Independent Verification of IPsec Functionality in FreeBSD

David Honig <honig@sprynet.com >

Revision: 44703

FreeBSD is a registered trademark of the FreeBSD Foundation.

Motif, OSF/1, and UNIX are registered trademarks and IT DialTone and The Open Group are trademarks of The Open Group in the United States and other countries.

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this document, and the FreeBSD Project was aware of the trademark claim, the designations have been followed by the “™” or the “®” symbol.

1999-05-03 by .

Abstract

You installed IPsec and it seems to be working. How do you know? I describe a method for experimentally verifying that IPsec is working.

Table of Contents

1. The Problem 2
2. The Solution 2
3. The Experiment 2
4. Caveat 3
5. IPsec---Definition 3
6. Installing IPsec 3
7. src/sys/i386/conf/KERNELNAME 4
8. Maurer's Universal Statistical Test (for block size=8 bits) 4

1. The Problem

First, lets assume you have installed *IPsec*. How do you know it is working? Sure, your connection will not work if it is misconfigured, and it will work when you finally get it right. netstat(1) will list it. But can you independently confirm it?

2. The Solution

First, some crypto-relevant info theory:

1. Encrypted data is uniformly distributed, i.e., has maximal entropy per symbol;

2. Raw, uncompressed data is typically redundant, i.e., has sub-maximal entropy.

Suppose you could measure the entropy of the data to- and from- your network interface. Then you could see the difference between unencrypted data and encrypted data. This would be true even if some of the data in "encrypted mode" was not encrypted---as the outermost IP header must be if the packet is to be routable.

2.1. MUST

Ueli Maurer's "Universal Statistical Test for Random Bit Generators"(MUST) quickly measures the entropy of a sample. It uses a compression-like algorithm. The code is given below for a variant which measures successive (~quarter megabyte) chunks of a file.

2.2. Tcpdump

We also need a way to capture the raw network data. A program called tcpdump(1) lets you do this, if you have enabled the *Berkeley Packet Filter* interface in your kernel's config file.

The command:

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

will capture 4000 raw packets to *`dumpfile.bin`* . Up to 10,000 bytes per packet will be captured in this example.

3. The Experiment

Here is the experiment:

1. Open a window to an IPsec host and another window to an insecure host.

2. Now start capturing packets.

3. In the "secure" window, run the UNIX® command yes(1), which will stream the y character. After a while, stop this. Switch to the insecure window, and repeat. After a while, stop.

4. Now run MUST on the captured packets. You should see something like the following. The important thing to note is that the secure connection has 93% (6.7) of the expected value (7.18), and the "normal" connection has 29% (2.1) of the expected value.

```
% 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. Caveat

This experiment shows that IPsec *does* seem to be distributing the payload data *uniformly*, as encryption should. However, the experiment described here *cannot* detect many possible flaws in a system (none of which do I have any evidence for). These include poor key generation or exchange, data or keys being visible to others, use of weak algorithms, kernel subversion, etc. Study the source; know the code.

5. IPsec---Definition

Internet Protocol security extensions to IPv4; required for IPv6. A protocol for negotiating encryption and authentication at the IP (host-to-host) level. SSL secures only one application socket; SSH secures only a login; PGP secures only a specified file or message. IPsec encrypts everything between two hosts.

6. Installing IPsec

Most of the modern versions of FreeBSD have IPsec support in their base source. So you will need to include the `IPSEC` option in your kernel config and, after kernel rebuild and reinstall, configure IPsec connections using setkey(8) command.

A comprehensive guide on running IPsec on FreeBSD is provided in FreeBSD Handbook.

7. src/sys/i386/conf/KERNELNAME

This needs to be present in the kernel config file in order to capture network data with tcpdump(1). Be sure to run config(8) after adding this, and rebuild and reinstall.

```
device bpf
```

8. Maurer's Universal Statistical Test (for block size=8 bits)

You can find the same code at this 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;
        }
      }
    }
  }
}
```