Uafhængig Verifikation af IPsec Funktionalitet i FreeBSD

```
Navne og email adresser på de teams af folk der arbejder
på specifikke opgaver.  Normalt er de bare mail aliaser der
er sat op på hub.FreeBSD.org
```

```
Brug disse entiteter når du refererer de korrekte teams.
```

```
Vær venlig at holde denne liste i alfabetisk orden efter
entitets navne.
```

```
VIGTIGT: Hvis du sletter navne fra denne fil *skal* du sikre dig at
         alle referencer til dem er fjernet fra håndbogens
         oversættelser.  Hvis de ikke er fjernet *vil* du
         ædelægge bygningen for de andre sprog, og vi
         vil drille dig i fuld offentlighed.
```

```
     $FreeBSD$
//
```

Resumé

Dansk version af Laust S. Jespersen Laust@doc.freebsd.dk.

Du installerede IPsec og det ser ud til at virke. Men hvordan ved du det? Jeg beskriver en metode til eksperimentalt at verificere at IPsec virker.

Indholdsfortegnelse

1. Problemet

Lad os antage, at du har Installering af IPsec. Hvordan ved du at det Undtagelse? Selvfølgelig virker din forbindelse ikke hvis den er miskonfigureret, og den vil virke når du endelig laver det rigtigt. netstat(1) lister den. Men kan du verificere det uafhængigt?

2. Løsningen

Først, noget krypto-relevant teoretisk information:

1. krypterede data er uniformt distribueret, som f.eks. har maksimal entropi pr. symbol;
2. rå, ukomprimerede data er typisk redundant, f.eks., har sub-maksimal entropi.

Hvis du kunne måle entropien af dataene til og fra dit netværks interface. Så kunne du se forskellen mellem ukrypterede og krypterede data. Det ville være tilfældet selvom nogle af dataene i "krypteret mode" ikke var krypterede, som f.eks. den yderste IP header skal være, hvis pakken skal kunne routes.

2.1. MUST

Ueli Maurers "Universal Statistical Test for Random Bit Generators" (MUST) måler hurtigt entropien af en stikprøve. Den bruger en kompressionsagtig algoritme. Maurers universelle statistiske test (for blok størrelse=8 bits) til en variant der måler successive (~kvart megabyte) store bidder af en fil.

2.2. Tcpdump

Vi har også brug for en måde at opsamle de rå netværksdata. Et program kaldet tcpdump(1) lader dig gøre dette, hvis du har slået *Berkeley Packet Filter* interfacet til i din src/sys/i386/conf/KERNELNAME.

Kommandoen

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

vil opfange 4000 rå pakker til *dumpfile.bin*. Op til 10.000 bytes per pakke bliver opfanget i dette eksempel.

3. Eksperimentet

Her er eksperimentet:

1. Åbn et vindue til en IPsec vært og et andet vindue til en usikker vært.
2. Start nu med at Tcpdump.
3. I det "sikre" vindue, køres UNIX® kommandoen yes(1), hvilket vil streame `y` karakteren. Stop dette efter et stykke tid. Skift til det usikre vindue, og gentag. Stop igen efter et stykke tid.
4. Kør nu Maurers universelle statistiske test (for blok størrelse=8 bits) på de opfangede pakker. Du skulle se noget lignende det følgende. Det vigtige at notere sig er, at den sikre forbindelse har 93% (6,7) af den ventede værdi (7.18), og den "normale" forbindelse har 29% (2.1) af den ventede værdi.

```
% 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. Undtagelse

Dette eksperiment viser at, IPsec *kan* se ud til at distribuere payload data *uniformt*, som kryptering skal. Men det eksperiment der er beskrevet her *kan ikke* detektere mange mulige fejl i et system (ingen af hvilke jeg har nogle beviser for). Disse inkluderer dårlig nøgle generering eller udveksling, data eller nøgler der kan ses af andre, brug af svage algoritmer, kernel undergravning, osv. Studér kildekoden; kend kildekoden.

5. IPsec---Definition

Internet Protokol sikkerheds udvidelser til IPv4; krævet for IPv6. En protokol til at forhandle kryptering og autentifikation på IP (vært-til-vært) niveau. SSL sikrer kun en applikationssocket; SSH sikrer kun et login; PGP sikrer kun en specifik fil eller besked. IPsec krypterer alting mellem to værter.

6. Installering af IPsec

De fleste af de moderne versioner af FreeBSD har IPsec support i deres base kildekode. Så du er sikkert nødt til at inkludere `IPSEC` optionen i din kernel konfig og, efter genbygning og reinstallation

af kernel, konfigurere IPsec forbindelser ved hjælp af setkey(8) kommandoen.

En udførlig guide om at køre IPsec på FreeBSD er tilrådighed i FreeBSD Håndbogen.

7. src/sys/i386/conf/KERNELNAME

Dette skal være til stede i kernel konfig filen for at være i stand til at opfange netværksdata med tcpdump(1). Vær sikker på at køre config(8) efter at tilføje dette, og genbygge og reinstallere.

```
device  bpf
```

8. Maurers universelle statistiske test (for blok størrelse=8 bits)

Du kan finde samme kode på dette link.

```
/*
  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;
        }
      }
    }
  }
}
```