Análise e alternativas de soluções para o problema envolvendo "Utilização de técnicas de esteganografia"

Guilherme Krzisch e Pedro Webber Neto
Faculdade de Informática - PUCRS

06 de maio de 2012

Resumo

Este artigo pretende demonstrar uma solução para o problema dado na disciplina de Programação para Software Básico. O problema consiste em esconder uma imagem dentro de outra, utilizando técnicas esteganográficas. São apresentadas diferentes soluções, cada uma evoluindo das anteriores e aprendendo com os erros delas.

1 Introdução

O problema proposto na disciplina de Programação para Software Básico consiste em ter que achar um algoritmo capaz de esconder uma imagem dentro de outra, sem que seja perceptível a alteração da imagem original. Em outras palavras, trabalharemos basicamento com três imagens: a imagem original, a imagem escondida e a imagem resultante. Esta última é o resultado da utilização de técnicas de esteganografia para esconder a imagem escondida na imagem original.

A linguagem utilizada para a implementação é C/C++, juntamente com a biblioteca SDL para tratamento das imagens.

Apresentamos diferentes soluções em sequência, explicando a sua lógica e os problemas encontrados, até chegarmos na solução definitiva, que foi a que se adequou melhor ao problema.

2 Desenvolvimento de soluções

Começamos pensando da forma mais intuitiva, que é esconder os pixels da imagem escondida na imagem original utilizando alguma sequência. Pensamos numa sequência simples, escondendo os pixels sempre a intervalos regulares. Chegamos então a fórmula $(altura_1 * largura_1)/(altura_2 * largura_2)$, convertendo o resultado para inteiro. Desta forma fazemos uma comparação entre

as duas imagens, vendo quanto a primeira é maior do que a segunda, e então escondendo os pixels na mesma proporção. Por exemplo, se a imagem 1 tem 100 pixels de altura por 100 pixels de largura, e a imagem 2 tem 50 pixels de altura por 50 pixels de largura, o resultado será o número quatro; este resultado será usado para fazer o intervalo para esconder os pixels (se o primeiro pixel for escondido no lugar 1, o segundo será no lugar 5, depois no 9, e assim sucessivamente).

Outro quesito do problema era definir os delimitadores de início e fim da imagem. Para o delimitador de início, resolvemos deixar os cinco primeiros pixels da imagem resultante com o mesmo RGB (red, green e blue); em vez de escolher um RGB aleatório, escolhemos o mesmo do primeiro pixel, assim diminuindo a probabilidade de alguém conseguir descobrir que a imagem continha técnicas esteganográficas. E em vez de utilizar o delimitador de fim da imagem, resolvemos esconder, também nos primeiros pixels, a largura e a altura da imagem escondida, pois só deste jeito poderíamos "montar" ela de novo (além de agora também sabermos aonde ela termina, só precisando fazer o cálculo do tamanho da imagem); se só usássemos o delimitador de fim da imagem, não seria possível saber as dimensões da imagem escondida.

Essa foi a primeira solução, porém quando a implementamos, ficou claro o uso de técnicas de manipulação na imagem (ficaram "traços" diagonais na imagem resultante). Então tivemos que achar outra alternativa para a resolução do problema.

Lendo o artigo sobre esteganografia em [1], vimos que há uma técnica para esconder informações que utiliza só o bit menos significativo da informação original, assim se tornando bem mais difícil a percepção de alterações.

Portanto implementamos esta ideia como a *política 1* do nosso programa. Assim cada pixel da imagem escondida deve ser espalhada pelos bits menos significativos de oito pixels da imagem original. Como um pixel tem três componentes (RGB) e cada um desses componentes armazena 8 bits de informação, para guardamos um pixel devemos ter 24 (8*3) "lugares". Então guardamos nos bits menos significativos das componentes RGB de oito pixels da imagem original, o que totaliza os 24 lugares requeridos.

O problema desta política é que a imagem original deve ser, pelo menos, oito vezes maior do que a imagem a ser escondida. Esta restrição nos motivou a procurar uma política melhor.

Assim chegamos na *política 2*, que consiste em esconder nos *dois* bits menos significativos da imagem original. Constatamos que mesmo alterando estes dois bits, a mudança é imperceptível. Agora a imagem original deve ser, pelo menos, quatro vezes maior que a imagem escondida.

Queríamos reduzir ainda mais esta restrição, portanto resolvemos converter a imagem escondida para tons de cinza, assim as três componentes RGB terão o mesmo valor, sendo assim só precisaremos armazenar um deles, pois os três são iguais. Se agora só temos que armazenar um número que tem oito bits, precisaremos de três pixels da imagem original (o RGB do primeiro pixel e do segundo pixel e o RG do terceiro pixel). Esta foi a *política 3*.

Para a *política 4*, utilizamos a mesma técnica de esconder a imagem em

tons de cinza. Porém agora resolvemos esconder o número de 8 bits em *um só pixel* da imagem original. Para isso devemos dividir esse número em 3 partes: a primeira parte, com 3 bits, irá nos bits menos significativos da componente R; a segunda parte, também com 3 bits, irá nos bits menos significativos da componente G; e por fim, para completar os 8 bits, os últimos 2 bits irão para os bits menos significativos da componente B. Agora conseguimos esconder uma imagem escondida dentro da imagem original, com as duas podendo ter o *mesmo* tamanho.

Utilizamos o mesmo processo descrito acima para a *política 5*, porém agora escondendo tudo ao contrário; começamos no fim da imagem e vamos indo para o início.

E, por fim, desenvolvemos a *política 6*, a definitiva. A base é a mesma da política 4, porém, em vez de ir escondendo de um em um pixel, fomos escondendo usando a mesma fórmula descrita e explicada no começo do artigo: $(altura_1 * largura_1)/(altura_2 * largura_2)$, assim deixando os bits escondidos mais espalhados.

3 Resultados e conclusão

Conforme íamos nos dando conta de pequenos detalhes, fomos evoluindo de implementação, até chegarmos na final, a *política 6*. O resultado foi uma imagem sem mostras de alteração, mas que, na realidade, esconde, nos seus bits menos significativos, outra imagem - e esta outra imagem pode ser até do mesmo tamanho. A única observação é que a imagem escondida ficará sem cor, pois foi transformada para tons de cinza, porém não achamos que a cor seja algo muito importante quando alguém deseja esconder uma imagem, portanto ficamos com essa implementação.

Também foi desenvolvido um módulo alternativo que esconde um código fonte escrito em C++ (.cpp), e depois o recupera, o compila e o executa.

Referências

[1] Steganography. Disponível em: <http://en.wikipedia.org/wiki/Steganography>. Acesso em: 06 de maio de 2012.