Vademecum hakera. Edycja plików binarnych

Autor: Damian Daszkiewicz
ISBN: 83-7361-067-7
Format: B5, stron: 290
Zawiera CD-ROM

Większość dostępnych obecnie aplikacji pozwala użytkownikowi dokonywać rozmaitych modyfikacji: od zmian wyglądu i dodawania nowych ikon począwszy, na zapisywaniu własnych makrodefinicji skończywszy. Nie zawsze to wystarcza. Żeby naprawdę zmusić program do działania zgodnie z naszymi oczekiwaniami, trzeba wejść nieco głębiej w jego kod.

Dzięki tej książce poznasz wiele technik, które umożliwią Ci samodzielną modyfikację rozmaitych programów, zarówno aplikacji użytkowych, jak i gier. Dowiesz się, jak „oszukać" grę, jak zmieniać teksty w programach, jak odnajdywać furtki pozostawione przez programistów. A przy okazji poznasz tajniki programowania, które sam będziesz mógł wykorzystać w swojej praktyce.

- Poznasz różne systemy liczbowe i nauczysz się przeliczać wartości pomiędzy nimi
- Nauczysz się modyfikować kody wynikowe programów za pomocą Hex Workshop i kHexEditor
- Dowiesz się, jak zmieniać zasoby (ikony, teksty, skróty klawiaturowe) używając programów Resource Hacker, EXEScope i PE Resource Explorer
- Nauczysz się edytować programy wewnętrznie skompresowane
- Dowiesz się, jak zabezpieczać własne aplikacje przed modyfikacjami
- Poznasz sposoby „oszukiwania" gier
- Dowiesz się, jak odczytywać z dyskietek ukryte informacje
- Poznasz podstawy pisania własnych kompilatorów

Do książki dołączony jest CD-ROM zawierający przydatne narzędzia i kody źródłowe.

„Vademecum hakera" to książka, dzięki której wykonasz w programach zmiany, które wydają się niewykonalne. Zadziwisz siebie i swoich znajomych!

Wydawnictwo Helion
ul. Chopina 6
44-100 Gliwice
tel. (32)230-98-63
e-mail: helion@helion.pl

Spis treści

Rozdział 9.

Ukrywanie tajnych informacji

Wstęp

W tym rozdziale opiszę kilka sposobów na ukrywanie tajnych informacji na dysku. Nie mam zamiaru opisywać jakichś programów do szyfrowania danych itp., gdyż są one na tyle popularne, że do wielu z nich powstały specjalne deszyfratory, łamiące kod i wydobywające z zaszyfrowanych plików cenne dane. Oczywiście nie wszystkie programy szyfrujące da się tak łatwo pokonać. Jednak w tym rozdziale mam zamiar opisać nieco inne, dość oryginalne sztuczki, pozwalające ukryć nasze cenne dane. Jeśli ukrywanie tajnych informacji jest dla Ciebie interesującym tematem, nie zapomnij przeczytać także rozdziału 10., w którym opiszę dość ciekawy sposób na ukrywanie informacji na dyskietce.

Jak w plikach BMP można ukryć tajne informacje, np. hasło?

Pliki *BMP* posiadają dość prostą budowę: na początku pliku znajduje się nagłówek, w którym zapisane są różne informacje dotyczące pliku (np. liczba kolorów, wymiary pliku itp.). Dalej umieszczone są już tylko informacje o poszczególnych pikselach. W zależności od rodzaju pliku *BMP* jednemu pikselowi odpowiada określona liczba bitów (szczegóły przedstawiono w tabeli 9.1). W plikach piksele są zapisywane w dość dziwnej kolejności. Wydawałoby się, że powinny być zapisywane od lewej do prawej i z góry do dołu. Niestety tak nie jest, gdyż są zapisywane z lewej do prawej, ale z dołu do góry! Jednak to nie jest dla nas aż taka istotna informacja.

Tabela 9.1. *Różne typy plików BMP*

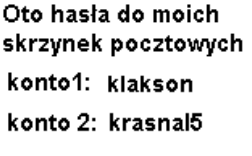
Typ pliku BMP	Ilość bitów opisująca jeden piksel
Monochromatyczny	1
16 kolorów	4
256 kolorów	8 (1 bajt)
24-bitowy (16,7 miliona kolorów)	24 (3 bajty)

Pojawia się drobne pytanko: jak można ukrywać informacje w plikach BMP? Odpowiedź brzmi: łatwiej niż myślisz. Na potrzeby książki wymyśliłem dwa ciekawe sposoby ukrywania informacji w plikach BMP. Pierwszy z nich to drobna ingerencja w nagłówek, a drugi sposób to przekształcenie pliku binarnego w plik BMP.

Ingerencja w nagłówek pliku BMP

Zanim będziemy ingerować w nagłówek pliku *BMP*, należy ten plik utworzyć. Do tego celu wystarczy nawet najprostszy program obsługujący pliki *BMP*, np. *Paint*. W programie tym należy zdefiniować rozmiar rysunku, np. 100×200 pixeli (ważne: oba te wymiary muszą być od siebie różne). Rysunek zapiszemy jako monochromatyczny, gdyż im więcej kolorów, tym więcej miejsca na dysku zajmuje plik. Kiedy rysunek został już zapisany, należy umieścić w nim jakiś napis, np. hasło do konta pocztowego. Można utworzyć tyle napisów, ile się zmieści — nie ma żadnych ograniczeń. Wygląd przykładowego pliku BMP zaprezentowano na rysunku 9.1.

Rysunek 9.1.
Oryginalny plik BMP



Skoro już wpisałeś jakieś hasła, zapisz zmiany dokonane w pliku. A teraz najprzyjemniejsza część, czyli ingerencja w nagłówek. Co chcemy zmienić w nagłówku? Dokonamy w nim zmiany wymiarów rysunku, tj. zmienimy rozmiar rysunku z 200×100 pixeli na 100×200 pixeli. Dzięki temu piksele inaczej się ułożą i utworzą chaotyczną „paplaninę punktów". Na rysunku 9.2 zaprezentowano wygląd pliku *BMP* po dokonaniu ingerencji w nagłówek.

Rysunek 9.2.
Plik BMP ze zmodyfikowanym nagłówkiem

Powstaje pytanie: jak dokonać zmian w nagłówku pliku *BMP*? Wystarczy plik *BMP* otworzyć w programie Hex Workshop i pod odpowiednimi offsetami zmienić wartości. Szerokość obrazu jest zapisana pod offsetem 12h, a wysokość — pod offsetem 16h. Wysokość i szerokość obrazu może wynosić maksymalnie 65 535 pikseli, a więc mamy do czynienia z liczbą 16-bitową (czyli pod offsetem 13h znajduje się starszy bajt szerokości obrazu — w naszym wypadku jest to zero — a pod offsetem 17h jest zapisany starszy bajt wysokości obrazu — w naszym wypadku również jest to zero).

Jednak z modyfikacją nagłówka nie należy przesadzać. Jeśli np. zdefiniujemy zarówno wysokość, jak i szerokość pliku jako 200 pikseli, takiego pliku nie będzie można otworzyć w żadnym programie graficznym, gdyż zostanie on uznany za uszkodzony (podczas odczytywania przez program opisów poszczególnych pikseli nagle pojawi się koniec pliku, mimo że w nagłówku zostały zdefiniowane większe wymiary niż są możliwe do odczytania; niektóre programy w takiej sytuacji zgłoszą błąd, inne z kolei odczytają poprawnie plik, wypełniając brakujące miejsca kolorem czarnym). Inna pułapka, która może na nas czyhać, to dobranie nieodpowiedniego rozmiaru pliku. Jeśli oryginalny plik ma wymiary 200×100 pikseli, a Ty zmienisz wymiary np. na 400×50 pikseli, istnieje prawdopodobieństwo, że rysunek będzie zawierał dwukrotnie napisane te same hasła, mające naturalną szerokość, ale wysokość o połowę mniejszą niż w oryginalnym pliku. Mimo to bez większych problemów będzie je można odczytać! Ten efekt występuje w wypadku edycji pliku 16-kolorowego, 256-kolorowego lub 24-bitowego. W przypadku plików monochromatycznych efekt ten może nie wystąpić (chyba że oba wymiary oryginalnego pliku były podzielne przez 8). Dlatego nie należy dobierać rozmiarów tak, aby pierwszy rozmiar był np. 2, 3, 4… razy większy od oryginalnego a drugi rozmiar — 2, 3, 4… razy mniejszy od oryginalnego. O ile w przypadku plików monochromatycznych ten efekt pojawia się sporadycznie, o tyle w plikach zawierających większą ilość kolorów występuje niemal zawsze.

Przekształcanie pliku binarnego w plik BMP

Powyższy sposób jest dość ciekawy, ale ma jedną wadę: nie można w pliku *BMP* o wymiarach 800×600 pikseli zapisać zbyt dużo informacji (raczej nikt normalny nie trzyma na dysku plików *BMP* o wymiarach np. kilka tysięcy pikseli na kilka tysięcy pikseli). Poza tym taki plik (800×600 pikseli), w którym da się zapisać kilkanaście (kilkadziesiąt) haseł, zajmuje na dysku dość sporo miejsca (1 MB). Jednak jest też inny sposób na ukrycie ważnych informacji w pliku *BMP*: otóż można dany plik tekstowy (lub binarny) przekształcić w plik *BMP*, tj. wstawić typowy nagłówek pliku *BMP*, zdefiniować odpowiednie rozmiary, a dalej skopiować zawartość naszego pliku. Po takowym sklejeniu otworzenie pliku *BMP* w dowolnym programie graficznym powinno być możliwe, tyle że pojawiłby się mały problem: znów plik *BMP* wyglądałby dość dziwnie (widoczne byłyby same chaotycznie rozmieszczone punkty). W listingu 9.1 zaprezentowano przykładowy plik tekstowy (plik ten wygenerowałem prostym programem tworzącym losowe pliki tekstowe).

Listing 9.1. *Fragment pliku dane.txt*

```
M]:U6;GV\Jf:fMXf4R<?=I3^U<:e483_IXG@Vg;1>R4a4DG>W?P
U]^9;dVGH<OY_==Gb^a\NIZ=@8]BW3@>M_bge@BQb2>JcbYWH8E
NLX13\`R6cALAKFO\chOR6I\Qh31D[[?;aRR;5^@>V;>F65g;RH
```

```
]IC?<=:;K3Q:Q\<bTGER74e=FX_bN;::J86B1G]Dg4OFRV[LCM9
\;WZTC2e@e3H61^HFCb8Eb<8_cPCTdJE^B5AAhZU96G[<fVT99A
]Z6b\FAKKUQ:C922g_Y\4cA8?XXfNC:6FV64[Hd\A8Q]Y<[;=NK
V;CfPU1\3=MbS[4]C?Y58R`9GUAOhXIZMbX_?>\XG@>Md_PT_EN
JP]\Q>R[AW:>Hgd\e84_OLT65fWJP>2PcFe59TI]VHNF1CfYOLZ
.... (plik zajmuje ponad 40KB, jednak tutaj przytoczylem tylko kilka pierwszych
wierszy) ....
```

Rysunek 9.3 prezentuje plik *BMP*, który został wygenerowany na podstawie powyższego pliku tekstowego.

Rysunek 9.3. *Plik BMP, w którym umieszczono plik tekstowy*

Niestety ten plik w książce wygląda tak ładnie tylko dlatego, że musiałem go zapisać w 256 odcieniach szarości. Tymczasem mój program zapisuje go w 256 kolorach, widoczne są więc takie kolory jak zielony, czerwony, żółty, niebieski itp.

Listing 9.2 zawiera kod źródłowy programu do generowania plików *BMP*.

Listing 9.2. *kodujbmp.pas — program, który z ważnych plików generuje pliki BMP*

```
Program KodujPlikiBMP;
uses crt;

var
 RozX:word; { wymiary pliku BMP}
 RozY:word; { -"-             }
 Reszta:word; {Ile ostatnich bajtow w pliku BMP to smiecie}
 RozmiarTXT:longint; {wielkosc pliku TXT}
 p,p2,p3:file of byte; {zmienne plikowe TXT/BMP/kodujbmp.dat}
 Plik1,Plik2:string; {Nazwy plikow}

Procedure NazwyPlikow;
Begin
 clrscr;
 write('KodujDoBMP v 1.0                                    (c) Damian Daszkiewicz');
 writeln('Program tworzy plik BMP, w ktorym opisy kolorow poszczegolnych pikseli to ');
 writeln('zawartosc innego pliku. Jest to dosc ciekawy sposob na ukrywanie ');
 writeln('informacji. Poza tym czasami powstaja calkiem interesujace pliki BMP');
 writeln('Program zostal napisany na potrzeby ksiazki Edycja Plikow Binarnych dla');
 writeln('wydawnictwa Helion. Nie zastrzezono do niego praw autorskich');
 writeln;

 Plik1:=ParamStr(1);
 if plik1='' then Begin
    Repeat
        write('Podaj nazwe pliku ktory chcesz zakodowac >>');
```

```
        readln(Plik1);
    until Plik1<>''
 end;

 Plik2:=ParamStr(2);
 if Plik2='' then Begin
   Repeat
     write('Podaj nazwe pliku BMP ktory chcesz utworzyc >>');
     readln(Plik2);
   until Plik2<>''
 end;
end;

{Jakie maja byc wymiary pliku BMP}
Procedure JakiRozmiar(X:longint);
var RX,RY:real;
Begin
   X:=X+2; {naglowek informujacy ile ma odczytac znakow}
   RX:=int(sqrt(X));
   RozX:=round(RX);
   RY:=int(X/RozX);
   RozY:=round(RY);
   Reszta:=X-(RozX*RozY)+1+RozY*2;
   if reszta>=2*rozy+2 then reszta:=reszta-2*rozy;
end;

Function OkresRozmiarPlikuTXT(PlikX:string):longint;
var f:file of byte;
Begin
 assign(f,PlikX);
 reset(f);
 OkresRozmiarPlikuTXT:=filesize(f);
 close(f);
end;

Procedure OtworzPliki;
Begin
  assign(p,plik1);
  assign(p2,plik2);
  assign(p3,'kodujbmp.dat');
  reset(p);
  rewrite(p2);
  reset(p3);
end;

Procedure KopiujNaglowek;
var a:longint;
    d:byte;
    S:longint;
    X:word;
Begin
  for a:=1 to 6 do read(p3,d);
  d:=66; write(p2,d); {B}
  d:=77; write(p2,d); {M}

  {Wielkosc pliku wpisz}
  S:=reszta+rozmiartxt+1079+1;
```

```
  d:=S mod 256;
  write(p2,d);
  d:=S div 256;
  write(p2,d);
  X:=S div 65536;
  d:=X mod 256;
  write(p2,d);
  d:=X div 256;
  write(p2,d);
  for a:=1 to 12 do
  begin
    read(p3,d);
    write(p2,d);
  end;
  {roz X}
  d:=lo(rozx);
  write(p2,d);
  d:=hi(rozx);
  write(p2,d);
  d:=0;
  write(p2,d,d);
  {roz Y}
  d:=lo(rozy);
  write(p2,d);
  d:=hi(rozy);
  write(p2,d);

  for a:=1 to 6 do read(p3,d);
  for a:=1 to 1054 do begin
    read(p3,d);
    write(p2,d);
  end;
  close(p3);
end;

Procedure KopiujPlikTXT;
var d:byte;
    i:longint;
Begin
     d:=lo(Reszta);
     write(p2,d);
     d:=hi(Reszta);
     write(p2,d);

     for i:=1 to RozmiarTXT do Begin {zapisz zawartosc pliku}
        read(p,d);
        write(p2,d);
     end;
     close(p);

     for i:=1 to Reszta do Begin {zapisz losowe smiecie}
       d:=random(256);
       write(p2,d);
     end;
     close(p2);
end;
```

```
BEGIN
  randomize;
  NazwyPlikow;
  RozmiarTXT:=OkresRozmiarPlikuTXT('dane.txt');
  JakiRozmiar(RozmiarTXT);
  OtworzPliki;
  KopiujNaglowek;
  KopiujPlikTXT;
END.
```

Program jest dosyć ciekawy, ale jeśli otworzymy plik *BMP* w Notatniku, zobaczymy zawartość wklejonego do niego pliku tekstowego. Ten program nadaje się jedynie do doklejania do bitmap plików binarnych, np. plików z rozszerzeniem *EXE*. Jednak nic nie stoi na przeszkodzie, aby dodać do programu malutki algorytm szyfrujący zapisywane dane. Jeśli masz zamiar ulepszać ten program, to warto zmienić procedurę czytania i zapisywania danych do pliku, gdyż czytanie po 1 bajcie jest dość wolne, szczególnie, jeśli program odczytuje duże pliki. Należy tak zmienić program, aby odczytywał naraz 1 000 znaków i tyle samo zapisywał. Wtedy program będzie dużo szybciej wykonywał swoje zadanie. Nawet jeśli nie chcesz ukrywać plików za pomocą tego programu, uruchom go, gdyż czasami potrafi wygenerować naprawdę ciekawe pliki *BMP*. Możesz dodatkowo zwiększyć bezpieczeństwo zakodowanego pliku, dodając ten plik jako nowy zasób do dowolnego pliku *EXE* za pomocą programu Resource Hacker. Aby to zrobić, należy:

1. Uruchomić program Resource Hacker.
2. Otworzyć w tym programie dowolny plik wykonywalny.
3. Z menu *Action* wybrać polecenie *Add a new resource*.
4. Odnaleźć na dysku nasz plik *BMP* zawierający zakodowany plik.
5. Podać jakąś nazwę zasobu np. *Tajne1*.
6. Kliknąć przycisk *Add resource*.
7. Zapisać zmiany dokonane w pliku.

Listing 9.3 prezentuje źródło programu, który z pliku BMP „wyciąga" wcześniej zakodowany plik.

Listing 9.3. *odkodujb.pas — program, który z plików BMP „wyciąga" plik zakodowany programem kodujbmp.pas*

```
Program OdkodujBMP;
Uses crt;
var P,P2:file of byte;
    Plik1,Plik2:string;
    Reszta:word;
    D:byte;
    a:word;
    MB,SB:byte;
    RozmiarBMP:longint;
    RozmiarTXT:longint;
```

```
Procedure NazwyPlikow;
Begin
 clrscr;
 write('odKodujBMP v 1.0                                    (c) Damian Daszkiewicz');
 writeln('Program wyciaga z plikow BMP pliki zakodowane programem KODUJBMP');
 writeln('UWAGA program nie odroznia zwyklych plikow BMP od plikow BMP stworzonych');
 writeln('przez program KODUJBMP! Przy zwyklych plikach BMP program moze sie wieszac!');
 writeln('Program zostal napisany na potrzeby ksiazki Edycja Plikow Binarnych dla');
 writeln('wydawnictwa Helion. Nie zastrzezono do niego praw autorskich');
 writeln;

 Plik1:=ParamStr(1);
 if plik1='' then Begin
    Repeat
        write('Podaj nazwe pliku, ktory chcesz odkodowac >>');
        readln(Plik1);
    until Plik1<>''
 end;

 Plik2:=ParamStr(2);
 if Plik2='' then Begin
   Repeat
     write('Jaka nadac nazwe odkodowanemu plikowi? >>');
     readln(Plik2);
   until Plik2<>''
 end;
end;

Function OkresRozmiarPliku(PlikX:string):longint;
var f:file of byte;
Begin
 assign(f,PlikX);
 reset(f);
 OkresRozmiarPliku:=filesize(f);
 close(f);
end;

Procedure OtworzPliki;
Begin
  assign(p,plik1);
  assign(p2,plik2);
  reset(p);
  rewrite(p2);
end;

Procedure OdczytajNaglowek;
Begin
  for a:=1 to 1078 do read(p,d);
  Read(p,mb);
  read(p,sb);
  Reszta:=mb+256*sb;
  RozmiarTXT:=RozmiarBMP-(1078+2+Reszta);
end;

Procedure WyciagajDane;
var i:longint;
```

```
Begin
   for i:=1 to RozmiarTXT do Begin
       read(p,d);
       write(p2,d);
   end;
end;

Procedure ZamknijPliki;
Begin
  close(p);
  close(p2);
end;

BEGIN
  NazwyPlikow;
  RozmiarBMP:=OkresRozmiarPliku(Plik1);
  OtworzPliki;
  OdczytajNaglowek;
  WyciagajDane;
  ZamknijPliki;
END.
```

Program ma jedną wadę: jeśli próbujemy nim „wyciągnąć" dane z pliku *BMP*, który nie został utworzony poprzednim programem, to albo zostaną odczytane jakieś „śmiecie", albo dojdzie do zatrzymania pracy programu. Aby się przed tym zabezpieczyć, należy tak przerobić program, aby „znakował" pliki *BMP* zawierające dane — np. oprócz nagłówka informującego, ile następnych bajtów w pliku *BMP* to zawartość innego pliku, umieszczona byłaby np. 4-znakowa sygnatura. Program dekodujący po natrafieniu na plik nie posiadający tej sygnatury informowałby, że jest to zwykły plik *BMP*, który nie zawiera w sobie cennych danych.

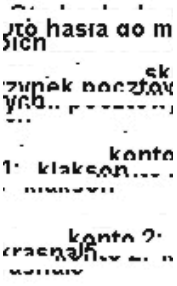
Ukrywanie tajnych informacji w innych plikach graficznych

Pliki *BMP* mają bardzo prostą budowę, dlatego bardzo dobrze nadają się do ukrywania tajnych informacji. Z innymi plikami graficznymi jest już gorzej, ale postanowiłem omówić kilka innych typów plików graficznych, aby docenić prostotę plików *BMP*.

JPG

Pliki *JPG* to skompresowane pliki bitmapowe. Jest to dość popularny typ plików graficznych, gdyż można za jego pomocą kompresować bitmapy, stosując różne współczynniki kompresji (im większy współczynnik, tym więcej miejsca na dysku zajmuje plik, ale zarazem jego jakość jest lepsza). Pliki *JPG* bardzo dobrze nadają się do kompresji różnych zdjęć. Niestety sztuczka polegająca na ingerencji w nagłówek nie jest dobra, gdyż w zmodyfikowanym pliku widoczne są fragmenty wyrazów z oryginalnego pliku (patrz rysunek 9.4).

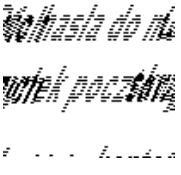
Rysunek 9.4.
Plik JPG ze zmodyfikowanym nagłówkiem



GIF

Pliki *GIF*, podobnie jaki pliki *JPG*, to skompresowane mapy bitowe. Jednak pliki *GIF* mają inny algorytm kompresji i bardziej nadają się do kompresowania bitmap, które zostały stworzone w jakimś programie graficznym. Niezbyt dobrze natomiast nadają się do kompresji zdjęć, gdyż maksymalnie mogą zawierać 256 kolorów, a minimalnie 16. Pliki *GIF* również nie nadają się do przechowywania tajnych informacji, gdyż — podobnie jak w plikach *JPG* — widoczne są napisy z oryginalnego pliku, o czym świadczy rysunek 9.5.

Rysunek 9.5.
Plik GIF ze zmodyfikowanym nagłówkiem



PCX

Pliki *PCX* to skompresowane pliki bitmapowe. Nie są tak skuteczne jak formaty *GIF* i *JPG*, ale za to mają bardzo prosty algorytm kompresji. Niestety, obecnie te pliki są już bardzo rzadko używane; zapis bitmapy do tego formatu był jeszcze możliwy w programie *Paint* w wersji dla Windows 3.11, ale już kolejne wersje tego programu dla systemów Windows 9x/Me/XP nie obsługiwały formatu *PCX*. Niestety sztuczka ze zmodyfikowaniem nagłówka również nie jest dobrym pomysłem, o czym świadczy rysunek 9.6.

Rysunek 9.6.
Plik PCX ze zmodyfikowanym nagłówkiem

TIFF

Jest to format grafiki bitmapowej. Pliki TIFF po części nadają się do wykorzystania ich w sztuczce z ingerencją w nagłówek, gdyż chyba wszystkie programy graficzne taki plik uznają za uszkodzony.

WMF

Format *WMF* jest dla grafiki wektorowej takim standardem, jakim dla grafiki rastrowej jest format *BMP*. Mimo że różne programy do tworzenia grafiki wektorowej zapisują grafiki we własnym formacie, to większość programów pozwala na zapisanie grafiki do pliku *WMF*. Z naszego punktu widzenia pliki *WMF* są mało użyteczne, gdyż po dokonaniu ingerencji w nagłówek plik i tak wyświetla oryginalne napisy, o czym świadczy rysunek 9.7.

Rysunek 9.7.
Plik WMF ze zmodyfikowanym nagłówkiem

Jak w plikach EXE ukryć tajne informacje?

W plikach *EXE* nie ma zbyt dużo miejsca, aby móc ukryć wiele tajnych informacji. Ale szczególnie w plikach *EXE* przeznaczonych dla systemu operacyjnego Windows znajdzie się kilkanaście lub kilkadziesiąt bajtów, które można swobodnie zastąpić naszymi informacjami. Niestety w różnych plikach *EXE* bajty te znajdują się pod różnymi offsetami. Pojawia się pytanie: jak można rozpoznać, czy dany fragment pliku *EXE* można zastąpić własnymi danymi? Odpowiedź jest prosta: należy zlokalizować w pliku *EXE* miejsce, w którym znajdują się same zera (tzn. kody ASCII 0). Takie miejsce istnieje na samym początku pliku, gdzie znajduje się kilka-kilkanaście zer. Te pola można śmiało zastąpić własnymi wartościami i sprawdzić, czy program się uruchomi. Jeśli się nie uruchomi, oznacza to, że te dane są jednak ważne i należy przywrócić plik z kopii zapasowej. Jeśli nam się poszczęści, można znaleźć całkiem spory fragment pliku *EXE*, w którym da się zmieścić nawet 1,5 KB danych! Przykładem takiego pliku *EXE* jest program *txt2ascii*, który znajduje się na płycie CD-ROM w katalogu *aplikacje/tools*. Pod offsetem 244h znajduje się 1,5 KB wolnego miejsca. Dotyczy to również wszystkich plików *EXE*, które były kompilowane w Microsoft Visual Basic 6.0. Niestety nie wszystkie pliki wykonywalne posiadają w swym wnętrzu aż tak dużo miejsca, które można zagospodarować. Ale nic nie stoi na przeszkodzie, aby do pliku *EXE* coś dopisać. Bez większego problemu na końcu pliku *EXE* dodałem plik tekstowy o rozmiarze 15 KB,

a program nadal działał poprawnie. Dotyczy to również plików *COM*. Jednak przed dodaniem na końcu pliku *EXE* pewnych informacji warto jest wykonać kopię takiego pliku. Aby do pliku *EXE* dopisać dane, należy wykonać następujące czynności:

1. Otworzyć plik *EXE* w programie Hex Workshop.
2. Wykonać kopię pliku. Najlepiej z menu *File* wybrać opcję *Save As...* i zapisać plik pod inną nazwą.
3. Otworzyć w Notatniku plik tekstowy.
4. Zaznaczyć cały tekst i skopiować go do schowka.
5. W programie Hex Workshop należy umieścić kursor na końcu pliku.
6. Z menu *Edit* wybrać pozycję *Paste Special*.
7. Pojawi się okno dialogowe, w którym można wybrać format, w jakim ma być wklejony tekst. Wybierz pozycję *CF_TEXT*. Upewnij się, czy „ptaszek" przy polu wyboru z napisem *Interpret as a hexadecimal string* **nie jest zaznaczony**.

Można również do pliku *EXE* dopisać zawartość innego pliku binarnego. Aby tego dokonać, należy wykonać następujące czynności:

1. W programie Hex Workshop otworzyć plik *EXE* oraz plik, który mamy zamiar wkleić do pliku *EXE*.
2. Wykonać kopię pliku. Najlepiej z menu *File* wybrać opcję *Save As...* i zapisać plik pod inną nazwą.
3. Zaznaczyć cały plik (podczas tej operacji zaznaczyć tę część, w której umieszczone są heksadecymalne cyfry).
4. Umieścić kursor na końcu pliku *EXE*.
5. Z menu *Edit* wybrać pozycję *Paste Special*.

Pojawi się okno dialogowe, w którym można wybrać format, w jakim ma być wklejony tekst. Wybierz pozycję *CF_TEXT*. Upewnij się, czy „ptaszek" przy polu wyboru z napisem *Interpret as a hexadecimal string* jest **zaznaczony**.

Być może wklejenie pliku binarnego nie jest trudne, ale trudniej jest potem „odkleić" od pliku *EXE* plik binarny. Dlatego napisałem malutki program, który wykona to za nas (patrz listing 9.4). Należy tylko podać nazwę pliku, z którego należy „odkleić" binarny fragment, nazwę nowego pliku oraz długość naszego nowego pliku binarnego (czyli ile ostatnich bajtów należy skopiować do nowego pliku). Programik napisałem w FPC, więc można go swobodnie uruchomić zarówno w systemie MS-DOS (Windows), jak i w systemie Linux.

Listing 9.4. *odklej.pas — program zapisujący do innego pliku ostatnie x bajtów dowolnego pliku*

```
Program Odklej;
uses crt;

var
 P1,P2:file of byte;
```

```
 WielkoscPliku,IleSkopiowac:longint;

Procedure OtworzPliki(Plik1,Plik2:string); forward;

Procedure Pytaj;
var
    Plik1,Plik2:string;
    kod:integer;
Begin
  clrscr;
  writeln('Odklej v 1.0 (c) Damian Daszkiewicz');
  writeln('Program do osobnego pliku kopiuje x ostatnich bajtow innego pliku');
  writeln;

  Plik1:=Paramstr(1);
  if Plik1='' then begin
    repeat
     write('Podaj nazwe pliku wejsciowego >> ');
     readln(Plik1);
    until plik1<>''
  end;

  Plik2:=Paramstr(2);
  if Plik2='' then Begin
    repeat
     write('Podaj nazwe pliku wyjsciowego >> ');
     readln(Plik2);
    until plik2<>'';
  end;

  val(paramstr(3),IleSkopiowac,kod);
  if IleSkopiowac=0 then Begin
     repeat
      write('Podaj ile ostatnich bajtow mam skopiowac >>');
      readln(IleSkopiowac);
     until IleSkopiowac<>0
  end;
  OtworzPliki(Plik1,Plik2);
end;

Procedure OtworzPliki(Plik1,Plik2:string);
var I:longint;
    D:byte;
Begin
  assign(p1,plik1);
  reset(p1);
  assign(p2,Plik2);
  Rewrite(P2);
  WielkoscPliku:=(filesize(p1));
  for i:=1 to WielkoscPliku-IleSkopiowac do Read(P1,D);

  for i:=1 to IleSkopiowac do Begin
    Read(P1,D);
    Write(P2,D);
  end;
```

```
  close(p1);
  close(p2);
end;

Begin
  Pytaj;
end.
```

Celowe uszkadzanie plików

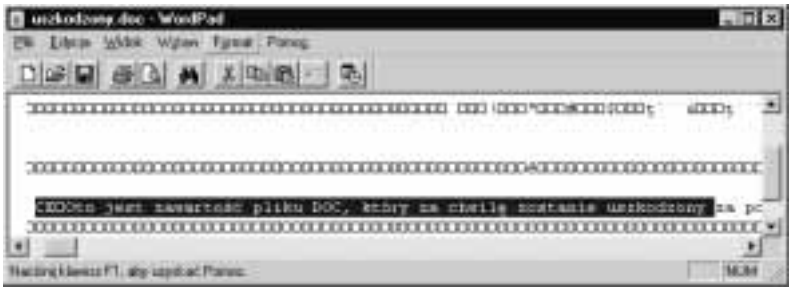
Dość prostym sposobem na ukrycie informacji jest uszkodzenie pliku. Być może brzmi to dziwnie, ale jest to bardzo proste i skuteczne rozwiązanie, gdyż większość programów nie będzie w stanie otworzyć uszkodzonych plików. Aby otworzyć uszkodzony plik, należy go naprawić. Poniżej opiszę kilka typów plików, które można uszkodzić i przedstawię metody ich uszkadzania.

Pliki DOC

Jest to dość popularny typ plików. Pliki te tworzone są przez program *Microsoft Word*, który jest najpopularniejszym edytorem tekstu w systemie Windows. Pliki *DOC* otwierane są przez kilka innych programów, np. *StarOffice* lub *OpenOffice* dla systemów Linux i Windows czy choćby zwykły *WordPad*, dołączony do systemu operacyjnego Windows. Pliki *DOC* oprócz tekstu mogą zawierać również rysunki i wykresy.

Aby uszkodzić plik *DOC*, nie trzeba się wcale natrudzić. Wystarczy otworzyć taki plik w programie Hex Workshop (lub dowolnym innym edytorze plików binarnych) i zmienić pierwszy bajt. Pierwszy bajt powinien mieć wartość `D0`, ale gdy zmienimy tę wartość na inną, program albo poinformuje, że plik jest uszkodzony, albo potraktuje plik jako plik tekstowy i wyświetli nieczytelne ciągi znaków („krzaczki"). Aby to naprawić, należy plik otworzyć w programie Hex Workshop i pierwszy bajt z powrotem zmienić na `D0`.

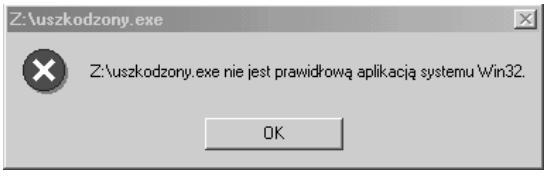
Pliki *DOC* nie nadają się do ukrywania danych, gdyż tekst w plikach *DOC* nie jest kodowany. Mimo że na ekranie pojawią się „krzaczki", między nimi będzie widoczny zwykły tekst (patrz rysunek 9.8).



Rysunek 9.8. *Uszkodzony plik DOC*

Pliki EXE

Pliki *EXE* są plikami wykonywalnymi. Uszkodzenie plików *EXE* nie jest trudne. Pliki *EXE* posiadają własny nagłówek, który zaczyna się od znaków `MZ` (lub `ZM`). Wystarczy zmienić te 2 znaki na jakieś inne, np. XX, i plik *EXE* będzie uznawany za uszkodzony. Na rysunku 9.9 przedstawiono komunikat, jaki pojawi się przy próbie uruchomienia uszkodzonego pliku *EXE*.



Rysunek 9.9. *Komunikat pojawiający się podczas próby uruchomienia uszkodzonego pliku EXE*

Przed uszkodzeniem pliku *EXE* można do niego dodać jakąś bitmapę, zawierającą np. hasła do kont pocztowych. Po uszkodzeniu pliku edytory zasobów nie otworzą go, gdyż uznają go za uszkodzony.

Pliki ZIP

Pliki *ZIP* są to pliki popularnego programu pakującego *WinZip*, służącego do pakowania innych plików, czyli zmniejszania ich objętości za pomocą skomplikowanych algorytmów. Plik *ZIP* może zawierać jeden skompresowany plik lub więcej plików. Pliki *ZIP* bardzo dobrze nadają się do ukrywania tajnych informacji, gdyż można spakować kilka ważnych plików do jednego pliku *ZIP*, a potem uszkodzić tylko jeden plik *ZIP* — nie trzeba po kolei uszkadzać wielu plików. Pliki *ZIP*, podobnie jak pliki poprzednio omówione, posiadają nagłówek. Dwa pierwsze bajty pliku PK informują, iż jest to plik *ZIP*. Po zmianie tych bajtów na dowolne inne plik automatycznie zostaje uszkodzony. Niektóre programy takowy plik otworzą i nawet wyświetlą pliki, jakie zostały skompresowane, ale już ich nie rozpakują; z kolei inne programy w ogóle nawet nie otworzą takiego pliku.

Inne pliki

Nie sposób tutaj omówić wszystkich możliwych plików, które można uszkodzić. Jednak chyba zauważyłeś pewien uniwersalny wzorzec: zwykle wystarczy zamienić 2 pierwsze bajty na inne, aby plik nie mógł być otworzony przez program, w którym został utworzony. Zachęcam do dalszego eksperymentowania!