Vérification indépendante du fonctionnement d'IPSec sous FreeBSD

Résumé

Vous avez installé IPSec et cela semble fonctionner. Comment pouvez-vous en être sûr? Je décris une méthode pour vérifier expérimentalement le fonctionnement d'IPSec.

Version française de Marc Fonvieille <blackend@FreeBSD.org>.

Table des matières

1. Le problème . . . 1
2. La solution . . . 1
3. L'expérience . . . 2
4. Mise en garde . . . 3
5. IPSec - Définition . . . 3
6. Installation d'IPSec . . . 3
7. src/sys/i386/conf/KERNELNAME . . . 3
8. Test statistique universel de Maurer (pour une longueur de bloc=8 bits) . . . 4

1. Le problème

Tout d'abord, supposons que vous avez Installation d'IPSec. Comment savez-vous si cela Mise en garde? Bien sûr, votre connexion ne fonctionnera pas si elle est mal configurée, et fonctionnera quand vous l'aurez enfin correctement configurée. netstat(1) le fera apparaître. Mais pouvez-vous le confirmer de façon indépendante?

2. La solution

Tout d'abord, quelques informations théoriques relatives à la cryptographie:

1. Les données chiffrées sont uniformément distribuées, i.e., ont une entropie maximale par symbole;
2. Les données brutes, non compressées sont en générale redondantes, i.e., n'ont pas une entropie maximale.

Supposez que vous pourriez mesurer l'entropie des données à destination et en provenance de

votre interface réseau. Alors vous pourriez voir la différence entre données non-chiffées et données chiffrées. Cela serait vrai même si certaines des données en "mode chiffré" n'étaient pas chiffrées --- comme l'en-tête IP externe, si le paquet doit être routable.

2.1. MUST

L'"Universal Statistical Test for Random Bit Generators"(MUST) d'Ueli Maurer, ou encore le "test statistique universel pour les générateurs aléatoires de bits", mesure rapidement l'entropie d'un échantillon. Il utilise une sorte d'algorithme de compression. Test statistique universel de Maurer (pour une longueur de bloc=8 bits) pour une variante qui mesure les morceaux (environ un quart de mégaoctet) successifs d'un fichier.

2.2. Tcpdump

Nous avons également besoin d'une manière de capturer les données réseau brutes. Un programme appelé tcpdump(1) vous permet de faire cela, si vous avez activé l'interface *Berkeley Packet Filter* (Filtre de Paquet de Berkeley) dans votre src/sys/i386/conf/KERNELNAME.

La commande

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

capturera 4000 paquets bruts dans le fichier *dumpfile.bin*. Dans cet exemple jusqu'à 10000 octets par paquets seront capturés.

3. L'expérience

Voici l'expérience:

1. Ouvrez une fenêtre sur un hôte IPSec et une autre sur un hôte non sécurisé.
2. Maintenant commencez à Tcpdump.
3. Dans la fenêtre "sécurisée", lancez la commande UNIX® yes(1), qui fera défiler le caractère y. Au bout d'un moment, arrêtez cela. Passez à la fenêtre non sécurisée, et faites de même. Au bout d'un moment, arrêtez.
4. Maintenant lancez Test statistique universel de Maurer (pour une longueur de bloc=8 bits) sur les paquets capturés. Vous devriez voir quelque chose de semblable à ce qui suit. Ce qui est important de noter est que la connexion non sécurisée a 93% (6,7) de valeurs attendues (7,18), et la connexion "normale" a 29% (2,1) de valeurs attendues.

```
% 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. Mise en garde

Cette expérience montre qu'IPSec *semble* distribuer les données utiles *uniformément* comme un chiffrement le devrait. Cependant, l'expérience décrite ici *ne peut pas* détecter les problèmes possibles dans un système. Ceux-ci peuvent être la génération ou l'échange d'une clé faible, des données ou clés visibles par d'autres, l'utilisation d'algorithmes faibles, code du noyau modifié, etc... Etudiez les sources, maîtrisez le code.

5. IPSec - Définition

Extensions de sécurité au protocole internet IPv4, requises pour l'IPv6. Un protocole pour le chiffrement et l'authentification au niveau IP (hôte à hôte). SSL sécurise uniquement une socket d'application; SSH sécurise seulement une session; PGP sécurise uniquement un fichier spécifique ou un message. IPSec chiffre tout entre deux hôtes.

6. Installation d'IPSec

La plupart des versions récentes de FreeBSD ont le support IPSec dans leurs sources de base. Aussi vous devrez probablement ajouter l'option `IPSEC` dans votre configuration de noyau et, après la compilation et l'installation du noyau, configurer les connexions IPSec en utilisant la commande setkey(8).

Un guide complet sur l'utilisation d'IPSec sous FreeBSD est fourni dans le Manuel de Freebsd.

7. src/sys/i386/conf/KERNELNAME

Ce qui suit doit être présent dans le fichier de configuration du noyau afin de pouvoir capturer les données réseau avec tcpdump(1). Soyez-sûr de lancer config(8) après avoir rajouté la ligne ci-dessous, et de recompiler et réinstaller.

```
device  bpf
```

8. Test statistique universel de Maurer (pour une longueur de bloc=8 bits)

Vous pouvez trouver le même code source ici.

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