FreeBSD の IPsec 機能を独立検証するには

概要

IPsec をインストールした時、 それがきちんと動作しているかどうか調べるにはどうしたら良いでしょう? ここでは、IPsec の動作を検証する実験的な方法を紹介します。

目次

1. 問題 1
2. 解決方法 1
3. 実験 2
4. 注意 3
5. IPsec の定義 3
6. IPsec のインストール 3
7. src/sys/i386/conf/KERNELNAME 3
8. Maurer's Universal Statistical Test (ブロックサイズ = 8 ビット) 4

1. 問題

まず、IPsec のインストールを前提に話を進めます。 IPsec が注意かどうか知るにはどうしたら良いでしょう?
もちろん設定が間違っていればネットワーク接続が行なえないでしょうし、
接続できたということは設定が合っているからだ、という認識は間違っていません。 接続状態は netstat(1) コマンドで確かめることができます。
しかし、それを独立して検証することは可能なのでしょうか?

2. 解決方法

最初に、暗号に使われている情報理論について考えます。

1. 暗号化されたデータは、一様に分布している。つまり、
   各情報源シンボルは最大のエントロピーを持っている。
2. 通常、未処理のデータや圧縮されていないデータは冗長である。
   つまり、各情報源シンボルのエントロピーは最大ではない。

ネットワークインターフェイスを入出力するデータのエントロピーを測定できると仮定すると、
「暗号化されていないデータ」と「暗号化されたデータ」の両者に、 違いを見ることができるはずです。
このことは、パケットのルーティングが行なわれる場合の一番外側の IP ヘッダなど、 データの一部が "暗号化モード" で暗号化されなかったとしても成立します。

2.1. MUST

Ueli Maurer 氏の "Universal Statistical Test for Random Bit Generators" (MUST) は、サンプルデータのエントロピーを高速に測定します。
これには圧縮と良く似たアルゴリズムが使われています。 Maurer's Universal Statistical Test (ブロックサイズ = 8 ビット)、 一つのファイル中で連続するデータ (最大 0.25 メガバイト) を測定するコードです。

2.2. Tcpdump

さて次に、上記に加えてネットワーク上の生データを捕捉するための手段も必要になります。
それを実現するプログラムに、tcpdump(1) と呼ばれるものがあります。 ただし、tcpdump を使うには、 src/sys/i386/conf/KERNELNAMEにおいて *Berkeley Packet Filter* インターフェイスが有効化されていなければなりません。

次のコマンド:

```
tcpdump -c 4000 -s 10000 -w dumpfile.bin
```

は、4000 個の生パケットを捕捉し、*dumpfile.bin* に記録します。 この例のでは 10,000 バイト以下のパケットのみ記録されます。

3. 実験

では、実験してみましょう。

1. IPsec ホストと IPsec を使っていないホストの両方にネットワーク接続してください。
2. そして Tcpdumpを開始します。
3. 次に、"IPsec を使っている" 接続で yes(1) という UNIX® コマンドを実行します。 これは、y という文字の連続データを出力するものです。 しばらくしたらコマンドを停止させ、IPsec を使っていない接続に対して同じコマンドを実行します。
   こちらも、しばらくしたらコマンドを停止させてください。
4. ここで、Maurer's Universal Statistical Test (ブロックサイズ = 8 ビット) を捕捉したパケットに実行すると、次のような出力が得られるはずです。
   この中で重要なのは、期待値 (7.18) に対して、 IPsec を使った接続が 93% (6.7)、 "通常の "接続が 29% (2.1) という結果になっていることです。

   ```
   % tcpdump -c 4000 -s 10000 -w ipsecdemo.bin
   % uliscan ipsecdemo.bin
   Uliscan 21 Dec 98
   L=8 256 258560
   Measuring file ipsecdemo.bin
   Init done
   Expected value for L=8 is 7.1836656
   6.9396 ----------------------------------------------------------
   ```

```
6.6177 ----------------------------------------------------
6.4100 ---------------------------------------------------
2.1101 -----------------
2.0838 -----------------
2.0983 -----------------
```

4. 注意

この実験は暗号化の理論が示すとおり、IPsec を使った通信では確かにペイロード中のデータに含まれるシンボルの生起確率が一様に分布する、ということを示しています。しかし、ここで示した実験ではシステム上の欠陥(あるのかどうか知りませんが)を検出することはできません。ここで言う「欠陥」とは、たとえば暗号鍵生成や交換の不備や、データや暗号鍵が他人に見られていないかどうかといった問題、あるいはアルゴリズムの強度はどうか、カーネルのバージョンは合っているかといったことです。
これらはソースを調べれば確かめることができます。

5. IPsec の定義

インターネットプロトコル セキュリティ拡張 (Internet Protocol security extensions) は IP v4 と IP v6 に適用され、IP v6 への実装は必須となっています。このプロトコルは IP (ホスト間) レベルで暗号化と認証を実現するためのものです。たとえば SSL は一つのアプリケーションソケット、SSH はログイン、PGP は特定のファイルやメッセージのみに対してそれぞれ安全性を提供しますが、IPsec は 2 ホスト間のすべての通信を暗号化します。

6. IPsec のインストール

FreeBSD の最近のバージョンでは IPsec のサポートが基本のソースコードに含まれています。それ故、あなたはおそらく IPSEC オプションをカーネルコンフィグファイルに追加し、カーネルを再構築/再インストールして setkey(8) コマンドで IPsec 接続を設定すればよいはずです。

FreeBSD で IPsec を実行する包括的なガイドは FreeBSD ハンドブック で提供されています。

7. src/sys/i386/conf/KERNELNAME

ネットワークデータを tcpdump(1) で補足するためにはカーネルコンフィグファイルには以下の行が必要です。追加後 config(8) を実行しカーネルの再構築/再インストールを 行なってください。

```
device  bpf
```

8. Maurer’s Universal Statistical Test (ブロックサイズ = 8 ビット)

同一のコードを このリンクから入手することができます。

```
/*
  ULISCAN.c   ---blocksize of 8

  1 Oct 98
  1 Dec 98
  21 Dec 98       uliscan.c derived from ueli8.c

  This version has // comments removed for Sun cc

  This implements Ueli M Maurer's "Universal Statistical Test for Random
  Bit Generators" using L=8

  Accepts a filename on the command line; writes its results, with other
  info, to stdout.

  Handles input file exhaustion gracefully.

  Ref: J. Cryptology v 5 no 2, 1992 pp 89-105
  also on the web somewhere, which is where I found it.

  -David Honig
  honig@sprynet.com

  Usage:
  ULISCAN filename
  outputs to stdout
*/

#define L 8
#define V (1<<L)
#define Q (10*V)
#define K (100   *Q)
#define MAXSAMP (Q + K)

#include <stdio.h>
#include <math.h>

int main(argc, argv)
int argc;
char **argv;
{
  FILE *fptr;
  int i,j;
  int b, c;
```

```
int table[V];
double sum = 0.0;
int iproduct = 1;
int run;

extern double   log(/* double x */);

printf("Uliscan 21 Dec 98 \nL=%d %d %d \n", L, V, MAXSAMP);

if (argc < 2) {
  printf("Usage: Uliscan filename\n");
  exit(-1);
} else {
  printf("Measuring file %s\n", argv[1]);
}

fptr = fopen(argv[1],"rb");

if (fptr == NULL) {
  printf("Can't find %s\n", argv[1]);
  exit(-1);
}

for (i = 0; i < V; i++) {
  table[i] = 0;
}

for (i = 0; i < Q; i++) {
  b = fgetc(fptr);
  table[b] = i;
}

printf("Init done\n");

printf("Expected value for L=8 is 7.1836656\n");

run = 1;

while (run) {
  sum = 0.0;
  iproduct = 1;

  if (run)
    for (i = Q; run && i < Q + K; i++) {
      j = i;
      b = fgetc(fptr);

      if (b < 0)
        run = 0;

      if (run) {
```

```
          if (table[b] > j)
            j += K;

          sum += log((double)(j-table[b]));

          table[b] = i;
        }
      }

    if (!run)
      printf("Premature end of file; read %d blocks.\n", i - Q);

    sum = (sum/((double)(i - Q))) /  log(2.0);
    printf("%4.4f ", sum);

    for (i = 0; i < (int)(sum*8.0 + 0.50); i++)
      printf("-");

    printf("\n");

    /* refill initial table */
    if (0) {
      for (i = 0; i < Q; i++) {
        b = fgetc(fptr);
        if (b < 0) {
          run = 0;
        } else {
          table[b] = i;
        }
      }
    }
  }
}
```