7 のようになっているが、シーケンス番号や SPI<sup>6</sup> といった

<sup>6</sup> 使用している認証アルゴリズム、暗号化アルゴリズムを格納する SAD という一種のデータベースに対するインデックス値のことである。認証や復号化をするためには、まず使用されているアルゴリズムを知る必要がある

データは ESP ヘッダに格納されているため、認証を行わない場合には少なくとも Replay Attack に対する防御は不可能になる<sup>7</sup>。このため、認証なしでの暗号化はセキュリティ上、非常に問題のある設定と言える。

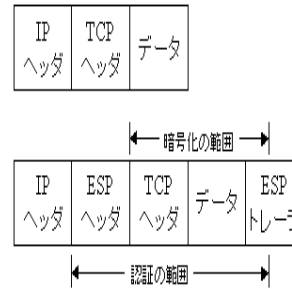


図 7: 通常の IP パケットと ESP パケット

## 2.3 鍵交換プロトコル IKE について

IKE(RFC 2409[18]) とは鍵交換のためのプロトコルであり、UDP のポート 500 で通信を行う。これは AH や ESP とは違い汎用的なプロトコルのため、IPSec だけではなく SNMPv3, OSPFv2, SSH など他にも数多くのプロトコルで利用されている。IKE は 3 つのプロトコル ISAKMP の基本部分 (RFC 2408[17])、Oakley(RFC 2412[17]) のモード、SKEME(RFC 2409<sup>8</sup>) の鍵交換を組み合わせて作成されたものである。

鍵交換の最大の目的は、インターネットなど安全ではないネットワークを通る場合でも安全に鍵を交換することと、定期的に鍵交換を行うことにより将来の安全を確保することにある。その場合、仮に鍵が破られたとしても、鍵交換により現在使用している鍵は既にその鍵とは違っている可能性が高い。これにより破られた鍵で暗号化されたデータは解読されてしまうが、その他のデータはまだ安全である。この事を一般的に Perfect Forward Secrecy - PFS と呼ぶ。

鍵交換のスキームとしては主に Diffie-Hellman 法が使われる。以下は Alice と Bob が Diffie-Hellman 法で鍵交換を行う際の式である。

$g$  および  $p$  は最初に両者で合意した 2 つの数値で、これは漏洩しても構わないためインターネットなど安

<sup>7</sup> シーケンス番号の詐称が可能であり、詐称であることを検証する手段が存在しないため

<sup>8</sup> SKEME 自身に関して記載された RFC は無いようだが IKE の RFC 2409 に記述がある

全ではないネットワーク経由で値を交換してもよい。a および b は両者がそれぞれ無作為に選択した秘密の値で、private key あるいは秘密鍵と呼ばれる。

$$A = g^a \text{ mod } p \qquad B = g^b \text{ mod } p$$

ここで算出された A および B は公開しても構わない値で、public key あるいは公開鍵と呼ばれる。この鍵 A B と private key a b の間には以下のような関係がある。

$$B^a \text{ mod } p = g^{ab} \text{ mod } p = A^b \text{ mod } p$$

つまり、例えば Alice が private key a を Bob に渡したい場合には Bob の public key B を元に  $B^a \text{ mod } p$  を計算した結果 (X とする) を Bob に渡す。Bob は受けとった X を元に、 $X = g^{ab} \text{ mod } p$  とする a の値を計算することにより、Alice の private key a を安全に受けとることができる。

RSA 暗号の public key, private key を利用しても同様にセキュアな鍵交換が可能になるが、RSA 暗号の仕様では法 (modulus)<sup>9</sup>より短いデータしか暗号化できず、変換結果は法と同じ長さになってしまう。法の長さは RSA 暗号の bit 長に依存している。Diffie-Hellman 法の場合にはそのような制約は存在しない。

### 3 FreeS/WAN — IPSec の実装

FreeS/WAN[3] とは、Linux における IPSec の実装の 1 つである。FreeS/WAN では KLIPS(Kernel IPSEC) が AH および ESP などパケットに関する全てを、Pluto daemon が IKE を実装している。KLIPS はカーネル内部で動作しているため、FreeS/WAN を使用するためにはカーネルにパッチをあてる必要がある。FreeS/WAN は今まで述べてきた IPSec の実装なので、IPSec の特徴がそのまま当てはまる。一般的には Tunneling mode で security gateway として使われることが多い。

鍵交換の方式は manual(鍵は固定) と auto(Pluto daemon 経由による鍵交換) があるが、manual はテスト時の疎通確認のみの使用を推奨している。Pluto

<sup>9</sup>上記の式の場合 A, B, a, b のこと

は Diffie-Hellman 法と RSA 暗号 (FreeS/WAN ver.1.4 以降) の両者に対応している。

ipchains による IP masquerade を実行しているゲートウェイ上で FreeS/WAN も実行することにより IP masquerade と IPSec を併用することは可能だが、Firewall との相性はあまり良くない。これに関しては 6 章で述べる。2000 年 11 月 06 日現在の最新版は ver.1.5, Snapshot 版は ver.1.7 だが、テスト用途であっても Snapshot 版の使用は勧められない。

## 4 Zebedee — IP トンネルの実装

### 4.1 IP トンネル

IPSec が使用できない状況というのはいくつか考えられるが、もっともありがちなものは、Firewall によるパケット書き換えが行われる環境である。このような場合、認証段階でパケットが改竄されていると見なされ通信ができない。

このような場合に活用出来るものが、アプリケーション層で実装されるトンネルツールである。トランスポート層以下の通信が正常に行える状況下においては、この層で実装されるトンネルツールが有用である。

### 4.2 Zebedee とは

Zebedee という名前は Zlib(圧縮), Blowfish[20](暗号化), Diffie-Hellman(鍵交換) の頭文字 (ZBD) をもじってつけられている。Zebedee は暗号化された IP トンネルを提供するツールである。同様のツールには ssh があるが、これについてはもともと安全なりモータールという位置付けで IP トンネルとしても使えるものの、

- 多機能ゆえに実装が複雑になっている
- シェルの使用が前提になっているため、クライアントを利用する側にも、ある程度のスキルが要求される
- 確立された単一の ssh コネクションを複数のトンネリングに使う場合、そのコネクションが切断された際にはすべてのトンネルが道連れになる

という問題がある。それに対し Zebedee は、

- 最初から IP トンネルツールとして作られているため、実装がシンプル
- 配布元より Windows 版のモジュールも配布されているため、クライアント側での導入が楽
- インストール時に工夫すれば、クライアント側で Zebedee の存在を意識する必要がない
- 1 トンネル 1 コネクションという実装になっているため、ssh で発生しうるような悲劇は起こりにくい

となっておりトンネル機能の提供を行うには、うってつけのものである。

Zebedee の提供する機能は、以下の通りである。

- IP トンネル
- blowfish アルゴリズムによるデータ暗号化
- zlib, bzip2 によるデータ圧縮

2000 年 11 月 06 日現在の最新版は ver.2.0.1, 開発版は ver.2.1.3 である。

### 4.3 簡単な使用例

Zebedee はサーバ/クライアントのホスト名以外の指定を行わない場合 telnet ポートに対するトンネルを提供する。サーバ chitei に対し、クライアント driller が Zebedee 経由で接続を使用とする場合のコマンドラインは以下ようになる。

```
サーバ側： zebedee -s
クライアント側： zebedee chitei
```

このようにすることで、driller と chitei の間で暗号化トンネルが開設される。

### 4.4 実用的な例

4.3 の例ではサーバ側で Zebedee を起動している場合、クライアント側でも Zebedee を用いられると誰でもサーバに接続できてしまうという問題がある。これを防ぐために Zebedee は秘密鍵／公開鍵 (FingerPrint) を用いた認証を用いることが可能である。なお、これらの鍵は Zebedee を用いて生成する。

秘密鍵生成

```
zebedee -p > private.key
```

FingerPrint 生成

```
zebedee -P -f private.key > public.id
```

としてやり、FingerPrint を何らかの方法でサーバ側にインストールすることで、FingerPrint と秘密鍵のペアを持ったサーバ/クライアントのペア以外の通信をはじくことが可能である。

## 5 パケットダンプによる実装比較

ここでとりあげるパケットダンプは図 8 の環境で no4 から telnet no1 を実行し、ID hoge, password moge でログインして ls -l /testdir を実行した時のものである。/testdir には下記ファイルが存在する。パケットのダンプには linux\_sniffer を使用した。

```
root root 0 Oct 30 17:32 a
root root 1024 Oct 30 17:34 b
root root 1048576 Oct 30 17:34 c
```

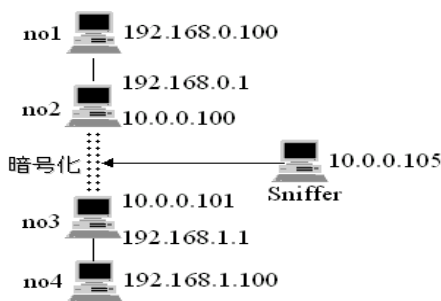


図 8: 機器構成

### 5.1 平文のダンプ結果

FreeS/WAN や Zebedee を実行していない (つまり暗号化されていない) 状態での telnet のダンプの抜粋を示す。

```
proto: 080000:06:29:d5:1b:62->00:06:29:38:4b:79 192.168.0.100[23]->192.168.1.100[1081]
0000 01 01 08 0a 01 1d f4 c7 - 01 0b 27 37 6c 6f 67 69 .....?logi
0010 6e 3a 20 - n:

proto: 080000:06:29:38:4b:79->00:06:29:d5:1b:62 192.168.1.100[1081]->192.168.0.100[23]
0000 01 01 08 0a 01 0b 27 7e - 01 1d f4 c7 68 .....?.....h
proto: 080000:06:29:38:4b:79->00:06:29:d5:1b:62 192.168.1.100[1081]->192.168.0.100[23]
0000 01 01 08 0a 01 0b 27 84 - 01 1d f5 0e 6f .....?.....o
proto: 080000:06:29:38:4b:79->00:06:29:d5:1b:62 192.168.1.100[1081]->192.168.0.100[23]
0000 01 01 08 0a 01 0b 27 8e - 01 1d f5 14 67 .....?.....g
```

```

proto: 080000:06:29:38:4b:79->00:06:29:d5:1b:62 192.168.1.100[1081]->192.168.0.100[23]
0000 01 01 08 0a 01 0b 27 97 - 01 1d f5 1e 65 .....e

proto: 080000:06:29:38:4b:79->00:06:29:38:4b:79 192.168.0.100[23]->192.168.1.100[1081]
0000 01 01 08 0a 01 1d f5 31 - 01 0b 27 a2 50 61 73 73 .....Pass
0010 7f 6f 72 64 3a 20 - word:

proto: 080000:06:29:38:4b:79->00:06:29:d5:1b:62 192.168.1.100[1081]->192.168.0.100[23]
0000 01 01 08 0a 01 0b 27 b8 - 01 1d f5 31 6d .....lm
proto: 080000:06:29:38:4b:79->00:06:29:d5:1b:62 192.168.1.100[1081]->192.168.0.100[23]
0000 01 01 08 0a 01 0b 27 bd - 01 1d f5 4a 6f .....Jo
proto: 080000:06:29:38:4b:79->00:06:29:d5:1b:62 192.168.1.100[1081]->192.168.0.100[23]
0000 01 01 08 0a 01 0b 27 c5 - 01 1d f5 4f 67 .....0g
proto: 080000:06:29:38:4b:79->00:06:29:d5:1b:62 192.168.1.100[1081]->192.168.0.100[23]
0000 01 01 08 0a 01 0b 27 e0 - 01 1d f5 57 65 .....We

proto: 080000:06:29:d5:1b:62->00:06:29:38:4b:79 192.168.0.100[23]->192.168.1.100[1081]
0000 01 01 08 0a 01 1d f5 74 - 01 0b 27 e5 4c 61 73 74 .....t.Last
0010 20 6c 6f 67 69 6e 3a 20 - 57 65 64 20 4e 6f 76 20 Login: Wed Nov
0020 20 31 20 31 38 3a 30 39 - 3a 33 31 20 66 72 6f 6d 1 18:09:31 from
0030 20 6e 6f 34 0d 0a - no4..

proto: 080000:06:29:d5:1b:62->00:06:29:38:4b:79 192.168.0.100[23]->192.168.1.100[1081]
0000 01 01 08 0a 01 1d f5 ad - 01 0b 29 1e 74 6f 74 61 .....).tota
0010 6c 20 31 30 33 32 0d 0a - 2d 72 77 2d 72 2d 2d 72 1 1032...rw-r-r
0020 2d 2d 20 20 20 31 20 72 - 6f 6f 74 20 20 20 20 20 20 -- i root
0030 72 6f 6f 74 20 20 20 20 - 20 20 20 20 20 20 20 20 20 root
0040 30 20 4f 63 74 20 33 30 - 20 31 37 3a 33 32 20 61 0 Oct 30 17:32 a
0050 0d 0a 2d 72 77 2d 72 2d - 2d 72 2d 2d 20 20 20 31 ..-rw-r-r- i
0060 20 72 6f 6f 74 20 20 20 - 20 20 72 6f 6f 74 20 20 root
0070 20 20 20 20 20 20 31 - 30 32 34 20 4f 63 74 20 1024 Oct
0080 33 30 20 31 37 3a 33 34 - 20 62 0d 0a 2d 72 77 2d 30 17:34 b...rw-
0090 72 2d 2d 72 2d 2d 20 20 - 20 31 20 72 6f 6f 74 20 r-r- i root
00a0 20 20 20 20 72 6f 6f 74 - 20 20 20 20 20 20 31 30 root 10
00b0 34 38 35 37 36 20 4f 63 - 74 20 33 30 20 31 37 3a 48576 Oct 30 17:
00c0 33 34 20 63 0d 0a 5b 68 - 6f 6f 65 40 6e 6f 31 20 34 c. [hoge]noi
00d0 68 6f 67 65 5d 2d 20 - hoge]#

```

ログインプロンプト、入力した ID, password, telnet の出力が全て平文のまま流れていることがわかる。また、通信をしているホストの IP アドレス (192.168.1.100 と 192.168.0.100)、ポート番号 (Telnet のポート番号 23 と、unprivileged なポート番号 1081) もわかる。

## 5.2 FreeS/WAN のダンプ結果

FreeS/WAN により中間のゲートウェイでトンネリングをしている状態での telnet の実行結果はこのようになる。

```

proto: 080000:06:29:38:4b:79->00:06:29:d5:1b:62 10.0.0.101[0]->10.0.0.100[512]
0000 cd 15 e6 0b dd ef fc a6 - b8 01 ee 19 5e 25 ac b9 .....%..
0010 35 05 58 aa 60 45 17 53 - 2b 52 35 84 ac 3b 76 81 5..X.E.S+R5...y.
0020 f6 33 dd c2 f5 d9 8d 66 - 61 21 05 e0 fe d9 ee id ..S.....fai.....
0030 76 f1 0f 46 d2 a3 10 52 - af 0a 25 4c d4 27 8d be v..F...R..M...

proto: 080000:06:29:d5:1b:62->00:06:29:38:4b:79 10.0.0.100[0]->10.0.0.101[512]
0000 16 9d c1 96 fe 04 16 10 - 34 06 6b 4f 51 3a c1 92 .....4.k0Q...
0010 36 5d b1 23 2a e4 c2 cb - 9b 40 da fb b8 10 fa 6a 6].*...0....j
0020 11 bb 49 c0 f1 91 e3 9d - 3e 1c 87 0a d4 44 30 84 ..I...>...DO.
0030 4a e1 2f 48 10 97 af 8c - 85 ab b3 2c 58 0a f2 b8 J./H.....X...
0040 ac 8b 63 e0 c3 e3 34 74 - d2 6c 48 7c 77 bd ac 3c ..c...4).lHw.<
0050 5d da f4 cb c8 1a f9 30 - a4 10 52 ae 26 60 32 67 l.....0..R.&'2g
0060 4b 45 4c aa 2c 17 6c 8d - b1 61 6a fd 95 32 cb 54 KEL...l..aj..2.T
0070 a8 57 f6 64 d2 37 27 44 - 28 06 6e 8b 15 fa f9 1e .W.d?').(m....
0080 31 c5 9f e4 e9 82 ff 10 - 13 37 ca a9 72 8e a3 87 1.....7..E....
0090 ae 44 75 87 32 85 fb 81 - 70 9c 5a e4 de a1 34 c9 ..Du.2..p.Z...d.
00a0 19 86 6a b0 ad fc 77 05 - 4e 63 8a 2e b5 28 d9 5a ..j...w.Nc...Z
00b0 ef 3a a8 fd 50 19 83 b1 - 32 81 66 15 b4 b6 58 88 .....P...2.f...X.
00c0 b8 00 0a 16 da 4f 0d 4c - 65 47 a4 97 39 fe 01 6b ....0.LoG...9..k
00d0 a6 08 c0 b0 6d 99 3a 94 - 18 99 46 40 3b 84 b8 87 .....m...0j...
00e0 41 a6 78 2c 1a 26 0e 5d - 50 21 69 91 01 05 0b 68 A.x.&.JPHi...h
00f0 6f 36 60 48 6b db 23 ad - 19 c4 86 d5 fe bf 7a 90 o6'Hk.#.....z.
0100 b7 b0 82 d0 a6 97 a4 97 - 31 83 50 d6 96 a1 c5 4b .....1.P....K

```

トンネリングにより、通信をしているホストの IP アドレスがゲートウェイのもの (10.0.0.100 と 10.0.0.101) に、ポート番号が 0 と 512 になっていることがわかる。トンネリング時のパケットのイメージは前述の図 3 を参照のこと。

ここで TCP ヘッダと ESP ヘッダの先頭 32bit を示す。図 7 および図 9 を見てわかる通り、パケットの SRC/DST のポート番号が入る位置に SPI がきている。上記ダンプは SPI = 0x200 の時に取ったもののため、0x200 の下位ビット 0x00 より SRC = 0, 上位ビット 0x02 より DST = 512 と算出されている<sup>10</sup>。SPI = 0x300 の時には DST = 768 となった。

ポート番号が上記のように誤認されていることはセキュリティを多少向上させるが、IP ヘッダのプロトコル番号に AH の場合は 51 が、ESP の場合は 50 が格納されているため、多少気のきいた packet sniffer の場合はポート番号と誤認せずに正しく AH/ESP ヘッダと判定ができる (そのように改造するのも簡単である)。だが、たとえ AH/ESP ヘッダということがわかっても AH の場合パケットの改竄が、ESP の場合は改竄およびポート番号の判定ができない<sup>11</sup>ため、セキュリティ上問題にはなりにくい。

通信しているポート番号が隠蔽されているというのは TCP の動作するトランスポート層の下である、インターネット層でセキュリティ機能を実装している IPSec の大きな利点である。

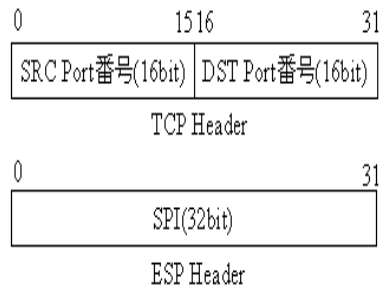


図 9: TCP ヘッダと ESP ヘッダ

## 5.3 Zebedee のダンプ結果

Zebedee を用いて、中間のゲートウェイでトンネリングをしている状態での telnet の実行結果はこのようになる。

```

proto: 080000:00:f4:95:94:49->00:e0:18:00:14:e8 10.0.0.101[19933]->10.0.0.100[11965]

proto: 080000:e0:18:00:14:e8->00:00:f4:95:94:49 10.0.0.100[11965]->10.0.0.101[19933]
0000 80 1d 06 06 11 a3 a5 fc - a2 81 cc b0 1f 4d 27 50 .....M'P
0010 f4 46 62 9a 72 70 a5 19 - 2a 9c 21 69 bf 65 a3 .....Fb.rp...i.e.

```

<sup>10</sup>このことより、SPI はリトルエンディアンで格納されていることがわかる

<sup>11</sup>ポート番号の格納されている TCP ヘッダは ESP により暗号化されている

```

proto: 080000:00:f4:95:94:49->00:e0:18:00:14:e8 10.0.0.101[19933]->10.0.0.100[11965]
proto: 080000:e0:18:00:14:e8->00:00:f4:95:94:49 10.0.0.100[11965]->10.0.0.101[19933]
0000 c1 9e e2 c7 9f 21 66 da - 33 40 23 e5 28 e5 1f 94 ...f,30#.(...
0010 5c dc 15 b2 e5 6a 00 4a - 09 50 15 cf 23 0b 63 92 \...J.P.#.c.
0020 98 47 86 6e cd 5e 0a ad - 2c 88 93 c5 19 da 80 e0 .G.a...
0030 bc 19 36 22 15 55 33 f3 - c6 8d 73 4d 3d a0 49 67 ...6.U3...SM=Ig
0040 c0 1a 68 85 cb 7d 2d b7 - de d9 a3 59 e4 af 1d 22 ...h.)-...Y...
0050 62 0f 81 b8 54 c8 7a c5 - a0 77 15 40 ad 5b 24 0a b...T.z...w.0.[8.
0060 52 14 9d d2 68 eb 58 07 - de e8 a5 3b c0 ff dc ec R...h.X...;...
0070 0f 7d 73 8f a5 95 89 2a - 27 b4 3a de 8c f3 ce 93 )s...*...
0080 67 54 a9 e1 e3 52 2f 02 - bf fc 0f fe a8 40 a2 bd gT...R/...o...
0090 ea 04 21 3b 31 16 07 36 - 62 33 70 b8 d9 56 ce 65 !;!;...6b5p...V.e
00a0 90 61 c8 84 df 0e 45 3d - df 46 56 07 11 25 09 af ...a...E...FV...X...
00b0 c8 a3 97 8f 2f ff 03 ac - a4 d9 e1 0e e4 a5 48 b0 .../...H...
00c0 2e fe a2 27 e0 11 1b b9 - e6 8b 7b 16 b0 dc 37 e3 .../...
00d0 76 b3 c0 fa ae b9 7a 59 - 36 86 16 cc 0f 6c e5 e7 v...z2f6...l...
00e0 21 a1 41 fa d5 c7 19 6f - 5a af b1 85 ae 51 15 04 !.A...oZ...Q...
00f0 8c fo f9 2e ce 7f e5 e6 - 49 3a dd 8f 8f 8c cb e4 ...I...
0100 3e d2 2f ef f3 b9 ac 9d - 3f e7 76 84 0b 45 c8 a8 >/...T...v...E...
0110 b7 19 9d 43 30 bf 53 08 - 2a 3e a1 f6 1d fd ea 31 ...C...S...>...1
0120 4f ad ec 6a fe cf 2e f9 - 94 0d af a0 3d 21 55 ab 0...J...
0130 99 dd 67 e8 d3 6c d4 dc - ac 57 2a 78 98 34 04 fe ...g...l...W*x...4...
0140 04 7f 27 43 ae a9 ce 8a - 14 c1 9d c2 f0 4a a7 68 ...C...J...J.h...
0150 82 bd ba c9 68 f1 16 ce - bd 38 0e ce 9d b8 0d b0 ...h...8...
0160 97 88 84 0e fe 09 90 32 - 25 df 21 71 86 c1 c5 43 ...e...2%...lq...C
0170 93 45 2e 4a b0 29 bc 19 - 2e d9 1b c5 3e 78 a3 34 ...E...J...>x...4
0180 ed d9 f5 f4 0e 59 02 ea - 4d 9e 57 cb 5d 5b 26 76 ...Y...M...W...[&v
0190 da 9e 79 3e 38 c0 e8 eb - ...y8...

```

Zebedee における暗号化の範囲は図 10 で示しているように、IPSec と異なり IP パケットを含んだ TCP のペイロードを暗号化対象としているため、Zebedee 自身が使用しているサービスポート (TCP/11965) がどれであるかはわかってしまう<sup>12</sup>。ただしこのことは別に Zebedee が特殊というわけではなく、アプリケーション層による IP トンネル実装では一般的なことである。

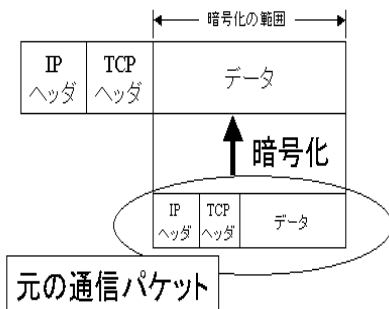


図 10: Zebedee における暗号化の対象

## 6 IPSec, IP トンネルの適用領域

これまで述べてきた通り IPSec はインターネット層での実装であり、標準化も進んでいることから今後 IPSec 対応のプロトコルスタックやハードウェアがリリースされることで、より広範囲に使われることが予想される。

<sup>12</sup>オリジナルのパケットのポート番号は暗号化されているため、わからない

IPSec はインターネット層でセキュリティ機能を追加するため、IP ネットワークを流れるあらゆる種類の通信を保護するのに適しており、上位のレイヤで動作するアプリケーションなどを修正する必要はない。また IPSec のパケット自身も IP パケットの一種にすぎないため、例えば図 11 のようにトンネルをネストすることにより、人事情報など機密情報にアクセスする場合などローカルのネットワーク内でもさらにセキュリティが必要になる時にも利用することができる。

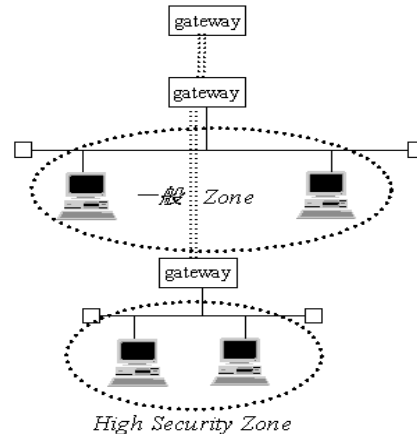


図 11: トンネルをトンネルした例

ただし、

- 機器の価格が高い
- 現状ではベンダー間の相互接続性が乏しい。特に IKE を用いた鍵交換は前途多難

という問題点もあるため、単一ベンダーのリリースする IPSec 実装で統一するか、もしくは (IKE による鍵交換を必要としない) プリシェアードキーを用いた IPSec 通信がこれからもしばらく使われ続けると考えられる。

また標準化が進み、これらの問題点が解決された時点で IPSec によるセキュアな通信が標準になったとしても、Firewall などによるパケット書き換えを IPSec のプロトコルレベルでパケット改竄とみなしてしまう可能性が高い。これは AH ヘッダや ESP ヘッダにはペイロードのチェックサムおよび IP ヘッダのチェックサムが同時に格納されるが<sup>13</sup>、IP ヘッダの書き換え (特

<sup>13</sup>ESP の場合 IP ヘッダは認証の対象外

に IP アドレスの書き換え) が行われた場合、AH ヘッダに格納されたチェックサム値と実際のヘッダ内容から算出されたチェックサム値が異なり、認証エラーが発生するためである。このため、完全に IPSec による通信のみでセキュアな通信路が確保できるとは言いがたい (IPSec などによる通信の認証・暗号化と Firewall によるアクセス制限・監査は別の話である)。

このような場合にはアプリケーション層での IP トンネルを使う意味がある。Zebedee の場合 Firewall によるパケットの書き換えは何の障害にもならない。また実装される層が異なるため、IPSec との併用も可能ではある。ただし Zebedee 自身がインターネット層レベルでのパケットルーティングを制御出来るわけではないため、IPSec ほど自由な通信を提供できるとは言いがたい。

適用範囲としては、

- IPSec は特定のグループ (ネットワーク) 間でセキュアな通信を行う場合に有効
- IP トンネル は特定のサービス (pop3 や imap4 など) を提供しているホストとの間でセキュアな通信を行う場合に有効
- 永続的なセッション利用は、TPO に応じて組み合わせ/選択的に使う

という住み分けになるだろう。

## 7 まとめ

ここまでの議論は、現時点でのインターネット層、アプリケーション層で確保可能なセキュリティについて実装の観点から述べたものである。今後 IPSec をプロトコル仕様に含んだ IPv6 が普及するに従い、IPv4 & IPSec という組み合わせから IPv6 へとインターネットプロトコルのシフトが進むことが予想される。ただし、これらの移行が早期かつスムーズに実現するかというところではなく、インターネットのどこかに IPv4 しか解釈できないホストが残る可能性は非常に高い。また、移行期のどさくさでセキュリティが確保しづらくなるような事態も考えられるだろう。このような場合、IPSec トンネルモードや IP トンネルを用いたセキュアな通信路開設を行う手段/実装が役に立つことがあるだろう。

## 参考文献

- [1] Naganand Doraswamy, Dan Harkins  
*IPSec: The New Security Standard for the Internet, Intranets, and Virtual Private Networks*  
ISBN0-1301-1898-2
- [2] W.Richard Stevens  
詳解 *TCP/IP*  
ISBN4-7973-0232-1
- [3] *FreeS/WAN*  
<http://www.freeswan.org/>
- [4] *Zebedee*  
<http://www.winton.org.uk/zebedee/>
- [5] *Zebedee* 紹介  
<http://www03.u-page.so-net.ne.jp/dc4/tnaka/zebedee/>
- [6] RFC1827  
*IP Encapsulating Security Payload (ESP)*  
<ftp://ftp.iij.ad.jp/pub/rfc/rfc1827.txt>
- [7] RFC2401  
*Security Architecture for the Internet Protocol*  
<ftp://ftp.iij.ad.jp/pub/rfc/rfc2401.txt>
- [8] RFC2411  
*IP Security Document Roadmap*  
<ftp://ftp.iij.ad.jp/pub/rfc/rfc2411.txt>
- [9] RFC2402  
*IP Authentication Header*  
<ftp://ftp.iij.ad.jp/pub/rfc/rfc2402.txt>
- [10] RFC2403  
*The Use of HMAC-MD5-96 within ESP and AH*  
<ftp://ftp.iij.ad.jp/pub/rfc/rfc2403.txt>
- [11] RFC2404  
*The Use of HMAC-SHA-1-96 within ESP and AH*  
<ftp://ftp.iij.ad.jp/pub/rfc/rfc2404.txt>
- [12] RFC2405  
*The ESP DES-CBC Cipher Algorithm With Explicit IV*  
<ftp://ftp.iij.ad.jp/pub/rfc/rfc2405.txt>

- [13] RFC2406  
*IP Encapsulating Security Payload (ESP)*  
<ftp://ftp.iij.ad.jp/pub/rfc/rfc2406.txt>
  
- [14] RFC2407  
*The Internet IP Security Domain of Interpretation for ISAKMP*  
<ftp://ftp.iij.ad.jp/pub/rfc/rfc2407.txt>
  
- [15] RFC2410  
*The NULL Encryption Algorithm and Its Use With IPsec*  
<ftp://ftp.iij.ad.jp/pub/rfc/rfc2410.txt>
  
- [16] RFC2451  
*The ESP CBC-Mode Cipher Algorithms*  
<ftp://ftp.iij.ad.jp/pub/rfc/rfc2451.txt>
  
- [17] RFC2408  
*Internet Security Association and Key Management Protocol (ISAKMP)*  
<ftp://ftp.iij.ad.jp/pub/rfc/rfc2408.txt>
  
- [18] RFC2409  
*The Internet Key Exchange (IKE)*  
<ftp://ftp.iij.ad.jp/pub/rfc/rfc2409.txt>
  
- [19] RFC2412  
*The OAKLEY Key Determination Protocol*  
<ftp://ftp.iij.ad.jp/pub/rfc/rfc2412.txt>
  
- [20] *Description of a New Variable-Length Key, 64-Bit Block Cipher (Blowfish)*  
<http://www.counterpane.com/bfsverlag.html>
  
- [21] *IP Security for Microsoft Windows 2000 Server*  
[http://msdn.microsoft.com/library/backgrnd/html/msdn\\_ip\\_security.htm](http://msdn.microsoft.com/library/backgrnd/html/msdn_ip_security.htm)
  
- [22] IBM redbooks  
*TCP/IP Tutorial and Technical Overview*  
<http://www.redbooks.ibm.com/abstracts/gg243376.html>