Experimento 1 - Remasterização de um sistema operacional Linux embarcado aplicado a um sistema de processamento de imagens

André Mateus R. Dantas
Matrícula: 11/0008359
Universidade de Brasília - Faculdade Gama
ymateus@hotmail.com

1. Objetivos

Remasterização do Damn Small Linux para uma plataforma embarcada aplicado a um sistema de processamento de imagens.

2. Introdução

Utilizou-se a remasterização para criarmos um sistema embarcado dedicado para criarmos a solução ao problema protosto.

Este experimento irá simular um sistema de interceptação (que será simulada por meio de arquivos) e decodificação de informações esteganografadas em vídeos (sequencia de imagens). A esteganografia é uma técnica utilizada para esconder uma informação dentro de outra. Observe que seu propósito é diferente ao da criptografia, onde o objetivo é proteger o conteúdo da mensagem.

3. Especificação dos Sistema

3.1. Descrição do sistema implementado

Uma das formas mais comuns da esteganografia é utilizando a técnica denominada *Least Significant Bits* (LSB), que consiste em utilizar os bits menos significativos para codificar a mensagem a ser escondida. Que foi a técnica utilizada nas imagens nesse experimento.Entretanto a esteganografia por LSB é muito vulnerável, pois é possível detectar e extrair a informação escondida com facilidade utilizando algumas técnicas de esteganálise. E portanto, foi inserido ao sistema de esteganografia um processo simples de embaralhamento. Neste caso, a esteganografia foi implementada em apenas um ou nenhum pixel para cada bloco de tamanho 3x3 pixels da imagem da seguinte forma:

1	4	7
2	5	8
3	6	9

O valor 0 indica que não existe nenhum pixel que foi esteganografado naquele bloco. Esta posição é obtida externamente utilizando uma chave decimal de forma circular. E cada bloco é analisado sem que haja sobreposição entre eles. Já os blocos da borda que não possuírem tamanho 3x3 foram desconsiderados, ou seja, não possuem informação esteganografada. E por fim deve ser feito um processo de flipagem nos bits decodificados o que causa um efeito visual melhor.

O sistema irá inserir um cabeçalho à imagem processada. O formato que será utilizado é o PGM (*portable graymap*), que contém um cabeçalho e a matriz correspondente da imagem. O cabeçalho pode ser feito inserindo a seguinte informação: P5 W H M I, onde P5 indica o formato PGM, W = largura em pixels, H = altura em pixels, M = valor máximo do pixel e I = imagem desesteganografada. Neste caso, o valor dos pixels são representados por um byte, ou seja, variam entre 0 e 255. Neste caso, o cabeçalho será inserido na imagem processada por um programa, utilizando a chamada de sistema ***system()***. Uma funcionalidade básica adicionada ao sistema é a capacidade de receber o sinal de interrupção (SIGINT), que é emitido aos processos do terminal quando as teclas de interrupção (por exemplo: CTRL+c) do teclado são acionadas, o código deve abortar o processamento inserindo zeros até o preenchimento dos pixels da imagem processada, em seguida inserir o cabeçalho à imagem parcialmente processada.

3.2. Implementação e prototipação

A solução proposta para o problema apresentado consiste em dois programas (um que desesteganografa a imagem e o outro insere o cabeçalho a esta imagem). O primeiro programa é basicamente composto pelas funções: aloca_matriz(int i, int j), colher_dados(), desembaralha(), flip(), interromper() e a main(). A função aloca_matriz simplismente aloca uma matriz com as dimensões i e j. A função colher_dados apenas lê os dados necessários do arquivo de parâmentros (parametros.txt).

Já na função desembaralha trabalho sobre a matriz que contém todo o vídeo, percorre essa matriz incrementado de 3 em 3 (tanto nas linhas quanto nas colonas), e a partir do valor da chave ele define qual pixel da matriz formada pelos valores nos índices:

$$\begin{pmatrix} (i,j) & (i,j+1) & (i,j+2) \\ (i+1,j) & (i+1,j+1) & (i+1,j+2) \\ (i+2,j) & (i+2,j+1) & (i+2,j+2) \end{pmatrix}$$

Como os valores de i e j são incrementados de 3 em 3, temos que todas as imagens do video são varridas em blocos de matrizes 3x3 e de acordo com a chave o valor correto é selecionado ou nenhum valor é. Isso se repete até que a imagem de saída esteja completa, esta e a função

flip, que inverte os bits de cada pixel da imagem montada na função desembaralha, são basicamente onde se encontra todo o processamento do sistema.

A função interromper(), esta associada a um sinal (SIGINT), que é disparado pelas teclas CTRL+C do teclado, fazendo com que o programa insira o cabeçalho na imagem processada até ali (os elementos não processados da imagem já terão o valor 0, pois a matriz foi alocada com o comando malloc, que já aloca a matriz e a preenche com zeros) e encerra o programa. Já na função main copia-se o valor dos elementos do vídeo para uma matriz dimanicamente alocada, inicializa-se as duas threads associadas as funções desembaralha e flip e se escreve a imagem resultante.

Para compilar esse código e executar o progama basta utilizar as seguintes linhas de comando na pasta onde os arquivos estao:

$gcc -o cabecalho cabecalho.c
$gcc -o exp1 exp1.c -lpthread
$./exp1

Agora vamos remasterizar o DSL, o primeiro passo é definir um local que será o ambiente de trabalho, para isso criou-se uma pasta chamada DSL no diretório home.

 # cd /home; mkdir DSL

Dentro de DSL baixou-se a distro dsl-4.4.10-embedded.zip, então criou-se duas pastas: a primeita denominada dsl-4.4.10-embedded, local onde foi descompactada o arquivo dsl-4.4.10-embedded.zip e a segunda nomeu-se remaster e continha três outros diretórios : image, para montar o sistema de arquivos da distro. Master, usada para inserir a nova distro que será gravada no cartão de memória. E source/KNOPPIX, que foi o diretório fonte da nova distro.

 # cd DSL; # mkdir dsl-4.4.10-embedded
 # wget –user=aluno –password=UnB_Gama
http://www.image.unb.br/mintsu/Gama/SistEmb/dsl-4.4.10-embedded.zip aluno
UnB_Gama

 # unzip dsl-4.4.10-embedded.zip -d
dsl-4.4.10-embedded
 # rm dsl-4.4.10-embedded.zip
 # mkdir remaster; cd remaster
 # mkdir -p image master source/KNOPPIX

Em seguida copiou-se a distribuição dsl-4.4.10-embedded para o diretório master.

 # rsync -Hav ../dsl-4.4.10-embedded/. master

Então fez-se a descompressão do KNOPPIX na forma de um arquivo temporário tmp.iso, para isso instalou-se o pacote cloop-utils.

 # apt-get install cloop-utils
#touch tmp.iso
 # extract_compressed_fs master/KNOPPIX/KNOPPIX
tmp.iso

O próximo passo foi montar o arquivo tmp.iso em image, e copiou-se o conteúdo para source/KNOPPIX.

 # mount -o loop tmp.iso image # rsync -Hav image/.
source/KNOPPIX # umount image

Removeu-se então o arquivo tmp.iso e a pasta image.

 # rm -R image tmp.iso

Nesse ponto entrou-se no diretório raiz do Damn Small Linux, disponível em source/KNOPPIX. Então mudou-se a flag de vdso-enabled para 0, permitindo assim o uso do comando chroot. Em seguida montou-se o diretório proc, com os comandos a seguir.

 # echo 0 > /proc/sys/abi/vsyscall32
 # chroot souce/KNOPPIX
 # mount -t proc /proc proc

O passo seguinte foi a inserção de pacotes do Debian, como o Mirror responsável pelos downloads de arquivos DSL estava desatualizado foi preciso modificá-lo pelo recente. Então habilitou-se o apt-get e o atualizou.

#echo Mirror:
distro.ibiblio.org/pub/linux/distributions/damnsmall/
>/opt/.dslrc; echo Protocol: http »/opt/.dslrc
 # dpkg-restore
 # dpkg -l
 # apt-get update

Enfim, possibilitou-se a instalação do minicom e do gcc para DSL de forma fácil:

 # apt-get install minicom
#apt-get install gcc

A partir desse ponto, tem-se o novo Damn Small Linux com os novos recursos, então desmontou-se /proc e saiu-se do chroot.

 # umount /proc
 # exit

Removeu-se o arquivo KNOPPIX da distro anterior e em seguida criou-se um novo arquivo KNOPPIX.

 # rm master/KNOPPIX/KNOPPIX
 # mkisofs -R -U -V -hide-rr-moved -cache-inodes -no-bak -pad source/KNOPPIX | create_compressed_fs - 65536 > master/KNOPPIX/KNOPPIX

Ápos esse ponto, bastou formatar o pendrive utilizando o Gparted, copiar os arquivos da pasta master para o dispositivo e instalar e torná-lo bootavel com o syslinux, com os seguintes comandos:

 # mount -t vfat /dev/sdb1 /mnt/card; # rsync -Hav /home/DSL/remaster/master/. /mnt/card
 # syslinux – install /dev/sdb1
 # umount /mnt/card

4. Resultados e Discussões

Como resultados deste experimento temos o total de quatro imagens geradas a partir dos arquivos de vídeo propostos. Duas eram fotos da Lena Söderberg com uma diferença de constraste, a terceira ela de um homem andando com um guarda-chuva e a terceira, que segue a baixo, todas rodaram no DSL rematerizado.

Figura 1. Alunos durante a prova.

5. Conclusão

A esteganografia digital, ou seja, a arte de esconder informações em meios digitais, vem sendo cada vez mais pesquisada e utilizada nos dias de hoje. Ela possui uma infinidade de aplicações, e talvez a mais importante delas seja a segurança da informação, já que, com a esteganografia, as mensagens ficam escondidas nos meios usados, e a informação passa despercebida por terceiros. Neste experimento

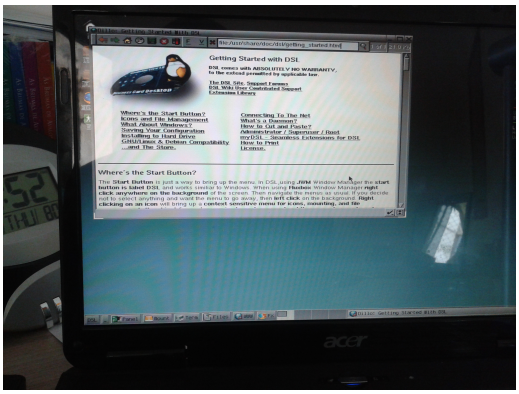

Figura 2. Damn Small Linux rodando.

mostrou-se uma técnica simples de esteganografada (LSB), e como ela pode ser utilizada para esconder imagens dentro de outros sem comprometer significativamente a qualidade visial das imagens. Em seguida implementou-se uma solução relativamente simples que utilizava vários recursos do sistema como : threads, processos (system) e sinais. A esteganografia digital, ou seja, a arte de esconder informações em meios digitais, vem sendo cada vez mais pesquisada e utilizada nos dias de hoje. Ela possui uma infinidade de aplicações, e talvez a mais importante delas seja a segurança da informação, já que, com a esteganografia, as mensagens ficam escondidas nos meios usados, e a informação passa despercebida por terceiros.

Referências

[1] http://image.unb.br/mintsu/Gama/SistEmb/. Roteiro do experimento 1. *Acessado em : 17/04/2014*.

[2] http://www.damnsmalllinux.org/. *Damm Small Linux webpage*. 2014.

6. Anexos

6.1. Código do Sistema de decodifica as imagens

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <math.h>
#include <pthread.h>
#include <signal.h>

/*----------------------Variaveis-----------------*/
int  w_video, h_video, w_imagem, h_imagem, num_bits;
int  h=0,i_img=0, j_img=0,num_frame=0;
unsigned char **frame, **imagem;
char nome_video[100],nome_chave[100];
FILE *video, *chave, *foto;


/*----------------------Funcoes-----------------*/
unsigned char** aloca_matriz(int i, int j){

        int k;
        unsigned char **matriz = (unsigned char **) malloc (i*sizeof(unsigned char*));
        if(matriz == NULL)
                {
                        system("clear");
                        fprintf(stderr, "Erro na alocacao de memoria.\n\n");
                        exit(1);
                }
        for(k=0 ; k<i;k++){
                matriz[k] = (unsigned char *) malloc (j*sizeof(unsigned char));
                if(matriz[k] == NULL)
                {
                        system("clear");
                        fprintf(stderr, "Erro na alocacao de memoria.\n\n");
                        exit(1);
                }
        }


        return matriz;
}
void colher_dados()
{
        int i, j;

        FILE* parametros;

        parametros=fopen("parametros.txt","rb");


        fscanf(parametros,"%s ",nome_video);

        fscanf(parametros,"%s ",nome_chave);

```

```
54          fscanf(parametros, "%d ",&w_imagem );
55
56          fscanf(parametros, "%d ",&h_imagem );
57
58          fscanf(parametros,"%d ",&w_video);
59
60          fscanf(parametros,"%d ",&h_video);
61
62
63          fscanf(parametros,"%d\n",&num_bits);
64          printf("%d\n",num_bits );
65          sleep(5);
66
67
68          fclose(parametros);
69
70
71  }
72
73  void *desembaralha(){
74
75          int i,j,vet[]= {9,8,7,6,5,4,3,2,1,0,0,0,0,0,0,0};
76          int valor=0,borda_h,borda_w;
77          borda_w = w_video - w_video%3;
78          borda_h = h_video - h_video%3;
79
80          for(i=0 ; i<borda_h*num_frame; i=i+3)
81                  for(j=0 ; j<borda_w;j=j+3){
82
83                          fscanf(chave,"%d ",&valor);
84                          if(feof(chave)){
85                                  rewind(chave);
86
87                          }
88
89                          if(i_img==(h_imagem))return;
90
91                          switch(valor){
92                                  case 0:
93                                          break;
94                                  case 1:
95                                          imagem[i_img][j_img]=frame[i][j];
96                                          j_img++; break;
97                                  case 2:
98                                          imagem[i_img][j_img]=frame[i+1][j];
99                                          j_img++; break;
100                                 case 3:
101                                         imagem[i_img][j_img]=frame[i+2][j];
102                                         j_img++; break;
103                                 case 4:
104                                          imagem[i_img][j_img]=frame[i][j+1];
105                                         j_img++;break;
106                                 case 5:
107                                         imagem[i_img][j_img]=frame[i+1][j+1];
108                                         j_img++; break;
109                                 case 6:
110                                         imagem[i_img][j_img]=frame[i+2][j+1];
111                                         j_img++; break;
```

```
                                case 7:
                                        imagem[i_img][j_img]=frame[i][j+2];
                                        j_img++; break;
                                case 8:
                                         imagem[i_img][j_img]=frame[i+1][j+2];
                                        j_img++;break;
                                case 9:
                                        imagem[i_img][j_img]=frame[i+2][j+2];
                                        j_img++; break;
                        }

                        if(j_img>=(w_imagem)){
                                i_img++;
                                j_img=0;
                        }

                }
}


void *flip(){

        int i,j,k;

        unsigned char temp, temp1 =0,temp2=0,mask1 = 0x1,mask2=0x80;

        for(i=0; i<h_imagem; i++)
                for(j=0; j<w_imagem; j++){

                        temp=(unsigned char )imagem[i][j];


                        temp1 =0;
                        temp2 = 0;
                        for(k=0; k<=3; k++){

                                temp1|=(imagem[i][j]&(mask1<<k))<<(7-2*k);
                                temp2|=(imagem[i][j]&(mask2>>k))>>(7-2*k);

                        }
                        temp= temp1|temp2;
                        temp = temp>>(8-num_bits);
                        temp = temp<<(8-num_bits);

                        imagem[i][j]=temp;

                }
}

void interromper(){
        int i,j;
        char temp[100] ;
        sprintf(temp,"./cabecalho result.y %d %d",w_imagem,h_imagem);
        system(temp);

        foto = fopen("result.y", "a+");
        if(!foto)
        {
```

```
                system("clear");
                fprintf(stderr, "Erro na abertura do frame da imagem de saida.\n\n");
                exit(1);
        }
        flip();

        for(i=0; i<h_imagem; i++)
                for(j=0; j<w_imagem; j++)
                        fputc(imagem[i][j], foto);
        printf("Imagem parcialmente gerada com sucesso!\n");

        fclose(foto);
        exit(0);
}
int main(int argc, unsigned char *argv[])
{


        int i,j,tam_arq=0;
        pthread_t thread1;
        pthread_t thread2;


        signal(SIGINT,interromper);

        colher_dados();

        video = fopen(nome_video, "rb");
        if(!video)
        {

                fprintf(stderr, "Erro na abertura do frame, verifique o nome dos frame de
                    entrada. \n\n");
                exit(1);
        }
        chave = fopen(nome_chave, "rb");
        if(!chave)
        {

                fprintf(stderr, "Erro na abertura da chave. \n\n");
                exit(1);
        }

        while(fgetc(video)!=    EOF) tam_arq++;
                rewind(video);


        num_frame= tam_ arq/(w_video*h_video);

        imagem = alocaMatriz(h_imagem,w_imagem);
        frame = alocaMatriz(h_video*num_frame,w_video);



        for(i=0; i<h_video*num_frame; i++)
                for(j=0; j<w_video; j++){
                        frame[i][j] = fgetc(video);
                }
```

```
227                 fclose(video);
228
229
230
231
232         pthread_create (&thread1, NULL, (void *)desembaralha, NULL);
233         pthread_join (thread1, NULL);
234         pthread_create (&thread2, NULL, (void *)flip, NULL);
235         pthread_join (thread2, NULL);
236
237         fclose(chave);
238
239
240
241         char temp[100] ;
242         sprintf(temp,"./cabecalho result.y %d %d",w_imagem,h_imagem);
243         system(temp);
244
245         foto = fopen("result.y", "a+");
246         if(!foto)
247         {
248                 system("clear");
249                 fprintf(stderr, "Erro na abertura do frame da imagem de saida.\n\n");
250                 exit(1);
251         }
252
253
254         for(i=0; i<h_imagem; i++)
255                 for(j=0; j<w_imagem; j++)
256                         fputc(imagem[i][j], foto);
257
258         printf("Imagem gerada com sucesso!\n");
259
260         fclose(foto);
261
262
263
264
265         for(i=0; i<h_video; i++)
266                 free(frame[i]);
267         free(frame);
268
269         for(i=0; i<h_imagem; i++)
270                 free(imagem[i]);
271         free(imagem);
272
273 }
```

6.2. Código do processo que adiciona cabeçalho as imagens

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4  int main (int argc, char* argv[])
5
6  {
7          FILE* foto;
8
```

```
9           foto = fopen(argv[1],"wb");
10          fprintf(foto, "P5 %d %d 255\n",atoi(argv[2]),atoi(argv[3]) );
11
12          fclose(foto);
13  }
```