Verificação Independente da Funcionalidade IPsec no FreeBSD

Resumo

Você instalou o IPsec e ele parece estar funcionando. Como você sabe? Eu descrevo um método para verificar experimentalmente se o IPsec está funcionando.

Índice

1. O problema . . . 1
2. A solução . . . 1
3. O Experimento . . . 2
4. Embargo . . . 3
5. IPsec --- Definição . . . 3
6. Instalando o IPsec . . . 3
7. src/sys/i386/conf/KERNELNAME . . . 3
8. Teste estatístico universal de Maurer (para tamanho de bloco = 8 bits) . . . 3

1. O problema

Primeiramente, vamos supor que você tenha o instalado o IPsec (Instalando o IPsec). Como você sabe se há algum problema (Embargo)? É verdade que sua conexão não funcionará se estiver mal configurada, e funcionará quando você finalmente acertar. O comando netstat(1) listará isso. Mas você pode confirmar isso de forma independente?

2. A solução

Em primeiro lugar, vejamos alguma informação teórica relevante em relação à criptografia:

1. Dados criptografados são uniformemente distribuídos, ou seja, possuem entropia máxima por símbolo;
2. Os dados brutos, não comprimidos são tipicamente redundantes, isto é, possuem entropia submáxima.

Suponha que você pudesse medir a entropia dos dados que entram e saem de sua interface de rede. Então, você poderia ver a diferença entre dados não criptografados e dados criptografados. Isso seria verdade mesmo que alguns dos dados no "modo criptografado" não estivessem criptografados - como o cabeçalho IP mais externo deve estar se o pacote for roteável.

2.1. MUST

O "Universal Statistical Test for Random Bit Generators" de Ueli Maurer (MUST) mede rapidamente a entropia de uma amostra. Ele usa um algoritmo semelhante ao de compressão. O [Código] de uma variante que mede blocos sucessivos (de cerca de um quarto de megabyte) de um arquivo.

2.2. Tcpdump

Também precisamos de uma maneira de capturar os dados brutos da rede. Um programa chamado tcpdump(1) permite que você faça isso, se você habilitou a interface do *Berkeley Packet Filter* no seu src/sys/i386/conf/KERNELNAME.

O comando:

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

serão capturados 4000 pacotes brutos e salvos em *dumpfile.bin*. Neste exemplo, serão capturados até 10.000 bytes por pacote.

3. O Experimento

Aqui está o experimento:

1. Abra uma janela para um host IPsec e outra janela para um host inseguro.
2. Agora, inicie o Tcpdump.
3. Na janela "segura", execute o comando UNIX® yes(1), que transmitirá o caractere `y`. Após um tempo, pare este comando. Mude para a janela "insegura" e repita o processo. Após um tempo, pare também esse comando.
4. Agora execute o Teste estatístico universal de Maurer (para tamanho de bloco = 8 bits) nos pacotes capturados. Você deverá ver algo como o seguinte. O importante a ser observado é que a conexão segura tem 93% (6,7) do valor esperado (7,18), enquanto a conexão "normal" tem 29% (2,1) do valor esperado.

```
% 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. Embargo

Este experimento mostra que o IPsec parece estar distribuindo os dados do payload *uniformly*, como a criptografia deveria fazer. No entanto, o experimento descrito aqui não pode detectar muitas possíveis falhas em um sistema (das quais não tenho nenhuma evidência). Estes incluem geração ou troca de chave fraca, dados ou chaves visíveis para outros, uso de algoritmos fracos, subversão do kernel, etc. Estude o código-fonte; conheça o código.

5. IPsec --- Definição

As extensões de segurança do Protocolo de Internet para o IPv4; obrigatório para o IPv6. Um protocolo para negociar criptografia e autenticação no nível IP (host-to-host). O SSL protege apenas um socket de aplicativo; O SSH protege apenas um login; O PGP protege apenas um arquivo ou mensagem especificada. O IPsec criptografa tudo entre dois hosts.

6. Instalando o IPsec

A maioria das versões modernas do FreeBSD possui suporte IPsec em seu código base. Portanto, você precisará incluir a opção `IPSEC` na configuração do seu kernel e, após recompilar e reinstalar o kernel, configurar as conexões IPsec usando o comando setkey(8).

Um guia abrangente sobre como executar o IPsec no FreeBSD está disponível no Handbook do FreeBSD.

7. src/sys/i386/conf/KERNELNAME

Isso precisa estar presente no arquivo de configuração do kernel para capturar dados de rede com o comando tcpdump(1). Certifique-se de executar config(8) após adicionar isso e reconstruir e reinstalar o kernel.

```
device  bpf
```

8. Teste estatístico universal de Maurer (para tamanho de bloco = 8 bits)

Você pode encontrar o mesmo código neste 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;
        }
      }
    }
  }
}
```