Независимое исследование работы IPsec во FreeBSD

Аннотация

Вы только что установили и настроили IPsec, и оно, кажется, заработало. Как это можно проверить? Я опишу метод экспериментальной проверки правильного функционирования IPsec.

Содержание

1. Постановка задачи . . . 1
2. Решение . . . 1
3. Эксперимент . . . 2
4. Замечание . . . 3
5. Определение IPsec . . . 3
6. Установка IPsec . . . 3
7. src/sys/i386/conf/KERNELNAME . . . 3
8. Универсальный Статистический Тест Маурера (размер блока - 8 бит) . . . 4

1. Постановка задачи

Для начала предположим, что Вы Установка IPsec. Как Вы узнаете, что IPsec Замечание? Несомненно, соединения не будет, если Вы неверно его сконфигурировали. И оно, конечно, появится в выводе команды netstat(1), когда Вы всё сделаете верно. Но можно ли как-то подтвердить сам факт функционирования IPsec?

2. Решение

Для начала немножко криптографической теории:

1. Шифрованные данные равномерно распределены по области определения, то есть каждый символ имеет максимальную энтропию;
2. "Сырые" и несжатые данные как правило избыточны, то есть их энтропия меньше максимально возможной.

Предположим, что у Вас имеется возможность измерить энтропию входящего и исходящего трафика на сетевом интерфейсе. В этом случае Вы сможете легко отличить зашифрованные данные от открытых, причём даже в том случае, когда часть данных в "режиме шифрования" передаётся в открытом виде, к примеру внешние заголовки IP, которые

используются для маршрутизации.

2.1. MUST

"Универсальный Статистический Тест для Генераторов Случайных Чисел" Уэли Маурера (Ueli Maurer's Universal Statistical Test for Random Bit Generators), сокращённо MUST позволяет быстро измерить энтропию последовательного набора данных. Используемый алгоритм похож на алгоритм сжатия. Универсальный Статистический Тест Маурера (размер блока - 8 бит) приведён исходный код, позволяющий измерять энтропию последовательных кусков данных размером около четверти мегабайта.

2.2. Tcpdump

Ещё нам нужен способ сохранения информации, проходящей через интерфейс. Программа tcpdump(1) позволяет сделать это в случае, если Вы src/sys/i386/conf/KERNELNAME с поддержкой *Пакетного Фильтра Беркли (Berkeley Packet Filter)*.

Команда

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

сохранит 4000 пакетов в файл *dumpfile.bin*. В данном примере объём записываемой информации в каждом пакете не может превышать 10,000 байтов.

3. Эксперимент

Повторите следующие шаги эксперимента:

1. Откройте два окна терминала и свяжитесь в одном из них с каким-нибудь компьютером через канал IPsec, а в другом - с обычным, "незащищённым" компьютером.
2. Теперь начните Tcpdump.
3. В "шифрованном" окне запустите команду UNIX® yes(1), которая будет выдавать бесконечный поток символов y. Немножко подождите и завершите её. Затем переключитесь в обычное окно (не использующее канал IPsec) и сделайте то же самое.
4. Заключительный этап: запустите Универсальный Статистический Тест Маурера (размер блока - 8 бит), передав ему для обработки только что сохранённые пакеты через командную строку. Вы должны увидеть что-то вроде изображённого чуть ниже. Заметьте, что безопасное соединение имеет 93% (6,7) от ожидаемого значения (7,18), а обычное соединение - всего лишь 29% (2,1).

   ```
   % 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. Замечание

Этот эксперимент показывает, что IPsec *действительно* распределяет передаваемые байты по области определения *равномерно*, как и любое другое шифрование. Однако этот метод *не может* обнаружить множество других изъянов в системе (хотя я таковых не знаю). Для примера можно привести плохие алгоритмы генерации или обмена ключами, нарушение конфиденциальности данных или ключей, использование слабых в криптографическом смысле алгоритмов, взлом ядра и т. д. Изучайте исходный код, узнавайте, что там происходит.

5. Определение IPsec

IPsec представляет собой протокол безопасного обмена информацией по Internet. Существует в виде расширения к IPv4; является неотъемлемой частью IPv6. Содержит в себе протокол шифрования и аутентификации на уровне IP (межмашинное "host-to-host" взаимодействие). SSL защищает только лишь конкретный прикладной сокет; SSH защищает вход на машину; PGP защищает определённый файл или письмо. IPsec шифрует всю информацию, передаваемую между двумя машинами.

6. Установка IPsec

Большинство современных версий FreeBSD уже имеют поддержку IPsec. Вероятно, Вы должны будете лишь добавить опцию `IPSEC` в конфигурационный файл ядра, и после сборки и инсталляции нового ядра, сконфигурировать соединение IPsec с помощью команды setkey(8).

Более подробно о том, как запустить IPsec во FreeBSD можно прочесть в Руководстве пользователя.

7. src/sys/i386/conf/KERNELNAME

Для того, чтобы захватывать сетевой трафик при помощи tcpdump(1), следующие строки должны присутствовать в конфигурационном файле ядра. Не забудьте после модификации запустить config(8), и, как обычно, пересобрать и установить новое ядро.

```
device  bpf
```

8. Универсальный Статистический Тест Маурера (размер блока - 8 бит)

Оригинал нижеприведённого кода находится по этому адресу.

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