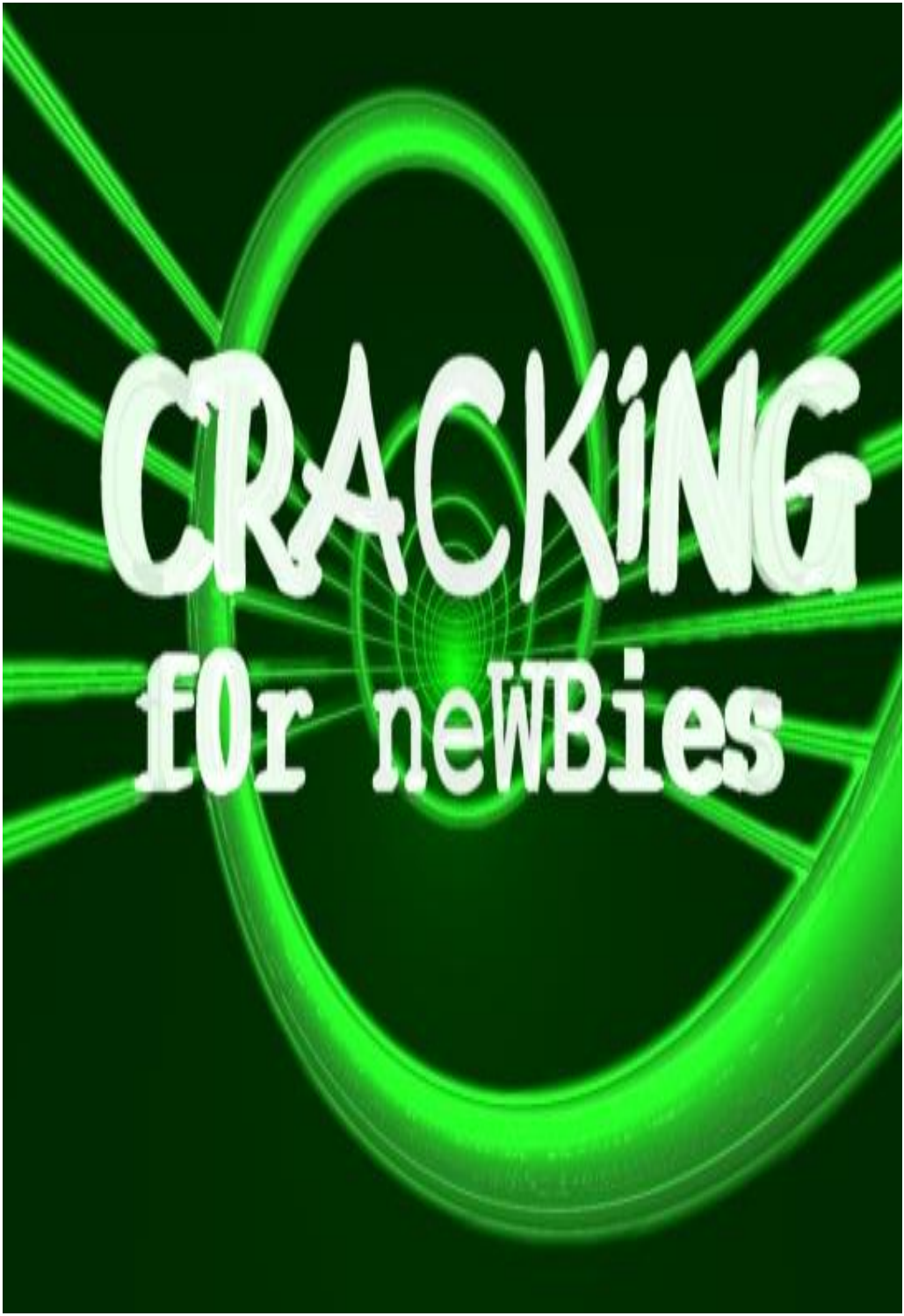

CRACKiNG
fOr neWBies

OBSAH

Úvod

Jelikož na českém internetu je toho o Reverse Engineeringu málo, napadlo mě udělat miniknížku ve formátu A5 o crackingu. Knížka je psaná v Microsoft Wordu 2oo7, ve formátu A5, a soubor má formát buď .doc nebo .pdf. Knížka je určená ke čtení na PC, ale samozřejmě si ji můžete i vytisknout. Vejdou se vám 2 listy na 1 list papíru A4, proto je psaná ve formátu A5, aby byla menší.

Tolik k teorii. Jak již jsem uvedl, na českých stránkách je hodně málo stránek o Reverse Engineeringu (dále jen RE, Rev. Eng. = Cracking), nejlepší ze stránek jsou asi picaso.poupe.net a www.t4c.ic.cz. Stránky už nejsou bohužel aktualizovány a na českých stránkách jsem vyloženě dobrou stránku o crackingu nenašel. Proto se snažím RE uvést znovu do provozu.

Dnešní aplikace, programy a hry jsou velmi kvalitně chráněny, ale „vždycky všechno jde". Podle mého názoru je „lepší" si zvolit cestu crackingu, než hackingu, protože u crackingu se jedná o kód, který je na našem počítači, a jde vždycky upravit, zatímco u hackingu „nikdy nevíte"…

Do RE se nemůžete pouštět jen tak. Musíte být aspoň průměrný programátor v Assembleru, a porozumět jednotlivým instrukcím. Předem říkám, že jsem začátečník, a proto promiňte některé nepřesné výrazy. Potom musíte mít určitou softwarovou výbavu. Ze základních patří tyto programy:

Olly Debugger (OllyDbg) – Toto je vynikající freewarový debugger, zobrazí vám jednotlivé instrukce v assembleru, můžete se podívat do paměti RAM, měnit flagy a měnit procesorové registry. Má komfortnější ovládání než SoftICE (viz níže) a proto pracuji právě v OllyDbg.

SoftIce – Debugger od společnosti Numega. Crackeři uvádějí, že je to jejich nejoblíbenější firma. Tento debugger je placený a pracuje v DOSovském prostředí. Má sice více možností než Olly, ale mě Olly vyhovuje více. Popravdě řečeno jsem tenhle debugger nainstaloval, jednou použil, a potom hned smazal…

DeDe – Delphi Decompiler. Dekompiluje vám aplikaci vytvořenou v Delphi, můžete se dívat na jednotlivé Formy, a procedury. *.pas soubor vám sice nevykouzlí, místo toho se vám objeví asm instrukce i s adresami.

IDA – Interactive DisAssembler, je to nejlepší DisAssembler (dále jen Dasm) vůbec. Je ale placený a složitý na ovládání. Ale na prohlídnutí asm kódu postačí. Navíc ukáže graf posloupnosti programu. Já osobně IDu nepoužívám, moc mi nevyhovuje, ale jednou za čas si v tomto Dasmu disassemblluju program.

W32Dasm – Výborný freewarový Dasm. Používám ho raději než IDu a umí jednu skvělou věc co IDA neumí, a to je vyhledávání stringů v jednotlivých částech kódu…

PEiD – Freewarový program, který zjistí, čím je program zkompilován, popřípadě zapakován nebo kryptován. Při Generic unpacker a UPX unpacker plug-inu, dokáže program i rozpakovat.

eXeScope – Tento program snad používám jenom já. Zobrazuje hlavičku exe souboru, importované a exportované funkce a resources. Popravdě řečeno, tímhle programem začínám vždycky crackovat.

PEditor – Editor hlaviček v exe souboru. Většinou zde měním characteristics sections. Při špatně nastavené sekci, nejde program načíst třeba do W32Dasmu. Nebo při kompresi UPX a vymazaných jménech sections, je zde doplním. NEŽ JAKÝKOLIV PROGRAM BUDETE CHTÍT UPRAVIT, ZÁLOHUJTE HO!!!

UPX – Ultimate Packer for eXecutables. Program zkomprimuje exe soubor (exe, dll, ocx a další PE soubory…). Program umí i dekomprimovat. Po zkomprimování a vymazání jmen sekcí a upravených characteristics, už nejde program normálně unpacknout pomocí UPX. Musí se sekce zpět přepsat, dumpnout, nebo genericky unpacknout.

ProcDump – Program programovaný v MASM assembleru. Dokáže dumpnout program z paměti RAM. Předem upozorňuji, že program takto dumpnutý většinou nefunguje, hlasí různé chyby a kód nedává smysl. Program dále umí rozpackovat exe pomocí asi 40ti unpackerů. Má v sobě i integrovaný editor hlaviček.

To je z nejdůležitějších programů asi vše. Ještě existuje program Hiew ale ten já nepoužívám, není proč.

Jak již jsem napsal, musíte umět aspoň základy assembleru. Nemusíte v něm nutně umět programovat (to neumím ani já, tedy…jen základy, nic moc extra), ale musíte rozumět jednotlivým instrukcím. Za chvíli je vypíšu…

K zahození nebude ani znalost jazyka C++, nebo Delphi (Pascal). V céčku si naprogramujeme patchy a cracky, a v Delphi si zkusíme naprogramovat jednoduché CrackMe (dále jen CMe). CrackMe je vlastně program, který je speciálně určený pro cracknutí.

Zde je slibovaný popis instrukcí:

CMP EAX, EBX – porovná registr EAX s EBX (samozřejmě může i jiné), pokud se sobě rovnají, nastaví zero flag na jedničku

JMP 004010F5 – Jump, skok na adresu (v tomto případě: 004010F5)

JE 004010F5 – Jestli je Zero-Flag (dále jen ZF) nastaven na 1, skoč…jestli je ZF=0, tak pokračuj…

JNZ 004010F5 – Úplný opak toho předchozího. Tedy: Jestli je ZF nastaven na 0, skoč…jestli je ZF=1, tak pokračuj…

CALL – volá instrukci, kde se něco počítá, je to něco jako „goto", můžete tuto instrukci v debuggeru přeskočit „F8" nebo „jít do callu" (trace into), tlačítkem „F7"

NOP – NO OPeration, žádná operace…tento příkaz je zkrátka zbytečný, ale i přes to je to jeden z nejdůležitějších příkazů

PUSH EAX – Uloží eax do stacku (zásobníku)

POP EAX – Pushnutý registr obnoví ze stacku (zásobníku)

RETN – pokud vstoupíte do CALLu, tak RETN (return) vrací zpět program na adresu, odkud call pocházel. Jednoduše je to ukončení CALLu.

XOR – početní operace, zapněte si kalkulačku ve Windows, nastavte ji na „Vědecký" mód, a operaci XOR tam taky máte. Kalkulačka s Poznámkovým blokem jsou taky vlastně aplikace, které budete s crackingem potřebovat…

V balíku „used_tools.rar" najdete většinu z popsaných programů na cracking

---

Lekce č.1

Cracknutí CrackMe

Zde vám popíšu nejjednodušší příklad. Zde je popsáno, co potřebujete:

Crackme.exe // CMe ze stránek Programujte.com
OllyDbg // Debugger
Znalosti C++ // na výrobu patchu… není nutné…

Nějaký c++ kompilátor popř. editor…Nejlepší bude Microsoft Visual C++ 2005 Express Edition, ale „jen" s Dev-Cpp to taky půjde…

Spustíme si CMe, vidíme EDITBOX a BUTTON na zaregistrování…

Bezva… Předem říkám, že CMe nemá žádnou proticrackerskou ochranu ani není zapakovaný, takže si ho rovnou můžeme otevřít v Olly…

Kód tohoto programu si můžeme vyloženě prohlédnout celý, protože CMe je opravdu malé…

Při projíždění kódem jste nemohli přehlídnout na adrese 004011AC string „Špatné sériové číslo!"… Jdeme nahoru po stopách kódu…

Tenhle kousek je zajímavý:

```
……
00401167    .  33C0          XOR     EAX, EAX                //EAX=0
00401169    .  33C9          XOR     ECX, ECX                //ECX=0
0040116B    .  A1 D3304000   MOV     EAX, DWORD PTR DS:[4030D3]   //do EAX
dej to, co je v EDITu
00401170    .  B9 5A415244   MOV     ECX, 4452415A   //ECX=4452415Ah
00401175    .  33C1          XOR     EAX, ECX                //Vyxoruj EAX
s ECX
00401177    .  33D2          XOR     EDX, EDX                //vynuluj EDX
00401179    .  BB 77770100   MOV     EBX, 17777              //EBX=17777h
0040117E    .  F7F3          DIV     EBX                     //vyděl RBX
00401180    .  35 86140000   XOR     EAX, 1486               //Vyxoruj EAX
s 1486h
00401185    . /75 1E         JNZ     SHORT crackme_.004011A5   //Jestli
není s/n správný skoč…**
00401187    . |6A 00         PUSH    0
00401189    . |68 CC304000   PUSH    crackme_.004030CC
0040118E    . |68 6A304000   PUSH    crackme_.0040306A……
```

**:Toto je naše důležitá instrukce…Zkušený cracker by podle instrukcí od adresy 401167 do 401180 vypočítal správné s/n…ale my jsme začátečníci a budeme rádi aspoň za něco…

Na adrese 401185 máme instrukci JNZ…ale my chceme, aby to skočilo, když zadáme i špatné s/n… Zkusíme instrukci změnit na JZ…poklepeme dvakrát na instrukci a přepište „JNZ SHORT 004011A5" na „JZ SHORT 004011A5" a klikněte na Assemble…

Spusťte program (F9) a zadejte nějaký blábol…a hle! Správné reg.č.! Chyba je v tom, že když to s/n zadáte dobré, tak se vám ukáže, že je špatné…tomu byste zamezili tím, že byste instrukci „vyNOPovali", TZN že byste místo instrukce „JNZ SHORT 004011A5" napsali jen „NOP" a klikli na assemble…

Mohli jste si všimnout tohoto:

```
00401185   . /75 1E        JNZ     SHORT crackme_.......
```

A po „zásahu" bylo :

```
00401185   . /74 1E        JZ      SHORT crackme_.......
```

Všimli jste si? 75h se proměnilo na 74h…

75 je HEX instrukce JNZ a

74 je HEX instrukce JZ…chápete?!

Zkusíme si napsat crack. Nejdříve si řekneme, co chceme, aby crack dělal. Chceme, aby instrukci 75h přepsal na 74h (jak již bylo psáno, správné č. se označí za špatné…).

Nejdříve si najdeme, na kterém hex. offsetu leží instrukce JNZ. To co v Olly vidíme vlevo je adresa, my chceme najít offset, ten najdeme tak, že si označíme instrukci JNZ a klikneme na ní pravým myšítkem, a klikneme na View->Executable File. Objeví se nám v HexEditoru všechny instrukce i s DisAssemblingem. A nalevo už není adresa, ale offset. Tam kde leží JNZ, by měl být offset 00000585, a měl by být označený. Už víme který to je offset a už víme, jak to chceme přepsat. To nám k výrobě cracku stačí.

Vytvoříme si patch v Céčku…základ je toto:

```
#include "stdafx.h"
#include <stdio.h>
#include <stdlib.h>
#include "Form1.h"
{
FILE *f;


int adr=0x585;      //offset
int owr=0x74;       //hex.74 = „JE“ (patch z 75 („JNZ“))

f = fopen( "crackme.exe" , "rb+");
if (f==NULL) { MessageBox::Show( "File can not found!",
"Error", MessageBoxButtons::OK,
MessageBoxIcon::Exclamation );
exit(1);
}


fseek(f,adr,SEEK_SET);


fwrite(&owr,1,1,f);
MessageBox::Show( "Cracked :-)", "Cracked...:-)"
,MessageBoxButtons::OK, MessageBoxIcon::Exclamation );
fclose(f);
}
```

Pokud si s kódem pohrajete ve Visual C++ můžete dospět k nádhernému výsledku……(CrackMe, projekt a source k Visual C++ je i v tomto balíčku)

V balíku „balik1.rar" najdete:

- **CrakMe.exe – z Programujte.com**
- **Patch v MSVC++2oo5 [DotNET] (exe+source)**

Lekce č.2

Serial fishing v Crackme

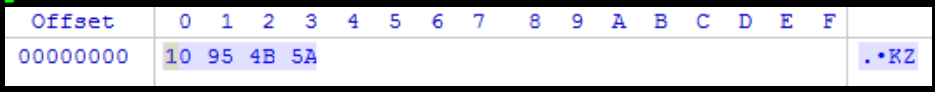
Zůstaneme u stejného CMe, ale nyní zkusíme vyluštit s/n. Natáhneme si tedy ORiGiNÁLNí soubor CrackMe.exe do Olly a nastavíme BreakPoint (dále jen BP) na adresu 0d0401167. Zde totiž začíná výpočet.

```
00401167   .  33C0          XOR     EAX, EAX                //EAX=0
00401169   .  33C9          XOR     ECX, ECX                //ECX=0
0040116B   .  A1 D3304000   MOV     EAX, DWORD PTR DS:[4030D3]   //do EAX
dej to, co je v EDITu
00401170   .  B9 5A415244   MOV     ECX, 4452415A   //ECX=4452415Ah
00401175   .  33C1          XOR     EAX, ECX                //Vyxoruj EAX
s ECX
00401177   .  33D2          XOR     EDX, EDX                //vynuluj EDX
00401179   .  BB 77770100   MOV     EBX, 17777              //EBX=17777h
0040117E   .  F7F3          DIV     EBX                     //vyděl RBX
00401180   .  35 86140000   XOR     EAX, 1486               //Vyxoruj EAX
s 1486h
00401185   . /75 1E         JNZ     SHORT crackme_.004011A5   //Jestli
není s/n správný skoč…
00401187   . |6A 00         PUSH    0
00401189   . |68 CC304000   PUSH    crackme_.004030CC
0040118E   . |68 6A304000   PUSH    crackme_.0040306A……
```

Cílem instrukce XOR EAX, 1486 je, aby v EAX bylo také 1486h, a pokud se XORují dvě stejné hodnoty, tak se nastaví ZF, protože v EAX bude 0. Takže celý fór je v početní operaci, kterou by měl zvládnout žák 8. Třídy(já =)).

Spustíme si tedy kalkulačku a přepneme se do Hex. Zadáme 1486 a vynásobíme číslem 17777 (vše v hex). Teď dáme XOR a číslo 4452415A a rovná se. Vyšlo nám 5A4B9510. Což je naše správné s/n. Číslo ale má být string, a tak tedy si spustíme nějaký hexeditor. Počítače x86 jsou Little-Endian, hex.č. se musí číst

pozadu:



Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
00000000	10	95	4B	5A													.•KZ

+•KZ - zajímavý výsledek… ale je správný. Podle návodu na Programujte.com se píše toto:

Takže pokud vyxoruji to, co mi vzniklo po XORu v EAX s ECX, tak získám původní EAX, tedy správné registrační číslo. Výsledkem této operace je hodnota 5A494C42h, ale heslem musí být nějaký řetězec, a proto převedeme toto číslo na znaky ASCII.

Poněvadž počítače x86 jsou little-endian, tak musíme číst text odzadu. Tím řetězcem je slovo „BLIZ". Teď jděte do nabídky „Debug" a zvolte volbu „Restart", nastavte breakpoint na adresu 00401167 a spusťte program (F9). Napište správný klíč a krokujte program klávesou F7 (pokud budete chtít přeskakovat instrukce CALL, tak potom krokujte klávesou F8). Koukejte se přitom na výpočet a kód v debuggeru a uvidíte, že to vše do sebe zapadá a nakonec se vám zobrazí věta „Registrace byla úspěšně dokončena".

5A494C42h - zajímalo by mě, jak k tomu mám dojít, když nevím jaký výsledek má být v EDX (zbytek po DIV EBX)…zde je pár komentářů přímo ze stránky:

1. **Pri konzulácii tohto článku ma kamarát Master upozornil i na ďalšie riešenie. A to 0x5A4B9510, až na tie prvé dva znaky... :) //což na to jsme došli taky…**

2. OK, takže úprava... možností je ďaleko viacej a to kvôli deleniu, resp. zvyšku po delení, ktorý sa zahadzuje.

Interval je <1E19D44A, 1E1B4BC0> a ľubovolné číslo z tohto intervalu stačí xornúť hodnotou 4452415A a dostávame riešenie. Otázkou je ASCII reprezentácia znakov, čiže kvôli ľahšiemu zadávaniu treba filtrovať tlačiteľné znaky. //Dlouho jsem přemýšlel, jak k tomu došel, ale už mě to napadlo:
1E19D44A – toto je vlastně 1486h * 17777h…to co jsme vlastně taky vypočítali…
1E1B4BC0 – toto je 1E19D44A + 17776h, takový může být vlastně maximální zbytek v EDX, který s hodnotou 1486h v EAX nemá nic společného. Tudíž autor s CMe trochu ujel…a zajímalo by mě, kde to číslo vzal. Když ten výsledek +•KZ dám do schránky (Ctrl+C) a pak vložím do Editu v CMe, vypíše nám to hlášku o úspěchu. Když napíšeme BLIZ taky to vypíše úspěch, protože zbytek v EDX se nepočítá, ale i přesto je v EAX 1486.

Je tam ale i tento komentář:

3. Hoy hezky...jen nejak nechapu jak kde si vzal ten zbytek z EDX? jak mam vedet jakej je kdyz nevim jaky jsou spravny hodnoty...? moh by mi nekdo osvetlit jak k tomu dojit? diky moc

Vlastně jsme si na to odpověděli, zbytek si autor někde přičaroval.

Pro tuto lekci není potřeba žádný balík :)

Lekce č.3

Seriál fishing v dalším CMe

Jedno CMe máme za sebou, jdeme na druhé.
Otevřeme si naše nové CMe cm_id5.exe, tentokráte v PEiD, protože o něm přece nic nevíme…:-)
Jdeme tedy na to úplně od začátku.
PEiD nám oznámí, že CMe je psané v Delphi 4.0 - 5.0.
Natáhneme tedy CMe do DeDe…
Najedeme si do Procedures->Unit1->Button1Click…
Vidíme přibližně toto (hned začátek…):

```
00441860   55                  push    ebp
00441861   8BEC                mov     ebp, esp
00441863   81C4D8FEFFFF        add     esp, $FFFFFED8
00441869   53                  push    ebx
0044186A   56                  push    esi
0044186B   57                  push    edi
0044186C   33C9                xor     ecx, ecx
0044186E   898DF0FEFFFF        mov     [ebp+$FFFFFEF0], ecx
00441874   894DF4              mov     [ebp-$0C], ecx
00441877   8BD8                mov     ebx, eax
00441879   BE30384400          mov     esi, $00443830
0044187E   33C0                xor     eax, eax
00441880   55                  push    ebp
00441881   68651B4400          push    $00441B65

***** TRY
|
00441886   64FF30              push    dword ptr fs:[eax]
00441889   648920              mov     fs:[eax], esp
0044188C   33C0                xor     eax, eax
0044188E   8906                mov     [esi], eax
00441890   8D55F4              lea     edx, [ebp-$0C]

* Reference to control TForm1.Edit1 : TEdit //Čtení námi zadaného seriálu z Edit1
|
00441893    8B83C4020000            mov     eax, [ebx+$02C4]  //Zde si v Olly nastavíme BP
```

Tady vidíte největší výhodu DeDe. Zjistíte přesné adresy, kde se co děje. Tady jsme třeba viděli co se přesně děje po kliknutí na Button1, vyčetli jsme, že se provádí čtení z Edit1. Spustíme Olly a otevřeme v něm toto CMe a nastavíme teda BreakPoint na adresu 00441893.
Nastavíme nějaké jméno: DjHacker
a sériové číslo : 123456789
a klikneme na „Validate…"
Zastaví se nám to na zadaném BPintu…

```
00441893   .  8B83 C4020000 MOV     EAX, DWORD PTR DS:[EBX+2C4]
00441899   .  E8 CE19FEFF   CALL    cm_id5.0042326C
0044189E   .  8B45 F4       MOV     EAX, DWORD PTR SS:[EBP-C]
004418A1   .  E8 5622FCFF   CALL    cm_id5.00403AFC
004418A6   .  83F8 05       CMP     EAX, 5  //delka jmena >5 znaku
004418A9   .  0F8C 95020000 JL      cm_id5.00441B44
004418AF   .  8D55 F4       LEA     EDX, DWORD PTR SS:[EBP-C]
004418B2   .  8B83 C4020000 MOV     EAX, DWORD PTR DS:[EBX+2C4]
```

Krokujeme pomocí F8, nesledujeme kód, sledujeme registry…krokujeme tak dlouho, než nenarazíme na adresu

```
00441B0D   .  8B15 38384400 MOV     EDX, DWORD PTR DS:[443838]
```

Tady se do EDX ukládá správné s/n. Po tomhle kroku si ho v EDX přečtěte a opište třeba do poznámkového bloku.

Jak jste si všimli, nesoustředili jsme se na kód, ale na registry a paměť. Sériové číslo se většinou ukládá do procesorových registrů, ale není to podmínkou. Až narazíte někde na nějaký podezřelý string „EKI-4568" nebo „9856-125-44459" nebo tak nějak, toto jsou samozřejmě jen příklady, tak se na

tento kód soustřeďte, ze začátku vám to řekne instinkt kde je s/n, později to budete muset vyhledávat tak, že prošmejdíte CALLy (Trace into [F7])…

Zadáme tedy jméno: DjHacker

a číslo : EKA-1497913

Zkuste si schválně dát BP na adresu 00441B13 a skočte do toho Callu [F7]čkou.

```
00441B0D   .  8B15 38384400 MOV     EDX, DWORD PTR DS:[443838]  //správné s/n do EDX
00441B13   .  E8 F420FCFF   CALL    cm_id5.00403C0C  //Do tohoto callu skočte
00441B18      75 07         JNZ     SHORT cm_id5.00441B21  //až vyskočíte z callu, bude nastaven Zero-Flag, jestli se bude rovnat nule, zadali jste špatné číslo, jestli jedničce, skočíte na oznámení Dobrého s/n…
```

V CALLu uvidíte asi toto:

```
00403C0C  /$  53            PUSH    EBX
00403C0D  |.  56            PUSH    ESI
00403C0E  |.  57            PUSH    EDI
00403C0F  |.  89C6          MOV     ESI, EAX
00403C11  |.  89D7          MOV     EDI, EDX
00403C13  |.  39D0          CMP     EAX, EDX  //porovnej EAX (naše zadané s/n) s EDX (správné s/n)
00403C15  |.  0F84 8F000000 JE      cm_id5.00403CAA  //jestli nesouhlasí, okamžitě skoč na BadBoy, jinak ještě kontroluj
00403C1B  |.  85F6          TEST    ESI, ESI
00403C1D  |.  74 68         JE      SHORT cm_id5.00403C87
```

Do slovníku:
BadBoy – kód s upozorněním na špatné s/n
GoodBoy – kód s oznámením o úspěchu…

V balíku „balik3.rar" najdete:
- cm_id5.exe – CrackMe

Lekce č.4

Další hledání seriálu v CMe

České CMe od sF…
Toto je první oficiální CMe ze stránek picasso.poupe.net. Dalo mě to pěknou práci, než jsem došel k úspěchu, ale potom jsem to celé vykrokoval po jednom kroku, a našel jsem správné s/n…
Dost mluvení, jdeme se na to mrknout…

Spustíme to…a otestujeme, zjistíme, že tlačítko OK se odblokuje až po napsání 11ctého znaku. Podíváme se PEiDem jestli není nějak soubor pakovaný…není a je vytvořený v Delphi.
Natáhneme do DeDe…a hle! Delphi 3…podíváme se na procedurku BtnOkClick…
a je tam tohle:

```
* Reference to: main.Proc_0042F1C8
|
0042F2C0   E803FFFFFF            call    0042F1C8
0042F2C5   C3                    ret
```

už toho víme až až…a hup to zdebuggnout v Olly…
Nastavuju BP na adresu 0042F2C0…zadáme nějaký blábol (12345678901) a hle, zastavilo se to na CALLu na adrese 0042F2C0…Dáme Trace Into [F7] a sleduju…

```
0042F1C8  /$  55            PUSH    EBP
0042F1C9  |.  8BEC          MOV     EBP, ESP
0042F1CB  |.  6A 00         PUSH    0
0042F1CD  |.  6A 00         PUSH    0
0042F1CF  |.  6A 00         PUSH    0
0042F1D1  |.  53            PUSH    EBX
0042F1D2  |.  56            PUSH    ESI
0042F1D3  |.  57            PUSH    EDI
0042F1D4  |.  8BF8          MOV     EDI, EAX
0042F1D6  |.  33C0          XOR     EAX, EAX
0042F1D8  |.  55            PUSH    EBP
0042F1D9  |.  68 B1F24200   PUSH    sF_crmeR.0042F2B1
0042F1DE  |.  64:FF30       PUSH    DWORD PTR FS:[EAX]
0042F1E1  |.  64:8920       MOV     DWORD PTR FS:[EAX], ESP
0042F1E4  |.  8D55 F4       LEA     EDX, [LOCAL.3]
0042F1E7  |.  8B87 EC010000 MOV     EAX, DWORD PTR DS:[EDI+1EC]
0042F1ED  |.  E8 2EADFEFF   CALL    sF_crmeR.00419F20
0042F1F2  |.  8B45 F4       MOV     EAX, [LOCAL.3]
0042F1F5  |.  8078 05 2D    CMP     BYTE PTR DS:[EAX+5], 2D   //6ty znak
musi byt pomlka!!!
0042F1F9  |.  0F85 88000000 JNZ     sF_crmeR.0042F287
0042F1FF  |.  B3 01         MOV     BL, 1
0042F201  |.  BE F0084300   MOV     ESI, sF_crmeR.004308F0           ;
ASCII "88451Ť@"  //to vypada nadejne :-)
0042F206  |>  8D55 F4       /LEA     EDX, [LOCAL.3]
0042F209  |.  8B87 EC010000 |MOV     EAX, DWORD PTR DS:[EDI+1EC]
0042F20F  |.  E8 0CADFEFF   |CALL    sF_crmeR.00419F20
0042F214  |.  8B45 F4       |MOV     EAX, [LOCAL.3]
0042F217  |.  33D2          |XOR     EDX, EDX
0042F219  |.  8AD3          |MOV     DL, BL
0042F21B  |.  8A4410 FF     |MOV     AL, BYTE PTR DS:[EAX+EDX-1]
0042F21F  |.  3A06          |CMP     AL, BYTE PTR DS:[ESI]
0042F221  |.  74 09         |JE      SHORT sF_crmeR.0042F22C
0042F223  |.  8BC7          |MOV     EAX, EDI
```

Dáme F9 ať se ten kód prožene a vyskočí BadBoy…
Dáme následující číslo:
88451-78901
...
už nás to pustí přes "kontrolu pomlky"…a dostali jsme se sem:

```
0042F206  |> /8D55 F4       /LEA     EDX, [LOCAL.3]
0042F209  |. |8B87 EC010000 |MOV     EAX, DWORD PTR DS:[EDI+1EC]
0042F20F  |. |E8 0CADFEFF   |CALL    sF_crmeR.00419F20
0042F214  |. |8B45 F4       |MOV     EAX, [LOCAL.3]
0042F217  |. |33D2          |XOR     EDX, EDX
0042F219  |. |8AD3          |MOV     DL, BL
0042F21B  |. |8A4410 FF     |MOV     AL, BYTE PTR DS:[EAX+EDX-1]
0042F21F  |. |3A06          |CMP     AL, BYTE PTR DS:[ESI]
0042F221  |. |74 09         |JE      SHORT sF_crmeR.0042F22C
      //jestli ZF=0 tak BadBoy…
0042F223  |. |8BC7          |MOV     EAX, EDI
0042F225  |. |E8 6A020000   |CALL    sF_crmeR.0042F494
      //call na badboy
0042F22A  |. |EB 62         |JMP     SHORT sF_crmeR.0042F28E
0042F22C  |> |43            |INC     EBX
0042F22D  |. |46            |INC     ESI
0042F22E  |. |80FB 06       |CMP     BL, 6
0042F231  |.^\75 D3         \JNZ     SHORT sF_crmeR.0042F206
```

Kupodivu to doopravdy byla kontrola čísel 88451…uf…aspoň nemusíme nic luštit…

ale kde je zbytek??…trasujeme dále….

nějak jsme se dostopovali k dalšímu stringu…tentokráte 13435…. Já jsem to zbrkle nastavil za pomlčku a BadBoy :-(…tak lehké to nebude…

Tak…číslo ponechám a jdeme dál trasovat…

dokrokovali jsme sem:

```
0042F264  |.  33D2          XOR     EDX, EDX
0042F266  |.  8BC3          MOV     EAX, EBX
0042F268  |.  E8 2B73FDFF   CALL    sF_crmeR.00406598
0042F26D  |.  81FB BF1B0100 CMP     EBX, 11BBF
//11BBFh = 72639 dec..
0042F273  |.  75 09         JNZ     SHORT sF_crmeR.0042F27E
      //jestli nesouhlasi...badboi
0042F275  |.  8BC7          MOV     EAX, EDI
0042F277  |.  E8 74010000   CALL    sF_crmeR.0042F3F0
      //call na badboy
0042F27C  |.  EB 10         JMP     SHORT sF_crmeR.0042F28E
0042F27E  |>  8BC7          MOV     EAX, EDI
```

Tudíž správné s/n je

88451-72639

zkusíme to...a jde to :-)
neptejte se mě, jak se to počítá...to už fakt ne...prostě když uvidím CMP někde před JE,JNZ,JMP a jinými "jumpy",
tak se mrknu do paměti, anebo si to překonvertuju z hex do dec...
Jak jste mohli vidět teď...něco ve mě trklo a řeklo mi,
že mám hodit 11BBF do dec a hle a bylo to...jako projížděl jsem CALLy, to jo..ale,
mám svůj styl crackování...a už do Vás hustím takový sena...asi sem toho o sobě prozradil dost :-)...
Jsem začátečník a toto CMe hodnotím na 3/10...vzhledem k tomu že jsem ho vyluštil i já...

Použité nástroje:DeDe
Na podívání, co skrývá btnOkClick za kód...
OllyDbg
Na zjištění s/n
Windows Kalkulačka...
Na konvertovaní dec -> hex a naopak...
WinHex
Tam jsem se podíval co je 2D za znak (pomlčka)

…just…
s/n:
88451-72639

Celé toto CMe řešení jsem zkopíroval z mojeho řešení, když jsem toto CMe řešil. Tak kdyžtak omluvte trochu nespisovštiny…

Ještě bych chtěl dodat…
13435, jak jsem si myslel že se jedná o správný kód (resp.o správnou druhou část), tak 13435h = 78901 dec. Tudíž to je moje ZADANÁ druhá polovina. Převod druhé poloviny do hex se nachází zde:
0042F25A - Zde vejděte do Callu,
004065F6 – Zde vejděte do Callu(hned první call), a zde co bude probíhat je převod druhé poloviny s/n do hex, do reg. EAX, ten se potom přesune do ESI a do finálního EBX. Zde jsou přesně ty instrukce, které provádí převod:

```
00402874  |> /80EB 30       /SUB     BL, 30
00402877  |. |80FB 09       |CMP     BL, 9
0040287A  |. |77 2A         |JA      SHORT sF_crmeR.004028A6
0040287C  |. |39F8          |CMP     EAX, EDI
0040287E  |. |77 26         |JA      SHORT sF_crmeR.004028A6
00402880  |. |8D0480        |LEA     EAX, DWORD PTR DS:[EAX+EAX*4]
00402883  |. |01C0          |ADD     EAX, EAX
00402885  |. |01D8          |ADD     EAX, EBX
00402887  |. |8A1E          |MOV     BL, BYTE PTR DS:[ESI]
00402889  |. |46            |INC     ESI
0040288A  |. |84DB          |TEST    BL, BL
0040288C  |.^\75 E6         \JNZ     SHORT sF_crmeR.00402874
```

V balíku „balik4.rar“ najdete:

- CrackMe od StealthFightera v1
- MojeReseni.txt :)

Lekce č.5

Cracknutí našeho prvního programu

Říkáte si že už je dost CMeček? Tak tedy jdeme crnknout skutečný program, který něco dělá. Crackneme si prográmek Delphi to C++Builder, který překonvertuje Delphi kód co C++ kódu. Provádí to tzv. „simply" tzn.že se na výsledek nedá 100% spolehnout, ale na jednoduché úpravy to stačí.

Spustíme to...oooh prográmek vypadá amatérsky...klikneme na register..napíšeme nějaký blábol...
Klikneme na Register a hle.. Invalid register…

Hupnem s tím do Ollyho…hledám stringy…a našel jsem pěkný ho*no…
Šup s tím do DeDe…
Klikneme na Procedures a najdeme Unit2 (ClassName= TRegForm) a mrkneme se na DisAsm výpis toho, co se skrývá pod tlačítkem Button1 (poklepem na něj…)..
A hle otevřelo se mi podobný okýnko jak v W32Dasmu…napadlo mě že sem měl najít stringy nejdřív v W32Dasmu…no…ale co už...
Čítám…a vidím nějaký…fakt drsný string:

```
* Possible String Reference to: 'Thislicenseappliestoanysoftwarecont
|                               aininganoticeplacedbythecopyrightho
|                               ldersayingthatitmaybedistributedund
|                               erthetermsoftheQtNon-CommercialLice
|                               nseversion1.0.Suchsoftwareishereinr
|                               eferredtoastheSoftware.Thislicensec
|                               oversdistributionoftheSoftware,useo
|                               fthird-partyapplicationprogramsbase
|                               dontheSoftware,andnon-commercialdev
|                               elopmentoffreesoftware,whichusesthe
|                               Software.C++Builder6extendsthepower
|                               ofBorland'sacclaimedANSIC++rapidapp
|                               licationdevelopmenttoolwithawealtho
|                               fnewfeaturesandenhancements'
```

Omg..ignoruju a čtu dál…

Narazím na toto:

```
* Reference to control Edit3 : TEdit
|
00408252   8B86F4020000          mov     eax, [esi+$02F4]
|
00408258   E8D7430500            call    0045C634
```

Procedura čtení z Edit3 (to je to kam zadáváte seriál)

……

```
* Possible String Reference to: 'OK,thank you for your support Ł˘'
|
00408636   BA91BE4800            mov     edx, $0048BE91
0040863B   8D45A0                lea     eax, [ebp-$60]
```

Když to regnem Sucessfully…

……..

```
* Possible String Reference to: 'Invalid register Ł˘'
|
004086DF   BAE2BE4800            mov     edx, $0048BEE2
```

……

Když to neregnem…

Už toho víme dost… natáhnem to do znovu do Olly a dáme BP na adresy 00408252 a 004086DF…

Spustíme F9
Klikneme na Register
zadáme opět nějaký blábol (‚1234‘)
a klikneme na Register…

Olly se nám instinktivně zastaví na adrese 00408252

Krokujeme pomocí F8…

```
00408271   .  8D55 E4       LEA     EDX, DWORD PTR SS:[EBP-1C]
00408274   .  58            POP     EAX
00408275   .  E8 EECE0600   CALL    d2c.00475168
0040827A      .   8985  68FFFFFF  MOV        DWORD  PTR  SS:[EBP-98],  EAX
//nahrávání zadaného s/n do EAX
00408280   .  FF4B 1C       DEC     DWORD PTR DS:[EBX+1C]
00408283   .  8D45 E4       LEA     EAX, DWORD PTR SS:[EBP-1C]
```

…nějaké bláboly, nerozumim vela :-)

Krokuju dál a sleduju registry, jestli se tam neobjeví něco zajímavého
Furt vidím v EDI string „HyA“ je to provokativní, ale správné s/n to není…na to je to moc krátký string…

…
A tady je to:

```
004083E0   .  51            PUSH    ECX                              ;
/Arg1
004083E1   .  E8 16EFFFFF   CALL    d2c.004072FC                     ;
\d2c.004072FC
004083E6   .  59            POP     ECX
004083E7   .  8D85 CCFDFFFF LEA     EAX, DWORD PTR SS:[EBP-234]
004083ED   .  66:C743 10 08>MOV     WORD PTR DS:[EBX+10], 8    // a hle!
V EAXu je správné s/n! :-)
004083F3   .  66:C743 10 5C>MOV     WORD PTR DS:[EBX+10], 5C
004083F9   .  8D55 F4       LEA     EDX, DWORD PTR SS:[EBP-C]
```

…**správné s/n je „ovmfSdsilo“**

```
0040840A   .  8D55 F4       LEA     EDX, DWORD PTR SS:[EBP-C]
0040840D   .  66:C743 10 08>MOV     WORD PTR DS:[EBX+10], 8
00408413   .  8D45 F8       LEA     EAX, DWORD PTR SS:[EBP-8]
00408416   .  E8 DDCB0600   CALL    d2c.00474FF8  //tady zavolá proceduru
která zkotroluje jestli je s/n správné
0040841B   .  84C0          TEST    AL, AL   //otestuje
0040841D   .  0F84 B6020000 JE      d2c.004086D9//a jestli je tak
pokračuj, jestli ne tak skoč na 4086D9
00408423   .  6A 01         PUSH    1
00408425   .  BA 6DBE4800   MOV     EDX, d2c.0048BE6D                ;
ASCII "registed"
0040842A   .  8D45 CC       LEA     EAX, DWORD PTR SS:[EBP-34]
0040842D   .  E8 DAC90600   CALL    d2c.00474E0C
00408432   .  FF43 1C       INC     DWORD PTR DS:[EBX+1C]
```

…

No nic, znovu pouštím F9, zadám s/n „ovmfSdsilo“ trochu krokuju a dívám se kde a jak se to počítá…
…s/n se nejspíš přepočítává nějakým výpočtem, co si hraje se stringem „Delphi to C++Builder“ jinak nic moc nevím…potom když vstoupíte do CALLu na adrese 00408416 a potom do CALLu na adrese 00475000 uvidíte nádhernou procedurku:

```
0042B378  /$  53            PUSH    EBX
0042B379  |.  56            PUSH    ESI
0042B37A  |.  57            PUSH    EDI
0042B37B  |.  89C6          MOV     ESI, EAX
0042B37D  |.  89D7          MOV     EDI, EDX
0042B37F  |.  39D0              CMP       EAX, EDX  //Zde se porovnávají
správné s/n a zadané s/n
0042B381  |.  0F84 8F000000 JE          d2c.0042B416  //jestli je s/n
správné, pokračuj jestli ne, okamžitě RETURNuj zpátky
0042B387  |.  85F6          TEST    ESI, ESI
0042B389  |.  74 68         JE      SHORT d2c.0042B3F3
0042B38B  |.  85FF          TEST    EDI, EDI
0042B38D  |.  74 6B         JE      SHORT d2c.0042B3FA
0042B38F  |.  8B46 FC       MOV     EAX, DWORD PTR DS:[ESI-4]
0042B392  |.  8B57 FC       MOV     EDX, DWORD PTR DS:[EDI-4]
0042B395  |.  29D0          SUB     EAX, EDX
0042B397  |.  77 02         JA      SHORT d2c.0042B39B
0042B399  |.  01C2          ADD     EDX, EAX
0042B39B  |>  52            PUSH    EDX
0042B39C  |.  C1EA 02       SHR     EDX, 2
0042B39F  |.  74 26         JE      SHORT d2c.0042B3C7
0042B3A1  |>  8B0E          /MOV     ECX, DWORD PTR DS:[ESI]
0042B3A3  |.  8B1F          |MOV     EBX, DWORD PTR DS:[EDI]
0042B3A5  |.  39D9          |CMP     ECX, EBX
0042B3A7  |.  75 58         |JNZ     SHORT d2c.0042B401
0042B3A9  |.  4A            |DEC     EDX
0042B3AA  |.  74 15         |JE      SHORT d2c.0042B3C1
0042B3AC  |.  8B4E 04       |MOV     ECX, DWORD PTR DS:[ESI+4]
0042B3AF  |.  8B5F 04       |MOV     EBX, DWORD PTR DS:[EDI+4]
0042B3B2  |.  39D9          |CMP     ECX, EBX
0042B3B4  |.  75 4B         |JNZ     SHORT d2c.0042B401
0042B3B6  |.  83C6 08       |ADD     ESI, 8
0042B3B9  |.  83C7 08       |ADD     EDI, 8
0042B3BC  |.  4A            |DEC     EDX
0042B3BD  |.^ 75 E2         \JNZ     SHORT d2c.0042B3A1
0042B3BF  |.  EB 06         JMP     SHORT d2c.0042B3C7
0042B3C1  |>  83C6 04       ADD     ESI, 4
0042B3C4  |.  83C7 04       ADD     EDI, 4
0042B3C7  |>  5A            POP     EDX
0042B3C8  |.  83E2 03       AND     EDX, 3
0042B3CB  |.  74 22         JE      SHORT d2c.0042B3EF
0042B3CD  |.  8B0E          MOV     ECX, DWORD PTR DS:[ESI]
0042B3CF  |.  8B1F          MOV     EBX, DWORD PTR DS:[EDI]
0042B3D1  |.  38D9          CMP     CL, BL
0042B3D3  |.  75 41         JNZ     SHORT d2c.0042B416
0042B3D5  |.  4A            DEC     EDX
0042B3D6  |.  74 17         JE      SHORT d2c.0042B3EF
```

```
0042B3D8  |.  38FD          CMP     CH, BH
0042B3DA  |.  75 3A         JNZ     SHORT d2c.0042B416
0042B3DC  |.  4A            DEC     EDX
0042B3DD  |.  74 10         JE      SHORT d2c.0042B3EF
0042B3DF  |.  81E3 0000FF00 AND     EBX, 0FF0000
0042B3E5  |.  81E1 0000FF00 AND     ECX, 0FF0000
0042B3EB  |.  39D9          CMP     ECX, EBX
0042B3ED  |.  75 27         JNZ     SHORT d2c.0042B416
0042B3EF  |>  01C0          ADD     EAX, EAX
0042B3F1  |.  EB 23         JMP     SHORT d2c.0042B416
0042B3F3  |>  8B57 FC       MOV     EDX, DWORD PTR DS:[EDI-4]
0042B3F6  |.  29D0          SUB     EAX, EDX
0042B3F8  |.  EB 1C         JMP     SHORT d2c.0042B416
0042B3FA  |>  8B46 FC       MOV     EAX, DWORD PTR DS:[ESI-4]
0042B3FD  |.  29D0          SUB     EAX, EDX
0042B3FF  |.  EB 15         JMP     SHORT d2c.0042B416
0042B401  |>  5A            POP     EDX
0042B402  |.  38D9          CMP     CL, BL
0042B404  |.  75 10         JNZ     SHORT d2c.0042B416
0042B406  |.  38FD          CMP     CH, BH
0042B408  |.  75 0C         JNZ     SHORT d2c.0042B416
0042B40A  |.  C1E9 10       SHR     ECX, 10
0042B40D  |.  C1EB 10       SHR     EBX, 10
0042B410  |.  38D9          CMP     CL, BL
0042B412  |.  75 02         JNZ     SHORT d2c.0042B416
0042B414  |.  38FD          CMP     CH, BH
0042B416  |>  5F            POP     EDI
0042B417  |.  5E            POP     ESI
0042B418  |.  5B            POP     EBX
0042B419  \.  C3            RETN  //z5 :-)
```

…**nějaké bláboly zase**…

Takže…
s/n=
ovmfSdsilo

A teď patch…nejjednodušší to bude přes registry…tedy přes *.reg soubor…v Ollym jste si určitě všimli nápadného stringu :

```
00408806   .  BA 30BF4800   MOV     EDX, d2c.0048BF30                ;
ASCII "software\\ahao's softwares\\wenku\\d2c"
```

No a to je ten klíč, kam se ukládá s/n…

Takže výsledný crack bude Reg.reg soubor :-D…
Bude vypadat asi takto…:

```
Windows Registry Editor Version 5.00

[HKEY_LOCAL_MACHINE\SOFTWARE\ahao's softwares\wenku\d2c\smtp]
"registed"="1"
"sc"="ovmfSdsilo"
"installed"="1"
"times"="0"
"stime"="2.8.2007"
"ltime"="2.8.2007"
"n1"=""
"n2"="Delphi To C++Builder"
"rc"=""
```

Jak jsem k tomu došel? Normálně, prostě jsem dumpnul registry :-) ---ne..najel jsem do adresy :
HKEY_LOCAL_MACHINE\SOFTWARE\ahao's softwares\wenku\d2c\smtp
a tu jsem pomocí funkce Exportovat, exportoval…
Soubor spusťte a máte registrovaný program :-)

Tento článek byl taky psaný mimo tuto knížku, tak prosím omluvte zase nějaké úlety :-)… Jinak jsem ještě zjistil kde se s/n počítá…

Na adrese 004083E1 se nachází CALL, ten trasujte, dostanete se mezi kupu instrukcí, a to je právě výpočet s/n. Nakonec na adrese 00407526 – 00407530 se nachází sled instrukcí, který zapisuje do paměti 0012E6F7. Když se na toto místo v paměti podíváme (Dumpneme v Olly) a krokujeme, vidíme jak jednotlivá písmena se generují, a nakonec uloží do ESI a pak do EAX.

V balíku „balik5.rar" najdete:

- **Program DelphiToC++Conventer v1.0**
- **Reg.reg – soubor pro zaregistrování**

Lekce č.6

Tajemství výroby ochran

Teď oddech od crackingu. Na chvíli se vrhneme do programování. Měli byste se alespoň trochu orientovat v Delphi (Pascal), a C++.
Vyrobíme si jedno CMe, v Delphi.
Tajemství spočívá, jak jinak, v šifrách. V Delphi je základ příkaz Ord(‚písmeno')… Samozřejmě my budeme chtít náš…třeba Nick…číst z EditBoxu. To uděláme takto: Ord(Edit1.Text[číslo písmena]);
Dost k teorii, jdeme k příkladu. Takto může vypadat s/n algoritmus:

```
Var
Sn, one:string;
Cykl:integer;
Begin
Sn:='';
One:=Edit1.Text;
for cykl:=1 to 18 do
   begin
      sn:=sn+inttostr(ord(one[cykl])+cykl*cykl-2);
   end;
end;
```

Toto je vážně jednoduchý algoritmus. Ještě jsem neřekl jednu podstatnou věc, a to, co to vlastně ono „Ord" je. Ord(písmeno) převede písmeno na číslo v ASCII tabulce. Tzn.že Ord('A') se rovná 65. Ord('a') je 97…atd…Musíme si uvědomit, že ord je integer, a musíme ho vždy převést příkazem IntToStr…

Provedeme si výpočet prvního písmena v zadaném nicku, a převedeme si ho na první číslo v s/n. Dejme tomu že první písmeno bude ‚D'…
Ord('D') := 68
Cykl^2 := 1
Cykl^2 -1 := -1

68+(-1) := 67…
První číslo v s/n bude tedy '67'…
Vyrobil jsem pro tento účel CMe. Source najdete v tomto balíku (i s Keygenem).
Nedivte se zpracování :-). Je to plnohodnotný komprimovací program, ale účel je stejný, dojít ke správnému s/n. Taky mě nemusíte říkat, že mám v source bordel :-)…vím to, a zvykejte si…

Najdete tam tuto šifrovací proceduru:

```
var
one,two,three,seven,sno,snt,snsf,snf :string;
four,five,six,eigth :integer;
begin
randomize;
if length(Edit1.Text) < 5 then
begin
ShowMessage('Nick musí mít aspoň 5 znaků!');
exit; end;
//////////výpočet serialu/////
  begin
    one:=Edit1.Text;
    one:=one+'ReGiStRaTiOn!';
    four:=Length(one);
    for five:=1 to 18 do
      begin
        two:=two+inttostr(ord(one[five])+five*five-2);
      end;
      seven:=inttostr(ord(two[1]));
      seven:=seven[2];
      eigth:=((strtoint(seven)div 2)+1);
    for six:=1 to 40 do
      begin
```

```
        if (six/3) = (round(six/3)) then
          begin
            three:=three+((two[six]));
            if ((six/eigth) = (round(six/eigth))) and (six<26) then
              begin
                three:=three+char(six+64);
              end;
          end;
      end;
        for six:=1 to 5 do
          begin
            sno:=sno+three[six];
          end;
        for six:=1 to 5 do
          begin
            snt:=snt+inttostr(ord(sno[six]));
          end;
        for six:=1 to 5 do
          begin
            snsf:=snsf+snt[six];
          end;

if Edit2.Text = ('DjH-'+sno+snsf) then begin regist; end;


          if Edit2.Text <> ('DjH-'+sno+snsf) then begin
          asm
          xor eax,eax
          xor ebx,ebx
          end;
          six:=ord(char(round(random(32)+64)));
          showmessage('Špatný reg.kód nebo Nick....'); end;

  end;
//////////výpočet serialu/////
end;
```

Zde vidíte pěkný příklad. Ale řekli byste, že i tento kód má ochranu asi tak 1/10 ?
Proč? Protože se vypočítávané číslo ukládá do paměti, a můžete si ho jednoduše přečíst v OllyDbg. Místo stringu 'DjH-', jsme mohli napsat Char(68) +

Char(106) + Char(72) + Char(45)… vyšlo by to nastejno, ale string by nebyl jen tak čitelný. Instrukce Char je úplný opak Ord. Z ASCII čísla vyrobí znak. Mohli jsme taky napsat:

```
If (Ord(Edit2.Text[1]) = 68) and (Ord(Edit2.Text[2]) = 106) and
(Ord(Edit2.Text[3]) = 72) and (Ord(Edit2.Text[4]) = 45)
```

……

Mohli jste také vidět: ……then Begin regist; end;… Jestli je s/n správné, skočí na proceduru Regist, kde je jen zápis hodnot do souboru… Při načítání se tyto hodnoty načtou, a ověří se jejich správnost, jako kdybyste to s/n zadávali v okně. Jestli program vyhodnotí s/n jako správné, pokračuje cestou kde vypíše v okně třeba ‚Registered to:' + vaše jméno… jestli vyhodnotí s/n jako špatné (třeba při prvním spuštění…) pokračuje tak, že do okna napíše třeba ‚Unregistered'… Mohli jste postřehnout také slabost. Když obskočíme hned toto kontrolování, program se může tvářit jako registrovaný pořád. Necháme program načíst jen Nick, a místo čtení s/n (ze souboru), přepíšeme na jump kde se (např.) vypisuje do okna ‚Registered to:'.
Zkusíme si moje CMe cracknout, nic složitého to není.
Víme, že program je kompilovaný v Delphi, načteme ho tedy do DeDe. Podíváme se do Procedures, Form3. Vidíme zde dvě tlačítka a dvě procedury. Regist a Before a Button1 a 2.Na obou Buttonech je jeden Call a Ret. Tzn., že toho moc nevyčteme. Ale když se pořádně podíváme, vidíme na Button1Click toto:

```
* Reference to : TForm3.before()
|
00470ECC   E893FCFFFF            call    00470B64
00470ED1   C3                    ret
```

Reference to TForm3.Before(). Tzn., že tento button volá proceduru ‚before', která vše kontroluje…
Při prohlédnutí procedury Before, vám musí být jasné, že se jedná o kontrolu s/n. Procedura regist, pak jen zapisuje do souboru.
Mrkneme se tedy na adresu z tlačítka Button1, a nastavíme v OllyDbg BP na adresu s CALLem. Spustíme program, a klikneme na Nápověda->Registrace. Zadáme nějaké jméno…Třeba DjH… a jako s/n 123456…klikneme na OK. A krokujeme (do callu vstupte pomocí F7).

```
00470BA1  |.  83F8 05       CMP     EAX, 5
00470BA4  |.  7D 0F         JGE     SHORT Compress.00470BB5
00470BA6  |.  B8 580E4700   MOV     EAX, Compress.00470E58          ;
ASCII "Nick musí mít aspoň 5 znaků!"
00470BAB  |.  E8 4C87FBFF   CALL    Compress.004292FC
```

Vyhodí nám to hlášku o tom, že nick musí mít aspoň 5 znaků. Nastavíme Nick: DjH2oo7 a s/n ponecháme na 123456.
A krokujeme zas…

```
00470BC6  |.  BA 800E4700   MOV     EDX, Compress.00470E80          ;
ASCII "ReGiStRaTiOn!"
00470BCB  |.  E8 603BF9FF   CALL    Compress.00404730
00470BD0  |.  8B45 FC       MOV     EAX, [LOCAL.1]
```

Když proženeme CALL, zjistíme, že k Nicku se nám přidá slovo „ReGiStRaTiOn!". (např. ‚DjH2oo7ReGiStRaTiOn!'). Dostaneme se sem:

```
00470BD3  |.  E8 503BF9FF   CALL    Compress.00404728
00470BD8  |.  BB 01000000   MOV     EBX, 1
00470BDD  |>  8B45 FC       /MOV     EAX, [LOCAL.1]
00470BE0  |.  0FB64418 FF   |MOVZX   EAX, BYTE PTR DS:[EAX+EBX-1]
00470BE5  |.  8BD3          |MOV     EDX, EBX
00470BE7  |.  0FAFD3        |IMUL    EDX, EBX
00470BEA  |.  03C2          |ADD     EAX, EDX
00470BEC  |.  83E8 02       |SUB     EAX, 2   //eax=eax-2
00470BEF  |.  8D55 DC       |LEA     EDX, [LOCAL.9]
00470BF2  |.  E8 4179F9FF   |CALL    Compress.00408538
00470BF7  |.  8B55 DC       |MOV     EDX, [LOCAL.9]
00470BFA  |.  8D45 F8       |LEA     EAX, [LOCAL.2]
```

```
00470BFD  |.  E8 2E3BF9FF   |CALL    Compress.00404730
00470C02  |.  43            |INC     EBX  //ebx:=ebx+1
00470C03  |.  83FB 13       |CMP     EBX, 13 //dec:19
00470C06  |.^ 75 D5         \JNZ     SHORT Compress.00470BDD
00470C08  |.  8D55 F0       LEA     EDX, [LOCAL.4]
00470C0B  |.  8B45 F8       MOV     EAX, [LOCAL.2]
00470C0E  |.  0FB600        MOVZX   EAX, BYTE PTR DS:[EAX]
00470C11  |.  E8 2279F9FF   CALL    Compress.00408538
```

Operace mezi adresami 00470BDD až 00470C06 bychom mohli označit jako

```
for five:=1 to 18 do
  begin
    two:=two+inttostr(ord(one[five])+five*five-2);
  end;
```

```
00470C03  |.  83FB 13       |CMP     EBX, 13 //dec:19
```

Zde vidíme důkaz…Proč je zde 13(19) a ne 12(18)? Protože při 19tém čísle je EBX 18, tzn. že 19ctá operace se neprovádí…

Na adrese 00470C0B nám to vyhodí velkou kupu čísel: 671087964134145102144180169224225283276320338392401

Je to vlastně zatím proměnná ‚two'… Uvidíme co se bude dít dál.

Od adresy 00470C3B – 00470CD4 vidíme

```
for six:=1 to 40 do
  begin
    if (six/3) = (round(six/3)) then
      begin
        three:=three+((two[six]));
        if ((six/eigth) = (round(six/eigth))) and (six<26) then
          begin
            three:=three+char(six+64);
          end;
```

A takhle bychom mohli pokračovat. Uvádím to jen pro orientaci, kdybychom crackovali program a neviděli source, asi stěží bychom rozeznali která procedura je která. Zkušení crackeři to však dokážou. Nakonec na adrese 00470D6F vidíme CALL, ten vystopujeme. Jedná se o CALL který všechny ty čísla přeluští a

vytvoří správné s/n. Dalo by se říct že je to tento kód(zvýrazněno zelenou…):

```
if Edit2.Text = ('DjH-'+sno+snsf) then begin regist; end;
```

potom si už můžeme na adrese 00470D77 přečíst v EDX správné s/n (DjH-1C7F449675). Kontrola mezi správným a zadaným s/n je na adrese 0040487B (po callu z adresy 00470D77).

Nyní zadáme Nick (DjH2oo7) a správné s/n (DjH-1C7F449675) a kliknem na OK, jestli chcete, tak můžete krokovat, nakonec vám vyskočí hláška o úspěšném zaregistrovaní (mj. odblokuje se vám v CMe možnost dekomprimace :-))

Jestli si vzpomínáte, mluvil jsem o chybě. O chybě, že když se načítá program a kontroluje Nick a s/n, můžete nechat načíst jen nick a místo čtení s/n, přepíšete příkaz na jump který odkazuje na GoodBoy. My si nyní (na tom stejném CMe) pokusíme najít takovou chybu, a napíšeme si patch. Nyní už by se tomu dalo říkat Crack.

Vymažeme po úspěšné registraci soubor reg.ini a spustíme program („jen tak" – bez debugování). Dole vidíme napsáno „CompressMan v 1.0:neregistrováno…". Nyní si načteme program do OllyDbgru a klikneme pravým myšítkem do oblasti s ASM kódem a klikneme na „Search for" -> „All referenced text strings" (toto lze také obejít přes program DeDe, ale chci vás naučit s Olly). Najdeme si string „neregistrováno"…je úplně dole na adrese

```
004718A9   .  BA 94194700   MOV     EDX, Compress.00471994          ;
ASCII "CompressMan v 1.0:neregistrováno..."
```

Půjdeme po stopách programu, ale nahoru (program nezapínejte!!!). Narazíme na adresu 00471867, kde vidíme po CALLu funkci JNZ a skáče na část s textem neregistrováno. Pokud bychom tento příkaz zaměnili za JE, program by skočil na část GoodBoy s každým špatným s/n. Ale pozor, kdybychom zadali správné s/n, program by jej v tom případě vyhodnotil jako špatné! Od tohoto problému by nás oddělil příkaz NOP. Ale zůstaneme u JE. Dvakrát poklepeme na příkaz, a zaměníme jen JNZ za JZ (nebo JE). Spustíme program (F9), a … neregistrováno! Jakto? Protože program nenašel žádný ‚reg.ini' soubor(protože jsme ho vymazali)! A tak vytvořil nový. Kdybychom spustili program znovu, Napsal by nám, že je registrován na „1". Podívejte se do souboru „reg.ini" Vidíme tam Nick=1 a sn=1. Zjistíme tedy, že naše první spuštění programu (po vymazání reg.ini), je jediná možnost kdy se můžeme zaregistrovat na libovolné jméno. Ale musí nám to „sežrat" okno Registrace… Takže… Stiskneme [Ctrl+F2] aby se program „resetoval". Vymazal se nám i patch na té adrese který jsme udělali, ale to nevadí, protože když klikneme na tlačítko [/] objeví se nám tam ve stavu removed. Potom ho stačí dát do stavu aktive a bude to :-), nemusíme tedy nic opět hledat. Takže uděláme práci navíc, a tuto stejnou chybu aplikujeme i na tlačítko pro ověření s/n. Pamatujeme si adresu pro Button1Click :00470ECC. Tam vidíme CALL na adresu 00470B64. Najdeme si tuto adresu (program je stále vypnutý!). Tam jsme vystopovali jump na adrese 00470D7D. Je tam příkaz JNZ, my ho zaměníme za příkaz JZ(JE). Je to vlastně JNZ po kontrole správnosti s/n (resp. po callu). Po tomto JNZ probíhá další kontrola. Pokud bychom to nechali

tak, jak to teď máme a uložili, klikli bychom na registrovat, tak by nám to vyplivlo hlášku, že jsme to úspěšně registrovali, a po odkliknutí této hlášky by se objevila nová, že jsme to regli špatně. Ale do souboru se to uloží, takže problém je akorát ten, že např. běžný uživatel by byl zmatený. Proto na adrese 00470DB9 vidíme příkaz JE, které po další kontrole s/n, by způsobil skok na BadBoy. Přepíšeme tento příkaz na JNZ. Já bych si byl jistý, že tyto tři patchlé byty by měli být všechno. Můžeme si tedy náš cracklý soubor uložit. Olly nám přímo umožňuje soubor uložit. Klikneme pravým myšítkem na ASM kód a klikneme na „Copy to executable" -> „All modifications". Zeptá se vás to na otázku. Stiskněte „Copy all". Objeví se nám nové okno. Na to klikneme opět levým myšítkem a stiskneme „Save to file". Uložíme si to třeba jako „CMe1[cracked].exe". Až vše uložíme, minimalizujeme si Ollyho, vymažeme soubor „reg.ini" a spustíme si náš cracklý soubor. Klikneme na Nápověda->registrovat. Zadáme jakýkoliv nick (DjH2oo7) a jako s/n můžete zadat třeba „na_s/n_ti_jebu!!!" :-), program to vyhodnotí jako správné s/n…

V balíku „balik6.rar" najdete:

- CrackMe – „CompressMan" – by DjH2oo7
- KeyGen – by DjH2oo7

Lekce č.7

Tajemství výroby cracků a patchů

Předem bych chtěl říct, že crackeři se většinou pokoušejí rozluštit s/n, a pokud možno vše bez patchování. Ale jsou i situace, kdy se tomu prostě nevyhnete (viz Lekce 8 :-)… Ale teď už k praxi…

Já osobně jazyk C++ moc neumím. Znám jen příkazy, které „musím". Nejvíce mi vyhovuje jazyk Pascal (Delphi), jak již jste mohli vidět. Akorát mě vážně se*e ta výstupní velikost *.exe souboru… V C++ bych asi ani CMe nenaprogramoval, ale i přesto dělám cracky právě v něm.

„Crack" co jsme vytvořili v minulé lekci je stejně velký jako program nepatchovaný. Proč? Protože jsme byty přepsali, ale nic jsme nepřidali. Program tedy zabírá něco kolem 540kB. Což je moc. Mohli bychom program zkomprimovat např. UPXem, nebo packovat do RARu. Ale není nic lepšího, než si vytvořit vlastní program, který vše zapatchuje.

Jak bylo psáno, crack si naprogramujeme v C++. Já budu programovat ve Visual C++ 2005 od Microsoftu. Můžete programovat třeba i v Dev-C++ (c++ jako c++), ale já přikládám source k Vis. C++ 2oo5. Dalo by se říct, že cracky, které budete vyrábět budou založené na této „kostře":

```
#include "stdafx.h"
#include <stdio.h>
#include <stdlib.h>
#include "Form1.h"
{
```

```
FILE *f;


int adr=0xadresa;       //offset
int owr=0xhex_prikaz;        //hex

f = fopen( "crackme.exe" , "rb+");
if (f==NULL) { MessageBox::Show( " File can not found!", "Error",
MessageBoxButtons::OK, MessageBoxIcon::Exclamation );
exit(1);
}


fseek(f,adr,SEEK_SET);


fwrite(&owr,1,1,f);
MessageBox::Show( "Cracked :-)", "Cracked...:-)" ,MessageBoxButtons::OK,
MessageBoxIcon::Exclamation );
fclose(f);
}
```

Ano, je to ta samá kostra z Lekce 1. Ale my si ji samozřejmě upravíme. Jelikož chceme patchnout 3 byty, musíme si najít jejich offsety, a tento cyklus (tohoto kódu), 3x zopakovat. Vypíšu zde offsety které patchnem (podle návodu v Lekci 1 si je můžete také najít sami), a za pomlčkou uvedu, na který byte (hex) to pachnem.

7017D – 74
701B9 – 75
70C67 - 74

Kód potom bude asi takový:
(Kompilovaný patch a source je přiložený…)

```
#include "stdafx.h"
#include <stdio.h>
#include <stdlib.h>
#include "Form1.h"
{
FILE *f;
```

```
int adr=0x7017D;       //offset
int owr=0x74;        //hex
f = fopen( "CompressMan_CrackMe_2.exe" , "rb+");
if (f==NULL) { MessageBox::Show( "File can not found!", "Error",
MessageBoxButtons::OK, MessageBoxIcon::Exclamation );
exit(1);
}
fseek(f,adr,SEEK_SET);
fwrite(&owr,1,1,f);


 adr=0x701B9;       //offset
 owr=0x74;        //hex
f = fopen( "CompressMan_CrackMe_2.exe" , "rb+");
if (f==NULL) { MessageBox::Show( "File can not found!", "Error",
MessageBoxButtons::OK, MessageBoxIcon::Exclamation );
exit(1);
}
fseek(f,adr,SEEK_SET);
fwrite(&owr,1,1,f);

 adr=0x70C67;       //offset
 owr=0x74;        //hex
f = fopen( "CompressMan_CrackMe_2.exe" , "rb+");
if (f==NULL) { MessageBox::Show( "File can not found!", "Error",
MessageBoxButtons::OK, MessageBoxIcon::Exclamation );
exit(1);
}
fseek(f,adr,SEEK_SET);
fwrite(&owr,1,1,f);



MessageBox::Show( "Cracked :-)", "Cracked...:-)" ,MessageBoxButtons::OK,
MessageBoxIcon::Exclamation );
fclose(f);
}
```

V balíku „balik7.rar" najdete:

- **CrackMe „CompressMan" - by DjH2oo7**
- **Crack for „CompressMan" - by DjH2oo7**

Lekce č.8

Crackneme si hru…

Promiňte mi to, že teď píšu až moc za sebe, ale víceméně, jedná se jen o MOJÍ :-) knížku o crackingu. Tak budu hodně mluvit za sebe.

Jako každej moderní puberťáckej kluk chodím do baru. Chodím tam většinou v pátek, a chodím do klubu, který se nazývá „9" -devítka-. Je tam hodně kulečníkových stolů, a celkově je to vlastně kulečníkový klub. A náhodou jsem zaštrachal mezi svými CD od PCWorldu a co nevidím: 3D Live Pool… Říkám si „bezva, konečně nějaký 3D kulec"… Přečtu si informace a co nevidím: trial 30. Říkám si OMG, proč ne, třeba to crnknu. V té chvíli jsem to bral spíše z legrace… Hra nainstalovaná a co nevidím…žádný trial 30, ale Times Run 10! Neboli hru spustíte 10krát a dost. Když jsem si hru otevřel v PEiD, uviděl jsem, že kulec je kódovaný ve Visual C++ 6. Podíval jsem se na Crypto analyser, a vidím nějaké Zlib, ADLER32…a nějaká sena, nerozumim tomu vela :-) Když jsem se na to podíval v Resource Hackeru, a našel jsem si v Dialogs okno, které mi říká, že x Runs zbývá, už jsem si myslel, že mám vyhráno, ale opak byl pravdou. Program jsem si natáhl do Olly, a hledám string „times"…nic se nenašlo. Asi půl hodiny jsem ten kulec hrál, a až mě to přestalo bavit, řekl jsem si, že risk je zisk. Spustil jsem tedy program už jen 7x. Nyní po startu program nenaběhne, jen se

objeví okno, že 0 runs zbývá a tlačítko quit. Hru byste měli najít i zde v balíčku.

Používám svůj styl crackování, možná ho používá většina hackerů, ale já mému stylu říkám „Lame Call'o'trasing" :-)

Jedná se o to, že v programu je hodně callů. Já zkusím je nejdříve přeskočit (F8), a když udělají práci, kterou já chci vystopovat, program restartuju (nebo třeba stisku tlačítko Registrovat…záleží na situaci…). Ještě předtím nastavím na ten call BreakPoint. Vykonám opět svou práci a když se zastaví Dbg na callu, na který jsem dal BP, tak ho vytrasuju (F7). Je možné že v i „v" tomto callu, budou další cally, které budeme muset trasovat, a které ne. Uvedu grafický příklad:

```
1.Call
2.Call
     Call
     Call
          Call
               Operace//zde je operace, kterou hledáme
     Call
3.Call
```

Přitom 1, 2 a 3 jsou po sobě jdoucí cally (když přeskočíme [F8] call č.2 vykoná se vlastně celá operace…) Jestli pořád nechápete, vysvětlím to na tomto příkladu, kde si hru crackneme.

Hru natáhněte do Olly, až vám bude „zbývat" 0 spuštění!!!
Tak, teď ji jen tak nanečisto spusťte, a ihned se vám objeví okno, že spuštění už jsou vyčerpané, a po kliknutí na quit hru ukončíte.

Takže natáhneme hru do Olly… a nespouštíme F9!! Jako první akci uděláme krok, tedy F8, hru nebudeme spouštět, celou ji vykrokujeme. Krokujeme až k prvnímu CALLu na adrese 0044BBDE…nic, okno nám nevyskočilo… Takto krokujem, dokud okno nevyskočí… Je to CALL na adrese 0044BC81. Když tento CALL přeskočíme [F8], Objeví se okno. Ale my chceme zjistit kde. Nastavíme tedy na tento call BP a restartujeme aplikaci [Ctrl+F2]…

Stiskneme tentokrát [F9] a Dbg se zastaví na našem zadaném BP. Ten my vytrasujeme [F7]. Opět jen trasujeme a nic nespouštíme… Objevíme se zde:

```
00412C80  /$  8B4424 04     MOV     EAX, DWORD PTR SS:[ESP+4]        ;
3D_Live_.00400000
00412C84  |.  B9 60CA4600   MOV     ECX, 3D_Live_.0046CA60
00412C89  |.  50            PUSH    EAX                              ;
/Arg1
00412C8A  |.  E8 71E4FEFF   CALL    3D_Live_.00401100                ;
\3D_Live_.00401100
00412C8F  |.  85C0          TEST    EAX, EAX
00412C91  |.  7D 05         JGE     SHORT 3D_Live_.00412C98
00412C93  |.  33C0          XOR     EAX, EAX
00412C95  |.  C2 1000       RETN    10
00412C98  |>  B9 60CA4600   MOV     ECX, 3D_Live_.0046CA60
00412C9D  |.  E8 CE04FFFF   CALL    3D_Live_.00403170
00412CA2  \.  C2 1000       RETN    10
```

Můžeme si všimnout jedné věci. Před námi jsou dva CALLy a na konci je RETN (RETURN, vrátí se k tomu CALLu, od kterého „přilétl"). To znamená, že jeden z těchto dvou Callů to musí být (protože za RETN bychom se nedostali…). Krokujeme [F8]ičkou, a zjistíme, že je to hned první call, který volá naše otravné okno. Nastavíme tedy na něj BP a aplikaci opět restartujeme. Stiskneme dvakrát [F9] (nejprve se zastaví na prvním BP a potom na druhém), a trasujeme tento call [F7]. Opět potom krokujeme pomocí [F8] dokud se nezobrazí okno. Na nic se nedívejte, na žádné EAX, EBX, to hoďte za hlavu, prostě klikejte na [F8] dokud se nezobrazí to otravné okno… Dokrokovali jsme nyní na CALL na adrese 004012B6. Nastavíme opět BP, restartujeme aplikaci a 3x stiskneme [F9] a pak [F7] (snad nemusím vysvětlovat proč). Opět krokujeme [F8] dokud se nezobrazí okno. A jsme tu! Dostali jsme se sem:

```
.0040223D  |.  85C0          TEST    EAX, EAX
.0040223F  |.  74 31         JE      SHORT 3D_Live_.00402272 //jestl je
ZF=1
 tak skoč na 00402272
.00402241  |.  8B86 E8A40200 MOV     EAX, DWORD PTR DS:[ESI+2A4E8]
.00402247  |.  8B96 F0A40200 MOV     EDX, DWORD PTR DS:[ESI+2A4F0]
.0040224D  |.  8B8E E4A40200 MOV     ECX, DWORD PTR DS:[ESI+2A4E4]
.00402253  |.  2BD0          SUB     EDX, EAX
.00402255  |.  6A 40         PUSH    40              ; /Flags =
SWP_SHOWWINDOW //zde se volá nějaké okno…
.00402257  |.  52            PUSH    EDX; |Height
.00402258  |.  8B96 ECA40200 MOV     EDX, DWORD PTR DS:[ESI+2A4EC]; |
.0040225E  |.  2BD1          SUB     EDX, ECX        ; |
.00402260  |.  52            PUSH    EDX             ; |Width
.00402261  |.  50            PUSH    EAX             ; |Y
.00402262  |.  8B86 D8A30200 MOV     EAX, DWORD PTR DS:[ESI+2A3D8]; |
.00402268  |.  51            PUSH    ECX             ; |X
```

```
.00402269  |.  6A FE         PUSH    -2; |InsertAfter = HWND_NOTOPMOST
.0040226B  |.  50            PUSH    EAX; |hWnd
.0040226C  |.  FF15 0C824500 CALL    NEAR DWORD PTR DS:[<&USER32.SetW>;
\SetWindowPos
.00402272  |>  8B86 E4A30200 MOV     EAX, DWORD PTR DS:[ESI+2A3E4]
```

Na adrese 0040226C se vám potom otevře okno. Ovšem aby na tuto adresu program nešel, stačí aby na adrese 0040223F by byl ZF=1. Toho docílíme tak, že přehodíme instrukci JE za JNZ. Zkusíme si to patchnout a uložit do executable. A hle! Hra naběhne v pohodě!! Při výrobě patchu v C++ musíme dbát na to, aby před aplikací patchu bylo počet zbývajících runs, nula. Jinak by třeba při zbývajících 7 runs program nenaběhl. Patch bude vypadat asi takto (c++ source):

```
#include "stdafx.h"
#include <stdio.h>
#include <stdlib.h>
#include "Form1.h"
{
FILE *f;


int adr=0x1400B;       //offset
int owr=0x74;        //hex

f = fopen( "3D Live Pool.exe" , "rb+");
if (f==NULL) { MessageBox::Show( "File can not be finded!", "Error",
MessageBoxButtons::OK, MessageBoxIcon::Exclamation );
exit(1);
}


fseek(f,adr,SEEK_SET);


fwrite(&owr,1,1,f);
MessageBox::Show( "Cracked :-)", "Cracked...:-)" ,MessageBoxButtons::OK,
MessageBoxIcon::Exclamation );
fclose(f);
}
```

Ještě malé upozornění:
Když programujete ve Visual C++, musí být v includes „stdafx.h". Třeba Dev-C++ vám nahlásí, že tento soubor nemá v includes. To platí pro všechny Source v C++ zde uveřejněné. Já jsem si crack upravil tak, aby v Title programu zobrazoval místo „3D Live Pool v2.21 - Unregistered" toto: „3D Live Pool v2.21- Cracked by DjH" :-). Source i hra jsou přiložené. Crack odzkoušený, běžíí!
P.S.:Setup hry zabírá pouze 1.46 MB, ale to neznamená, že hra není dobrá! Hra je vynikající, až někdy na tu fyziku…doporučuju ji všem milovníkům kulecu :-)

Na závěr…
Just Enjoy!!

V balíku „balik8.rar" najdete:

- **Hru – 3D Live Pool v2.21**
- **Crack v MSVC++2oo5 [DotNET] (Src+exe)**

Lekce č.9

Tajemství ochran podruhé…

Probrali jsme si jeden typ ochrany, a to typu Nick-s/n.
Nyní si probereme nový typ, a to jen s/n. Nemyslím pevně dané, ale třeba typu ABC-xxxyyy. Jak takové s/n počítání funguje? Jednoduše. Dám jednoduchý příklad. Známe příkaz Ord. Správné s/n bude typu ABC-xxyyyy. Kde xx, jsou dvě náhodná čísla, a první dvě yy, je ord prvního x, a druhá dvojice yy, je druhé ord x. Dáme si příklad: Víme, že se s/n počítá takto, jak jsem napsal. Dejme tomu, že chceme vytvořit třeba keygen. Máme ABC-29yyyy. Provedeme:
Ord(‚2‘) = 50
Ord(‚9‘) = 57
Tedy správné s/n by bylo ABC-295057.
Samozřejmě, pokud chceme ochranu zvýšit, započteme mezi čísla i písmena. Podle stejného vzorce by bylo ABC-sEyyyy = ABC-sE11569. s/n má ale čísel ‚y‘ pět, ale my chceme aby byly jen čtyři. Zkrátíme tedy na ABC-sE1156. Samozřejmě že tato ochrana je primitivní. Je to ale podle mě lepší ochrana než typ Nick-s/n. Pokud děláte tyto ochrany v Delphi, moc je neochráníte, protože Delphi vše ukládá do proc. registrů. Museli byste použít tzv.vložený ASM. A to takto:

```
Asm
;asm prikazy, napr:
Mov eax, ebx
Xor eax, eax
End;
```

Vložený asm, začíná „asm" a končí „end;". Jednoduché viďte. Já osobně ale vložený asm nepoužívám. Moc mi to nejde. Lepší je na ochrany vytvořit DLL knihovnu, anebo celý program psát v C++. V aplikacích na bázi .NET FW, uvidíte v DisAsm taky pěkné prt :-), moc se v těchto aplikacích nevyznám, ani v Olly. Víceméně, vyzkoušejte sami… Ale teď uvedu zde source k dalšímu našemu (mému) CMe, které si crackneme:

```
var
s1,s2,s3,s4,s5,sn:string;
i1,i2,i3,i4,i5:Integer;
begin
s1 :=  Edit1.Text;
if (s1[1] = 'C') and (s1[2] = 'M') and (s1[3] = 'e') and (s1[4] = '-')
and (length(Edit1.Text)=12) then
  begin
    i1:=ord(s1[5]);
    i2:=ord(s1[6]);
    i3:=ord(s1[7]);
    i4:=ord(s1[8]);
    i1:=strtoint(inttostr(i1)[2]);
    i2:=strtoint(inttostr(i2)[2]);
    i3:=strtoint(inttostr(i3)[2]);
    i4:=strtoint(inttostr(i4)[2]);
    if Edit1.Text = 'CMe-
'+s1[5]+s1[6]+s1[7]+s1[8]+inttostr(i2)+inttostr(i4)+inttostr(i3)+inttost
r(i1) then Button4.Click;
  end;
```

Snad chápete, co kontrola provádí. Spustíme si tedy naše CMe, a jako, že jsme tento kód neviděli :-), napíšeme ‚123456' a klikneme na Check, a nic se nestalo. Otevřeme si CMe programem PEiD, a co nevidíme, program je packovaný UPXem. Zkoušíme unpacknout přes PEiD – nejde, zkoušíme přes UPX, nejde. Když se podíváte na jednotlivé sekce, jsou přepsané a vlastnosti characteristics jsou také přepsané. Proto nelze provést UnPack. Zkusíme poslední možnost, a to je Generic Unpacker. Bomba! Už to jde! :-)

Nyní jsme si ukázali perfektní ukázku crackování packovaného programu…Když program přetáhneme do eXeScope, tváří se pořád jako packovaný. Zkusíme program natáhnout do DeDe, ať se podíváme, co se děje při kliknutí na „check"… Program ohlásí „Dump sucessfull" ale v Procedures nic není! Bude to složité… Natáhneme si program do Olly a projedeme si kód. Stringy většinou bývají až na konci, tak se díváme… Nic na konci není. Proč? Protože jsme soubor unpackli a dokonce Genericky, takže kód nemusí být originální, a také, že není. Našli jsme stringy „CMe-" a „Registration completed sucessfully! :-)"… Podíváme se na to druhé, a stopujeme pozpátku (program nezapínáme), hledáme „to pravé místo pro BP". Bohužel jsme dostopovali asi jen dva řádky nahoru, našli jsme tam totiž NOP a RETN. Nastavíme tedy BP na NOP (0046C773) a na MOV (0046C774) a zkusíme zjistit, odkud sem skáče. Spustíme program, napíšeme blábol, stiskneme „Check"

a…nic…jakto? Protože jsme nastavili BP na funkci, která probíhá po SPRÁVNÉM zaregistrování, což my jsme neudělali. Co teď? Najdeme si ten string „CMe-" a krokujeme nahoru. Někde by měla být směsice nesmyslných dat, ale my si nastavíme BP předtím. Poslední DB je na adrese 0046C4A9. Potom následuje MOV. Nastavíme si BP na adresu 0046C4B9 (je tam příkaz JNZ) a zkusíme…a hle! Zastavilo se to tam… Vidíme že je to nějaký cyklus, a tak vyčkáme než přestane. Nyní se už jen soustředíme na nalezení s/n… krokujeme, a co nevidíme…:

```
0046C4DF   .  8038 43       CMP     BYTE PTR DS:[EAX], 43  //první znak
musí být C (43 hex = 67 dec a to je písmeno „C" (Ord(,C') = 67)
0046C4E2   .  0F85 AC010000 JNZ     cme_exe_.0046C694
0046C4E8   .  8B45 FC       MOV     EAX, DWORD PTR SS:[EBP-4]
0046C4EB   .  8078 01 4D    CMP     BYTE PTR DS:[EAX+1], 4D  //Druhé
písmeno musí být „M"
0046C4EF   .  0F85 9F010000 JNZ     cme_exe_.0046C694
0046C4F5   .  8B45 FC       MOV     EAX, DWORD PTR SS:[EBP-4]
0046C4F8   .  8078 02 65    CMP     BYTE PTR DS:[EAX+2], 65  //třetí „e"
0046C4FC   .  0F85 92010000 JNZ     cme_exe_.0046C694
0046C502   .  8B45 FC       MOV     EAX, DWORD PTR SS:[EBP-4]
0046C505   .  8078 03 2D    CMP     BYTE PTR DS:[EAX+3], 2D  //čtvrtý
znak musí být pomlka „-"
0046C509   .  0F85 85010000 JNZ     cme_exe_.0046C694
0046C50F   .  8D55 F0       LEA     EDX, DWORD PTR SS:[EBP-10]
0046C512   .  8B83 F8020000 MOV     EAX, DWORD PTR DS:[EBX+2F8]
0046C518   .  E8 CBB6FCFF   CALL    cme_exe_.00437BE8
0046C51D   .  8B45 F0       MOV     EAX, DWORD PTR SS:[EBP-10]
0046C520   .  E8 E77BF9FF   CALL    cme_exe_.0040410C
0046C525   .  83F8 0C       CMP     EAX, 0C  //počet znaků musí být 12
0046C528   .  0F85 66010000 JNZ     cme_exe_.0046C694
```

Jinak nás to automaticky háže na Badboy. Nastavíme: CMe-56789012 (max. poč. znaků = 12). Tamějším testem

projde v pohodě, a soustředíme se opět na nalezení s/n.

Narazíme na toto:

```
0046C67E   .  E8 D57BF9FF   CALL    cme_exe_.00404258
0046C683   .  75 0F         JNZ     SHORT cme_exe_.0046C694
```

Kde v CALLu se porovnává zadané s/n se správným a jnz už jen skočí buď na badboye, nebo ne. Když call vytrasujeme, můžeme si na adrese 0040425F v EDX najít správné s/n. Tedy „CMe-56784653". „CMe-5678xxxx" jsme napsali pamatujete? Jen se vypočítaly poslední 4 čísla. Zadáme s/n „CMe-56784653", a jde to :-). Zkusíme si zadat s/n „CMe-DjH4xxxx", a nastavíme BP na CALL, který volá porovnávání, abychom si hned přečetli správné s/n… Správné s/n pro prvních 8 znaků (CMe-DjH4) je „CMe-DjH40228"… V balíčku najdete Source, CMe (originální), CMe (Packované) a CMe (Genericky UnPackované) + source a exe ke keygenu.

V balíku „balik9.rar" najdete:

- **DjH's CMe v3 [ExE + Src]**
- **Keygen for DjH's CMe v3 [ExE + Src]**

Lekce č.10

Keygenning…

Než jsem došel alespoň na úplné základy keygenningu, dalo mě to moc a moc práce. Velkou pauzu mezi lekcí 9 a touto, zapříčinil i začátek nového školního roku, kde se na nás usmívají naše milé paní učitelky, s ještě hodnějšími pány učiteli. Promiňte mi hrubky jestli nějaké jsou… Ale nebudeme odbočovat…

Keygenning mě hodně vzal… jednak se mi zdá asi vrcholem samotného RE, protože vlastně musíme opravdu porozumět kódu, který vidíme… Tudíž se naučíme pár nových příkazů:

IMUL EAX,EAX, 5 – Vynásobí EAX * 5 a uloží do EAX

SUB EAX, 5 – Zmenší EAX o 5 (Eax := Eax – 5)

ADD EAX, 5 – Zvětší EAX o 5 (Eax := Eax + 5)

SHL EAX, 5 – Posune EAX o pět číslic do leva… je to těžké vysvětlit, ale dám příklad: SHL EAX, 5… Eax = 12345678 a po operaci bude 67800000

SHR EAX, 5 – Posune EAX o pět číslic doprava… je to těžké vysvětlit, ale dám příklad: SHR EAX, 5… Eax = 12345678 a po operaci bude 00000123

Jestli nechápete, nebojte, až na ně narazíte v Olly, pochopíte to :-)

A víte co? Pojďte si hnedka jedno CMe keygennout :-). Stáhl jsem ho z www.crackmes.de a obtížnost je nejlehčí. Keygenovat fakt ještě, dalo by se říct, skoro vůbec neumím :-(, ale naučím se to :-)…

A jupí to cracknout :-)
……
Takže CMe 3 od kaiZera si otevřeme v PEiDu. Naskočí nám, že je to made in Delphi. Takže tentokráte vynecháme DeDe, a jdeme na Olly :-)

V Olly vyhledáme stringy a úplně dóle najdeme zajímavé stringy… Třeba „Solved…"… A mrkáme nahoru… V Delphi většinou procedury začínají „Pushem"…ale na to pak přijdete :-)… Takže nalezli jsme určitě adresu…

```
00458814   .  55            PUSH    EBP
```

To je ten první PUSH, tam začíná procedura. My si ale nastavíme BP na první Call (00458840) a spustíme si CMe a dáme jako jméno třeba „DjH2oo7" a s/n „1234567890" a klikneme na „OK". Olly se nám zastaví na callu, který přeskočíme a krokujeme dál…

Vidíme toto:

```
00458848   .  E8 87B9FAFF   CALL     CrackMe_.004041D4
0045884D   .  8BF0          MOV      ESI, EAX
0045884F   .  83FE 01       CMP      ESI, 1 //zjistime, zda-li
jmeno obsahuje aspon 1 znak
00458852   .  7D 12         JGE      SHORT CrackMe_.00458866
00458854   .  33D2          XOR      EDX, EDX
00458856   .  8B83 44030000 MOV      EAX, DWORD PTR DS:[EBX+344]
0045885C   .  E8 BB9EFDFF   CALL     CrackMe_.0043271C
00458861   .  E9 09010000   JMP      CrackMe_.0045896F
00458866   >  83FE 63       CMP      ESI, 63 //maximalne muze mit
63h znaku
00458869   .  7E 12         JLE      SHORT CrackMe_.0045887D
0045886B   .  33D2          XOR      EDX, EDX
0045886D   .  8B83 44030000 MOV      EAX, DWORD PTR DS:[EBX+344]
00458873   .  E8 A49EFDFF   CALL     CrackMe_.0043271C
00458878   .  E9 F2000000   JMP      CrackMe_.0045896F //Pod
timto zacina vypocet… (vsimnete si, jak porovnavane cislo, kolik
znaku obsahuje nick, bylo v ESI, nyni tedy pracujeme s poctem
pismen v nasem zadanem nicku…)
0045887D   >  6BFE 75       IMUL     EDI, ESI, 75  //Do Eax uloz
Esi * 75h (Eax := Esi *5) (Esi := Length(nick.text))
00458880   .  81C7 3E150000 ADD      EDI, 153E //Edi := Edi+153Eh
00458886   .  81EF 74150000 SUB      EDI, 1574 //Edi := Edi -
1574h
0045888C   .  8BC6          MOV      EAX, ESI  //Eax := pocet
pismen v zadanem nicku
0045888E   .  83E8 22       SUB      EAX, 22  //Eax := Eax - 22h
00458891   .  69C0 F0110000 IMUL     EAX, EAX, 11F0  //Eax := Eax
* 11F0h
00458897   .  03F8          ADD      EDI, EAX //Edi := Edi + Eax
(Edi z minula :-))
00458899   .  81C7 4C520E00 ADD      EDI, 0E524C  //Edi := Edi +
E524Ch
```

Cracking 4 newbies…

```
0045889F   .  68 BC894500   PUSH    CrackMe_.004589BC
;  ASCII "668r9\\5233"
004588A4   .  8D55 EC       LEA     EDX, DWORD PTR SS:[EBP-14]
004588A7   .  8BC7          MOV     EAX, EDI
004588A9   .  E8 C6F5FAFF   CALL    CrackMe_.00407E74 //Pricist
na zacatek "668r9\\5233" ("668r9\\5233" + Edi)
004588AE   .  FF75 EC       PUSH    DWORD PTR SS:[EBP-14]
004588B1   .  68 D0894500   PUSH    CrackMe_.004589D0
004588B6   .  68 DC894500   PUSH    CrackMe_.004589DC
;  ASCII "k329[43}"
004588BB   .  8D45 F0       LEA     EAX, DWORD PTR SS:[EBP-10]
004588BE   .  BA 04000000   MOV     EDX, 4
004588C3   .  E8 CCB9FAFF   CALL    CrackMe_.00404294 //Pricist
na konec "k329[43}" ("668r9\\5233" + Edi + "k329[43}")


…


004588D6   .  85F6          TEST    ESI, ESI
004588D8   .  0F8E 91000000 JLE     CrackMe_.0045896F
004588DE   .  BF 01000000   MOV     EDI, 1
004588E3   >  8B45 FC       MOV     EAX, DWORD PTR SS:[EBP-4]
004588E6   .  0FB64438 FF   MOVZX   EAX, BYTE PTR DS:[EAX+EDI-1]
//První znak z nicku do eax
004588EB   .  8945 F4       MOV     DWORD PTR SS:[EBP-C], EAX
004588EE   .  8D4D F8       LEA     ECX, DWORD PTR SS:[EBP-8]
004588F1   .  BA 02000000   MOV     EDX, 2
004588F6   .  8B45 F4       MOV     EAX, DWORD PTR SS:[EBP-C]
004588F9   .  E8 8AF6FAFF   CALL    CrackMe_.00407F88
004588FE   .  8D55 E4       LEA     EDX, DWORD PTR SS:[EBP-1C]
00458901   .  8B83 44030000 MOV     EAX, DWORD PTR DS:[EBX+344]
00458907   .  E8 E09DFDFF   CALL    CrackMe_.004326EC
0045890C   .  FF75 E4       PUSH    DWORD PTR SS:[EBP-1C]
//Pushne si aktualni s/n ("668r9\\5233" + Edi + "k329[43}")
0045890F   .  FF75 F8       PUSH    DWORD PTR SS:[EBP-8]
//Pushne si ten první znak (resp. jeho hex. Hodnotu)
00458912   .  68 F0894500   PUSH    CrackMe_.004589F0
```

```
00458917   .  8D45 E8       LEA     EAX, DWORD PTR SS:[EBP-18]
0045891A   .  BA 03000000   MOV     EDX, 3
0045891F   .  E8 70B9FAFF   CALL    CrackMe_.00404294 //Pricte
se to aktualni s/n + ta hodnota prvniho pismena z nicku + znak
„$"
00458924   .  8B55 E8       MOV     EDX, DWORD PTR SS:[EBP-18]
00458927   .  8B83 44030000 MOV     EAX, DWORD PTR DS:[EBX+344]
```

Potom ještě dělá CMe cosi se s/n (přičítá další hex hodnoty písmen…) ale bylo na crackmes.de, že jde o autorův nechtic bug.

Myslím, že by to mělo být všecko potřebné, co jsme chtěli vědět :-)

Můžeme jít na Keygen… a aby to nebylo jen tak, tak Vám zkusím přiblížit Delphi v API f-cích. Jde o to, že se do výsledné exace, nelinkuje VCL (kvůli které má výsledné exečko kolem 400 kB). Ale počítejte s tím, že je to o to těžší…

Vypíšu tedy celý popsaný zdroják (je to jen jeden soubor :-) (+ rsrc.res ale to je v balíčku :-))

```
program KeyGen;

uses Windows, Messages,sysutils,BeRoXM; //BeRoXM - this is pckg, where
are components for playing music in *.xm. The format use a few keygens,
cracks and trainers. Pckg is included v balí čku :-D

{$R rsrc.res}

const
id_name  = 102;    //id of EditBox "Name"
id_sn = 103;       //id of EditBox "Serial no"
id_gen = 101;      //id of Button  "GEN!"

var
  mWnd: HWnd;      //HWND
  i:    integer;   //i := Length(id_name)
  sn2, sn3:   integer;   //sn :-)
  XM:TBeRoXM;

function GetText: string;      //id_name.text
begin
  i := SendDlgItemMessage(mWnd, id_name, WM_GETTEXTLENGTH, 0, 0);
  SetLength(result, i);
  i := GetDlgItemTextA(mWnd, id_name, pchar(result), i + 1);
  SetLength(result, i);
end;

function sn: string;        //Computing s/n
begin
gettext; //get "i" :-)
if (i >= 3) then
  begin
    sn2:=0;
    sn3:=0;
  sn3:=i;
  sn3:=sn3 * $75;
  sn3:=sn3 + $153e;
  sn3:=sn3 - $1574;
  sn2:=i;
```

```
  sn2:=sn2 - $22;
  sn2:=sn2 * $11f0;
  sn3:=sn3 + sn2;
  sn3:=sn3 + $0e524c;
  sn := '668r9\5233' + IntToStr(sn3) + '-k329[43}' +
inttohex(ord(GetText[1]),2) + '$';                    //:-)
  end else
  begin
    sn := 'Input at least 3 chars!!!';
  end;
end;

function engine(wnd, wmsg, wParam, lParam: dword): dword; stdcall;
begin
 mWnd:=wnd;

  case wmsg of

    WM_COMMAND:
      case loword(wparam) of
      id_gen: SetDlgItemTextA(mWnd, id_sn, Pchar(sn));   //Jestli je
stisknuto tlacitko, vykonej funkci sn a nastav ji jako "id_sn.text"
      end;

    WM_INITDIALOG:
      begin
        Randomize;
        SetDlgItemTextA(mWnd, id_name, 'DjH2oo7');  //Implicitni…
        SetDlgItemTextA(mWnd, id_sn, '668r9\5233815353-k329[43}44$');
//…Hodnoty
        SetFocus(mWnd);
      end;

    WM_CLOSE:
      begin
        EndDialog(wnd, 0);
        MessageBoxA(mWnd,pchar('Hi all rockers!!!' +#13+ 'Keygen
programmed by: DjH2oo7' +#13+ 'Relase: 9.September
2oo7'+#13+#13+#13+'Thx to kaiZer-by for CrackMe =)!!'), pchar('Keygen by
DjH...'), 0); //When we click „X" button :-)
        result := 1
```

```
      end;
  end;
end;

begin
 XM:=TBeRoXM.Create(41000,2048,4,TRUE,10);            //Music...
 XM.ResamplingMethod:=BeROXMMixerWindowedFir;
 XM.Clipping:=TRUE;
 XM.Module.MasterVolume:=256;
 XM.Module.LoadFromResource(hInstance,'MSC',RT_RCDATA);
 XM.Play;

 DialogBox(hinstance, pchar(200), 0, @engine);   //Application.run
end.


//Sorry for my English :-(
```

Komponenta pro přehrávání je jen „na okrasu" :-), Keygeny ji mívají. Tak jsem Vám chtěl přinést tajemství, kterým se to dělá (googlil jsem kvůli tomu asi 2 hodiny…).
Hudbu uložíte do resource (třeba pomocí ResHackeru) pod jménem „MSC" a funkcí
`XM.Module.LoadFromResource(hInstance,'MSC',RT_RCDATA);`
ji loadnete.
To byla taková třešnička na dortu (bohužel to moc zvětší binárku (cca o 100 kB) ale UPX to napraví (asi na 56 kB)). Můžete ji vymazat z uses a ty příkazy před DialogBoxem a binárka bude mít 56 kB samotná (po UPX zásahu, i 20kB :-))…
V balíku, jak je již řečeno, je CMe + Keygen + Sources na keygen + komponenta XM
Příště bych se chtěl mrknout na ochranu proti crackování, a vytvořím CMe, pokusíte se ho překonat

:-). Pokusím se přiblížit i pár antidebugových triků…

V balíku „balik10.rar" najdete:

- **kaiZer's CMe**
- **Keygen for kaiZer's CMe by DjH [exe+src]**
- **BeRoXM - komponenta pro Delphi, pro přehrávání XM Tracků**

Lekce č.11

Opět chráníme náš software…

Takže teď bych chtěl uvést pár antidebugových triků, které jsem za poslední dobu zatím stihl postřehnout.

1. **Ten první je, použít API funkci „IsDebuggerPresent". Bohužel proti tomuto je snadná obrana :-(, stačí mít v OllyDebuggeru plugin (je jich spousta, třeba „IsDebugPresent" nebo „Hide Debugger"), a když narazíte na program, který tuto f-ci používá, a nějak to neřešíte, nepostřehnete ani, že tam takováto funkce je… Ale i přes to, ji používám :-).**
2. **Druhá možnost je, na kterou jsem přišel já, sám :-), nevím jestli je využívaná, nebo nějak extra známá… Ale i přes to Vám to povím. Jde o to, že zkusíme najít okno podle Title, třeba „OllyDbg". To má OllyDebugger pořád v Captionu okna. Na oba typy ochran Vám ukážu source (pro Delphi)…**

IsDbgPresent:

```
function IsDebuggerPresent : boolean; stdcall; external kernel32 name
'IsDebuggerPresent';


function checkdbg : boolean;
begin
if IsDebuggerPresent = true then
begin
  asm
      CALL ExitProcess
  end;
end;
end;
```

Pro ověření, jestli je program debugovaný, zavolejte funkci „CheckDbg".

Podle Title okna:

```
var
  item:string;
  hwndShell: HWnd;

begin
  HWNDShell:=GetWindow(Handle,GW_HWNDFIRST);  //Zacneme hledat okna…
    repeat
      SetLength(item,255);
      HWNDShell:=GetNextWindow(HWNDShell,GW_HWNDNEXT);  //Zacneme hledat
okna…
      GetWindowText(HWNDShell,Pchar(item),8);   //Zjistime prvnich 7
pismen…
        if  length(string(Pchar(item)))>0 then
          begin
```

```
            if (String(Pchar(item))) = 'OllyDbg' then begin
application.Terminate; end; //Jestli je tech 7 pismen rovno stringu
„OllyDbg" ukonci program…
          end;
    until HWNDShell=0;
end;
```

Aby nebyl string viditelný, je opět lepší jej alespoň trochu zneviditelnit, třeba pomocí funkce Char, a vypsat všech 7 písmen pomocí Char(x) + Char(x+1) … Promiňte, teď se mi to nechce převádět, alespoň máte co na práci pokud se vám to zalíbí :-)… Háček je v tom, že stačí mít Ollyho jen zapnutého, a program se ukončí. Takže mu je jedno jestli je debuggovaný nebo ne… Ale to si myslím ani tak nevadí.

Tolik k antidebuggingu v Olly, ještě se pokusím přijít na to, jak v Delphi udělat SMC (Self-Modyfing Code), které je velmi působivé :-)… (btw: místo „OllyDbg" můžete nahradit (přidat) jiný string (Soft-Ice)… ten ale nepoužívám, takže nevím přesně, jaký má caption :-))

Jak ochránit s/n, aby nebylo jen tak čitelné v paměti??
Jde o to, kontrolovat jeden znak po druhém, a ještě lépe, na přeskáčku, a pokud je v nějakém písmeně zádrhel, okamžitě skočit na BadBoi. Asi bych to opět měl popsat na ukázce (VCL, Delphi…):

```
if sn.Text[1]    =  inttostr(length(nick.Text)) then
if sn.Text[3]    =  inttostr(ord(sn.text[5]))[2] then
if sn.Text[7]    =  char(77 xor 96) then
if sn.Text[10]   =  b then
if sn.Text[aa]   =  inttostr((length(nick.Text)-1) xor ord(sn.text[2])
xor ord('D') * (length(nick.Text)-1) xor ord(sn.text[2]) xor
ord('D'))[2] then
if sn.Text[cc]   =  inttostr(aa - ord(b[1]))[length(inttostr(aa -
ord(b[1])))] then
 begin
       showmessage('Registrováno úspěšně...');
  end;
end;
```

Proměnné (b, aa…) si můžete udělat podle svého algoritmu, dávejte ale pozor, aby se Vám dvakrát nekontrolovalo jedno písmeno! Potom by bylo CME neřešitelné (vyřešil by to jen patch). Třeba aby sedmé písmeno nebylo `char(77 xor 96)` **(pomlka „-") a zároveň by se pak kontrolovalo, jestli není sedmé písmeno třeba „8" (TŘEBA..! :-)). To je velká chyba pak. Ještě bych chtěl doplnit, že na tyhle typy „CheckSn" je docela těžké udělat keygen (myslím že ze zdrojového kódu to jde vidět :-)). Třeba v Delphi byste si museli pohrát s Arrayem…**
Nemám moc času, ukončím to dnes dříve, možná postupem času článek dopíšu, nebo vydám v 12té lekci pokračování na toto téma. Snad jste vše teda pochopili, a jestli ne, tak se moc omlouvám :-(,

ale pokud se vyznáte v Delphi, pochopit by to neměl být problém…

P.S. Na www.crackmes.de mám (zatím v téhle době) 2 CMe, pokuste se je prosím vyřešit, jsou i zde v balíku, tak kdyžtak na mail, nebo sem do komentářů, můžete přidávat komenty, popřípadě chyby. Solutions bych radši až v tom e-mailu. Uveřejňovat je zde nebudu (možná časem…:-))…

V balíku „balik11.rar" najdete:

- **src pro Delphi – příklady antidebuggingu v Delphi**
- **Oficiální CMe1 a CMe2 by DjH2oo7**

Lekce č.12

Úplné začátky MASM32…

Předem bych chtěl říct, že v assembleru jako takovém (MASM32) umím pouze začátky, a měli by je pochopit také začátečníci. Assembler jsem se naučil, protože se v něm lépe dělají keygeny, a takové různé jednoduché sarapetičky a výsledná binárka je hodně malá =). Používám na zvýraznění syntaxe program WinAsm Studio v 5.1.1.0, který je mimochodem také celý naprogramovaný v MASMu. Chci Vám – neznalým - tedy tak nějak poradit, jak se alespoň minimálně naučit tento programovací jazyk (MASM32)… Nejdůležitější je, naučit se všechny, nebo nejhlavnější API funkce, jako je třeba GetDlgItemText, SetDlgItemText, MessageBox, ShellExecute, CloseHandle… Je jich opravdu moc, takže se naučte tyto základní. Pokud máte Delphi 7, můžete najít v helpu „Windows SDK", a tam naleznete všechny API funkce (btw: píšou se tam i Win bugy, je to zajímavá příručka =))… Je možné, že ji na internetu seženete samostatně, ale vzhledem k tomu, že to Delphi už mám, nebudu Vám vypisovat externí linky =)…

Tak jdeme do samostatného programování…
Nainstalujte si WinAsm(freeware), který je v balíku (pozn: potřebujete i kompletní balík MASM!

www.masm.com), a klikněte na „File" -> „New Project", dále na Tab „Dialog" -> „Base", a hurá, probili jste se k základnímu rozvrhu =), výhoda je, že WinAsm obsahuje *.rc editor, tudíž můžete upravovat okna rovnou v něm…

Většina WinAPI funkcí má několik parametrů, které se do každého programovacího jazyka zadávají jinak, ale většinou do závorek (např.:
MessageBox(0, pchar(‚Text'), pchar(‚Caption'), nil);
)
Ale jako výstup (dissasemble) můžete vidět, že tyto hodnoty jsou „pushovány" a to odzadu (z minula, v pořadí :„nil, caption, text, 0"), a na konci je CALL, který volá funkci ze systémové knihovny (kernel32, user32…).
Například v NASMu byste tyto funkce museli pushovat, ale MASM má takové „makro"… Místo závorky, zde použijete funkci „invoke"… Nejlepší je to ukázat na ukázce:

```
.data
sText db "Text",0
sCapt db "Caption",0

.code
invoke MessageBox, NULL, addr sText, addr sCapt, NULL
```

Kde:
MessageBox: API funkce
NULL : Handle okna, NULL by mělo být hlavní okno…
Addr sText: odkazuje na adresu, kde je string

Addr sCapt: to stejné (string musí být ukončen hexnulou!)
NULL : typ MsgBoxu, jakou bude mít ikonu a jaká tlačítka, najdete je ve WinApi, mohou se kombinovat znaménkem „+", třeba: MB_YESNOCANCEL + MB_ICONSTOP

Za „.data" se ukládají různé stringy atp… dalo by se říct, že to jsou jakési proměnné… Pokud chcete udělat „pole" které bude mít 256 bytů vyhrazených pro sebe, nemusíte psát doaleluja

```
sString db 0,0,0,0,0,0,0……
```

i na to má MASM „makro"…

```
sString      db 256 dup (0)
```

kde 256 znamená „kolik" a (0) znamená „čeho" v hexčísle…
Tedy funkci invoke, můžete používat jako v ostatních jazycích, akorát místo závorky, napíšete

```
invoke API_FCE, par1, par2, parX ;par=parametr
```

A to je vlastně to nejhlavnější…invoke…
…Po troše zkoumání jsem zjistil, že jdou zde dělat poměrně těžší CMe než v jiných jazycích =)… Všechno se lépe pochopí na příkladu (pěkně okomentovaném =)), takže Vám vypíšu moje naprogramované CMe, i s keygenem (vše v MASM)…kompletní WinAsm projekt najdete v balíku…

```
.486
.model flat, stdcall          ;model...
option casemap :none          ; case sensitive

include        base.inc       ;includovano(najdete v baliku =))


.data
sZadano db 256 dup (0)        ;opakovat 256x (na adresu zapsat "0") tzn,
vypsat 256 hexanul
str1    db      "XoR!",0      ;promenna str1…
str2    db      "BadBoi!",0
str3    db      "GoodBoi! Plz send sltion!",0

.const          ;konstanty...„promenne"(nejsou to promenne, jsou to
takove…identifikatory =))...treba nazev Itemu za jeho ID

IDC_SN          equ 1002
IDC_NAME        equ 1001



.code                         ;cast pro kod...
;-------------------------------------------------------------------
;Dalsi "CALLinY"

hash:
       cmp eax, 8     ;Funkce GetDlgItemText vraci v EAX Length z EDITu
       jnz konec2     ;tedy pokud nemá SN 8 znaku, skakej na badboi
       mov eax, dword ptr ds:[sZadano]  ;první 4 znaky ze jmena do EAX
(pozadu, hex hodnoty pismen, eax je 32bitovy, to znamena ze se do nej
vleze (32/8) 4 byty (znaky…))
       cmp eax, 2D525450h ;První 4 znaky musí byt „PTR-" (pozadu, tedy
50h je „P" 54h je „T" 52h je „R" a 2D je „-")
       jnz konec2     ;jestli nejsou první 4 znaky „PTR-" skakej na
badboi
       xor ecx,ecx    ;vycisti registry (priprava na operaci…)
       xor edx,edx
       mov cx, word ptr ds:[sZadano+4]  ;do cx dej 2 byty ze zadaneho SN
(od 4teho znaku, tedy PTR-XXyy, a XX je v CX) (proc 2? Protože do CX se
vic nevejde, je dvoubytovy)
```

```
        mov dx, word ptr ds:[sZadano+6]  ;do dx dej 2 byty ze zadaneho SN
(od 6teho znaku, tedy PTR-XXyy, a yy je v DX) (proc 2? Protože do DX se
vic nevejde, je dvoubytovy)
        xor cx, word ptr ds:[str1]  ;vyxoruj první dva byty ze zadaneho SN
(cx) s „Xo" (proc Xo? Protože nemuzeme porovnavat db s dw, a xoruje se
s hodnotou která je rovna opacnemu poradi hex hodnot těchto pismen)

        cmp cx, dx      ;vyxorovana hodnota se musí rovnat poslednim dvoum
znakum v SN (tedy DX)
        jnz konec2      ;jestli se nerovna, badboy
        mov eax,22h     ;jen takove identifikovani, jestli je GoodBoi ci
ne (nahraska za Boolean ;))

        xor ebx,ebx     ;a vycistime…
        xor ecx,ecx
        xor edx,edx
        jmp konec3      ;a GoodBoi
konec2:
        mov eax,99h     ;id - BadBoi =)
konec3:
        xor ebx,ebx     ;cisteni…
        xor ecx,ecx
        xor edx,edx
        cmp eax,22h     ;nastaveni ZF, jestli je Goodboi nebo ne,
porovnavat se bude, az se vyjede z CALLu =)
ret     ;"vyjeti" z CALLu

start:                                  ;label, pro start... OEP
        invoke  GetModuleHandle, NULL   ;Vytvoreni okna…
        mov     hInstance, eax          ;instance
        invoke  DialogBoxParam, hInstance, 101, 0, ADDR DlgProc, 0   ;na
adresu DlgProc uloz data, pac se tvori dialog s ID 101 =)
        invoke  ExitProcess, eax        ;Když vyjedeme z obsluhy okna,
ukonci se app
; -------------------------------------------------------------------------
DlgProc proc    hWin    :DWORD,                 ;Zpracovani dialogu...
                                uMsg    :DWORD,
                                wParam  :DWORD,
                                lParam  :DWORD

        .if     uMsg == WM_COMMAND      ;Jestli bylo "neco" stisknuto
```

```
                .if     wParam == IDC_OK        ;jestli "Check!"
; ------------------------------------------------------------------------
;                       Po kliku na OK
; ------------------------------------------------------------------------
                invoke GetDlgItemText, hWin, IDC_NAME, ADDR sZadano, 256
        ;nastavi do addr sName string z IDCNAME  ;"vezmeme" si string ze
SN a ulozime na adresu sZadano…

                                call hash       ;hlavni fce, kde se
porovnava SN (viz vyse ;))


                                jnz badboy      ;jestli není ZF, badboi

                                invoke SetDlgItemText, hWin, IDC_SN   ,
ADDR str3                       ;nastavi SNtext na TestString ;nastaveni
druheho editboxu na zpravu o GoodBoiovi

                                jmp konec1      ; skoc na konec1

                                badboy: ;nastaveni editboxu na zpravu o
badboiovi
                                invoke SetDlgItemText, hWin, IDC_SN   ,
ADDR str2                       ;nastavi SNtext na TestString
; ------------------------------------------------------------------------
        .elseif         wParam == IDC_IDCANCEL          ;jestli "x!t"
                        invoke EndDialog,hWin,0         ;tak ukonci pomoci
api EndDialog
                .endif
        .elseif uMsg == WM_CLOSE                        ;jestli byl stisknut
krizek
                invoke  EndDialog,hWin,0                ;tak ukonci pomoci
api EndDialog
        .endif

        konec1:
        xor     eax,eax         ;vycistit eax
        ret     ;a cyklus obsluhy okna
DlgProc endp
end start       ;konec labelu start
```

Žádné vykecávání, pojdme si to cracknout ;)
Zkusíme, co nám ukáže Olly…
Na adrese 0040109A se volá GetDlgItemText a hned pod touto adresou se volá „hashovací" funkce… Dáme si BP na adresu 0040109A a schválně co to udělá…
Jako blbí dáme jako s/n 123456 a skončí nám to na cmp eax,8, takže jakože už chytřejší zadáme „PTR-1234" a krokujeme… 0040109F -> tracujeme…

```
00401000  /$  83F8 08      CMP     EAX, 8
00401003  |.  75 37         JNZ     SHORT base.0040103C
00401005  |.  A1 00304000   MOV     EAX, DWORD PTR DS:[403000]
0040100A  |.  3D 5054522D   CMP     EAX, 2D525450
0040100F  |.  75 2B         JNZ     SHORT base.0040103C
```

Přes toto nás to už pustí ;)

```
0040102A  |.  66:3BCA      CMP     CX, DX
```

Zde nám vyšlo v CX „5D69" což je 69;5D; což je „i]"…
Tedy správné s/n by mělo být „PTR-12i]"…
Zjistili jsme, že s/n je závislé na prvních dvou znacích, a z nich se „vypočítavají" poslední dva znaky které se porovnávají. Tato „s/n" funkce je malá, mohli bychom ji přepsat celou do MASM, ale proč se namáhat, máme přece zdroje k Cme =), ale ty Vám normálně v životě nebudou k dispozici…

Takže…co víme? Víme to, že na adrese 0040109F se volá „s/n" funkce a na 0040104A skáče z5…

I když máme zdroje k Cme, nebude jen tak jednoduché keygen vytvořit. Zaprvé, doteď nevím, jak v MASM

vytvořit náhodné číslo, a za druhé v CMe to sériové číslo není v paměti jen tak čitelný string, tudíž to bude o to těžší. Dá se to udělat tak, že budu mít už řetězec ‚PTR-' a k němu připojím jakékoliv náhodné dvojčíslí (prostě dva znaky =)) a z těch „vyrobím" ty poslední dva znaky a celé to sloučím dokupy =), snad jste to pochopili, jestli ne, zde je kousek zdroje:

```
.data
sZadano db 256 dup (0)
str1    db      "XoR!",0
……
.code
hash:
        mov cx, word ptr ds:[sZadano]
        xor cx, word ptr ds:[str1]
        mov ax,cx

        mov cx, word ptr ds:[sZadano]
        mov dword ptr ds:[sZadano+96] , '-RTP' ;musí byt pozpatku
        mov  word ptr ds:[sZadano+100], cx
        mov  word ptr ds:[sZadano+102], ax
ret
……
call hash

invoke SetDlgItemText, hWin, IDC_SN   , ADDR sZadano+96
        ;nastavi SNtext
invoke SetDlgItemText, hWin, IDC_SN2   , ADDR sZadano+102
        ;nastavi SNtext
……
```

Trochu jsem si Cme předělal no =), KeyGen funguje tak, že uživatel zadá jakékoliv dva znaky, ze kterých se vygenerují poslední dva znaky ;)
Snad jste aspoň trošku pochopili, jak MASM pracuje, v balíku najdete source k Cme i KeyGenu a WinAsm studio, který má už v sobě pár examples…tak to snad pochopíte alespoň z těch…

V balíku „balik12.rar" najdete:

- WinAsm v 5.11
- CMe + Keygen v MASM32

Lekce č.13

Výroba patchů podruhé - Delphi…

Proč se znovu hrabat ve výrobě cracků? Protože tedka Vám ukážu, jak naprogramovat crack v Delphi ;)

V C++ jsou funkce – fopen, fwrite, fseek a fclose…
V Delphi jsou f-ce- FileOpen, FileWrite, FileSeek a FileClose…
Je to velmi podobné… Ale teď, jak to použít? Skoro stejně jako v céčku… tedy syntaxe je takováto:

```
Procedure CrackIt;
Var
FileHandle, Address, BytesToWrite, NbrOfBytes:integer;
FlName:string;

begin
    FileHandle := FileOpen (FlName, fmOpenReadWrite);
//Otevrit soubor, a nastavit FileHandle jako handle souboru…
                  FileSeek (FileHandle, Address, 0);
//Presun se na offset(address) od zacatku (poslední parametr (0)…0=from
begin; 1=current position; 2=end; (2 se pouziva vetsinou pro zjisteni
velikosti souboru
                  FileWrite(FileHandle, BytesToWrite, NbrOfBytes);
//Prepis byty, NbrOfBytes je, kolik bytu se ma zapsat…
                  FileClose(FileHandle);
End;
```

Takže tohle byl podprogram „CrackIt", (PS:nemusí to být Procedura, může to být i Function ;)), Zavoláme ji „příkazem" „CrackIt;"…

Musíme ale nadefinovat ještě proměnné… Když budeme chtít cracknout naše (jejich =)) CrackMe z lekce 1 a 2, bude to vypadat takto:

```
//#####################################################################//
  Address     :=$585;         //Hexadecimalni offset ($ znaci hex)
  BytesToWrite:=$74 ;         //Hexadecimalni instrukce k zapsani
  NbrOfBytes  :=1   ;         //Pocet bytu...
  FlName      :='crackme.exe';//Implicitni filename...
//#####################################################################//
  CrackIt;   //Start procedury (funkce)...
```

Samozřejmě je lepší, to skombinovat s OpenDialogem ;)…
V balíku máte příklady v Delphi (VCL i bez VCL) komentáře jsou anglicky, a předem se omlouvám za špatnou angličtinu =), ale česky mě to psát nebaví… je to takové divné…

BTW: jestli NÁHODOU nevíte, jak dělat dialogy, icony, bitmapy atd… Pořiďte si freeware prográmek ResHacker, kde si vše vytvoříte hned, potom by nebylo na škodu si pořídit freeware ResEditor, nebo 30ti denní trial Resource Builder (ale ten stojí za to =), ten je nejlepší)… Pokud nevíte, jak udělat bitmapu:
V ResHackerovi klikněte na upravovaný Dialog pravým myšítkem, klikněte na Insert Control, vyberte BITMAP, a jako „Caption", napište název Resource bitmapy (v mém příkladu na WinAPI: „PIC1" (BACHA, JE TO CASE SENSITIVE!!))

V balíku „balik13.rar" najdete:
- **Crack [Src+Exe] na CMe z 1. lekce – Delphi7, {bez použití VCL + s použitím VCL}**
- **CMe z 1. lekce**

Lekce č.14

Patch - Search&Replace engine v Delphi DÍL[1]

Znáte dUP? Je to diablo2oo2's Ultimate Patcher. Pomocí tohoto prográmku si vytvoříte svůj vlastní patch. Do programu zadáte pár dat, ten vytvoří kdovíkam *.asm soubor, který se náhle zkompiluje pomocí MASM. Diablo2oo2 je proslavený jako „velkej cracker" a „čaroděj MASMu", tak není co divu, že i samotný dUP je vyvíjený v MASMu.

O co tedy vlastně jde? Dejme tomu, že PATCHUJETE 5MB program, patchnete 1 byt (třeba JZ za JMP) a chcete vyrobit patch. Minule jsem Vám ukazoval, jak to udělat pomocí pevného zadání offsetu přímo do zdrojového kódu, ale s&r engine pracuje úplně jinak… Jak?

Dejme tomu, že jsme v OllyDbg a crackujeme opět to naše CMe z lekce 1 (resp. 2), a jsme na adrese 00401185 kde je instrukce JNZ, kterou my chceme zaměnit, třeba vyNOPovat. My tak učiníme, odzkoušíme a bingo, udělá to to, co my chceme (vždycky goodboy). Každá instrukce je složená z nějakých bytů, ale to vy už snad víte… Označíme si v Olly TEN úsek, kde se nachází instrukce, označíme ho však tak dostatečně dlouhý, abychom si mohli být jistí, že se takový kousek kódu nikde jinde v programu

nevyskytuje, ale ne zase tak dlouhý aby to byla půlka z celé binárky… Dejme tomu že označíme toto:

```
0040117E   .  F7F3          DIV     EBX
00401180   .  35 86140000   XOR     EAX, 1486
00401185   .  75 1E         JNZ     SHORT crackme.004011A5 ;TOTO CHCEME
ZAMENIT!
00401187   .  6A 00         PUSH    0                                ;
/Style = MB_OK|MB_APPLMODAL
00401189   .  68 CC304000   PUSH    crackme.004030CC                 ;
|Title = "Zpráva"
```

Nyní si instrukci změníme (vynopujeme, nebo si patchnutí v Olly obnovíme v okně [/]), a vybereme ten stejný počet bytů, vlastně ten stejný úsek:

```
0040117E   .  F7F3          DIV     EBX
00401180   .  35 86140000   XOR     EAX, 1486
00401185      90            NOP      ;ZMENENO
00401186      90            NOP      ;ZMENENO
00401187   .  6A 00          PUSH    0                                ;
/Style = MB_OK|MB_APPLMODAL
00401189   .  68 CC304000    PUSH    crackme.004030CC                 ;
|Title = "Zpráva"
```

Tudíž změnili jsme byty 751E za 9090. No jo, jenže takových se v kódu může vyskytovat několik, proto jsme vybrali větší kousek :-). Nyní si odstraníme patchnutí z5 na „JNZ" a označíme si ten kousek co je výše, a klikneme na kód pravým myšítkem, klikneme na Binary->Binary Copy… vyhodí nám to do ClipBoardu toto:

```
F7 F3 35 86 14 00 00 75 1E 6A 00 68 CC 30 40 00
```

Opět to patchneme, vybereme opět tu část a opět dáme Binary copy a máme toto:

F7 F3 35 86 14 00 00 ***90 90*** 6A 00 68 CC 30 40 00

Po takovém rozepsání snad už začínáte chápat, co to je ten s&r engine. Jde o to, že v prográmku dUP zadáte v políčku „Find bytes:" toto:
F7 F3 35 86 14 00 00 ***75 1E*** 6A 00 68 CC 30 40 00

A v políčku „Replace with:" toto:
F7 F3 35 86 14 00 00 ***90 90*** 6A 00 68 CC 30 40 00

A výsledný crack bude při patchovací proceduře dělat toto:
Vezmi byte -> je byte F7? Ne? Tak popojdi o byte dál…
Vezmi další byte -> je byte F7? Ano?
Vezmi další byte -> je byte F3? Ne? Popojdi dále…
Vezmi další byte -> je byte F7? Ano?
Vezmi další byte -> je byte F3? Ano?
Vezmi další byte -> je byte 35? Ano?
……
Vezmi další byte -> je byte 40? Ano?
Vezmi další byte -> je byte 00? Ano?
Vrať se o 15 bytů z5 a podle zadaných dat „Replace with" přepiš data…

Je to jednoduché, co říkáte? :) Prográmek dUP verze 1 a zatím poslední v této době (2.16) je v balíku. Program má pro výrobu patchu ohromné množství možností a obrovská výhoda je ta, že je výsledný

patch v MASMu, tzn. malá binárka. Můžete vytvořit i Loader, nebo si do patchu přidat XM písničku… je toho opravdu moc :)…

Jenomže mě MASM moc nesedí, tak mi vrtalo hlavou… Co si takhle přepsat s&r engine do Delphi? Proč ne, trvalo mi to sice asi hodku ale poradil jsem si, výsledný engine v pascalu je zde:

```
begin
     //0F 8C 01 02 00 00 8B 86 84 A3 02 00 85 C0 EB 31 8B 86 E8 A4 02
00 8B 96 F0 A4 02 00
       maxL    := 7;  //in array is declared max value 128...

       rasd[1] := $0F;  rawd[1] := $0F;
       rasd[2] := $8C;  rawd[2] := $8C;
       rasd[3] := $01;  rawd[3] := $01;
       rasd[4] := $02;  rawd[4] := $07;
       rasd[5] := $00;  rawd[5] := $00;
       rasd[6] := $00;  rawd[6] := $00;
       rasd[7] := $8B;  rawd[7] := $8B;
//rasd=search byte, rawd=byte to replace(write)
end;

for j := 1 to FileSeek (FlHandle, 0, 2) do
  begin
  l:=1;
  m:=1;
                  FileSeek (FlHandle, n, 0);                    //Seek the
offset "addres" from Begin (0)
                  FileRead (FlHandle, Bfr, 1);      //Actually  byte  to
bfr

                  lopS:
                    if bfr = rasd[l] then  //Compare with „search for“
byte
                    begin
                      if l = maxl then  //if all „search for“ bytes are
equals
```

```
                        begin
                          //MessageBox(mwnd,   pchar(inttohex(bfr,2)+   '
FOUND!!'), pchar(inttohex(n, 2)),0); //Debug :)
                          SetDlgItemText(mwnd, id_process, Pchar(GetText
+ #13#10 + 'Bytes at address ' + inttohex(n,2) + ' equals to the "find
data"! Now Replacing...')); ScrollText;  //id_proces = Memo :), process
info
                          goto lopR;
                        end else
                        begin
                          FileSeek (FlHandle, (n + 1), 0);
                          FileRead (FlHandle, Bfr, 1);
                          inc (1);
                          goto lopS;
                        end;
                      end;
                        goto end2;


                  lopR:
                      if m = (maxl + 1) then
                        begin
                          //MessageBox(mwnd,           pchar('CRACKED!'),
pchar(inttohex(n,2) + '!!' + inttostr(m)),0);
                          SetDlgItemText(mwnd, id_process, Pchar(GetText
+  #13#10  +  'Bytes  at  address  '  +  inttohex(n,2)  +  '  sucessfully
patched!')); ScrollText;
                          Inc(HowManyCrkd);
                          Inc(CrkdProc);
                          goto end2;
                        end else
                        begin
                          FileSeek (FlHandle, (n - 1 + m), 0);
                          FileWrite(FlHandle, rawd[m], 1);
                          inc (m);
                          goto lopR;
                        end;
      end2:
        inc(n);
    end;
```

Jelikož po nějaké slušné době, mě přestalo bavit psát (deklarovat) ty byty ručně, vymyslel jsem si na to prográmek, do kterého prostě zkopčíte byty z Ollyho a vyflusne Vám to rovnou deklaraci, kterou pastnete do zdrojáku v Delphi :). Zatím se mi nějak nechtělo to řešit, dělám to takto, možná se na to někdy pořádně mrknu, a možná vymyslím i něco takového, jako je dUP :).

Takže teď Vám napíšu, jak by asi vypadal patch s použitím s&r na to CMe z lekce 1 a 2(CMe from Programujte.com =)):
Spustím si můj superprográmek a zadám do něho ty čísla bytů z Ollyho, vyplivne mi toto:

```
//Created by DjH's BTP4C v0.5
MaxL := 16;
       rasd[1] := $F7;       rawd[1] := $F7;
       rasd[2] := $F3;       rawd[2] := $F3;
       rasd[3] := $35;       rawd[3] := $35;
       rasd[4] := $86;       rawd[4] := $86;
       rasd[5] := $14;       rawd[5] := $14;
       rasd[6] := $00;       rawd[6] := $00;
       rasd[7] := $00;       rawd[7] := $00;
       rasd[8] := $75;       rawd[8] := $90;
       rasd[9] := $1E;       rawd[9] := $90;
       rasd[10] := $6A;      rawd[10] := $6A;
       rasd[11] := $00;      rawd[11] := $00;
       rasd[12] := $68;      rawd[12] := $68;
       rasd[13] := $CC;      rawd[13] := $CC;
       rasd[14] := $30;      rawd[14] := $30;
       rasd[15] := $40;      rawd[15] := $40;
       rasd[16] := $00;      rawd[16] := $00;
```

Když odstraníme veškeré zbytečnosti (ovládání oken, různé deklarace a procedury), zbude nám asi toto:

```
procedure Declare;
begin
  if o= 1 then
    begin
      //Created by DjH's BTP4C v0.5
        MaxL := 16;
              rasd[1] := $F7;       rawd[1] := $F7;
        rasd[2] := $F3;     rawd[2] := $F3;
              rasd[3] := $35;       rawd[3] := $35;
              rasd[4] := $86;       rawd[4] := $86;
              rasd[5] := $14;       rawd[5] := $14;
              rasd[6] := $00;       rawd[6] := $00;
              rasd[7] := $00;       rawd[7] := $00;
              rasd[8] := $75;       rawd[8] := $90;
              rasd[9] := $1E;       rawd[9] := $90;
              rasd[10] := $6A;      rawd[10] := $6A;
              rasd[11] := $00;      rawd[11] := $00;
              rasd[12] := $68;      rawd[12] := $68;
              rasd[13] := $CC;      rawd[13] := $CC;
              rasd[14] := $30;      rawd[14] := $30;
              rasd[15] := $40;      rawd[15] := $40;
              rasd[16] := $00;      rawd[16] := $00;
    end;
end;

function CrackIt: string; //Search and Replace engine, Delphi Code, by
DjH2oo7
label lopS;//loop in search...
label lopR;//loop in replace...
label end2;//jmp
begin
  HowManyCrkd := 0;
  //SucCrkd := False;
  j:=0;
  n:=1;   //WE CAN'N USE "J", I DON'T KNOW WHY??!!!
  //====================\\
  //    FlHandle    := FileOpen (FlName, fmOpenReadWrite);    //Handle se
urcuje jeste pred touto prockou
  //====================\\

for j := 1 to FileSeek (FlHandle, 0, 2) do
```

```
  begin
  l:=1;
  m:=1;
                  FileSeek (FlHandle, n, 0);                    //Seek the
offset "addres" from Begin (0)
                  FileRead (FlHandle, Bfr, 1);

                  lopS:
                    if bfr = rasd[1] then
                    begin
                      if l = maxl then
                        begin
                          SetDlgItemText(mwnd, id_process, Pchar(GetText
+ #13#10 + 'Bytes at address ' + inttohex(n,2) + ' equals to the "find
data"! Now Replacing...')); ScrollText;
                          goto lopR;
                        end else
                        begin
                          FileSeek (FlHandle, (n + l), 0);
                          FileRead (FlHandle, Bfr, 1);
                          inc (l);
                          goto lopS;
                        end;
                      end;
                        goto end2;


                  lopR:
                      if m = (maxl + 1) then
                        begin
                          SetDlgItemText(mwnd, id_process, Pchar(GetText
+ #13#10 + 'Bytes at address ' + inttohex(n,2) + ' sucessfully
patched!')); ScrollText;
                          Inc(HowManyCrkd);
                          Inc(CrkdProc);
                          //SucCrkd := True;
                          goto end2;
                        end else
                        begin
                          FileSeek (FlHandle, (n - 1 + m), 0);
                          FileWrite(FlHandle, rawd[m], 1);
```

```
                        inc (m);
                        goto lopR;
                      end;
      end2:
      inc(n);
    end;
  SetDlgItemText(mwnd, id_process, Pchar(GetText + #13#10 + 'Cracked ' +
inttostr(HowManyCrkd) + ' same procedures!')); ScrollText;
end;

//#########################################################################
###//

function GetInfo: string;
begin
crkdproc:=0;    //Kolik bylo cracknuto procedur...porovna se to s tim,
kolik jich cracknuto melo byt a pokud se sobe cisla rovnaji (tedy bylo
vsechno cracknuto), crack vy\hodnoti akci za uspesnou :), nekdy je i
vsak mozne, ze se crackne procedur vice (je jich deklarovano 3 a
cracknou se 4, proc? Protoze treba kusu kodu jako je deklarace 2 muze
byt v programu vice... nekdy to muze byt umysl, nekdy vsak ne...)
  if not FileExists(FlName) then
  begin
    SetDlgItemText(mwnd, id_process, Pchar(GetText + #13#10 + 'File ' +
FlName + ' not found, please, find it manually ;)')); ScrollText;
    OpenDlg;
    FlName:=Opn.lpstrFile;
    SetDlgItemText(mwnd, id_process, Pchar(GetText + #13#10 + 'File ' +
FlName + #13#10 + '[LOADED]'));  ScrollText;

      begin
        IsBkuped;       //Kontrola, jestli je zaskrtly CheckBox pro
zalohovani
      //-----------------------------------------------------\\
         FlHandle      := FileOpen (FlName, fmOpenReadWrite); //nyni
oteverme soubor a zjistime jeho handle
      //-----------------------------------------------------\\
         for o := 1 to nop do    //nop je "number of procedures" :),
tudiz kolik procedur se mam cracknout, je to deklarovano uplne dole :)
         begin
           Declare;   //Deklarovani bytu pro "search" a pro "write"
```

```
            CrackIt;    //Spusteni s&r enginu, cracknuti
          end;
       //-----------------------------------------------------\\
         FileClose(FlHandle);    //zavreni handle, souboru
       //-----------------------------------------------------\\
       end;

    SetDlgItemText(mwnd, id_process, Pchar(GetText + #13#10 + '[JOB
COMPLETED]'));
        if (crkdproc >= nop) then
          begin
            SetDlgItemText(mwnd, id_process, Pchar(GetText + #13#10 +
inttostr(crkdproc) + ' procedures cracked of ' + inttostr(nop) + #13#10
+ '[:' + product + ':] sucessfully cracked!! ;)'));  ScrollText;
            MessageBox(mwnd,pchar(product + ' sucessfully cracked ;)'),
pchar('Cracked ;)'), 0);
          end else
          begin
            SetDlgItemText(mwnd, id_process, Pchar(GetText + #13#10 +
product + ' - ERROR! ' + inttostr(crkdproc) + ' procedures cracked of '
+ inttostr(nop) + '!! (Maybe yet cracked, or you cracking update (or
other) version)')); ScrollText;
            MessageBox(mwnd,pchar(product    +    '    -    ERROR!    '    +
inttostr(crkdproc) + ' procedures cracked of ' + inttostr(nop) + '!!
(Maybe yet cracked, or you cracking update (or other) version)'),
pchar('NOT Cracked !!'), 0);
          end;
  end else

  //To same, akorat ze patch zjisti, ze program (v tomto pripade CMe) je
ve stejne slozce jako je patch, tudiz nemusime nic hledat a okamzite se
crackuje :)
    begin
    SetDlgItemText(mwnd, id_process, Pchar(GetText + #13#10 + 'File ' +
FlName + #13#10 + '[LOADED]'));  ScrollText;
      begin
        IsBkuped;
       //-----------------------------------------------------\\
          FlHandle   := FileOpen (FlName, fmOpenReadWrite);
       //-----------------------------------------------------\\
          for o := 1 to nop do
```

```
          begin
            Declare;
            CrackIt;
          end;
        //-----------------------------------------------------\\
          FileClose(FlHandle);
        //-----------------------------------------------------\\
        end;
    SetDlgItemText(mwnd,   id_process,   Pchar(GetText   +   #13#10   +
'[CRACKED]'));
        if (crkdproc >= nop) then
          begin
            SetDlgItemText(mwnd, id_process, Pchar(GetText + #13#10 +
product + ' sucessfully cracked ;)'));  ScrollText;
            MessageBox(mwnd,pchar(product + ' sucessfully cracked ;)'),
pchar('Cracked ;)'), 0);
          end else
          begin
            SetDlgItemText(mwnd, id_process, Pchar(GetText + #13#10 +
product  +  '  -  ERROR!  NOTHING  CRACKED!  (Maybe  yet  cracked)'));
ScrollText;
            MessageBox(mwnd,pchar(product + ' - ERROR! NOTHING CRACKED!
(Maybe yet cracked)'), pchar('NOT Cracked !!!'), 0);
          end;
  end;
end;
```

Kde ScrollText je moje vlastnoručně napsaná procka, která posunuje scrollbar v Memu (který informuje o procesu :))

Kompletní kód je samozřejmě v balíku :)

V balíku tentokrát naleznete:

- dUP v1
- dUP v2.16
- Mojí šablonu pro s&r engine do Delphi (bez VCL)
- Prográmek, do kterého dáte binární data z Ollyho a on Vám vyflusne Pascal deklaraci :)
- Zdroje (D7 noVCL) a exe pro patch na CMe z lekce 1 a 2, s použitím s&r enginu (je to už asi spíš jen taková demonstrace :)

PS: mám takový dojem, že těch patchů na to CMe jsme vyrobili už asi tři =)

V balíku „balik14.rar" najdete:

- Crack [Src+Exe] na CMe z 1. lekce - Delphi7, Search&Replace Engine, no VCL
- dUP v1
- dUP v2
- s&r příklad - S&R Crack na 3D Live Pool 2.21
- šablona pro cracky v Delphi [S&R] - (no VCL)
- Originální ASM src enginu s&r
- Návod - how.to.make.nice.search.and.replace.patches.pdf
- Program na konvertování binary dat z Olly do šablony pro cracky [BTDP4Cv0.5] by DjH2oo7

Lekce č.15

Patch - Search&Replace engine v Delphi DÍL[2]

Minule již jsme nakousli s&r, dnes doděláme poslední nedostatky :)

Co když třeba budu chtít nějaký byt ignorovat? V dUPu tato možnost je, byt ignorujete tak, že místo hex. Hodnoty bytu, zadáte ‚??'. Poupravil jsem mou šablonu a úplně jsem předělal „engine" (lol) programu, co generuje kód :), promiňte, nevím jak ho mám nazvat, říkejme mu třeba BTDP4C :). Nyní Vám i v balíčku přenechávám zdrojové kódy…

Kdybych nechal program BTDP4C „tak, jak je", po zadání ‚??' by mi vyplivl ‚… := $??' , což je kravina a Delphi nám to nesežere :), proto, když narazí v EditBoxu na ‚??', přidá ‚… := $FFF' což bude v mé šabloně taková „identifikace", že se má byte ignorovat :).

Jak jsem šablonu přeměnil, Vám teď ukážu, žádné velké štrachy jsem neudělal:

```
for j := 1 to FileSeek (FlHandle, 0, 2) do
  begin
  l:=1;
  m:=1;
                  FileSeek (FlHandle, n, 0);   //Seek the offset
"address" from Begin (0)
                  FileRead (FlHandle, Bfr, 1);

                  lopS:
                    if (bfr = rasd[l]) or (rasd[l] = $FFF) then    //if
read byte = "search byte" or search byte = '??' ($FFF)
                    begin
                      if l = maxl then    //If all bytes equals...
                        begin
                          SetDlgItemText(mwnd, id_process, Pchar(GetText
+ #13#10 + 'Bytes at address ' + inttohex(n,2) + ' equals to the "find
data"! Now Replacing...')); ScrollText;
                          goto lopR;   //Now replace...
                        end else
                        begin
                          FileSeek (FlHandle, (n + l), 0);  //read byte
and increase l
                          FileRead (FlHandle, Bfr, 1);
                          inc (l);
                          goto lopS;   //loop...
                        end;
                      end;
                        goto end2;   //We found nothing...


                lopR:
                    if m = (maxl + 1) then
                      begin
                            SetDlgItemText(mwnd, id_process,
                     Pchar(GetText + #13#10 + 'Bytes at address ' +
                     inttohex(n,2) + ' sucessfully patched!'));
                     ScrollText;
                        Inc(HowManyCrkd);
                        Inc(CrkdProc);
                        //SucCrkd := True;
                        goto end2;
```

```
                        end else
                        begin
                          if (rawd[m] = $FFF) then goto ignore;    //If
replace byte eqals to $FFF, ignore (increase and loop)
                          FileSeek (FlHandle, (n - 1 + m), 0);
                          FileWrite(FlHandle, rawd[m], 1);
                          ignore:
                          inc (m);
                          goto lopR;
                        end;
      end2:
      inc(n);
    end;
```

K čemu je používání ‚??', a vůbec, celého Search&Replace dobré? Možná jsem to už říkal, ale pro jistotu to řeknu ještě jednou :), v minulé lekci v balíku jste měli přidaný soubor ‚how.to.make.nice.search.and.replace.patches.pdf' ve kterém to bylo všechno pěkně popsáno, ale anglicky… Zkusím aspoň kousek z toho článku přeložit:

…

Proč s&r?

Odpověď je jednoduchá: když použijete patchování přesně zadaných offsetů, patch Vám poběží jen na verzi, kterou zrovna chcete patchnout, ale s&r patch, může při troše štěstí (a zacházení) fungovat i do budoucích (minulých) verzí programu. Ovšem, jak jsem řekl, musí se to udělat dobře :)

Pár pravidel…

Je ovšem pár pravidel, které musíte dodržet, aby jste dosáhli dobrého s&r patchu. Hlavně potřebujete velké znalosti assembleru a hexadecimálních

instrukcí (třeba co je za instrukci 75h?! Přece JNZ =))
Při zadávání šablony, podle které se bude patchovat, si vlastně uvědomme, z čeho vlastně se jednotlivé příkazy v hex. podobě skládají. Při zavolání (CALL) je vždy po instrukci CALL (většinou ‚E8') napsány 4 byty, které znamenají, jak daleko je od aktuálního offsetu instrukce, na kterou se volá. Stejně tak je na tom i JZ nebo JNZ. Vždy po bytu 75 nebo 74 následují 4 byty, které znamenají, jak daleko je instrukce, na kterou by se skákalo. Pokud je instrukce, na kterou JNZ nebo JZ ukazuje, blízko, označuje se JNZ SHORT popř. JZ SHORT. V tom případě je za bytem 74 nebo 75 jen jeden byte. Třeba instrukce vzdálená 14 bytů směrem nahoru by vypadala takto: 75 F1 (směrem nahoru 14 znaků, tj. „-14" což je FFh – F1h = Eh = 14d :)). Pokud stále nechápete, podívejte se na následující…(‚??' je to, co může být jakékoliv):

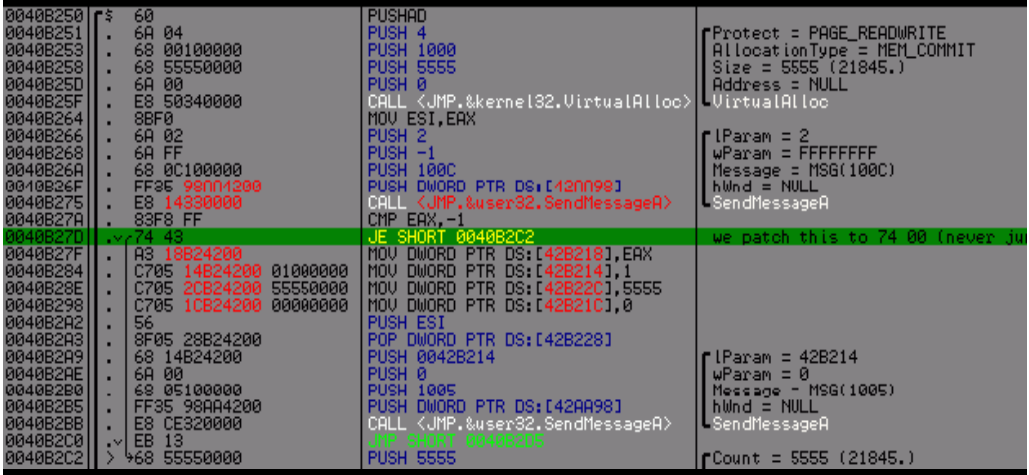
CALLy
ASM: CALL ?? ?? ?? ??
HEX: E8 ?? ?? ?? ??

Memory Addresses
ASM: CMP BYTE PTR [?? ?? ?? ??],1
HEX: 80 3D ?? ?? ?? ?? 01

Long Jumps
ASM: JE LONG ?? ?? ?? ??
HEX: OF 84 ?? ?? ?? ??

To jsou asi nejběžnější instrukce…

Uvedu příklad:



To co nám je „ukradený", je označeno červeně. Tyto hodnoty se můžou měnit (třeba podle verze), proto nás zajímají jen ostatní instrukce (cally, přesuny atp. ale číslo, na které JE volá, nás vážně nezajímá, protože podle verze se může změnit i délka, na kterou JE skáče)
Vlevo je zápis v HexaBytech, v pravo v assembleru…

Dejme tomu, že chceme, aby skok na adrese 0040B27D nikdy neskočil :)

```
0040B27D |. /74 43 JE SHORT 0040B2C2
0040B27D |. /74 00 JE SHORT 0040B27F
```

Vybereme si delší kus kódu před a po instrukci:

```
PUSH 2
PUSH -1
PUSH 100C
PUSH DWORD PTR DS:[?? ?? ?? ??]
CALL <?? ?? ?? ??>
CMP EAX,-1
JE SHORT ??
MOV DWORD PTR DS:[ ?? ?? ?? ?? ],EAX
MOV DWORD PTR DS:[ ?? ?? ?? ?? ],1
MOV DWORD PTR DS:[ ?? ?? ?? ?? ],5555
MOV DWORD PTR DS:[ ?? ?? ?? ?? ],0
```

A kus hexkódu, podle kterého budeme chtít hledat byty, bude vypadat asi takto:

```
6A 02 6A FF 68 0C 10 00 00 FF 35 ?? ?? ?? ?? E8 ??
?? ?? ?? 83 F8 FF 74 ?? A3 ?? ?? ?? ?? C7 05 ?? ??
?? ?? 01 00 00 00 C7 05 ?? ?? ?? ?? 55 55 00 00 C7
05 ?? ?? ?? ?? 00 00 00 00
```

A nahradit tímto:

```
?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??
?? ?? ?? ?? ?? ?? ?? 00 ?? ?? ?? ?? ?? ?? ?? ?? ??
?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??
?? ?? ?? ?? ?? ?? ?? ?? ??
```

Snad jste si všimli, proč jsem to označil a podtrhl :)…

Snad jsem to už dostatečně polopaticky vysvětlil, je to opravdu jednoduché, měl by to pochopit opravdu každý :)

Co se týče mé šablony, poupravil jsem ji, víceméně jste to mohli i postřehnout, když jsem Vám ten nový

s&r engine v Delphi ukazoval :), ale pokud to stále nechápete, přidal jsem do balíku zdrojové kódy opět ke hře 3D Live Pool, s využitím ‚??' ($FFF).
A co se týče BTP4C, uvolnil jsem zdroje, tudíž BTP4C je OpenSource =), ale základní úpravy jsem mu dodal, jako je třeba to, že text, který zadáváte do EditBoxů (binary data) se Vám minule vlezly jen na ten řádek, nyní můžou být binary data nekonečně dlouhá :). A vlastně ještě úprava ta, že podporuje konvertování ‚??' na $FFF, které engine v šabloně chápe jako „ignore byte" :)…

V balíku „balik15.rar" najdete:

- **BTDP4Cv0.6 [Exe+Src]**
- **Crack na 3D Live pool v2.21**

Lekce č.16

Cracknutí dalšího prográmku - VirtualDj

Asi tři díly jsme probíraly patchy a programování cracků, jako takových. Dlouho jsme si ale nic necrackli, hm, co říkáte? :)

Tak to teď jdu napravit. Znáte prográmek VirtualDj od firmy AtomiX? Je to paráda, po spuštění se před Vámi objeví mixážní pult, a je jen na Vás, co budete mixovat. Nebo můžete jen experimentovat, nebo si jen tak přehrávat hudby :). Myslím, že by tento program měla znát většina z Vás. Jinak ještě neplacená reklama: www.virtualdj.com :). (jen tak mimochodem…byl i na CD od Světa počítačů(5/2oo7)…)

Tento program je vyvíjen poměrně již dlouhou dobu, možná i přes 5 let. Aktuální je verze 4.3 (v době psaní článku). Tu já bohužel nemám, ale mám verze 2.0;2.1;3.0;3.1;4.0;4.1;4.2. Proč to říkám, se dozvíte za chvíli :)

Vše co tady budu provádět, bude na verzi 4.2. Ke stažení jsou dva balíky, jeden s VDJ4.2 a jeden bez něj. Instalační program VDJ zabírá 18.9 MB.

Takže…

Pokud bude pátrat po informacích o VDJ, zjistíte, že Atomix vydává VDJ jako 20ti denní trial. Po uplynutí této doby program sice spustíte, ale po dvou minutách se automaticky vypne.

Snad už máte VDJ v4.2 nainstalovaný, jestli ne, učiňte tak prosím =). V licenčních podmínkách ignorujte prosím tyto řádky:
This license does NOT allow you to:
- decompile, reverse engineer, disassemble, or otherwise reduce the Software to a human-perceivable form. =) a klikněte na „I accept" =)…
Po nainstalování zjistíme, že už jen samotná binárka virtualdj_trial.exe má 8.81 MB. Bude to teda pěkný maso vyznat se v 9 milionech znaků :).
(btw: zkoušel jsem to packovat UPXem, packovalo se to asi minutu a soubor se zkomprimoval na 1.7 MB (--best), potom ještě Upackem, to je podle mě nejlepší z nejlepších EXE packerů, je i v balíku, tím se to zkomprimovalo na 1.56MB. Nevím proč ten soubor nezkomprimovali, ale je to jen můj názor který jsem chtěl mermomoci prosadit :))
Podíváme se, co nám ukáže PEiD, a hle, je to jak jinak než Visual C++. Nyní si už konečně soubor otevřeme v Ollym. Jak již jsem říkal, program se po uplynutí 20ti zkušebních dnů po DVOU minutách sám vypne. Jelikož nám to ani žádná message neoznámí, že jsme už „vyčerpali" všechny dny, budeme muset najít funkci, tedy nastavení nějakého časového intervalu. Tu má na starost WinAPI „SetTimer", v knihovně USER32.DLL. Pokud chceme Timer zrušit, použijeme WinAPI KillTimer, najdete to ve WinAPI příručce, ale to teď není důležité :), teď je důležité najít ten hlavní „SetTimer". Takže hon na SetTimer začneme asi takto: Klikneme v Ollym na asm kód pravým myšítkem,

a zvolíme „Search for" -> „All intermodular calls". Bingo, objeví se nám všechny cally a WinApi, na které program volá. Seřadíme si je podle „Destination" a sjedeme dolů, někam k „S", a tam už SetTimer najdeme :). No jo, SetTimer se tu ale objevil 34x! Vezmeme to teoreticky… Můžeme předpokládat, že se timer spustí hned při startu, nebo hned po něm. Můžeme tedy nastavit BP na všechny CALLy, které volají Api „SetTimer". Uděláme to tak, že na obrazovce s asm kódem [C], stiskněte kombinaci [Ctrl + N]. Najděte si Call na USER32.SetTimer, označte ho, klikněte na něj „pravým", a klikněte na „Set breakpoint on every reference". Teď se Vám na všechny cally na tuto api, nastavily breakpointy. Nezbývá nám nic jiného, než aplikaci spustit, a doufat, že se nějaký breakpoint „chytne" :)…
Po chvilce se vážně Olly zastavil na breakpointu. Není to vlastně ani call, je to přesun adresy callu na SetTimer, do registru ESI:

```
0041868E  |.  57            PUSH    EDI             ; |hWnd
0041868F  |.  FF15 D8836E00 CALL    NEAR DWORD PTR DS:[<&SHELL32.Dra>;
\DragAcceptFiles
00418695  |.  8B15 947CB100 MOV     EDX, DWORD PTR DS:[B17C94]
0041869B  |.  8B35 24846E00 MOV     ESI, DWORD PTR DS:[<&USER32.SetT>;
USER32.SetTimer
004186A1  |.  6A 00         PUSH    0       ; /Timerproc = NULL
004186A3  |.  6A 32         PUSH    32      ; |Timeout = 50. ms
004186A5  |.  68 A0000000   PUSH    0A0     ; |TimerID = A0 (160.)
004186AA  |.  52            PUSH    EDX     ; |hWnd => 003F0922
('VirtualDJ',class='VIRTUALDJ')
004186AB  |.  FFD6          CALL    NEAR ESI   ; \SetTimer
004186AD  |.  A1 947CB100   MOV     EAX, DWORD PTR DS:[B17C94]
004186B2  |.  6A 00         PUSH    0   ; /Timerproc = NULL
```

```
004186B4  |.  68 C1D40100   PUSH    1D4C1   ; |Timeout = 120001. ms
004186B9  |.  68 9F000000   PUSH    9F      ; |TimerID = 9F (159.)
004186BE  |.  50            PUSH    EAX     ; |hWnd => 003F0922
('VirtualDJ',class='VIRTUALDJ')
004186BF  |.  FFD6          CALL    NEAR ESI  ;\SetTimer
004186C1  |.  833D 647DA400>CMP     DWORD PTR DS:[A47D64], 0
004186C8  |.  74 04         JE      SHORT virtuald.004186CE
004186CA  |.  33C0          XOR     EAX, EAX
```

Takže je to přece jenom ono :), je to trochu skrytější funkce, protože to není přímý call, ale oklikou přes adresu v ESI. Na adrese 004186B4 můžeme vidět, jak se PUSHuje ten dvouminutový interval, resp. 120 tisíc tisícin :), a na adrese 004186BF se už z ESI volá Api SetTimer.
Teď ale jak to patchnout. Máme více možností, můžeme třeba všechny pushování a následný CALL ESI vyNOPovat, ale je to zbytečně složité. Myslím si, že lepší by bylo si na adrese 004186B2 udělat odskok až za CALL ESI, takže se vlastně timer nenastaví. Poklepeme na instrukci „PUSH 0" na adrese 004186B2 a přepíšeme ji na „JMP 004186C1". Bajty 6A 00 (PUSH 0) se nahradily za EB 0A (JMP +$0D (+14)). Nyní restartujeme aplikaci [Ctrl + F2], najedeme do záložky Patches [/], dáme naše patchlé byty do stavu [Active], odstraníme všechny breakpointy, a necháme jen ten na adrese 0041869B. Spustíme aplikaci [F9]. A hle! Ani po dvou minutách se nevypne :). Docela to ale pořád kazí po spuštění ten nápis „Trial: X days left". Co si tam napsat něco jako „Cracked by DjH2oo7"?! Jak to udělat? Klikněte do oblasti RamDump pravým tlačítkem, a zvolte „Search For" ->

„Binary String". Nebo můžete použít klávesovou zkratku [Ctrl + B] :). Tam zadejte třeba slova „days left" a klikněte na tlačítko OK. Pokud se nic nenajde, asi nejste v oblasti s daty. V tom případě klikněte pravým na RamDump, a zvolte „Go to" -> „Expression", a zadejte „006E8000" bez uvozovek. Zde začínají data. Jak jsem k tomu došel? Podívejte se v části s asm kódem, úplně dolů. Co vidíte za adresu? „006E7FFF". No a hned bajt za tím jsou data :)… Pokud jste string našli, označte ho celý:

```
006F2CB6  54 72 69 61 6C 3A 20 25 69 20  Trial: %i
006F2CC6  64 61 79 73 20 6C 65 66 74 00  days left.
```

Až ho označíte, klikněte na něj pravým, a zvolte „Binary" -> „Edit" popřípadě [Ctrl + E]. Místo stringu Trial…blabla… Napište třeba „Cracked by DjH2oo7". Ovšem string musí končit hexnulou. Takže po znaku „7" najeďte do části kde se zapisují data hexadecimálně, a za „sedmičku" napište „00". Může to vypadat třeba takto:

```
006F2CB6  43 72 61 63 6B 65 64 20 62 79       Cracked by
006F2CC6  20 44 6A 48 32 6F 6F 37 00 00       DjH2oo7..
```

Pokud bychom si vybrali třeba kratší string, např.: „DjH", nebyl by problém, kdyby to bylo třeba takto:

```
006F2CB6  43 72 61 63 6B 65 64 20 62 79       Cracked by
006F2CC6  20 44 6A 48 00 6C 65 66 74 00       DjH.left.
```

Protože za „H" je hexnula, tudíž tam string končí, a nikoho to nezajímá =)… S delším stringem by byl už ale problém…

Nyní jdeme na patch, jak jinak než s&r, podle mé Delphi šablony :). Zkusíme si vybrat kus kódu, do toho podle minulého článku přidat na správná místa otazníky ‚??' a patch otestovat. Vyberu třeba tento kód:

```
004186B2      6A 00          PUSH    0
004186B4  |.  68 C1D40100    PUSH    1D4C1     ; |Timeout = 120001. ms
004186B9  |.  68 9F000000    PUSH    9F        ; |TimerID = 9F (159.)
004186BE  |.  50             PUSH    EAX       ; |hWnd => NULL
004186BF  |.  FFD6           CALL    NEAR ESI  ; \SetTimer
004186C1  |.  833D 647DA400>CMP     DWORD PTR DS:[A47D64], 0
004186C8  |.  74 04          JE      SHORT virtuald.004186CE
004186CA  |.  33C0           XOR     EAX, EAX
```

Hex. data vypadají takto:

```
6A 00 68 C1 D4 01 00 68 9F 00 00 00 50 FF D6 83 3D 64 7D A4 00
00 74 04 33 C0
```

Podle minulé lekce už víme, kam nejlépe otazníky dát. Podle mě by to bylo nejlepší takto:

```
6A ?? 68 ?? ?? ?? ?? 68 ?? ?? ?? ?? 50 FF D6 83 3D ?? ?? ?? ??
?? 74 ?? 33 C0
```

A patch bajty:

```
EB 0D ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??
?? ?? ?? ?? ??
```

Když toto zadám do mého prográmku BTDP4C, vyplivne mi toto:

```
//6A ?? 68 ?? ?? ?? ?? 68 ?? ?? ?? ?? 50 FF D6 83 3D ?? ?? ?? ?? ?? 74
?? 33 C0
//EB 0D ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??
?? ?? ??

//Created by DjH's BTP4C v0.6
MaxL := 26;
       rasd[1] := $6A;       rawd[1] := $EB;
       rasd[2] := $FFF;      rawd[2] := $0D;
       rasd[3] := $68;       rawd[3] := $FFF;
       rasd[4] := $FFF;      rawd[4] := $FFF;
       rasd[5] := $FFF;      rawd[5] := $FFF;
       rasd[6] := $FFF;      rawd[6] := $FFF;
       rasd[7] := $FFF;      rawd[7] := $FFF;
       rasd[8] := $68;       rawd[8] := $FFF;
       rasd[9] := $FFF;      rawd[9] := $FFF;
       rasd[10] := $FFF;     rawd[10] := $FFF;
       rasd[11] := $FFF;     rawd[11] := $FFF;
       rasd[12] := $FFF;     rawd[12] := $FFF;
       rasd[13] := $50;      rawd[13] := $FFF;
       rasd[14] := $FF;      rawd[14] := $FFF;
       rasd[15] := $D6;      rawd[15] := $FFF;
       rasd[16] := $83;      rawd[16] := $FFF;
       rasd[17] := $3D;      rawd[17] := $FFF;
       rasd[18] := $FFF;     rawd[18] := $FFF;
       rasd[19] := $FFF;     rawd[19] := $FFF;
       rasd[20] := $FFF;     rawd[20] := $FFF;
       rasd[21] := $FFF;     rawd[21] := $FFF;
       rasd[22] := $FFF;     rawd[22] := $FFF;
       rasd[23] := $74;      rawd[23] := $FFF;
       rasd[24] := $FFF;     rawd[24] := $FFF;
       rasd[25] := $33;      rawd[25] := $FFF;
       rasd[26] := $C0;      rawd[26] := $FFF;
```

Ještě musíme nahradit string „Trial…" za „Cracked by…". Takže… Hexhodnoty stringu pro vyhledání (Trial):

```
54 72 69 61 6C 3A 20 25 69 20 64 61 79 73 20 6C 65 66 74 00
```

A string „Cracked by":

```
43 72 61 63 6B 65 64 20 62 79 20 44 6A 48 32 6F 6F 37 00 00
```

Záměna stringu je takováto:

```
//54 72 69 61 6C 3A 20 25 69 20 64 61 79 73 20 6C 65 66 74 00
//43 72 61 63 6B 65 64 20 62 79 20 44 6A 48 32 6F 6F 37 00 00

//Created by DjH's BTP4C v0.6
MaxL := 20;
       rasd[1] := $54;       rawd[1] := $43;
       rasd[2] := $72;       rawd[2] := $72;
       rasd[3] := $69;       rawd[3] := $61;
       rasd[4] := $61;       rawd[4] := $63;
       rasd[5] := $6C;       rawd[5] := $6B;
       rasd[6] := $3A;       rawd[6] := $65;
       rasd[7] := $20;       rawd[7] := $64;
       rasd[8] := $25;       rawd[8] := $20;
       rasd[9] := $69;       rawd[9] := $62;
       rasd[10] := $20;      rawd[10] := $79;
       rasd[11] := $64;      rawd[11] := $20;
       rasd[12] := $61;      rawd[12] := $44;
       rasd[13] := $79;      rawd[13] := $6A;
       rasd[14] := $73;      rawd[14] := $48;
       rasd[15] := $20;      rawd[15] := $32;
       rasd[16] := $6C;      rawd[16] := $6F;
       rasd[17] := $65;      rawd[17] := $6F;
       rasd[18] := $66;      rawd[18] := $37;
       rasd[19] := $74;      rawd[19] := $00;
       rasd[20] := $00;      rawd[20] := $00;
```

Tohle si pastneme do našeho zdrojáku, a ještě v šabloně úplně dole, jak se deklarují stringy, musíte změnit proměnnou „nop" (number of process) na

hodnotu 2 (patchujeme dvakrát – SetTimer a String). A proč jsem na začátku říkal že mám všechny ty verze? Jednoduše, na všechny ty verze můj patch funguje :)

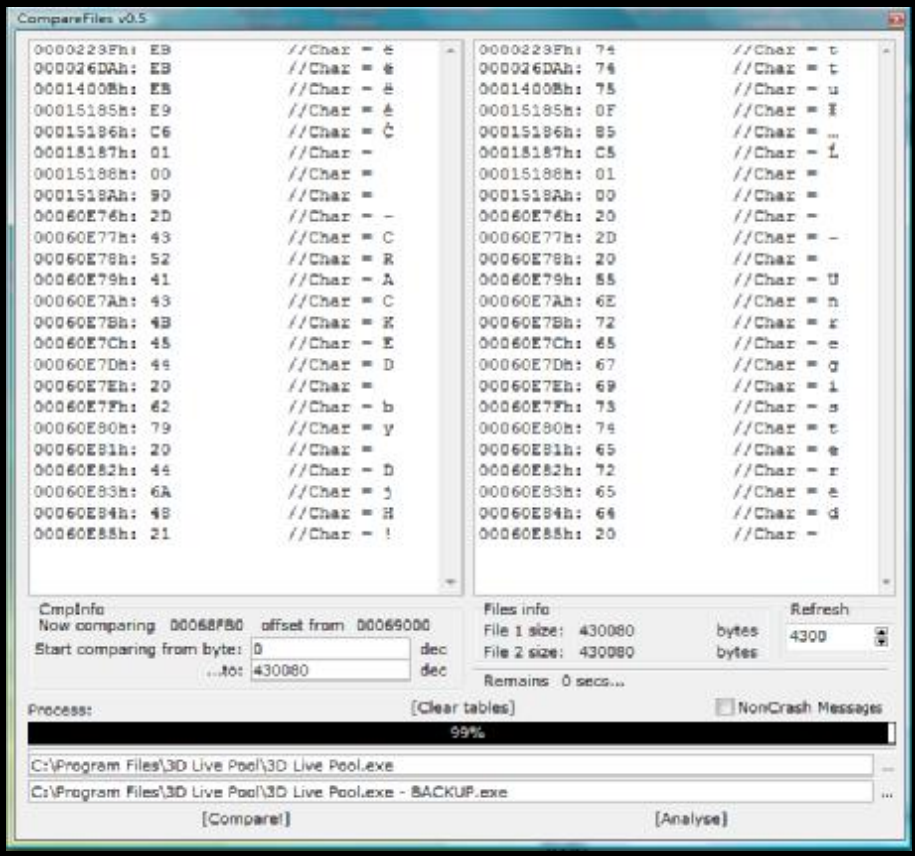
V balíku „balik16.rar" najdete:

- **Crack [Src+Exe] na Virtual Dj {2.x;3.x;4.x} – Delphi7, Search&Replace Engine, no VCL**
- **WinUPack v0.39**
- **v balíku „balik16b.rar" je navíc setu VirtualDj4.2**

Lekce č.17

Porovnávání souborů

Máš patch, a chtěl by si zjistit, co všechno v daném souboru patchnul, nebo si jen chceš zkontrolovat jestli to, co jsi patchl, jsi patchnul správně? Vím, že programů na porovnávání souborů je hodně, ale žádný mě nevyhovoval, tak jsem si udělal svůj. Nejdřív mě to napadlo, když jsem chtěl zjistit jestli v programu 3D Live Pool byly patchlé bajty opravdu ty, které jsem chtěl patchnout. Naprogramoval jsem prográmek, zatím pod názvem CmpFiles. Funguje na tom principu, že si vyberete dva soubory (třeba patchlý a nepatchlý) a stisknete „Analyse" a potom „Compare". Analýza Vám zjistí velikost souboru a doporučí refresh hodnot. Po stisku Compare začne bajt po bajtu prohledávání souborů a když se dva bajty sobě nerovnají, vypíšou se i s hex. adresami a bajt se vypíše v ASCII hodnotě. Schválně si zkuste dát jako první soubor originální 3D Live Pool.exe a jako druhý cracklý Live Pool. Ukáže vám to patchlé bajty s originálními. Naopak když je člověk línej (nebo lama), a stáhne si patch (třeba vyrobený pomocí dUP), a aplikuje patch, snadno si zjistí, které byty se mu v souboru patchly, potom se může podívat do Ollyho a najet si na adresu, která se mu v CmpFiles ukázala, a tak snadno zjistí, co a jak se patchlo :) Zdrojové kódy jsou přiložené, náhodou se mi zase chtělo použít VCL =), asi lenost tou začínající zimou :)… Ještě ukážu screenshot z porovnávání dvou souborů (3D Live Pool) cracklý a originál:



V balíku „balik17.rar" najdete:

- **CmpFiles v0.5 [exe+src] – Delphi7 – VCL**

Lekce č.18

Trocha programování v Delphi – komponenta DjH

DÍL[1]

Tento díl je celý určen jen programátorům v Delphi, takže pokud nejste programátor v Delphi, s klidným srdcem můžete tuto lekci přeskočit. Pokud se chcete přiučit tak nemusíte, čtěte dále :). V této nepravidelné řadě budeme vyvíjet komponentu „DjH", která obsahuje hodně používané funkce, ale ty jsou bohužel v unitách, které se na binárce poznamenají i několika desítkami nebo stovkami kilobajtů, nebo funkce, které nikde jinde nejsou. I přesto se použitá unita v binárce snaží nepřesáhnout 3 kB.

Znáte to, děláte keygen bez pomoci VCL, ale přece jen někde musíte použít funkci „IntToStr", „IntToHex", „StrToInt" atp… Tyto funkce Vám bez přidané Unit „SysUtils" v Uses nepůjdou. Jenže s přidáním SysUtils do Uses, se binárka zvětší asi o 30kB, protože v SysUtils jsou i funkce pro práci se soubory (FileCreate, FileSeek…) a různé funkce, které VŮBEC v keygenu nejsou potřeba! Bohužel Borland pro nás nenechal ani nezkompilovaný soubor „SysUtils.pas", abychom se mrkli, jak se takový integer převádí na string, místo toho nám už nechal zkompilovanou unitu „SysUtils.dcu". Ještě jsem neřekl co mám v plánu…

Mám v plánu vytvořit Unitu, podobnou SysUtils, ale jen s vybranými funkcemi (StrToInt. IntToStr…).
Takže, zaprvé, jak dostat ze zkompilované unity kód? Přece pomocí DeDe :), ano, DeDe umí dekompilovat i DCU soubory, opět Vám jen ukáže ASM kód, ale to nám vůbec nevadí :)…
Otevřete si DeDe, zvolte z menu „Dumpers" -> „DCU dumper" a vyberte soubor SysUtils.dcu a klikněte na Process. Potom uložte z „Mema" kód nebo označte RadioButton „Save output to file". Chvíli to potrvá, ale dočkáte se. Uloží se Vám asi soubor s 25000 řádky :). Ten si otevřete v nějakém programu, co zvýrazňuje syntaxi, třeba opensource „Notepad ++" nebo PSPad. Moc nedoporučuju otevírat to přímo v Delphi.
Hon na IntToHex…
Vyhledejte si string „IntToHex" a asi třetí z vrchu je procedura v asm. Pod tímto (4. z vrchu) je funkce IntToHex, která importuje Int64 (což je výhoda, integer podporuje int64). Takže se zaměříme na něj:

```
function IntToHex (Value: Int64; Digits: Integer): AnsiString; overload;
var
  result Result: AnsiString;
begin
   00000000 : 55                          PUSH EBP
   00000001 : 8B EC                       MOV EBP,ESP
   00000003 : 83 F8 20                    CMP EAX,32
   00000006 : 7E 02                       JLE +2; (0xA)
   00000008 : 31 C0                       XOR EAX,EAX
   0000000A : 56                          PUSH ESI
   0000000B : 89 E6                       MOV ESI,ESP
   0000000D : 83 EC 20                    SUB ESP,32
   00000010 : B9 10 00 00 00              MOV ECX,$00000010
   00000015 : 52                          PUSH EDX
   00000016 : 89 C2                       MOV EDX,EAX
```

```
   00000018 : 8D 45 08                          LEA EAX,DWORD PTR
[EBP+8{Value}]
   0000001B : E8(00 00 00 00                    CALL CvtInt64{0x41A}
   00000020 : 89 F2                             MOV EDX,ESI
   00000022 : 58                                POP EAX
   00000023 : E8(00 00 00 00                    CALL
@LStrFromPCharLen{0x18B}
   00000028 : 83 C4 20                          ADD ESP,32
   0000002B : 5E                                POP ESI
   0000002C : 5D                                POP EBP
   0000002D : C2 08 00                          RET NEAR,8
end;
```

Myslím si, že není co dodat, a můžeme ihned zapisovat. První dva řádky, Push EBP a MOV EBP, ESP vynecháme, Delphi je přidává automaticky a poslední dva, Pop ebp a Return taky. Za JLE je adresa +2 (0A), to znamená že skáče v tomto kódu na adresu 0A, kde je příkaz PUSH ESI… Vytvoříme si tedy label a bude to vypadat asi takto:

```
function IntToHex(Value: Int64; Digits: Integer): string;
asm
        CMP     EAX, 32         // Digits < buffer length?
        JLE     @A1
        XOR     EAX, EAX
@A1:    PUSH    ESI
        MOV     ESI, ESP
        SUB     ESP, 32         // 32 chars
        MOV     ECX, 16         // base 10
        PUSH    EDX             // result ptr
        MOV     EDX, EAX        // zero filled field width: 0 for no
leading zeros
        LEA     EAX, Value;
          CALL    CvtInt64

        MOV     EDX, ESI
        POP     EAX             // result ptr
          CALL    System.@LStrFromPCharLen
        ADD     ESP, 32
        POP     ESI
end;
```

Funkci LstrFromPCharLen má v Delphi každý program, proč to tedy nepoužít, že? Ještě k řádku 18h, kde je LEA EAX, Value… Snad Vám došlo, že PTR v tom DCU dumpu měl v {závorce} hodnotu Value, tudíž takhle to jde použít. Větší problém je přeložit CvtInt64, což je funkce v unitě SysUtils, která převádí Int64 na text. Není to přesně IntToStr, ale je to kód, který „předpřevede" integer na text (ne na string). Přeložit ji musíme :)… Dump je:

```
procedure CvtInt64;
begin
   00000000 : 08 C9                        OR CL,CL
   00000002 : 75 30                        JNE +48; (0x34)
   00000004 : B9 0A 00 00 00               MOV ECX,$0000000A
   00000009 : F7 40 04 00 00 00 80         TEST DWORD PTR
[EAX+4],$80000000
   00000010 : 74 22                        JE +34; (0x34)
   00000012 : FF 70 04                     PUSH DWORD PTR [EAX+4]
   00000015 : FF 30                        PUSH DWORD PTR [EAX]
   00000017 : 89 E0                        MOV EAX,ESP
   00000019 : F7 1C 24                     NEG DWORD PTR [ESP]
   0000001C : 83 54 24 04 00               ADC DWORD PTR [ESP+4],0
   00000021 : F7 5C 24 04                  NEG DWORD PTR [ESP+4]
   00000025 : E8(34 00 00 00               CALL +CvtInt64{0x41A}+52
   0000002A : C6 46 FF 2D                  MOV BYTE PTR [ESI-1],$2D
   0000002E : 4E                           DEC ESI
   0000002F : 41                           INC ECX
   00000030 : 83 C4 08                     ADD ESP,8
   00000033 : C3                           RET NEAR
   00000034 : 56                           PUSH ESI
   00000035 : 83 EC 04                     SUB ESP,4
   00000038 : D9 7C 24 02                  FSTCW  [ESP+2]
   0000003C : D9 3C 24                     FSTCW  [ESP]
   0000003F : 66 81 0C 24 00 0F            OR WORD PTR [ESP],$0F00
   00000045 : D9 2C 24                     FLDCW  [ESP]
   00000048 : 66 89 0C 24                  MOV WORD PTR [ESP],CX
   0000004C : D9 E8                        FLD1
   0000004E : F7 40 04 00 00 00 80         TEST DWORD PTR
[EAX+4],$80000000
   00000055 : 74 27                        JE +39; (0x7E)
   00000057 : FF 70 04                     PUSH DWORD PTR [EAX+4]
   0000005A : FF 30                        PUSH DWORD PTR [EAX]
   0000005C : 81 64 24 04 FF FF FF 7F      AND DWORD PTR
[ESP+4],$7FFFFFFF
```

```
   00000064 : 68 FF FF FF 7F              PUSH $7FFFFFFF
   00000069 : 68 FF FF FF FF              PUSH $FFFFFFFF
   0000006E : DF 6C 24 08                 FILD TWORD PTR [ESP+8]
   00000072 : DF 2C 24                    FILD TWORD PTR [ESP]
   00000075 : D8 C2                       FADD ST,ST2
   00000077 : DE C1                       FADDP ST1,ST
   00000079 : 83 C4 10                    ADD ESP,16
   0000007C : EB 02                       JMP +2; (0x80)
   0000007E : DF 28                       FILD TWORD PTR [EAX]
   00000080 : DF 04 24                    FILD WORD PTR [ESP]
   00000083 : D9 C1                       FLD ST1
   00000085 : 4E                          DEC ESI
   00000086 : D9 F8                       FPREM
   00000088 : DF 1C 24                    FISTP WORD PTR [ESP]
   0000008B : DC F9                       FDIVR ST1,ST
   0000008D : 8A 04 24                    MOV AL,BYTE PTR [ESP]
   00000090 : 04 30                       ADD AL,$30
   00000092 : 3C 3A                       CMP AL,$3A
   00000094 : 72 02                       JB +2; (0x98)
   00000096 : 04 07                       ADD AL,$07
   00000098 : 88 06                       MOV BYTE PTR [ESI],AL
   0000009A : D9 C1                       FLD ST1
   0000009C : D8 D3                       FCOM ST3
   0000009E : 9B                          WAIT
   0000009F : DF E0                       FSTSW AX
   000000A1 : 9E                          SAHF
   000000A2 : 73 E1                       JNB -31; (0x85)
   000000A4 : D9 6C 24 02                 FLDCW  [ESP+2]
   000000A8 : 83 C4 04                    ADD ESP,4
   000000AB : DD C3                       FFREE ST3
   000000AD : DD C2                       FFREE ST2
   000000AF : DD C1                       FFREE ST1
   000000B1 : DD C0                       FFREE ST
   000000B3 : 59                          POP ECX
   000000B4 : 29 F1                       SUB ECX,ESI
   000000B6 : 29 CA                       SUB EDX,ECX
   000000B8 : 76 10                       JBE +16; (0xCA)
   000000BA : 29 D6                       SUB ESI,EDX
   000000BC : B0 30                       MOV AL,$30
   000000BE : 01 D1                       ADD ECX,EDX
   000000C0 : EB 03                       JMP +3; (0xC5)
   000000C2 : 88 04 32                    MOV BYTE PTR [EDX+ESI],AL
   000000C5 : 4A                          DEC EDX
   000000C6 : 75 FA                       JNE -6; (0xC2)
   000000C8 : 88 06                       MOV BYTE PTR [ESI],AL
   000000CA : C3                          RET NEAR
end;
```

Nelekejte se, je to přece jen o trošičku delší, ale postup je úplně stejný :). Tam kde jsou jumpy, vytvářejte labely a poslední return vypusťte. Po chvilce trápení byste měli vykouzlit asi přibližně toto:

```
procedure CvtInt64;
{ IN:
    EAX:  Address of the int64 value to be converted to text
    ESI:  Ptr to the right-hand side of the output buffer:  LEA ESI,
StrBuf[32]
    ECX:  Base for conversion: 0 for signed decimal, or 10 or 16 for
unsigned
    EDX:  Precision: zero padded minimum field width
  OUT:
    ESI:  Ptr to start of converted text (not start of buffer)
    ECX:  Byte length of converted text
}
asm
        OR      CL, CL
        JNZ     @start              // CL = 0  => signed integer
conversion
        MOV     ECX, 10
        TEST    [EAX + 4], $80000000
        JZ      @start
        PUSH    [EAX + 4]
        PUSH    [EAX]
        MOV     EAX, ESP
        NEG     [ESP]               // negate the value
        ADC     [ESP + 4],0
        NEG     [ESP + 4]
        CALL    @start              // perform unsigned conversion
        MOV     [ESI-1].Byte, '-'   // tack on the negative sign
        DEC     ESI
        INC     ECX
        ADD     ESP, 8
        RET

@start:   // perform unsigned conversion
        PUSH    ESI
        SUB     ESP, 4
        FNSTCW  [ESP+2].Word      // save
        FNSTCW  [ESP].Word        // scratch
        OR      [ESP].Word, $0F00   // trunc toward zero, full precision
        FLDCW   [ESP].Word
```

```
        MOV     [ESP].Word, CX
        FLD1
        TEST    [EAX + 4], $80000000  // test for negative
        JZ      @ld1                  // FPU doesn't understand unsigned
ints
        PUSH    [EAX + 4]             // copy value before modifying
        PUSH    [EAX]
        AND     [ESP + 4], $7FFFFFFF  // clear the sign bit
        PUSH    $7FFFFFFF
        PUSH    $FFFFFFFF
        FILD    [ESP + 8].QWord      // load value
        FILD    [ESP].QWord
        FADD    ST(0), ST(2)         // Add 1.  Produces unsigned
$80000000 in ST(0)
        FADDP   ST(1), ST(0)         // Add $80000000 to value to replace
the sign bit
        ADD     ESP, 16
        JMP     @ld2
@ld1:
        FILD    [EAX].QWord          // value
@ld2:
        FILD    [ESP].Word           // base
        FLD     ST(1)
@loop:
        DEC     ESI
        FPREM                        // accumulator mod base
        FISTP   [ESP].Word
        FDIV    ST(1), ST(0)         // accumulator := acumulator / base
        MOV     AL, [ESP].Byte       // overlap long FPU division op with
int ops
        ADD     AL, '0'
        CMP     AL, '0'+10
        JB      @store
        ADD     AL, ('A'-'0')-10
@store:
        MOV     [ESI].Byte, AL
        FLD     ST(1)            // copy accumulator
        FCOM    ST(3)            // if accumulator >= 1.0 then loop
        FSTSW   AX
        SAHF
        JAE @loop

        FLDCW   [ESP+2].Word
        ADD     ESP,4

        FFREE   ST(3)
```

```
        FFREE   ST(2)
        FFREE   ST(1);
        FFREE   ST(0);

        POP     ECX             // original ESI
        SUB     ECX, ESI        // ECX = length of converted string
        SUB     EDX,ECX
        JBE     @done           // output longer than field width = no
pad
        SUB     ESI,EDX
        MOV     AL,'0'
        ADD     ECX,EDX
        JMP     @z
@zloop: MOV     [ESI+EDX].Byte,AL
@z:     DEC     EDX
        JNZ     @zloop
        MOV     [ESI].Byte,AL
@done:
end;
```

Všimněte si, že v kódu už jsou převedené „hexhodnoty" na „chary" ($2D -> ,-'). Nyní už jen vytvořit komponentu, nebo ještě lépe, package :). Když už jste v tom PSPadu, nebo kde vlastně (já v Notepadu++), vytvoříme si tam tu komponentu. Po chvíli čachrování byste měli napsat asi něco takového:

```
unit DjH;

interface
     function IntToHex (Value: Int64; Digits: Integer): String;
overload;

implementation

uses Windows,Messages;

//############################################################################//
procedure CvtInt64;
asm
……
end;
//######################INTTOHEX#################################//
function IntToHex (Value: Int64; Digits: Integer): String;
asm
……
end;
```

Za „……" si doplňte výše uvedený kód :). Stačí komponentu už jen nakopčit do …\Borland\DelphiX\Lib (pod názvem „DjH.pas") nebo ji zaregistrujte přes menu „Component" -> „Install component…". Nyní si vytvořte aplikaci (bez VCL) a místo SysUtils napište DjH (SysUtils vymažte). Třeba si udělejte takovouto jednoduchou aplikaci:

```
program DjHExample;
uses
  Windows,DjH;

begin
  Randomize;
  MessageBox(0,PChar(IntToHex((Random(100000)+1),8)),'Náhodné číslo <0-
100000>',0);
end.
```

Zkompilujte, a hle! Program má jen 14.5 kB! Pro zajímavost: Po zkomprimování Upackem jen 7.87 kB! Nyní si místo DjH v uses, napište SysUtils a zkompilujte [Ctrl + F9], a co se nestalo, aplikace má rázem 38,5 kB! Po zásahu Upacku má soubor sice míň, ale 18.2 kB je stejně „moc". Toto (a ještě méně!) my máme s komponentou DjH i bez komprimace :).

Tak, a běžte machrovat s touto komponentou! =), cože, že je to málo? Ok, fajn, uděláme si ještě „IntToStr", „StrToInt" a „StrToHex". Třeba funkci StrToHex nikde nenajdete, a ta naše funkce bude pracovat na tomto principu:

```
function StrToHex(S:String; Value:Integer):String;
begin
  result := IntToHex(StrToInt(S), Value);
end;
```

Prosté… A třeba maníky v Borlandu to (asi) nenapadlo, aby nám takto usnadnili práci…
Ale nepředbíhejme, jdeme si vyštrachovat funkci IntToStr.
Našli jsme a výpis je zde:

```
function IntToStr (Value: Int64): AnsiString; overload;
var
  result Result: AnsiString;
begin
   00000000 : 55                            PUSH EBP
   00000001 : 8B EC                         MOV EBP,ESP
   00000003 : 56                            PUSH ESI
   00000004 : 89 E6                         MOV ESI,ESP
   00000006 : 83 EC 20                      SUB ESP,32
   00000009 : 31 C9                         XOR ECX,ECX
   0000000B : 50                            PUSH EAX
   0000000C : 31 D2                         XOR EDX,EDX
   0000000E : 8D 45 08                      LEA EAX,DWORD PTR
[EBP+8{Value}]
   00000011 : E8(00 00 00 00                CALL CvtInt64{0x41A}
   00000016 : 89 F2                         MOV EDX,ESI
   00000018 : 58                            POP EAX
   00000019 : E8(00 00 00 00                CALL
@LStrFromPCharLen{0x18B}
   0000001E : 83 C4 20                      ADD ESP,32
   00000021 : 5E                            POP ESI
   00000022 : 5D                            POP EBP
   00000023 : C2 08 00                      RET NEAR,8
end;
```

Opět je výhodnější počítat s Int64. Vypustíme první dva a poslední dva řádky a zbyde nám toto:

```
function IntToStr(Value: Int64): String;
begin
asm
   PUSH   ESI
   MOV    ESI, ESP
   SUB    ESP,32
   XOR    ECX,ECX
   PUSH   EAX
   XOR    EDX,EDX
   LEA    EAX,Value
      CALL  CvtInt64
   MOV    EDX,ESI
   POP    EAX
      CALL  System.@LStrFromPCharLen
   ADD    ESP,32
   POP    ESI
end;
end;
```

Vlastně už nic neupravujeme, akorát přidáme před LstrFromPCharLen string „System.", vymažeme úplně nesmyslný „Var" a místo „AnsiString" použijeme jen „String".

Hon na StrToInt:

Zde bude potřeba menší úprava…

Dump:

```
function StrToInt (S: AnsiString): Integer;
var
  Result: Integer;
  E: Integer;
begin
   00000000 : 53                          PUSH EBX
   00000001 : 56                          PUSH ESI
   00000002 : 83 C4 F4                    ADD ESP,-12
   00000005 : 8B D8                       MOV EBX,EAX
   00000007 : 8B D4                       MOV EDX,ESP
   00000009 : 8B C3                       MOV EAX,EBX
   0000000B : E8(00 00 00 00              CALL @ValLong{0x1A4}
   00000010 : 8B F0                       MOV ESI,EAX
   00000012 : 83 3C 24 00                 CMP DWORD PTR [ESP],0
   00000016 : 74 19                       JE +25; (0x31)
   00000018 : 89 5C 24 04                 MOV DWORD PTR [ESP+4],EBX
   0000001C : C6 44 24 08 0B              MOV BYTE PTR [ESP+8],$0B
   00000021 : 8D 54 24 04                 LEA EDX,DWORD PTR [ESP+4]
   00000025 : A1(00 00 00 00              MOV EAX,DWORD PTR
[SInvalidInteger{0x4}[0]]
   0000002A : 33 C9                       XOR ECX,ECX
   0000002C : E8(00 00 00 00              CALL ConvertErrorFmt{0x317}
   00000031 : 8B C6                       MOV EAX,ESI
   00000033 : 83 C4 0C                    ADD ESP,12
   00000036 : 5E                          POP ESI
   00000037 : 5B                          POP EBX
   00000038 : C3                          RET NEAR
end;
```

Na adrese „12" se kontroluje délka stringu, který se má převést na integer, pokud je délka 0, tedy string = „", je skok na chybu, ta se ale volá na adrese „2C" (ještě k tomu z podcallu). Můžeme si to nechat

být, a celou podmínku zrušit, a při zavolání funkce s prázdným („“) stringem bude result „0“, nebo funkci JE přepsat tak, aby skákala někam na námi udělaný MessageBox, který říká, že „“ není platný integer. Naopak když se zavolá funkce s „Integerem“ kde je neplatný znak, třeba StrToInt(‚123TR54‘);, tak nám SysUtils řekne, že ‚123TR54‘ je neplatný integer. Pokud my neuděláme podmínku, převede se string na integer do prvního neplatného znaku, v tomto případě tedy „123“. Já si zvolím cestu první, tedy že i neplatný string se bude konvertovat:

```
function StrToInt (S: String): Integer;
asm
   PUSH EBX
   PUSH ESI
   ADD ESP,-12
   MOV EBX,EAX
   MOV EDX,ESP
   MOV EAX,EBX
      CALL System.@ValLong
   ADD ESP,12
   POP ESI
   POP EBX
end;
```

Nyní už funkce StrToHex:

```
function StrToHex(S:String; Value:Integer):String;
begin
  result := IntToHex(StrToInt(S), Value);
end;
```

Funkce IntToHex, IntToStr, StrToInt, StrToHex. Myslím že naše komponenta už je alespoň k něčemu a na rozdíl od SysUtils, zabere v binárce méně než 1 kB (SysUtils asi 25 kB). Příště si do ní přidáme pár funkcí a průběžně (nepravidelně) si tuto komponentu

budeme vylepšovat. Budu ji vydávat jak ve verzi „All in 1", tak v dílech – „DjHStringUtils". Příště se podíváme na práci jako je třeba otevírání a zavírání mechaniky a práce se soubory, bude se to jmenovat třeba „DjHDriveUtils" a „DjHFileUtils". V „All In 1" komponentě budou všechny unity spojené :). Je to jen pro to, že až toho v All in 1 bude hodně, budou třeba 3 kB použity v binárce úplně na nic. Proto si jen vyberete, jaké unity chcete :)

PS: Borland nám nedal zdrojáky k SysUtils!? My se nedáme! =), zde jste mohli vidět, jak jsme trochu Borlándka setřeli… Všechno jde… Jen, když se chce…

PS2: Takovýmto způsobem si můžete vytvořit vlastní komponentu, která má jen funkce, které vážně chcete, ale pozor! Když budete chtít takovéto zdrojové kódy dávat do světa, nezapomeňte do balíku se zdrojáky, přidat i tu Vámi vyrobenou komponentu, poněvač tu si počítač nedomyslí =), tu musíte FAKT přidat :

V balíku „balik18.rar" najdete:

- DelphiPacKage – DjH – To nejpoužívanější z unity SysUtils
- Extrahování resources do souboru [exe+src] – Delphi7 – NoVCL

Lekce č.19

Trocha programování v Delphi – komponenta

DjH

DÍL[2]

Minule jsme probrali myslím dostatečně práci se stringy. Nyní se pustíme do souborů. Ukážu Vám dvě funkce, které jsem sepsal, jak jinak :), kompletně já. Jedná se o funkce „ExtractResource" a „Run". První funkce uloží nějaký resource ze „sebe sama" do nějakého souboru. Fukce je takováto:

```
ExtractResource(FileName:String; Resource:PCHar;
DeleteIfExists: Boolean);
```

Myslím že není co dodat. Další funkce (procedura) je:

```
Run(FileName:string);
```

Jde o to, že se spustí jakýkoliv soubor v defaultním „prohlížeči toho souboru". Tedy pokud použiju „Run(,linkin park.mp3')" otevře se mi tato empétrojka ve WinAmpu nebo Mediaplayeru, prostě v defaultním přehrávači MP3jek :). To je ten rozdíl od API „WinExec", který umí spouštět jen *.exe, *.com atp… Funkce Run je vlastně tento řádek:

```
ShellExecute(0,'open',PChar(FileName),nil,nil,0);
```

Napsal jsem tuto funkci, protože tenhle zápis mě furt nebavil psát dokola :)

Za to na funkci ExtractResource jsem se „vyřádil na to tata". Nejdřív jsem tuto funkci složil ze „SysUtilovských" FileCreate, FileWrite… Potom jsem

si uvědomil, že by vlastně narostla exace o 25 kB, tak jsem použil assembler. Nějak jak jsem k tomu došel popisovat nebudu, mělo by to být jasné z komentářů :)

```
procedure ExtractResource(FileName:String; Resource:PCHar;
DeleteIfExists: Boolean);
//ExtractResource - Function by DjH2oo7
Var
  BytesWritten:Integer;
begin

  if DeleteIfExists = True then DeleteFile(PChar(FileName));  //Delete
the file, if exists

  CreateFile(PChar(FileName),
  GENERIC_READ + GENERIC_WRITE,
  FILE_SHARE_READ + FILE_SHARE_WRITE,
  nil,
  CREATE_NEW,
  FILE_ATTRIBUTE_NORMAL,
  0);                              //Create the file

  asm                             //Now we will work in assembler :)
                                PUSH EBX                //Push the exports
of
                                PUSH EDI                //func. CreateFile
                                MOV EDI,EAX             //EDI = EAX =
fileHandle
  end;
    FindResource(0,Resource,RT_RCDATA);   //Find resource
  asm
                                MOV EBX,EAX             //In EAX is
Address...

        push EAX
        push 0
        Call LoadResource       //Invoke LoadResource,0,eax - Get the
address, where resource starts

                                PUSH EAX
        push ebx
        push 0
        Call SizeofResource    //Invoke SizeOfResource,0,ebx,eax - Get
the size of res.
```

```
                              POP ECX                 //Get the handle

                              lea edx,BytesWritten    //BytesWritten -
Temp value :)

        push 0
        push edx
        push eax
        push ecx
        push edi
                              call WriteFile          //Invoke
WriteFile,edi,ecx,eax,edx,0

                              push edi
        call CloseHandle      //Close handle of file (EDI)

        pop  edi      //Zustanou nam ve stacku dve hodnoty,
        pop  edi      //musime je "vyPOPovat", jinak by RETN neskocil
spravne! :)
  end;
end;
```

Jak jednoduché, že?! :)
Uděláme si další čtyři funkce, na otevření a zavření mechaniky a na vypnutí a restartování počítače :). Tyhle čtyři funkce budou zvlášť v unitě DjHHWUtils (HW = HardWare :)). Unita je přiložená v balíčku…

```
//#########################################################################//
//########################OpenDrive#######################################//
procedure OpenDrive;
begin
  mciSendString('Set cdaudio door open', nil, 0, hinstance);
end;

//#########################################################################//
//########################CloseDrive######################################//
procedure CloseDrive;
begin
  mciSendString('Set cdaudio door closed wait', nil, 0, 0);
end;
```

A v Uses nesmí chybět MMSystem :). A reset a Turn Off:

```
//######################################################################//
//#####################ShutDownPC#######################################//
Procedure ShutDownPC;
begin
  ExitWindowsEx(EWX_Force,0); // potlačí hlášky systému
  ExitWindowsEx(EWX_SHUTDOWN,0); // vypnutí počítače
end;

//######################################################################//
//#############################RestartPC################################//
Procedure RestartPC;
begin
  ExitWindowsEx(EWX_Force,0); // potlačí hlášky systému
  ExitWindowsEx(EWX_REBOOT,0); // restart počítače
end;
```

Tyto dva díly Vám měli přiblížit vytváření vlastních komponent, naučit se používat vkládaný assembler v Delphi a naučit využívat DeDe naplno :). Neříkám, že tyhle funkce jsou potřeba, ale někdy se ocitne situace, kdy tyto funkce potřebujete. A proč tam ty funkce nedat, když v binárce zaberou stejně jen kolem 1 kilobajtu… Někdy se unita DjH prostě hodí :)

V balíku „balik19.rar" najdete:

- DelphiPacKage – DjH – To nejpoužívanější z unity SysUtils - Vylepšení

Lekce č.20

Keygenning po druhé

Už se kolem CrackMe ze stránek Programujte.com a hry 3D Live Pool motáme pořád do kola, co říkáte? Víceméně se jedná o naučný seriál, a ne o nějakou „Jak se stát warezmanem" :). Ale nebojte, postupně si toho crackneme více. Ale tedka se opět podíváme na CrackMe z první lekce, a vytvoříme si na něj keygen :). Pokusíme se o keygen v MASMu i v Delphi.

Nejdříve ale zapátráme, jak se vlastně s/n vypočítávalo:

```
00401167   .  33C0          XOR     EAX, EAX                //EAX=0
00401169   .  33C9          XOR     ECX, ECX                //ECX=0
0040116B   .  A1 D3304000   MOV     EAX, DWORD PTR DS:[4030D3]   //do EAX
dej to, co je v EDITu (první 4 znaky)
00401170   .  B9 5A415244   MOV     ECX, 4452415A   //ECX=4452415Ah
00401175   .  33C1          XOR     EAX, ECX                //Vyxoruj EAX
s ECX
00401177   .  33D2          XOR     EDX, EDX                //vynuluj EDX
00401179   .  BB 77770100   MOV     EBX, 17777              //EBX=17777h
0040117E   .  F7F3          DIV     EBX                     //vyděl EBX
00401180   .  35 86140000   XOR     EAX, 1486               //Vyxoruj EAX
s 1486h
00401185   . /75 1E         JNZ     SHORT crackme_.004011A5   //Jestli
není s/n správné skoč na badboy
00401187   . |6A 00         PUSH    0
00401189   . |68 CC304000   PUSH    crackme_.004030CC
0040118E   . |68 6A304000   PUSH    crackme_.0040306A……
```

Oživíme si trochu paměť z první lekce:
Dlouho jsem nechápal, jak může fungovat funkce DIV jen s jednou zdrojovou proměnnou. Stačí se mrknout do IronScrewa popisu ASM instrukcí (v balíku), a zjistíte, že DIV přesně znamená (a používá se…):
DIV zdroj - AL:=AX div zdroj; AH:=AX mod zdroj
…
kde

- **DIV – početní operace; dělení bez lomítka**
- **MOD – početní operace, která zjistí zbytek po dělení**

Tudíž, už v první lekci bylo řečeno:
Libovolné číslo z intervalu <1E19D44A, 1E1B4BC0> stačí xornout hodnotou 4452415Ah a vyjde nám číslo, které převedeme odzadu na ASCII a vyjde nám správný string…
Proč takový interval jsem taky vysvětloval:
1E19D44A – toto je vlastně 1486h * 17777h…
1E1B4BC0 – toto je 1E19D44A + 17776h, takový může být vlastně maximální zbytek v EDX, který s hodnotou 1486h v EAX nemá nic společného.

Stačí tedy deklarovat číslo 1E19D44Ah a k tomu přičíst hodnotu „Random($17777);" (Delphi - $17776 ne, protože čísla v závorce se stejně nikdy nedosáhne, maximální číslo je tedy $17777 – 1, tedy $17776, mohli bysme tedy napsat $17776 + 1 ale je to zbytečné :)), no a tuto hodnotu xornout 4452415Ah a pomocí vkládaného assembleru vypsat string. Keygen v Delphi by vypadal asi takto:

```
program Keygen;
uses
  Windows,DjH;

var
  Serial :int64;     //sn musi byt int64, aby mel pro sebe 8bytu (integer
ma 4, a nemel by ukonceny string hexnulou)
  PtrSn  :pointer;   //Pointer na Serial (push addr serial)
  Caption:pchar;     //Caption MsgBoxu
begin
Randomize;
  Caption := 'Keygen4CMe1-Programujte.com';
  serial := ((($1E19D44A) + Random($17777)) xor $4452415A);
  PtrSn := Addr(Serial);
  asm
    push 0
    push caption
    push PtrSn
    push 0
    call messagebox
  end;
end.
```

Po každém spuštění se Vám zobrazí platné sériové číslo. To ale může obsahovat i neplatné znaky, zkusíme si tedy ještě přidat kontrolu každého bajtu:

```
program Keygen;
uses
  Windows,DjH;

var
  Serial :int64;     //sn musi byt int64, aby mel pro sebe 8bytu (integer
ma 4, a nemel by ukonceny string hexnulou)
  PtrSn  :pointer;   //Pointer na Serial (push addr serial)
  Caption:pchar;     //Caption MsgBoxu
label start;         //pozn. autora:
begin                //JNGE cíl (je menší)
Randomize;           //JNLE cíl (je větší)
start:
  Caption := 'Keygen4CMe1-Programujte.com';
  serial := ((($1E19D44A) + Random($17777)) xor $4452415A);
  PtrSn := Addr(Serial);
```

```
  asm
    push edi
    xor edi,edi
//########################################//
@az:                                      //
    cmp byte ptr ds:[serial+edi], 'a'     //
    jnge @@AZ                             //
@az2:                                     //
    cmp byte ptr ds:[serial+edi], 'z'     //
    jnge @dale2                           //
//########################################//
@@AZ:                                     //
    cmp byte ptr ds:[serial+edi], 'A'     //
    jnge @09                              //
@@AZ2:                                    //
    cmp byte ptr ds:[serial+edi], 'Z'     //
    jnge @dale2                           //
//########################################//
@09:                                      //
    cmp byte ptr ds:[serial+edi], '0'     //
    jnge start                            //
@092:                                     //
    cmp byte ptr ds:[serial+edi], '9'     //
    jnge @dale2                           //
//########################################//
    pop edi                               //
    jmp start                             //
//########################################//
@dale2:
    inc edi
    cmp edi, 3
    jnz @az
    pop edi
  end;

  asm
    push 0
    push caption
    push PtrSn
    push 0
    call messagebox
  end;
end.
```

Jak vidíte, musíme začít kontrolovat od znaku s největší ASCII hodnotou, a postupně je zmenšovat.

Největší ASCII hodnotu mají ‚a' – ‚z', potom je ‚A' – ‚Z' a nakonec ‚0' – ‚9'. Pokud je v serialu nalezen byť jeden neplatný ASCII znak, vygeneruje se nové s/n a kontrola se provádí znovu, dokud neprojde v pohodě všemi kontrolami. Podobným způsobem bysme mohli udělat keygen i s oknem a tlačítkem „Generate", ale myslím, že stačí že je v balíku, je to velmi obdobné a vypisovali bysme pořád dokola ten stejný kód. V balíku je i „Bruteforcer". Není to sice přesný název, ale dejme tomu… V balíku jsou dva:

1. **uloží do souboru VŠECHNY platná s/n (i s netisknutelnými znaky)**
2. **uloží do souboru jen ty, která mají znaky tisknutelné…**

Nyní si uděláme keygen v MASM. V Delphi už jsme „nakousli" tuto část vkládaným assemblerem. Jestli používáte WinAsm studio (v 12. balíku) vytvořte si nový project s dialogem. Na dialog si dejte jeden button(ID_BTN – id=100) a jeden editbox(ID_EDT – id = 102). Po kliku na ID_BTN bude tato operace:

```
; ------------------------------------------------
;                    Po kliku na OK
; ------------------------------------------------
       call hash
       invoke SetDlgItemText, hWin, ID_EDT  , ADDR serial
       ;nastavi SNtext na SN
```

Kde hash, je tato funkce:

```
hash:
        ;randomize:
        ADD     ESP, -8
        PUSH    ESP                         ;/pPerformanceCount
        CALL    QueryPerformanceCounter  ;\QueryPerformanceCounter
        TEST    EAX, EAX
        JE      lbl1
        MOV     EAX, DWORD PTR SS:[ESP]
        MOV     DWORD PTR DS:[rndprom], EAX
        POP     ECX
        POP     EDX
        jmp rnd
lbl1:
        CALL    GetTickCount       ;[GetTickCount
        MOV     DWORD PTR DS:[rndprom], EAX
        POP     ECX
        POP     EDX

rnd:
        mov eax, 17777h
        ;call random:
        XOR     EBX, EBX
        IMUL    EDX, DWORD PTR DS:[EBX+rndprom], 8088405h
        INC     EDX
        MOV     DWORD PTR DS:[EBX+rndprom], EDX
        MUL     EDX
        MOV     EAX, EDX

        add eax, 1E19D44Ah
        xor eax, 4452415Ah

        mov dword ptr ds:[serial], eax

    xor edi,edi
```

```
;//########################################//
@az:                                       ;//
    cmp byte ptr ds:[serial+edi], 'a'      ;//
    jnge @@AZ                              ;//
@az2:                                      ;//
    cmp byte ptr ds:[serial+edi], 'z'      ;//
    jnge @dale2                            ;//
;//########################################//
@@AZ:                                      ;//
    cmp byte ptr ds:[serial+edi], 'A'      ;//
    jnge @09                               ;//
@@AZ2:                                     ;//
    cmp byte ptr ds:[serial+edi], 'Z'      ;//
    jnge @dale2                            ;//
;//########################################//
@09:                                       ;//
    cmp byte ptr ds:[serial+edi], '0'      ;//
    jnge rnd                               ;//
@092:                                      ;//
    cmp byte ptr ds:[serial+edi], '9'      ;//
    jnge @dale2                            ;//
;//########################################//
    jmp rnd                                ;//
;//########################################//
@dale2:
    inc edi
    cmp edi, 3
    jnz @az
ret
```

Kompletní kód je samozřejmě v balíku :). Co se týče funkce Randomize a Random(x);. Tyto funkce jsem, jak jinak, zjistil z DeDe, stačí si dekompilovat unitu System.dcu. Pokud chcete mít IntToStr, StrToInt atd. v MASMu, není problém si je tam takto přidat, vždyť je to vlastně assembler, a ten prostě fungovat musí…všude (třeba ‚Randomize' z Delphi funguje („dumplá") v MASM…).

PS: Tento keygen vytvořený v MASM má 3kB (3072b). UPX ho zkomprimovat už nedokáže, ale Upack ano, po zkomprimování Upackem, má program 1,46kB (1500b)!!!

Ještě můžeme do rozpoznávání uMsg dosadit tento řádek:

```
        .elseif uMsg == WM_INITDIALOG        ;API - OnCreate ;)
               call hash
               invoke SetDlgItemText, hWin, ID_EDT   , ADDR serial  ;hned
po startu je v editboxu nahodny serial :)
```

Myslím, že není co dodat k Msg wm_initdialog.

Po spuštění Bruteru, je jedno, ať už toho, co filtruje printable chars nebo ne, všimněte si jedné věci. Správné sériové číslo je vlastně ve tvaru „xxHZ", kde „xx" jsou úplně libovolné tisknutelné znaky. Tedy například „00HZ", „F8HZ", „4ZHZ", „ZHHZ"… Funguje i „xxIZ", ale s čísly je to horší. Ale aspoň platí, že i v tomto případě lze za „x" dosadit libovolné písmeno. Rázem se dostáváme asi na 1500 řešení (pomocí tisknutelných znaků, malá písmena se CMe automaticky převádějí na velká). Jinak asi na 20'000 (!) s použitím všech znaků (tedy i netisknutelných).

Myslím si, že tento díl nebyl až tak nezajímavý, jak by se mohlo zprva zdát :)…

V balíku „balik20.rar" najdete:

- **CMe ze stránek Programujte.com**
- **Keygen na toto CMe v:**
  - **MASM – [src{WinAsm project přibalený}+exe]**
  - **Delphi 7 – [exe+src]**
- **BruteForcer na toto CMe – dva typy:**
  - **PrintableOnly – Delphi(noVCL) [exe+src] – {výsledek asi 50kB řešení}**
  - **AllChars – Delphi(noVCL) [exe+src] – {výsledek asi 750kB řešení}**
- **Popis ASM instrukcí by Iron Screw**

Lekce č.21

Keygenning Crueheadova CrackMe

No tak vidíte, dočkali jste se nového CrackMe :). Tentokráte si vezmeme do latě Crueheadovo CrackMe. Já si myslím, že můžeme hned začít :). CrackMe si otevřte a vidíte, že máte před sebou jen menu, tak zvolte „Help" -> „Register" a trochu si ho můžete omakat :). Pokud ze zeptáme pana PeiDa, co na CrackMe říká, odpoví Vám MASM32/TASM32. Žádný crypter, žádný packer, můžeme jít na věc a natáhnout prográmek do Ollyho. CrackMe je velmi jednoduché, krátké, kód si můžete celý projet, máte to celé na dlani :). Na adrese 004012B5 můžete vidět, jak nám začíná funcke GetDlgItemTextA a pod ní je tato funkce další. Můžeme tedy určit, že se zde bere Name a Serial. Nastavíme BP na výše uvedenou adresu a zkusíme si spustit CMe. Do editu name napíšeme třeba „DjH2oo7" a do editu serial „123456" a stiskneme OK. Pokud budeme krokovat, program kamsi skočí, a dostaneme se sem:

```
004011E6   >  5B          POP     EBX
004011E7   . |5F          POP     EDI
004011E8   . |5E          POP     ESI
004011E9   . |C9          LEAVE
004011EA   . |C2 1000     RETN    10
```

A RETN nás hodí do knihovny USER32.DLL (můžete vyčíst ze stacku:

```
0012F4B8   77AB1A10  RETURN to USER32.77AB1A10
```

A tam už bysme se asi nevyznali, proto si najdeme adresu, kde už se pracuje se stringem. Popojedeme nahoru a vidíme:

```
00401223   .  83F8 00       CMP     EAX, 0
00401226   .^ 74 BE         JE      SHORT CRACKME1.004011E6
00401228   .  68 8E214000   PUSH    CRACKME1.0040218E  ;  ASCII "DjH2oo7"
0040122D   .  E8 4C010000   CALL    CRACKME1.0040137E  ;  vypocet cisla z
[name]
00401232   .  50           PUSH    EAX
00401233   .  68 7E214000   PUSH    CRACKME1.0040217E  ;  ASCII "123456"
00401238   .  E8 9B010000   CALL    CRACKME1.004013D8  ;  vypocet cisla z
[sn]
0040123D   .  83C4 04       ADD     ESP, 4
00401240   .  58            POP     EAX
00401241   .  3BC3         CMP     EAX, EBX    ;  hodnoty se porovnaji
```

nastavíme BP na adresu 00401223 a krokujeme. Call na adrese 0040122D vytracujeme pomocí [F7] a skočíme sem:

```
0040137E  /$  8B7424 04     MOV     ESI, DWORD PTR SS:[ESP+4]          ;
CRACKME1.0040218E
00401382  |.  56             PUSH    ESI
00401383  |>  8A06           /MOV     AL, BYTE PTR DS:[ESI]             ;
tato procedurka zajisti,
00401385  |.  84C0           |TEST    AL, AL                            ;
ze v editboxu pro
00401387  |.  74 13         |JE      SHORT CRACKME1.0040139C           ;
[name] budou jen velka pismena
00401389  |.  3C 41         |CMP     AL, 41
0040138B  |.  72 1F         |JB      SHORT CRACKME1.004013AC
0040138D  |.  3C 5A         |CMP     AL, 5A
0040138F  |.  73 03         |JNB     SHORT CRACKME1.00401394
00401391  |.  46            |INC     ESI
00401392  |.^ EB EF         |JMP     SHORT CRACKME1.00401383
00401394  |>  E8 39000000   |CALL    CRACKME1.004013D2
00401399  |.  46            |INC     ESI
0040139A  |.^ EB E7         \JMP     SHORT CRACKME1.00401383
0040139C  |>  5E            POP     ESI
0040139D  |.  E8 20000000   CALL    CRACKME1.004013C2                  ;
pokud jsou, skoc...
004013A2  |.  81F7 78560000 XOR     EDI, 5678
004013A8  |.  8BC7          MOV     EAX, EDI
004013AA  |.  EB 15         JMP     SHORT CRACKME1.004013C1
004013AC  |>  5E            POP     ESI
004013AD  |.  6A 30         PUSH    30                                 ;
/Style = MB_OK|MB_ICONEXCLAMATION|MB_APPLMODAL
004013AF  |.  68 60214000   PUSH    CRACKME1.00402160                  ;
|Title = "No luck!"
004013B4  |.  68 69214000   PUSH    CRACKME1.00402169                  ;
|Text = "No luck there, mate!"
004013B9  |.  FF75 08       PUSH    [ARG.1]                            ;
|hOwner
004013BC  |.  E8 79000000   CALL    <JMP.&USER32.MessageBoxA>          ;
\MessageBoxA
004013C1  \>  C3            RETN
```

Zde jsme zjistili, že v EditBoxu name, musí být jen písmena. Jak jsme k tomu došli?
Takto: AL, tedy aktuální znak, který tam přiřadila funkce MOV AL, BYTE PTR DS:[ESI] (z cyklu) se porovnává s hodnotou 41h, tedy 65 dec. což je v ASCII „A" a hodnota 5Ah je 90 dec, což je „Z", a pokud není v tomto rozhraní, CMe skočí sem:

```
004013D2  /$  2C 20         SUB     AL, 20
004013D4  |.  8806           MOV     BYTE PTR DS:[ESI], AL
004013D6  \.  C3            RETN
```

Tím se zajistí, že kdyby náhodou přišlo na malý znak, odečtením hodnoty 20h (32 dec) dostaneme znak velký, a podprogram skočí zpět. Tam pokračuje v kontrole, a pokud i po „akci podprogramu" není znak v tomto rozmezí, CMe skočí na badboi. Tedy číslice nepřipadají v úvahu…

Takže necháme CMe projet, a nastavíme jméno „DJH" a s/n necháme na „123456".
Nyní si dáme breakpoint na adresu 0040139D protože už jsme si jistí, že k tomuto skoku dojde. Cyklus necháme proběhnout [F9] a Olly se zastaví na tomto BreakPointu. Ten my vytracujeme [F7] a skočíme sem:

```
004013C2  /$  33FF          XOR     EDI, EDI                        ;
tato procedurka vypocita
004013C4  |.  33DB          XOR     EBX, EBX                        ;
ordinalni soucet ASCII znaku
004013C6  |>  8A1E          /MOV     BL, BYTE PTR DS:[ESI]          ;
v editu [name]
004013C8  |.  84DB          |TEST    BL, BL
004013CA  |.  74 05         |JE      SHORT CRACKME1.004013D1
004013CC  |.  03FB          |ADD     EDI, EBX
004013CE  |.  46            |INC     ESI
004013CF  |.^ EB F5         \JMP     SHORT CRACKME1.004013C6
004013D1  \>  C3            RETN
```

Myslím že kód je dostatečně popsaný. Cykl sečte ordinální hodnoty ASCII znaků. Je to něco jako:

```
For i:=1 to Length(Edit1.Text) do
begin
       soucet := soucet + ord(Edit1.Text[i]);
end;
```

Až cyklus skončí, a dorazí na „RETN", opíšeme si (třeba do poznámkového bloku) hodnotu z EDI. To je vlastně ta hodnota. Převeďte si ji (třeba pomocí Win kalkulačky) na decimální hodnotu, a tu si taky zapište. Se jménem „DJH" by v EDI mělo být D6h. Super, vše máme, krokujeme tedy dále:

```
0040139D  |.  E8 20000000   CALL    CRACKME1.004013C2               ;
pokud jsou, skoc...//program se nam vlastne vrati…
004013A2  |.  81F7 78560000 XOR     EDI, 5678                       ;
soucet vyxoruj s 5678h
004013A8  |.  8BC7          MOV     EAX, EDI                        ;
a uloz do EAX
004013AA  |.  EB 15         JMP     SHORT CRACKME1.004013C1
```

Po xoru EDI s 5678h si EDI znovu někam zapište (v hex. i v dec.). Po skoku se dosaneme zpět sem:

```
00401232   .  50           PUSH    EAX
00401233   .  68 7E214000   PUSH    CRACKME1.0040217E               ;
ASCII "123456"
00401238   .  E8 9B010000   CALL    CRACKME1.004013D8               ;
vypocet cisla z [sn]
```

A call na adrese 00401238 vytracujeme a dostaneme se sem:

```
004013D8  /$  33C0          XOR     EAX, EAX
004013DA  |.  33FF          XOR     EDI, EDI
004013DC  |.  33DB          XOR     EBX, EBX
004013DE  |.  8B7424 04     MOV     ESI, DWORD PTR SS:[ESP+4]
004013E2  |>  B0 0A         /MOV     AL, 0A                          ;
prevedeni cisla v [sn] na hex tvar
004013E4  |.  8A1E          |MOV     BL, BYTE PTR DS:[ESI]           ;
ktery se ulozi do registru EDI
004013E6  |.  84DB          |TEST    BL, BL
004013E8  |.  74 0B         |JE      SHORT CRACKME1.004013F5
004013EA  |.  80EB 30       |SUB     BL, 30
004013ED  |.  0FAFF8        |IMUL    EDI, EAX
004013F0  |.  03FB          |ADD     EDI, EBX
004013F2  |.  46            |INC     ESI
004013F3  |.^ EB ED         \JMP     SHORT CRACKME1.004013E2
004013F5  |>  81F7 34120000 XOR     EDI, 1234                       ;
cislo se vyxoruje s cislem 1234h
004013FB  |.  8BDF          MOV     EBX, EDI                        ;
a presune do EBX
004013FD  \.  C3            RETN
```

Myslím že není co dodat. Cyklus na adrese <004013E2 až 004013F3> vypočítá ze zadaného SN platné číslo, tedy z „123456" vytvoří v EDI „1E240h", což je vlastně oněch 123456 v dec. tvaru. Toto EDI se ovšem zase XORuje, tentokráte s hodnotou 1234h a vyxorovaná hodnota se přesune do EBX. RETN nás vrátí sem:

```
0040123D    .  83C4 04      ADD     ESP, 4
00401240    .  58           POP     EAX
00401241    .  3BC3         CMP     EAX, EBX                        ;
hodnoty se porovnaji
00401243    .  74 07        JE      SHORT CRACKME1.0040124C
```

A to už je jasné :). Takže to co nám vzniklo po xoru 5678h se musí rovnat číslu v editboxu serial. Toto číslo ale musí být zase vyxorováno hodnotou 1234h a převedeno na dec. Zní to složitě, ale opět je opak pravdou… Možná to pochopíte takto:

```
For i:=1 to Length(Edit1.Text) do
begin
  soucet := soucet + ord(Edit1.Text[i]);
end;
  soucet := soucet xor $5678;
  soucet := soucet xor $1234;
  Edit2.Text := IntToStr(Soucet);
```

Kde Edit1 je EditBox pro jméno a Edit2 pro serial. Vlastně máte před sebou už keygen v Delphi :). Takže když už keygen, tak procedura by po úplném zkrácení měla vypadat takto:

```
procedure ProcGen;
var
sName:string; //sName - String Name...
sz,j:integer; //sz = soucet ord znaku, i = docasna promenna
begin
sName := GetName;
  sz := 0;
    for j := 1 to i do
      begin
        sz := sz + ord(sName[j]);
      end;
      //sz := sz xor $5678;
      //sz := sz xor $1234;
      //5678h xor 1234h = 444Ch
      sz := sz xor $444C;
SetDlgItemText(mWnd,id_serial,pchar(IntToStr(sz)));
end;
```

Takto to je v API, keygen je přiložený i se zdrojáky v Delphi bez VCL. Vzhledem k tomu, že plánuju do

budoucna více keygenů, rozhodl jsem se i pro šablonu na keygeny v Delphi. Stačí si jen pozměnit proceduru pro výpočet a nějaké stringy, víceméně podívejte se sami do zdrojových kódů :)

Myslím si, že dnes byl díl jeden z nejzajímavějších, hlavně pro nováčky. Kód, co jsme projížděli v Olly jsem vysvětloval jen jednou protože si myslím, že kód byl pochopitelný velmi snadno. Kdyby náhodou jste pořád něco nechápali, pročtěte si článek ještě jednou, nebo tolikrát než to pochopíte. Vězte, že po delším začtení to pochopit musíte :)…

V balíku „balik21.rar" najdete:

- CMe ze od CrueHeada
- Keygen na toto CMe v Delphi 7 bez VCL- [exe+src]
- Source lze použít jako šablonu pro keygeny

Knihu napsal kompletně – DjH2oo7
Nejedná se (snad) o poslední verzi knížky, čekejte další! S dalšími lekcemi!:-)
ps: lekce, soubor a logo jsou pod autorským právem DjH2oo7! :)
Více informací na www.studioadler.wz.cz, www.soom.cz
a otázky a připomínky uvítáme na e-mailu DjH@soom.cz
nebo na ICQ 319-960-895

Knížka založena :27.07.2oo7 v [22:10]
Tato verze uložena :03.10.2oo7 v [23:14]

Stat:		
Stránek	-	148
Slov	-	24 796
Znaky bez mezer	-	121 886
Znaky s mezerami	-	170 362
Řádky	-	5 069