Politechnika Śląska
Wydział Informatyki, Elektroniki i Informatyki

Programowanie Komputerów 2

Steganografia w pliku BMP

autor	Wojciech Janota
prowadzący	dr inż. Mirosław Skrzewski
rok akademicki	2019/2020
kierunek	informatyka
rodzaj studiów	SSI
semestr	2
termin laboratorium	środa, 15:30 – 17:00
sekcja	51
termin oddania sprawozdania	2020-07-27

1 Treść zadania

Proszę napisać program umożliwiający ukrycie / odczyt w/z pliku BMP innego pliku dowolnego typu. Program powinien przyjmować liczbę bitów każdej ze składowych RGB, które zostaną przeznaczone na ukrycie informacji. Program powinien działać dla różnych głębi kolorów. Program uruchamiany z linii komend z przełącznikami: -i plik do ukrycia, -p plik obrazu, -o plik wyjściowy, -r odczyt ukrytej informacji, -b liczba bitów składowej RGB.

Uwagi: Jako strukturę danych proszę zastosować tablicę dynamiczną.

2 Analiza zadania

Zagadnienie przedstawia problem zapisania dowolnego pliku na ostatnich n bitach składowych tablicy pixeli obrazu w formacie BMP.

2.1 Struktury danych

Program wykorzystuje tablice dynamiczne do przechowywania poszczególnych składowych pixeli oraz bajtów plików do zapisania. Wykorzystuje także struktury przechowujące nagłówki plików BMP oraz specjalny dodatkowy nagłówek steganograficzny, który przechowuje informacje o tym, czy dany plik zawiera ukryte informacje oraz rozmiar ukrytego pliku. Struktury szczegółowo omówiłem w dokumentacji, która jest w załączniku.

2.2 Algorytmy

Program ukrywa pliki dowolnego typu w podanym pliku BMP. Nie wykorzystałem żadnych skomplikowanych algorytmów, jedynie proste operacje na maskach bitowych, które umożliwiły mi zapis oraz odczyt danych do oraz z pliku BMP.

Maski bitowe których używałem dla plików 24- i 32-bitowych pozwalają mi zapisywać i odczytywać dane na ostatnich n bitach pełnego bajta danych. Obliczam je w następujący sposób: `mask = (1 << n) - 1`. Wtedy mogę za pomocą operatora bitowego `AND` oraz `NOT` "wyczyścićóstatnie n bitów, czyli je wyzerować, a następnie za pomocą operatora `OR` zapisać tam podane dane. Użycie operatora `OR` jest poprawne, ponieważ te ostatnie miejsca są wyzerowane.

2.3 Analiza pliku BMP

Struktura pliku BMP jest następująca[2]:

Nazwa struktury	Rozmiar	Opcjonalny	Znaczenie
Nagłówek główny	14 bajtów	Nie	Przechowuje najważniejsze informacje o pliku.
Nagłówek DIB	7 różnych wersji	Nie	Przechowuje najważniejsze informacje o obrazie i definiuje format piksela.
Dodatkowe maski	12 lub 16 bajtów	Tak	Definiuje format piksela.
Tablica kolorów	zmienny	Tak/Nie	Definiuje kolory użyte przez obraz (tablicę pikseli)
Przerwa	zmienny	Tak	Struktura dostosowująca
Tablica pixeli	zmienny	Nie	Definiuje wartości poszczególnych pikseli
Przerwa	zmienny	Tak	Struktura dostosowująca
Profil kolorów ICC	zmienny	Tak	Definiuje profil kolorów dla systemu zarządzania kolorem

Pliki BMP zawierają na początku nagłówek główny, nazywany przez mnie dalej nagłówkiem `BMPHEADER`. Jego struktura jest następująca[1]:

Offset	Rozmiar	Przeznaczenie
0x0	2 bajty	nagłówek determinujący, czy plik jest plikiem BMP, dla pliku BMP wynosi BM
0x2	4 bajty	Rozmiar całego pliku w bajtach
0x6	2 bajty	zarezerwowany
0x8	2 bajty	zarezerwowany
0xA	4 bajty	Offset, początkowy adres tablicy pixeli

Następny jest nagłówek `DIBINFOHEADER`, nazywany przeze mnie `DIBHEADER`. Jego struktura jest następująca[1]:

Offset	Rozmiar	Nazwa	Opis zawartości
0x14	4 bajty	biSize	Wielkość nagłówka informacyjnego
0x18	4 bajty	biWidth	Szerokość obrazu w pikselach
0x22	4 bajty	biHeight	Wysokość obrazu w pikselach
0x26	2 bajty	biPlanes	Liczba warstw kolorów, zwykle 1
0x28	2 bajty	biBitCount	Liczba bitów na piksel
0x30	4 bajty	biCompression	Algorytm kompresji
0x34	4 bajty	biSizeImage	Rozmiar samego obrazu
0x38	4 bajty	biXPelsPerMeter	Rozdzielczość pozioma
0x42	4 bajty	biYPelsPerMeter	Rozdzielczość pionowa
0x46	4 bajty	biClrUsed	Liczba kolorów w palecie
0x50	4 bajty	biClrImportant	Liczba ważnych kolorów w palecie (gdy 0 to wszystkie są ważne)

Po tych nagłówkach na offsecie podanym w nagłówku BMPHEADER znajduje się tablica pixeli. Dla obrazów 16 bitowych składowe jednego pixela są zapisane na 2 bajtach, dla 24 bitowych na 3 bajtach a dla 32 bitowych na 4 bajtach. W każdym wierszu tablicy pixeli zapisanym do pliku musi być wielokrotność 4 bajtów, więc brakujące bajty sa nadpisywane zerami[2].

3 Specyfikacja zewnętrzna

Program jest uruchamiany z linii poleceń. Po podaniu flagi -h wyświetla się pomoc, która podpowiada prawidłową semantykę wywołania programu. Do programu należy przekazać następujące flagi:

- -w — zapisywanie danych do obrazu

- -r — odczytywanie danych z obrazu
- -p — obraz wejściowy
- -i — plik z danymi do ukrycia
- -o — plik/obraz wyjściowy
- -b — liczba bitów na których ma być zapisana informacja

Poprawne wywołanie programu powinno zawierać flagi: -h, -w -i -o -p -b, lub -r -o -b.

Poza pierwszym argumentem, kolejność flag nie ma znaczenia.

4 Specyfikacja wewnętrzna

4.1 Ogólna struktura programu

W funkcji głównej najpierw odczytywane są przekazane pierwsze flagi. Następnie, w przypadku podania flagi -h wyświetlana jest pomoc, natomiast w przypadku, gdy pierwszymi argumentami są -r lub -w rozpoczyna się wykonanie programu.

Dla flagi -w najpierw odczytywane są pozostałe flagi oraz dane, następnie po sprawdzeniu ich poprawności oraz otwarciu odpowiednich plików do pamięci za pomocą funkcji `read_headers()` zostają wczytane nagłówki pliku BMP: `BMPHEADER` oraz `DIBHEADER`. Po wypisaniu podstawowych informacji o pliku z nagłówków w zależności od głębi kolorów danego pliku wywołana zostaje funkcja `pixelArray_read_nbit()`, gdzie n to 16, 24 lub 32. Funkcja ta wczytuje do tablicy dynamicznej `pixelArray` tablicę pixeli w pliku BMP. Następnie funkcja `read_file_to_memory()` wczytuje do tablicy `dataArray` plik do ukrycia w pliku obrazu. Następnie po sprawdzeniu poprawności wczytania danych funkcja `write_data_to_image()` zapisuje dane do pliku. Po tej operacji zostają zamknięte pliki oraz zwolniona pamięć.

Dla flagi -r najpierw odcztytwane są pozostałe flagi i dane, następnie po sprawdzeniu ich poprawności oraz otwarciu odpowiednich plików do pamięci za pomocą funkcji `read_headers()` zostają wczytane nagłówki pliku BMP: `BMPHEADER` oraz `DIBHEADER`, analogicznie do wywołania z flagą -w. Po sprawdzeniu poprawności podanych danych oraz odczytu nagłówków także zostaje wywołana funkcja `pixelArray_read_nbit()`, tak samo jak to miało miejsce dla flagi -w. Następnie wywoływana jest funkcja `write_data_from_image()`, która odczytuje dane z obrazu i zapisuje je do pliku wynikowego. Po zwolnieniu pamięci i zamknięciu plików następuje zakończenie pracy programu.

4.2 Szczegółowy opis typów i funkcji

Szczegółowy opis typów i funkcji zawarty jest w załączniku.

5 Testowanie

Program został przetestowany dla wszystkich obsługiwanych rodzajów plików BMP tj. 16, 24 i 32-bitowych. Ukrywanie pliku pustego spowoduje późniejsze odczytanie pliku pustego, ponieważ program zapisuje także wielkość ukrytego pliku. W wyniku limitacji powstałych przez zapisywanie wielkości pliku w wolnym miejscu w nagłówku BMP wielkość ukrywanych danych nie może przekroczyć 65kB, ponieważ do dyspozycji pozostało jedynie 16 bitów, na których można zapisać maksymalnie liczbę 65535. Przy próbie zapisania większego pliku program zwróci błąd, mówiący o zbyt dużym pliku.

Program został także sprawdzony pod kątem wycieków pamięci przy pomocy narzędzia `valgrind`. Wyniki tego testu, napierw dla komendy -w, a potem -r pozostawiam poniżej:

```
    ==33417== Memcheck, a memory error detector
==33417== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward
==33417== et al.
==33417== Using Valgrind-3.16.0.GIT and LibVEX; rerun with -h
==33417== for copyright info
==33417== Command: ./main -w -i data.bin -p
==33417== ../images/image16.bmp -o image_new.bmp -b 1
==33417==
==33417== HEAP SUMMARY:
==33417==     in use at exit: 0 bytes in 0 blocks
==33417==   total heap usage: 13 allocs, 13 frees, 5,323,078 bytes
==33417==   allocated
==33417== All heap blocks were freed -- no leaks are possible
==33417==
==33417== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)

==33769== Memcheck, a memory error detector
==33769== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward
==33769== et al.
==33769== Using Valgrind-3.16.0.GIT and LibVEX; rerun with -h
==33796== for copyright info
==33769== Command: ./main -r -p image_new.bmp -o newdata.bin -b 1
==33769==
```

```
Odczytywanie ukrytego pliku z pliku BMP
Dane zapisane poprawnie!
==33769==
==33769== HEAP SUMMARY:
==33769==     in use at exit: 0 bytes in 0 blocks
==33769==   total heap usage: 10 allocs, 10 frees, 5,314,670 bytes
==33796==   allocated
==33769==
==33769== All heap blocks were freed -- no leaks are possible
==33769==
==33769== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

6 Wnioski

Program korzystający ze steganografii do ukrywania danych w obrazach nie jest bardzo skomplikowanym programem, jednak częstość korzystania z masek bitowych oraz pewne cechy plików BMP takie jak potrzeba wyrównania pliku do wielokrotności 4 bajtów sprawiły mi pewne problemy. Mój pomysł zakładający używanie wolnych miejsc w nagłówku `BMPHEADER` także spowodował, że możliwe jest zapisanie tylko plików o maksymalnym rozmiarze 65kB. Przy testowaniu na innym środowisku graficznym dla systemu Linux zauważyłem, że domyślne narzędzia dla środowiska KDE Plasma 5 nie potrafią odczytać plików BMP 16 i 32-bitowych, a program GIMP w dziwny sposób interpretuje maski bitowe dla obrazów 32-bitowych. Używając narzędzia Eye Of Gnome byłem w stanie bezproblemowo wyświetlić dowolne obrazy 16, 24 i 32-bitowe przed i po ukryciu informacji. Podejrzewam, że te narzędzia w inny sposób interpretują nagłówki plików BMP

Literatura

[1] Microsoft (2018) *Bitmap Reference - Win32 apps — Microsoft Docs*, `https://docs.microsoft.com/pl-pl/windows/win32/gdi/bitmap-reference`

[2] S. Leonard, Penango Inc (September 2016) (ISSN: 2070-1721) *RFC 7903 - Windows Image Media Types*, `https://tools.ietf.org/html/rfc7903`

Dodatek
Szczegółowy opis typów i funkcji