The Art Of Cracking

Got Cracked ?

by Ap0x

The art of cracking

(for newbies and advanced crackers)

by Ap0x

Predgovor

Mi zivimo nase svakodnevne zivote nesvesni sitnica i malih stvari koje nam se desavaju ispred ociju. Prelazimo preko ociglednih stvari jer nam se cine jednostavnim i logicnim. Ne osecamo potrebu da zavirimo ispod jednolicne jednostavne spoljasnosti stvari, ili samo to ne zelimo? Postavimo sebi pitanje: Kada smo zadnji put pogledali neki objekat i zapitali se kako on to radi? Kakvi se to procesi kriju iza njegovog nama dostupnog lica? Zasto se nesto desava bas tako kako se desava? Kada smo neki objekat poslednji put razlozili na sastavne delove snagom svojih misli? Kada smo zadnji put videli iza ociglednog i dostupnog? Odgovori na ova pitanja leze u nama samima i predstavljaju samu osnovu reversnog inzinjeringa. Sama teznja da se pronikne u pocetke i uzroke stvari, da se do pocetnih uslova dodje od rezultata otvara neverovatne mogucnosti, samo ako promenimo tacku gledista, samo ako se sa mesta pasivnog posmatraca pomerimo u mesto reversera, samo tako cemo doci do same srzi reversnog inzinjeringa. Imajte na umu da se reversni inzinjering ne primenjuje samo na kompjuterima, svuda oko nas je reversni inzenjering, samo to treba da uocimo.

Knjiga je posvecena svim ljudima koji su ostavili neizbrisiv trag u mom zivotu: porodici, najboljim prijateljima, prvoj ljubavi, mentorima, ostalim prijateljima, neprijateljima i ostalim koji nisu ovde nabrojani ali predstavljaju manje ili vise bitan deo mog zivota.

“I am not cracking beacause of the money and the girsl, the devil made me do it :)”

Ap0x

01 Intro to cracking

Kao prvo poglavlje u knjizi ovo poglavlje ima za cilj da vas uvede u reversni inzenjering i da vam pokaze samu osnovu crackovanja, nacin razmisljanja i neke osnovne trikove sa alatima za cracking. Prvo cete usvojiti neke osnovne pojmove vezane za cracking, naucicete kako se konfigurisu alati koje cemo koristiti, i konacno cemo napraviti nas prvi crack.

What is R.C.E.?

Reverse Code Engineering je tehnika pomocu koje dobijamo pocetne vrednosti neke funkcije pocinjuci od njenih rezultata. Primeticete da sam upotrebio uopsenu definiciju RCEa, a ne onu koja bi se odnosila na RCE primenjen na compjuterske aplikacije. RCE sam definisao ovako jer je on upravo to, bez obzira na oblast na koju se primenjuje RCE prdstavlja tehniku razmisljanja, odnosno postupak resavanje nekog problema iz drugog ugla. Ali ako se ogranicnimo samo na kompjutere onda cemo RCE definisati kao tehniku modifikacije nepoznatog koda, kada izvorni kod samog programa nije dostupan. Koristeci tehnike opisane u ovoj knjizi cete shvatiti osnovne probleme pri modifikaciji ili analizi nepoznatog koda kada nije poznat izvorni kod samog problema. Tada moramo pristupitu reversnom posmatranju problema, to jest trazenja uzroka razlicitih ponasanja programa, pocevsi od samih posledica, da bi smo stigli do pocetnik uzroka. Naravno kao i svaka druga oblast ljudskog delovanja, pa tako i u RCEu imamo razlicite probleme koji u vecini slucaja nemaju jedinstvena resenja. Ovo ujedno znaci da se vecina problema moze resiti na veliki broj nacina i da u vecini slucajeva postoje samo najlaksa, odnosno najbrza resenja, i ona koja to nisu. Posto nas u vecini slucajeva nezanima vreme potrebno da se neki problem resi, glavni faktor u resavanju RCE problema ce biti tacnost dobijenih rezultata. Ova tacnost je surova kada se radi o RCE problemima, jer u vecini slucajeva postoje samo dva slucaja resavanja problema. Postoje samo tacno reseni problemi i oni koji to nisu. RCE problemi koji su netacno reseni mogu dovesti do nestabilnosti sistema, kao i do pucanja samog operativnog sistema koji koristimo kao osnovu za RCE. Ovo ujedno znaci da RCE nije platformski definisan jer se moze primenjivati, a i primenjuje se, na svim kompjuterskim platformama.

Iako je ova knjiga “pompezno” nazvana The Art Of Cracking ona se ne odnosi na pravo znacenje reversnog inzinjeringa, niti je tako nesto uopste moguce. Ne, ova knjiga je sebi stavila za cilj da se ogranici samo na jednu usko definisanu oblast, popularno nazvanu Cracking, tezeci da opise sto je vise moguce pojava vezanih za sam Cracking. Naravno i ovo je tesko izvodljivo ali cu se potruditi da vas kroz ovu knjigu uverim da nepostoji stvar koja se naziva “sigurna aplikacija”. Od sada pa na dalje ovaj termin izbrisite iz svoga recnika. Termin koji cete usvoji umesto toga je: “aplikacija teska za reversing!”, sto znaci da svaka aplikacija koja moze da se pokrene moze biti “slomljena” i da netreba verovati takozvanim komercijalnim aplikacijama za zastitu vasih ili tudjih programa. Ova knjiga ce unistiti zablude onih koji veruju da su njihovi passwordi bezbedni u bazama podataka, da su njihovi passwordi bezbedni iza “zvezdica”. Ovakve zablude ce pasti u vodu posle citanja ove knjige. Programeri spremite se, jer vase aplikacije ce biti stavljene na obiman test...

Tools of Trade

Kao i za svaku drugu delatnost na kompjuteru za reversni inzenjering su vam potrebni neki alati (*programi*) kako bi mogli brze i lakse da dodjete do informacije koja vam treba. Vecina alata koje cu vam ja ovde preporuciti mogu se besplatno preuzeti sa interneta i kao freeware proizvodi koristiti i distribuirati. Pre nego sto pocnem sa listom programa koje ce te morati preuzeti sa interneta prvo cu u par recenica objasniti kakvi su nam to alati potrebni i za sta oni sluze.

Debugger – Ovo je osnovni alat svakog crackera ali i svakog programera koji zeli da brzo i lako eliminise greske iz svog koda. Ono sto nam debugger pruza je mogucnost da nadgledamo izvrsavanje naseg ili tudjeg koda bas onako kako ga procesor vidi. Da, to znaci da ce te morati da nauciti osnove assemblera (*masinskog jezika*) kako bih mogli da razumete i kontrolisete izvrsavanje koda. Ovakav tekst sam vec napisao i objavio na sajtu *www.EliteSecutiry.org* a nalazi se i u ovom izdanju knjige na strani 8 (*ovo je dopunjeno izdanje*) cije je citanje neophodno radi lakseg snalazenja i razumevanja tekstova iz ove knjige.

Dissassembler – Ovo je dodatni alat za debugger. Naime ako vam debugger ne da dovoljno informacija o "meti" onda mozete koristiti neke od dissassemblera kako bi lakse uocili informaciju koja vam treba. Sa vremenom ce te sve manje koristiti ovaj alat posto ce te se naviknuti na asemblerov kod tako da vam ovi alati nece biti potrebni.

PE identifikatori – Ne dajte da vas ovaj naslov zbuni, PE fajlovi su samo obicni exe fajlovi koji mogu da sadrze neki dodatan kod, koji je najcesce paker koji sluzi za smanjenje velicine exe fajla. Posto postoji veliki broj ovakvih pakera i enkriptera pa su potrebni posebni programi za prepoznavanje istih. Naravno ovo se donekle moze raditi i runo sto cu vas takodje nauciti.

Hex Editori – su alati koji nam daju tacan izgled fajla na hard disku i sluze za fizicko menjanje koda za razliku od menjanja koda u memoriji sto nam omogucuju debuggeri.

Resource Vieweri – Sluze za pregled, ekstakciju, izmenu ili dodavanje exe resursa. Resursi su podaci koji su ukljuceni u sam exe fajl a mogu biti slike, dijalozi, multimedija, stringovi ili neki drugi tipovi podataka. Ovi programi u principu nisu neophodni ali nam mogu olaksati posao.

Process dumperi – Sluze prevashodno kod odpakivanja zapakovanih PE fajlova i omogucavaju nam da celokupnu "sliku" nekog aktivnog programa snimimo na hard disk.

Import rekonstrukteri – su programi koji sluze za popravljanje pogresnih ili nedefinisanih poziva ka windows api funkcijama.

ASM basics

[Registri]

Registri su osnovna mesta gde programi cuvaju svoje podatke. Ti podaci mogu biti tekst ili brojevi. Kada govorimo o AMSu moramo znati da se ne govori o obicnim decimalnim brojevima nego da procesor radi sa Hex t.j. hexadecimalnim brojevima (0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F).

Najvazniji registri za crackere su (E)AX,(E)BX,(E)CX,(E)DX - (E)AX = EAX ili AX u zavisnosti da li radimo sa 16-bit (AX) ili 32-bitnim (EAX) programima. Takodje postoje parovi registera koji se koriste za uporedjivanje stringova (obicnog teksta)

```
DS:SI   ; Koriste se kao osnova za operacije sa stringovima
ES:DI   ; Koriste se kao rezultat za orepacije sa stringovima
```

Registri su najvazniji deo kod razumevanja ASMa, ali dovoljno je da ih shvatite kao mesta na kojima se cuvaju vrednosti promenjivih (variabli)

```
Primer(1):
----------
MOV EAX, 1          <- Dodeli vrednost 1 EAXu
MOV EBX, 2          <- Dodeli vrednost 2 EBXu
SUB EBX, EAX        <- Oduzmi od EBXa EAX
RET                 <- Vrati se
```

Ovo gore je ekvivalentno matematickim izrazima:

```
EAX = 1
EBX = 2
EBX = EBX - EAX
```

samo sto je izrazeno tako da procesor moze da razume sta nas program zeli da uradi.

[Flags]

Flags ili zastave su slicne registrima samo sto mogu da drze samo dve vrednosti (0 ili 1/True ili False) Flags ustvari drze logicke promenjive, a da li ce one sadrzati vrednosti 0 ili 1 zavisi iskljucivo od ASM operacija koje se koriste za uporedjivanje dva registra.

```
Primer(2):
----------
CMP BX,AX
; Uporedi BX sa AX, ako su BX i AX jednaki onda je Flag = 1 u suprotnom je Flag = 0
JNE 00402323
; Ako BX i AX nisu jednaki (Ako je Flag = 0) onda skoci na hex adresu 0040232
```

Ovo gore je jednostavno uporedjivanje registera koje se jako cesto pojavljuje u fajlu posto se u samim programima cesto uporedjuju vrednosti. Ovo je takozvano programsko grananje...

```
Primer(3):
----------
IF BX = AX THEN .....
```

Ovo je tipican primer kako se u programskim jezicima uporedjuju vrednosti a u ASMu je ekvivalentan primeru 2.

[STACK]

Pre bilo koje funkcije t.j. calla u ASMu, toj funkciji se moraju proslediti vrednosti potrebne za nju samu. Ovo jednostavno znaci da se svaka funkcija t.j. call sastoji od odredjenog broja parametara koji se prvo slazu na stack, a tek kada su svi parametri prosledjeni funkcija se poziva i izvrshava. Razmisljajte o ovome kao o gomili ploca naredjanih jedna na drugu. Vi polako skidate jednu po jednu sa vrha to jest od poslednje krecete a zavrshavate na prvoj. E tako se parametri i prosledjuju funkciji, u obrnutom redosledu - od poslednjeg do provog.
U slucaju da vam ovo nije jasno sledi primer:

```
Primer(4):
----------
Windowsova Api funkcija GetDlgItemText zahteva sledece parametre:
(1) Handle dialog boxa
(2) Identifikator kontrole iz koje se cita tekst
(3) Adresa u koju se smesta tekst
(4) Maximalna duzina teksta
```

Stoga ce iscitavanje teksta u ASMu izgledati ovako:

```
MOV  EDI,[ESP+00000220]          ; Handle dialog boxa se smesta u EDI
PUSH 00000100                    ; PUSH (4) Maximalna duzina teksta
PUSH 00406130                    ; PUSH (3) Adresa u koju se smesta tekst
PUSH 00000405                    ; PUSH (2) Identifikator kontrole iz koje se cita tekst
PUSH EDI                         ; PUSH (1) Handle dialog boxa
CALL GetDlgItemText              ; CALL the function
```

Parametri potrebni funkciji se njoj prosledjuju u reversnom redu.

[Spisak komandi]

```
AND
---
Komanda  :  AND meta,izvor
Sluzi za :  Izvodi logicku operaciju AND.
            Meti se logicki dodaje izvor (META = META AND IZVOR)
Primer   :  AND BX, 04h
--------------------------------------------------------------------------------
ADD
---
Komanda  :  ADD meta,izvor
Sluzi za :  Izvodi matematicku operaciju sabiranje.
            Meti se dodaje izvor (META = META + IZVOR)
Primer   :  ADD BX, 04h
--------------------------------------------------------------------------------
CALL
----
Komanda  :  CALL hex_adresa
Sluzi za :  Poziva odredjenu funkciju koja se nalazi na toj hex adresi
Primer   :  CALL 00401000

Kada se taj CALL zavrshi program nastavlja sa izvrshavanjem odmah ispod CALLa.
--------------------------------------------------------------------------------
```

CMP

Komanda : CMP meta,izvor
Sluzi za : Uporedjuje metu i izvor, ako su oni razliciti Flag postaje jednak 0 a ako su isti 1
Primer : CMP BX, AX

DIV

Komanda : DIV meta,izvor
Sluzi za : Izvodi matematicku operaciju deljenje.
Meta se deli sa izvorom (META = META / IZVOR)
Primer : DIV BX, AX

SUB

Komanda : SUB meta,izvor
Sluzi za : Izvodi matematicku operaciju oduzimanje.
Od mete se oduzima izvor (META = META - IZVOR)
Primer : SUB BX, AX

IMUL

Komanda : IMUL meta,izvor
Sluzi za : Izvodi matematicku operaciju mnozenje.
Meti se mnozi sa izvorom (META = META * IZVOR)
Primer : IMUL BX, AX

TEST

Komanda : TEST meta,meta
Sluzi za : Proverava da li je meta uneta onda je Flag jednak 1 ili ako nije onda je
Flag jednak 0
Primer : TEST EAX,EAX

JMP

Komanda : JMP hex_adresa
Sluzi za : Ekvivalentaj je BASIC komandi GOTO i sluzi za preskakanje odredjenog
dela koda
Primer : JMP 00401000

JMP je bezuslovni skok, t.j. on se izvrshava bez obzira na vrednos Flaga ili na neku drugu vrednost.
Postoji vishe vrsta uslovnih skokova:

Hex:	Asm:	Znaci
75 or 0F85	jne	skoci ako nije jednako
74 or 0F84	je	skoci ako je jednako
EB	jmp	bezuslovni skok
77 or 0F87	ja	skoci ako je iznad
0F86	jna	skoci ako nije iznad
0F83	jae	skoci ako je iznad ili jednako
0F82	jnae	skoci ako nije iznad ili jednako
0F82	jb	skoci ako je ispod
0F83	jnb	skoci ako nije ispod
0F86	jbe	skoci ako je ispod ili jednako
0F87	jnbe	skoci ako nije ispod ili jednako
0F8F	jg	skoci ako je vece
0F8E	jng	skoci ako nije vece
0F8D	jge	skoci ako je vece ili jednako
0F8C	jnge	skoci ako nije vece ili jednako
0F8C	jl	skoci ako je manje
0F8D	jnl	skoci ako nije manje
0F8E	jle	skoci ako je manje ili jednako
0F8F	jnle	skoci ako nije manje ili jednako

Skokovi obicno prate funkciju CMP ili TEST.

```
MOV
---
Komanda   :  MOV meta,izvor
Sluzi za  :     Dodeljuje vrednost izvora meti.
             Ekvivalentno matematickom izrazu META = IZVOR
Primer    :     MOV EAX, EBX
Postoji vishe varijanti operacije MOV ali sve one rade istu stvar.
-------------------------------------------------------------------------------------------
OR
--
Komanda   :  OR meta,izvor
Sluzi za  :     Izvodi se logicka operacija OR a meti se dodeljuje dobijena vrednost
             (META = META OR IZVOR)
Primer    :     OR BX, AX
-------------------------------------------------------------------------------------------
XOR
---
Komanda   :  XOR meta,izvor
Sluzi za  :     Izvodi se logicka operacija XOR a meti se dodeljuje dobijena vrednost
             (META = META XOR IZVOR)
Primer    :     XOR BX, AX

Najcesce se koristi za enkriptovanje vrednosti, posto je revezibilna funkcija.

40 Xor 2 = 42;    42 xor 2 = 40;
-------------------------------------------------------------------------------------------
RET
---
Komanda   :  RET / RETN
Sluzi za  :     Oznacava kraj funkcije i izlazenje iz nje tako da se nastavi sa izvrshavanje koda
             ispod calla
Primer    :     RET
-------------------------------------------------------------------------------------------
```

NAPOMENA: Ako zelite da razumete sve sto se nalazi u knjizi razumevanje osnovnih ASM komandi je neophodno za dalje citanje knjige. Ako niste sve razumeli ili ste nesto propustili predlazem da se vratite i procitate ovaj deo knjige opet.

Reading time table:

Chapter	Required Level	Minimum reading time	
Intro to Cracking	newbie	1 day	1 week
NAG Screens	newbie	1 day	1 week
Cracking Serials	newbie	2 days	1 week
Making KeyGens	advanced newbie	4 days	1 week
CD Checking	advanced newbie	1 day	3 days
Code Hacking	advanced coder	1 day	1 week
"Getting caught"	advanced newbie	3 days	1 week
Cracking it	newbie	2 days	1 week
Decrypt me	expert coder	1 week	3 weeks
Unpacking	advanced newbie	1 week	3 weeks
Patching	advanced newbie	1 day	3 days
Nightmare	advanced cracker	1 week	4 weeks

Download links:

Debugger – OllyDBG v1.10 http://home.t-online.de/home/Ollydbg
Dissasembler - W32dsm89 http://www.exetools.com
PE identifikator – PeID http://peid.has.it
Hex Editor – Hiew 6.83 http://www.exetools.com
Resource Viewer - Res. Hacker http://rpi.net.au/~ajohnson/resourcehacker
Import rekonstrukter - Import Reconstructer 1.6 http://www.exetools.com
Process dumper – LordPE http://y0da.cjb.net

Ostali alati:
.ap0x Patch Creator – http://ap0x.headcoders.net

Napomena: Ovo su samo neki alati koji ce biti korisceni u knjizi. Njih smatram osnovnim i pre nego sto pocnete sa daljim citanjem ove knjige trebalo bi da ih nabavite.

Configuring Tools of Trade

Vecina gore navedenih alata je vec konfigurisana kako treba i stoga samo treba da promenimo izvesne sitnice kako bih jos vise olaksali sebi rad sa ovim alatima.

OllyDBG v.1.10

Podesicemo sledece opcije u OllyDBG programu. Otvorite Olly i idite u meni za konfiguraciju debuggera **[Hint:** Options -> Debugging options, ili Alt + O **]** Idemo na Exeptions i tu treba da iskljucimo sve otkacene opcije. Dalje idemo na Strigs i tamo treba da odkacimo: Decode Pascal strings i Dump non printable ASCI codes as dots, od mode of string decoding opcija izaberite Plain opciju. U Disasm opciji kao sintaksu izaberite MASM, a u CPU tabu selektujte sledece: uderline fixups, show directions of jumps, show jump path, show grayed path if jump is not taken, show jumps to selected command.

Sledece sto mozemo da podesimo je takozvana kolor sema koja nam omogucava da lakse uocimo odredjene komande u dissasemblovanom kodu, kao sto su skokovi i call funkcije. Idemo na Options -> Appearance i tamo u provom tabu **[Hint:** General **]** treba da iskljucimo restore window position and appearance. Dalje u defaults mozete izabrati temu koja vam odgovara. Ja preferiram Black on White kao temu, a Christmas tree kao highlithing sintakse. Vi mozete izabrati bilo koji ali su po mom misljenju ova dva najpreglednija. To je sve sto treba da se konfigurise u Olly.

Napomena: OllyDBG je samo jedan od velikog broja debuggera koje mozete naci na internetu. Izmedju ostalih popularnih mozete jos koristiti i SoftIce, TRW2000 ili neke debuggere koji se koriste specijalno za odredjenje vrste kompajlera (programske jezike kao sto su Visual Basic, Delphi, .Net itd.), ali samo je Olly besplatan, univerzalan i jednostavno najbolji debugger za pocetnike ali i za iskusne korisnike. Preporucujem da ga koristite jer ce se dalje u ovoj knjizi on veoma cesto pominjati i koristiti.

Numega Smart Check v.6.03

Smart Check je specijalni debugger koji se koristi za trazenje gresaka u programima koji su pisani u Visual Basicu (verzije 5 i 6). Jako je koristan a u vecini slucajeva je pregledniji i laksi za obradu podataka od Olly-a. Stoga ako je program koji pokusavate da "razbijete" pisan u Visual Basicu prvo probajte sa Smart Checkom pa tek onda sa ostalim alatima.

Idemo na *Program -> Settings -> Error detection*
Otkacite sve checkboxove osim *Report Errors immediatly*.

Idemo na *Advanced*:
Otkacite samo sledece:

- *Report errors caused by other errors*
- *Report errors even if no source code available*
- *Report each error only once*
- *Check all heap blocks on each memory function call*

Sve ostalo NE treba da bude otkaceno, stisnemo [OK] pa idemo dalje...

Idemo na *Reporting*:
Otkacimo sve osim - *Report MouseMove events from OCX*, [OK], zatvorimo ovaj meni za konfiguraciju.

Idemo na *Program -> Event Reporting*

Pritisnite zelenu strelicu u toolbaru da pokrenete selektovani program. Ovo moze biti bilo koji program pisan u Visual Basicu i onda samo treba da odkacimo sledece menije:

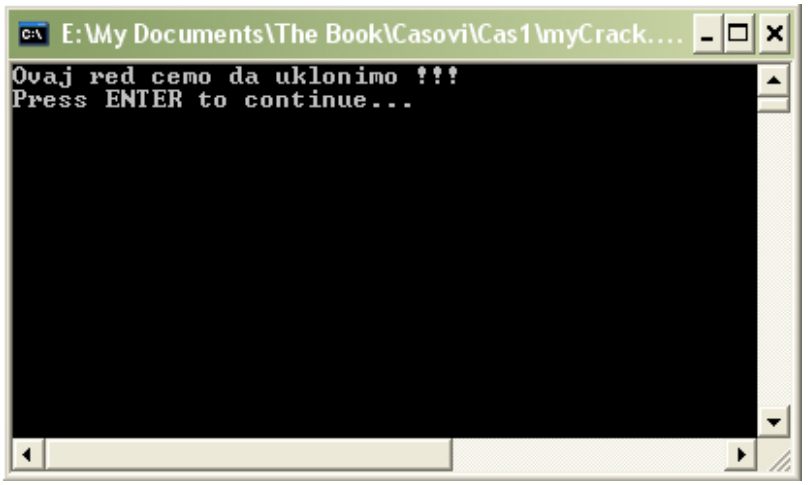
View -> Arguments ...
View -> Sequence Numbers ...
View -> Suppresed Errors ...
View -> Show errors and specific…
Window -> [2] ... - Program Resaults

PeID v.0.92

PeID je program koji cemo dosta cesto koristiti kroz ovu knjigu a kao sto mu ime fajla kaze njegova glavna upotreba je da nam pokaze informacije o "meti" koju pokusavamo da reversujemo. Ti podaci ukljucuju verziju kompajlera kao i da li je program pakovan nekim pakerom i kojom verzijom ako jeste. Ovaj program je po defaultu podesen kako treba ali cemo samo par stvari podesiti kako bi sebi olaksali zivot. Upalite program i idite na opcije. Tamo treba da selektujete *Hard Core Scan* i *Register Shell Extensions*. Kliknite na Save i program je uspesno konfigurisan.

My first Crack

Posto smo konfigurisali alate koji su nam potrebni sada cemo krenuti sa njihovim koristenjem. Startujte myCrack.exe koji se nalazi u folderu ...\Casovi\Cas01. Ono sto vidimo je sledeca poruka na ekranu:



Kao sto se vidi na slici ovo ce biti jednostavan primer u kome cemo samo ukloniti prvi red, tako da ce program na ekran izbacivati samo poruku koja se trenutno nalazi u drugom redu.

Ovaj zadatak mozda izgleda komplikovano pocetnicima ali uz pomoc prvih par alata necemo imati nikakvih problema da uspesno i brzo resimo ovaj problem. Za resenje ovog problema trebace nam samo dva alata: W32Dasm i Hiew. Ali pre nego sto pocenemo sa crackovanjem moram da vam objasnim same osnove crackovanja.

Crackovanje je tehnika menjanja vec kompajlovanih (*gotovih*) exe (*ali i drugih tipova*) fajlova. Modifikacija fajlova se radi na asemblerskom nivou, sto znaci da je pozeljno poznavati osnovne principe funkcionisanja asembleskih komandi. Naravno ovo nije i obavezno jer mozemo da znamo sta radi neka asemblerska komanda pomocu logickog razumevanja same komande. Ovo na najjednostavnijem primeru znaci da logikom mozemo doci do zakljucka sta radi ova komanda: MOV EAX,1... Naravno pogadjate da je ova komanda ekvivalentna matematickoj funckiji EAX = 1. Osnovna komanda koja ce nam trebati za ovaj prvi cas je NOP. Ona je osnovna crackerska komanda i znaci da se na redu u kome se ona nalazi ne desava bas nista. To znaci da ce program proci preko nje i nece uraditi bas nista. Zasto nam je ovo bitno??? Prilikom crackovanja postoje komande koje zelimo da izmenimo ili da obrisemo. Posto u crackovanju nije pozeljno brisati deo fajla jer bi tako nesto ispod reda koji brisemo moglo da se poremeti, zato koristimo ovu NOP komandu da izbrisemo mesta koja nam ne trebaju. Naravno ova mesta nece biti fizicki izbrisana nego program jednostavno nece raditi nista na mestu na kojem se nalazila komanda koju smo obrisali i nastavice dalje. Prikazano asemblerski to izgleda ovako. Ako imamo sledece asemblerske komande:

```
MOV EAX,1
INC EAX
ADD EAX,5
```

mozemo da predpostavimo sta se ovde desava. Matematicki to izgleda ovako:

```
EAX = 1
EAX = EAX + 1
EAX = EAX + 5
```

Mislim da je svima jasno koji je rezultat EAXa posle izvrsavanja ove tri komande. Recimo da nam taj rezultat ne odgovara i da bi smo hteli da EAX na kraju izvrsavanja ove tri komande bude manji za jedan nego sto je sada. Jednostavno cemo spreciti da se EAXu doda 1 u drugom redu i resicemo problem. Sada bi to izgledalo ovako:

```
MOV EAX,1
NOP
ADD EAX,5
```

Vidmo da smo jednostavno NOPovali drugi red i da se on sada ne izvrsava kao INC EAX nego kao NOP, zbog cega se rezultat EAXa ne menja posle izvrsavanja tog reda.

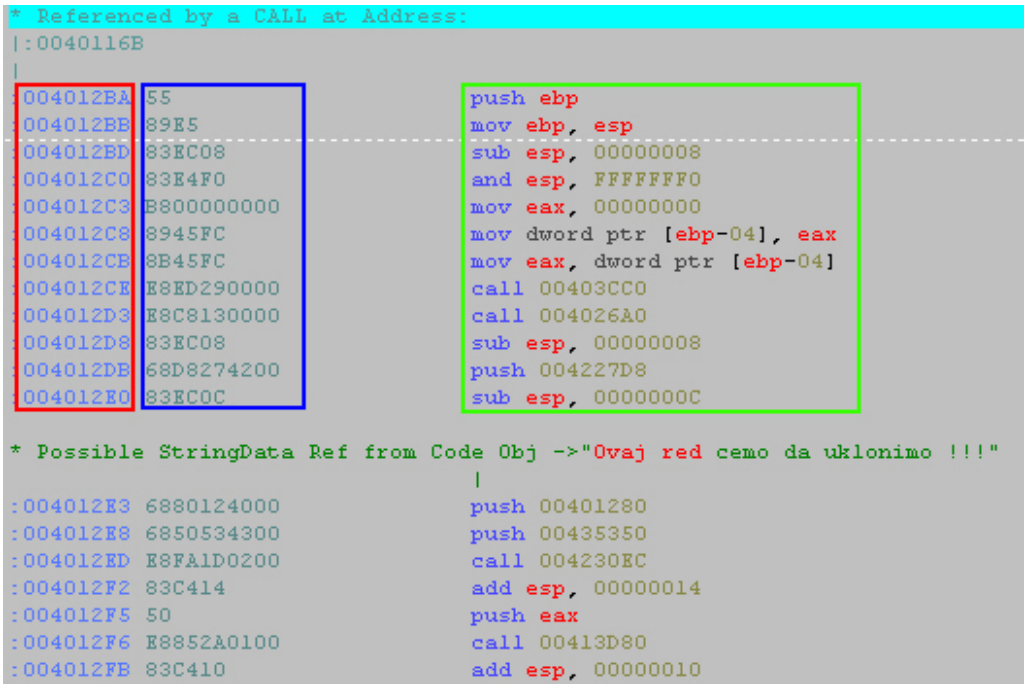
Prvo otvorimo ovu “metu” u W32Dasmu i u njemu potrazimo tekst koji zelimo da uklonimo. Kada se program ucita u W32Dasm onda preko Find opcije potrazimo tekst poruke koju treba da uklonimo. Idemo na *Search -> Find -> Ovaj red -> OK.* Videcemo ovo:

```
* Referenced by a CALL at Address:
|:0040116B
|
:004012BA 55                    push ebp
:004012BB 89E5                  mov ebp, esp
:004012BD 83EC08                sub esp, 00000008
:004012C0 83E4F0                and esp, FFFFFFF0
:004012C3 B800000000            mov eax, 00000000
:004012C8 8945FC                mov dword ptr [ebp-04], eax
:004012CB 8B45FC                mov eax, dword ptr [ebp-04]
:004012CE E8ED290000            call 00403CC0
:004012D3 E8C8130000            call 004026A0
:004012D8 83EC08                sub esp, 00000008
:004012DB 68D8274200            push 004227D8
:004012E0 83EC0C                sub esp, 0000000C

* Possible StringData Ref from Code Obj ->"Ovaj red cemo da uklonimo !!!"
                                 |
:004012E3 6880124000            push 00401280
:004012E8 6850534300            push 00435350
:004012ED E8FA1D0200            call 004230EC
:004012F2 83C414                add esp, 00000014
:004012F5 50                    push eax
:004012F6 E8852A0100            call 00413D80
:004012FB 83C410                add esp, 00000010
```

Kao sto vidimo prvi red ispod nadjenog teksta je zaduzen za prikazivanje poruke na ekran. Ono sto zakljucujemo je da ako samo NOPujemo taj red onda cemo ukloniti poruku sa ekrana. POGRESNO !!! Ovde se namece jedan problem, ako NOPujemo samo taj red onda cemo dobiti gresku u programu i program ce se srusiti jer ga nismo pravilno modifikovali. Problem lezi u cinjenici da je za prikazivanje teksta na ekranu zaduzeno vise redova. Shvatite ovo kao funkciju koja ima vise parametara. Na primer:

prikazi_tekst_na_ekran(parametar1, parametar2, parametar3,...)

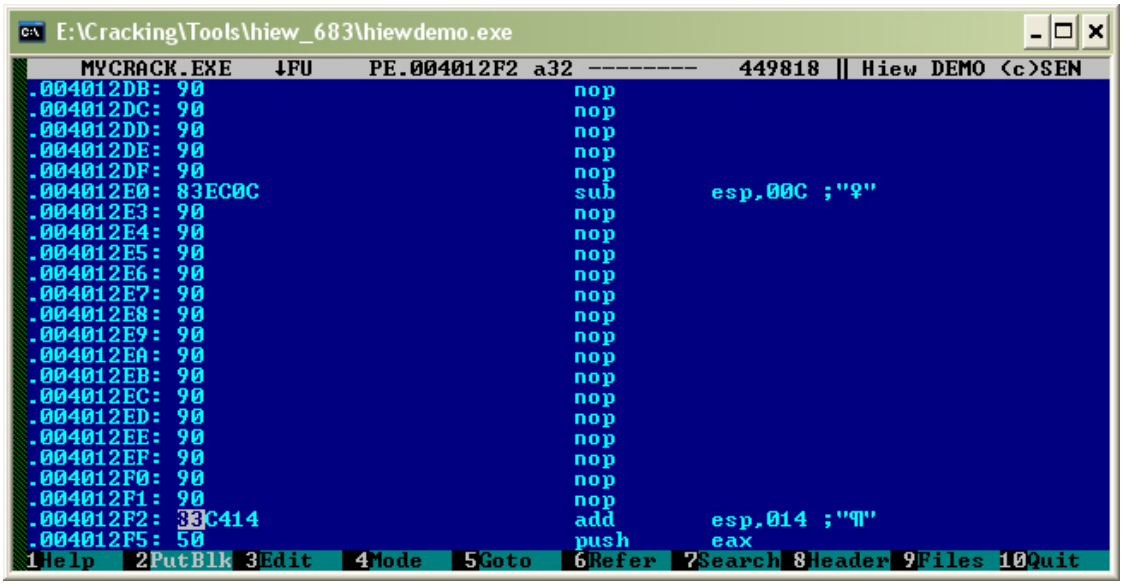
ova funkcija za prikazivanje teksta na ekranu ima tri parametra koja su joj potrebna da prikazu tekst na ekranu. Ono sto smo mi hteli da uradimo je da joj oduzmemo jedan parametar zbog cega bi se program srusio. Problem lezi u tome sto svaka funkcija mora imati dovoljan broj parametara, ovaj broj ne sme biti ni veci ni manji od potrebnog za tu funkciju. Ono sto je zgodno kod asemblera je da mi ne moramo znati koliko parametara kojoj funkciji treba. Primeticete par PUSH komandi koje predhode jednoj CALL komandi. Ove komande predstavljaju ono sto sam objasnjavao na primeru od gore, pri cemu PUSH komande predstavljaju parametre funkcije a sam CALL predstalja funkciju. Ono sto je takodje bitno da zapamtite je da se u asembleru parametri funkcijama prosledjuju u obrnutom redosledu. Prvo se prosledjuje poslednji parametar, a pre samog pozivanja funkcje prosledjuje se i prvi parametar. Ovo znaci da ne smemo samo da NOPujemo CALL, ili samo da NOPujemo PUSH funkcije, mi da bi smo sklonili ispisivanje poruke sa ekrana moramo da NOPujemo sve PUSHeve i na kraju sam CALL. Primecujete da se tri reda ispod poruke koju zelimo da uklonimo sa ekrana nalazi CALL koji sluzi za prikazivanje poruke na ekran. Pre njega se nalaze ne dva nego tri PUSHa, dva ispod teksta poruke i jedan iznad. Nedajte da vas zbuni to sto se izmedju PUSH komandi nalaze neke druge ASM komande, one nas ne interesuju, interesuju nas samo PUSHevi i CALL. Pre nego sto pocnemo sa samim crackovanjem detaljnije cu objasniti prozor koji trenutno vidite u W32Dasmu. Ono sto vidimo je ovo:



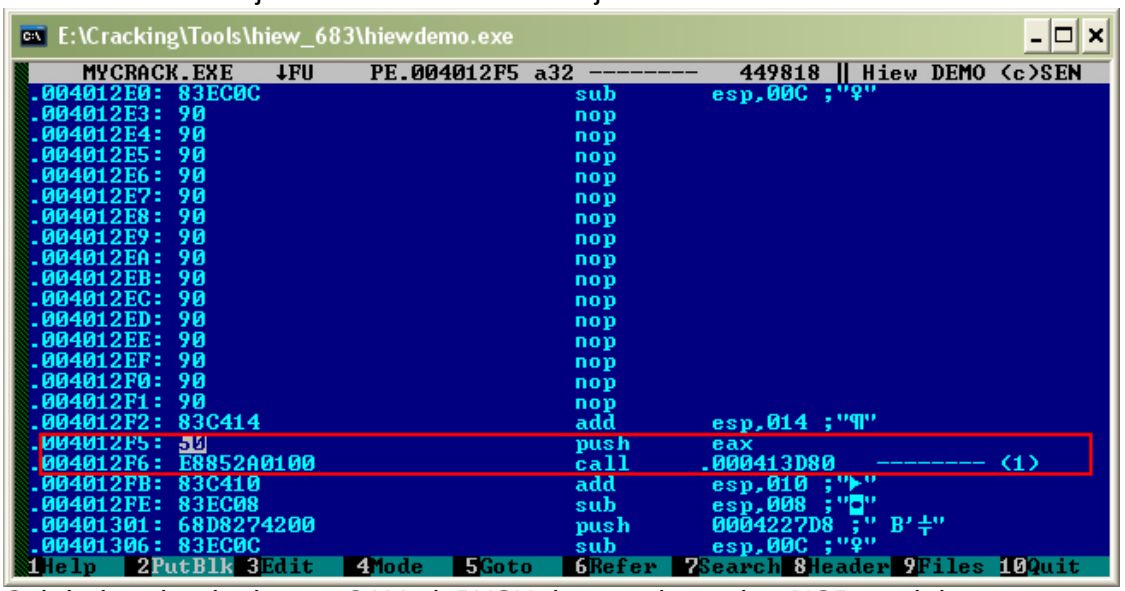
Crveno uokvireni deo koda oznacava virualne adrese na kojima se nalazi ta linija koda. Posmarajte ove brojeve kao da su brojevi 1,2,3 i da predstavljaju red izvrsavanja programa. U ovom primeru prva adresa od koje krece

izvrsavanje tkz. OEP (original entery point) je 0040100, a sve ostale adrese se povecavaju za odredjeni broj. Ovaj broj ne mora biti uvek jedan, veoma cesto i nije jedan. Taj broj direktno zavisi od sadrzaja koji se nalazi uokviren plavo na slici. Ovi hex brojevi predstavljaju komande koje su u fajlu zapisane u HEX obliku. Tako je HEXa decimalno 55 ekvivalentno ASM komandi PUSH EBP. Primeticete da se ovi HEX brojevi pisu iskljucivo u paru po dva. Samo dvocifreni HEX brojevi predstavljaju komande koje su ekvivalentne onima koje su na slici uokvirene zelenom bojom. Nama je za ovaj primer bitno da se ASM komanda NOP u HEX obliku pise kao 90. Ovo znaci da je jedno 90 ekvivalentno jednoj komandi NOP. Da bi smo NOPovali jedan red moramo da zamenimo sve dvocifrene HEX brojeve iz tog reda sa 90. Na primer ako brisemo red koji se nalazi na adresi 004012E0 moracemo da zamenimo sadrzaj ove adrese (83EC0C) sa tri NOP komande (909090) jer postoje tri para dvocifrenih brojeva u toj liniji. Primetite kako se to adrese povecavaju, videcete da se recimo iza adrese 004012C0 nalazi 004012C3. Logicno je za ocekivati da se iza 004012C0 nalazi 004012C1, ali ovde se nalazi 004012C3. Ovo se desava zato sto u ASMu svakom dvocifrenom broju dodeljuje samo jedna adresa. Dakle ako je adresa 004012C0 onda se na toj adresi nalazi samo jedan dvocifreni broj i to 83, na adresi 004012C1 se nalazi E4, a na adresi 004012C2 F0. Jedini razlog zasto su ove tri adrese spojene u jedan red je zato sto te tri adrese predstavljaju samo jednu ASM komandu, pa zbog toga je prva sledeca adresa, na kojoj se nalazi sledeca ASM komanda, 004012C3. Adrese redova odgovaraju adresi prvog dvocifrenog hex broja (bajta) koji se nalazi u toj “liniji” koda.

Posto smo naucili kako da patchujemo (*modifikujemo*) program predjimo na samu modifikaciju. Program mozete editovati u Hiewu ali i u bilo kom drugom hex editoru. Posto je Hiew najbolji za pocetnike sva objasnjenja u prvom poglavlju bice vezana za njega. Dakle startujete Hiew u otvorite myCrack.exe pomocu njega. Prvobitan prikaz u Hiewu nam je nerazumljiv pa cemo ovaj prikaz pretvoriti u prikaz koji je istovetan prikazu iz W32Dasma. Ovo radimo pritiskom na F4 i selekcijom Decode moda. Sada vec vidimo ASM komande. Ako niste zapamtili pogledajte gornju sliku ponovo i prepisite adrese koje treba da NOPujemo, te adrese su: 004012DB, 004012E3 i 004012E8. Ako zelimo da odemo na te adrese u Hiewu treba da pritisnemo dugme F4 i ukucamo prvo tacku pa adresu na koju zelimo da odemo. Kada odemo na prvu adresu videcemo istu onu PUSH komandu iz W32Dasma. Posto smo naucili da treba da NOPujemo sve dvocifrene brojeve iz tog reda, zapamticemo dva zadnja bajta. Dakle zadnja dva bajta koja cemo NOPovati su 42 i 00. Sada pritisnite F3 da udjete u Edit mod i kucajte nove bajtove kojima zelite da zamenite stare, ti bajtovi su 90. Moracemo da unesemo 5x 90 da bi smo NOPovali ceo red. Postavite kursor na sledecu adresu koju treba da NOPujete, to jest na sledecu PUSH komandu i sa njom uradite isto. Isti postupak ponovite i sa zadnjom PUSH komandom i sa CALLom. Kada zavrsite rezultat bi trebalo da izgleda kao na sledecoj slici:



Dakle sve adrese sa PUSH i CALL komandama su NOPovane. Da bi smo snimili promene pritisnimo F9, a za izlazak iz programa pritisnite F10. Probajte slobodno da startujete program koji ste malo pre patchovali i videcete da on nece raditi!!! Iako smo sve dobro uradili, patchovali smo sve PUSH i CALL komandu program pravi gresku. Ono sto nismo ocekivali, a trebalo je, da je ovo DOS program i da posle ispisivanja poruke na ekran program prebacuje kursor u novi red. Ovo se desava odmah ispod ispisivanja poruke koju smo uklonili, sto znaci da sledeci CALL sluzi bas za ovo. Otvorimo ponovo HIEW i ukucajmo adresu 004012F2. Odmah ispod ove adrese videcemo jednu PUSH komandu i jedan CALL.



Ocigledno je da i ovu CALL i PUSH komandu treba NOPovati jer program nemoze da prebaci kursor u sledeci red bez ispisivanja poruke na ekran. Znaci NOPovacemo i adrese 004012F5 i 004012F6, snimicemo fajl i pokusacemo ponovo da startujemo program. Videcemo da smo sada uspeli i da se poruka vise ne pojavljuje. Uspeli smo da crackujemo nas prvi program.

My second Crack

Posto smo konfigurisali alate koji su nam potrebni sada cemo krenuti sa njihovim koristenjem. Startujte myFirstCrack.exe koji se nalazi u folderu ...\Casovi\Cas01. Ono sto cete videti je obican Dos prozor koji kaze da nije crackovan. Ovo cemo da promenimo. Otvorimo ovaj program u W32Dsamu **[Hint:** *Dissasembler -> Open File to dissasemble* **]**, sacekajmo koji trenutak da bi se pojavio dissasemblovani program. Posto je ovo drugi cas necu mnogo analizirati nego cemo uraditi samo ono sto je potrebno da bi smo crackovali ovaj program. Podsetimo se one poruke koju nam je program izbacio, sada bi trebalo da saznamo odakle se ona poziva. Ovo mozemo uraditi na dva nacina. Prvi je preko find opcije u programu, a drugi, nama malo korisniji, je preko opcije String Reference u W32dsamu. Znaci pritisnimo predzadnje dugme u toolbaru na kojem pise Str Ref i novi prozor ce se otvoriti. U tom prozoru cemo pronaci poruku koju je program izbacio (*Nisam crackovan :P*) i kliknucemo 2x na nju, sto ce nas odvesti do tacnog mesta u kodu ovog exe fajla odakle se ova poruka poziva. Program ce nas odvesti ovde:

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0040128C(C)
|
:004012EF 837DFC00              cmp dword ptr [ebp-04], 00000000
:004012F3 754C                  jne 00401341
:004012F5 83EC08                sub esp, 00000008
:004012F8 6848284200            push 00422848
:004012FD 83EC0C                sub esp, 0000000C

* Possible StringData Ref from Code Obj ->"Nisam crackovan :P"
|
:00401300 6880124000            push 00401280 <- Ovde smo
:00401305 6850534300            push 00435350
:0040130A E84D1E0200            call 0042315C

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004012A8(C)
|
:0040130F 83C414                add esp, 00000014
:00401312 50                    push eax
:00401313 E8D82A0100            call 00413DF0
:00401318 83C410                add esp, 00000010
:0040131B 83EC08                sub esp, 00000008
:0040131E 6848284200            push 00422848
:00401323 83EC0C                sub esp, 0000000C
```

Vidimo poruku “Nisam crackovan :P”. Znaci odavde se poziva ova poruka. Ono sto je predmet ovog casa je da vas naucim kako da se umesto ove

poruke pojavi neka druga poruka koja se vec nalazi u istom exe fajlu. Ako pogledamo malo iznad ove poruke videcemo jedan uslovni skok:

```
:004012F3 754C                  jne 00401341
```

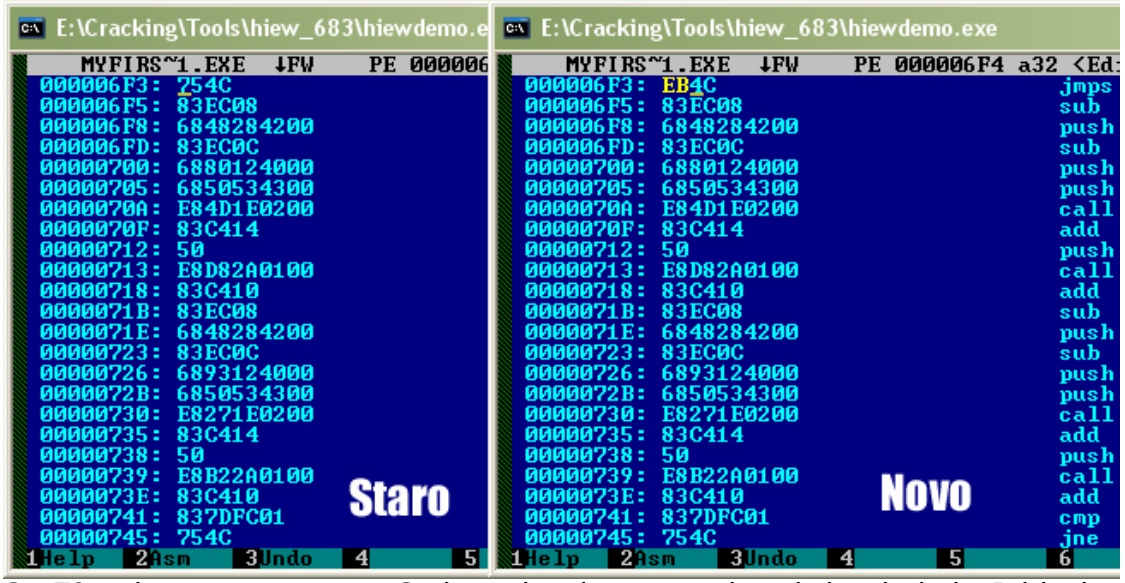
Ovo znaci ako nesto nije jednako program ce skociti na 00401341. Ako odemo malo dole da vidimo sta se nalazi na toj adresi videcemo sledece:

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004012F3(C)
|
:00401341 837DFC01              cmp dword ptr [ebp-04], 00000001
:00401345 754C                  jne 00401393 <-Vazan skok koji kad se izvrsi
:00401347 83EC08                sub esp, 00000008 ; poruka se preskace.

:0040134A 6848284200            push 00422848
:0040134F 83EC0C                sub esp, 0000000C

* Possible StringData Ref from Code Obj ->"Uspesno si me crackovao :)"
|
:00401352 68AE124000            push 004012AE
```

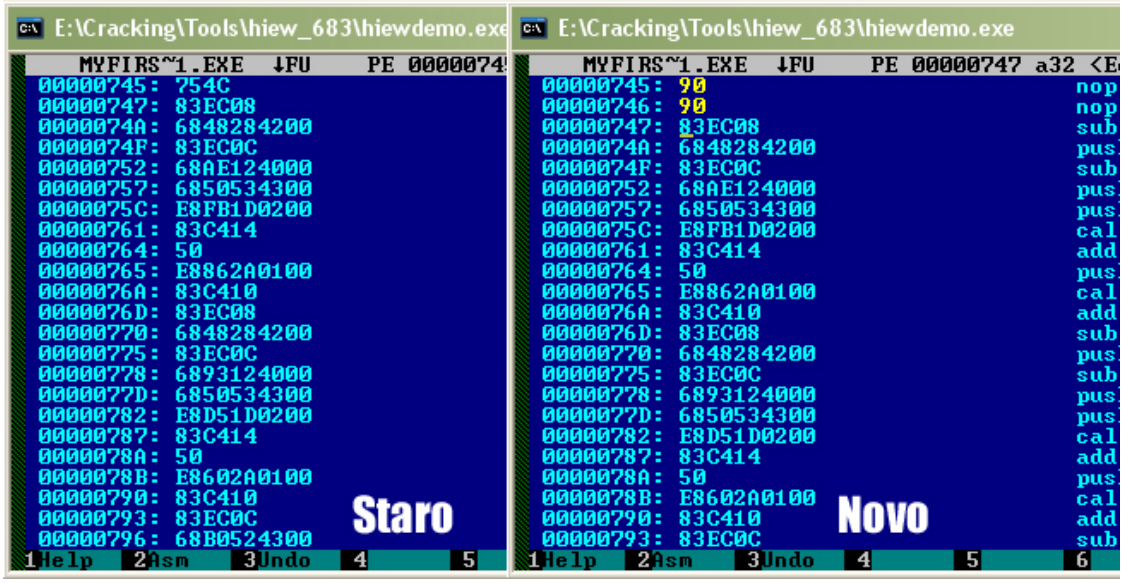
Primeticemo da se tu nalazi poruka “Uspesno si me crackovao :)” koja ce se prikazati na ekranu samo ako se skok sa adrese 004012F3 uvek izvrsava. Ono sto moramo da primetimo je da se izmedju adrese 00401341 i adrese 00401352 nalazi jos jedan skok koji treba da izmenimo. Da bi smo uspesno crackovali ovaj program treba da izmenimo onaj skok na adresi 004012F3 sa JNE (*75 hex*) u JMP (*EB hex*) da bi se uvek izvrsavao to jest da bi progam uvek stizao do adrese 00401341, a skok na adresi 00401345 da promenimo tako da se nikad ne izvrsava, to jest da ga promenimo u dva NOPa (*No Operation*). Kada ovo zavrsimo uvek ce se na ekranu prikazivati poruka o uspesnom crackovanju. Ako vam ovaj deo nije jasan obratite paznju na hex adrese na koje vode skokovi. Videcete da nas oni vode ispod i preko poruka koje zelimo da se prikazu na ekranu. Mi moramo da izmenimo ove skokove tako da se uvek na ekranu prikazuje poruka koju mi zelimo. Ovo je mozda najbitniji deo knjige, sama osnova. Da bi ste uspesno napredovali dalje morate da shvatite kako menjanje odredjenih komandi utice na ponasanje programa. Stoga predlazem da ako se prvi put srecete sa crackovanjem uradite ovu prvu oblast temeljno i da ne prelazite na druge oblasti bez predhodnog razumevanja ove i bez samostalnog resavanja prve vezbe koja se nalazi na kraju ovog poglavlja. Radjenje vezbi je preporucljivo jer ce vam pomoci da se osamostalite i da sami resavate probleme, koje niste pre toga videli i obradili.

Da bi smo izvrsili ove promene u programu treba da ga iskopiramo u direktorijum gde se nalazi Hiew i otvorimo ga pomocu njega.

Posto nam ovaj prvobitan prikaz nije bas razumljiv, pritiskom na F4 2x prelazimo u Decode oblik koji je isti kao onaj koji smo videli u W32Dsamu. Mozemo da skrolujemo dole do adrese 004012F3 koju hocemo da izmenimo a mozemo i da idemo do te adrese sa komandom GoTo. Pritisnite F5 i prvo unesite *tacku*, pa adresu 004012F3 i onda enter. Sada kada smo na toj adresi, mozemo je menjati prelaskom u edit mode sa F3. Stavite kursor na prvi bajt 75 i kucajte EB. Evo kako to treba da izgleda:



Sa F9 snimamo promene. Sada treba da promenimo i drugi skok. Pritisnite F5 i unesite tacku, pa adresu 00401345 i onda pritisnite enter. Sada kada smo na toj adresi, mozemo je menjati prelaskom u edit mode sa F3. Stavite kursor na prvi bajt 75 i kucajte 9090. Evo kako to treba da izgleda:



Sa F9 snimamo promene a sa F10 izlazimo iz Hiew-a. Sada mozete startovati ovaj exe fajl i videcete da ce on uvek prikazivati istu poruku, “Uspesno si me crackovao :)”

Mozete pogledati fajl …\Casovi\Cas1\main.cpp da vidite kako taj program izgleda u programskom jeziku C++. Kao sto vidite:

```
#include <iostream>

using namespace std;

int main (int argc, char *argv[])
{
  int i;
  i = 0;
 if (i == 0){
  cout << "Nisam crackovan :P" << endl;
  cout << "Press ENTER to continue..." << endl;
 }
 if (i == 1){
  cout << "Uspesno si me crackovao :)" << endl;
  cout << "Press ENTER to continue..." << endl;
 }
  cin.get();
  return 0;
}
```

postoje dva uslova. Ako je I jednako nula onda se pokazuje poruka da program nije crackovan, a ako je I jednako jedan onda se prikazuje poruka da je program crackovan. Posto je I uvek jednako nula ono sto smo mi uradili je kao da smo zamenili uslove, to jest kao da je za prvu poruku potrebno da je I jednako jedan a za drugu da je I jednako nula. Ovo je malo komplikovaniji primer jer je potrebno izmeniti dva skoka ali kada savladate ovo mocicete da crackujete veliki broj programa za pocetnike jer se skoro svi zasnivaju na ovom ili slicnom principu.

Vezba:

Ako zelite mozete proveriti dosada steceno znanje na istom primeru ili na slicnom vec pripremljenom fajlu. …\Casovi\Cas1\myFirstTest.exe je fajl koji treba da crackujete sami. Sam postupak se ne razlikuje mnogo od predhodnog primera. U ovom testu treba ispraviti samo jedan skok. Za one koji znaju ili po malo razumeju C++ tu je i …\Casovi\Cas1\test.cpp da pogledaju razlike izmedju prvog primera i ovog testa, a fajl …\Casovi\Cas1\myFirstTest .cracked.exe je primer kako treba da izgleda konacan crackovan program. Radjenje vezbi kroz ovu knjigu je preporucljivo kako bi lakse zapamtili postupke crackovanja na slicnim primerima.

Resenje:

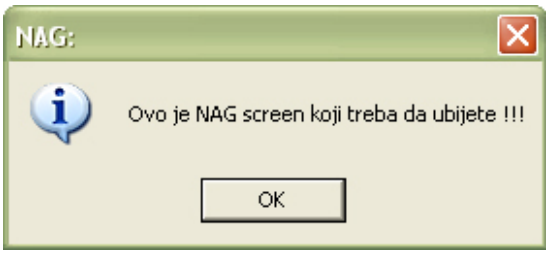
Treba promeniti skok na adresi 00401391 sa JNE (*755E*) u JMP (*EB5E*).

02 NAG Screens

U prvom poglavlju smo savladali osnovne tehnike pronalazenja date informacije u exe fajlu i menjanja isto, a u drugom cemo nauciti kako da se izborimo sa osnovnim problemima u reversovanju.

Killing NAGs – *MsgBoxes*

NAG ekrani su one dosadne poruke koje se pojavlju pre ulaska u neki program ili na samom izlasku iz njega, a njihova glavna funkcija je da vas podsete da niste platili program koji koristite. Postoji vise vrsta NAGova ali su najcesce zastupljena dva standardna tipa, *message boxovi i dialozi.* U ovom primeru cu vam pokazati kako da se resite obicnog *message box* NAGa. On izgleda upravo ovako:



a nalazi se u fajlu ...\Casovi\Cas2\NAG.exe. I za ovaj primer cemo upotrebiti iste alate (*W32Dsam i Hiew*) kao i za prvi primer.

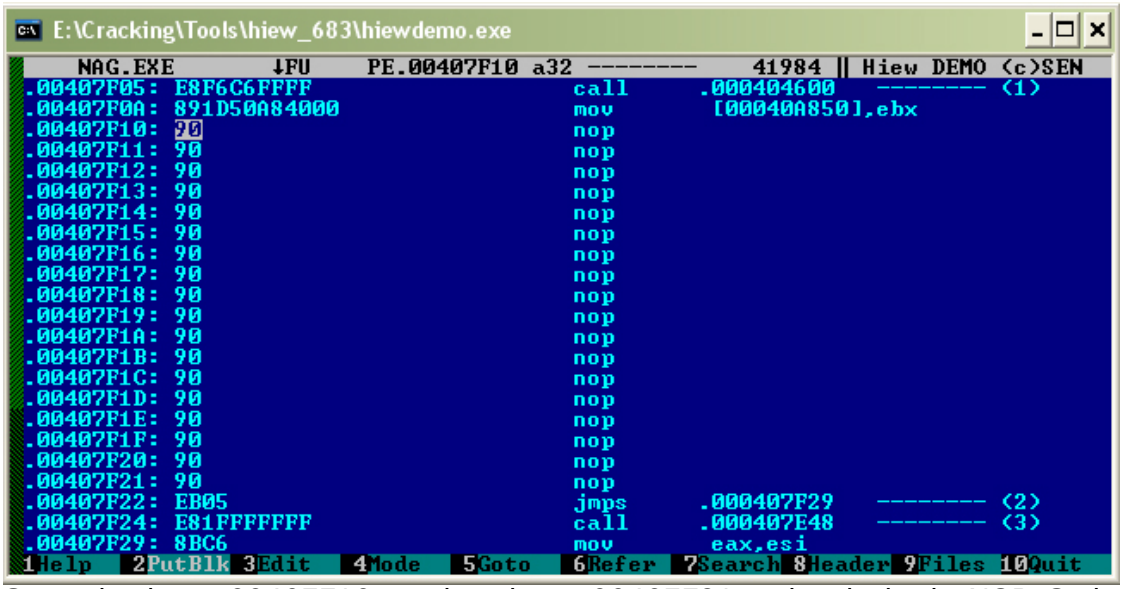
U W32Dsamu otvorite ovaj exe fajl i sacekajte da se disasembluje. Najlaksi nacin za ubijanje ovog NAGa je trazenje teksta koji se u njemu pojavljuje. Otvorite opet String Refernce i nadjite tekst iz ovog message boxa. Duplim klikom na taj tekst zavrsicemo ovde:

```
* Reference To: user32.SetWindowTextA, Ord:0000h
|
:00407F05 E8F6C6FFFF            Call 00404600
:00407F0A 891D50A84000          mov dword ptr [0040A850], ebx
:00407F10 6A40                  push 00000040

* Possible StringData Ref from Code Obj ->"NAG:"
|
:00407F12 68407F4000            push 00407F40

* Possible StringData Ref from Code Obj ->"Ovo je NAG screen koji treba da "
                                       ->"ubijete !!!" <- Ovde smo
|
:00407F17 68487F4000            push 00407F48
:00407F1C 53                    push ebx

* Reference To: user32.MessageBoxA, Ord:0000h
|
:00407F1D E8C6C6FFFF            Call 004045E8
:00407F22 EB05                  jmp 00407F29
```

Ono sto je specificno za ove message box NAGove je da se oni iz bilo kog programskog jezika pozivaju na isti nacin:

```
MessageBoxA(hwnd, Text, Naslov, MB_TIPMESSAGEBOXA);
```

sto znaci da se CALL funkciji koja generise ovaj message box prosledjuju cetiri parametra. Ovo prosledjivanje ide u obrnutom redosledu pa se tako pre CALL funckije na adresi 00404F1D nalaze cetiri PUSH funkcije koje prosledjuju ove parametre u obrnutom redosledu. Ako vam ovo nije jasno predlazem da procitate deo STACKu sa strane 8. Kao sto predpostavljate ova funkcije ne treba da se izvrsava nikada stoga je treba NOPovati. Obratite paznju samo na jedno, a to je da ako NOPujete samo CALL onda ce doci do greske u programu. Pravi nacin ubijanja ovakvih NAG ekrana je da treba da NOPujete sve PUSH funkcije koje predhode CALLu a onda i sam CALL. To u HIEWu treba da izgleda ovako:



Sve od adrese 00407F10 pa do adrese 00407F21 treba da bude NOP. Sada mozete startovati NAG.exe i videcete da NAGa vise nema.

Vezba:

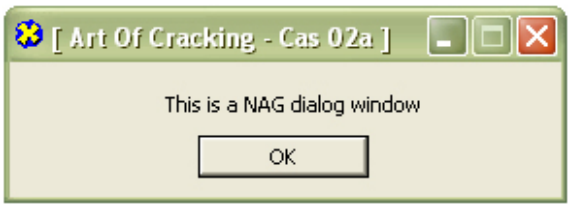
Ako zelite mozete proveriti dosada steceno znanje na istom primeru tako sto ce te ubiti tekst koji se pojavljuje kada korisnik klikne na dugme **?.** kada se pojavljuje About dialog.

Resenje:

Sve od adrese 00407EA0 pa do adrese 00407EB3 treba da bude NOPovano.

Killing NAGs – *Dialogs*

U proslom primeru je objasnjeno kako se skidaju *messagebox* NAGovi a u ovom ce biti objasnjeno kako se skidaju *dialog* NAGovi. Taj dialog moze izgledati bas ovako:



Razlika je mozda neprimetna ali za razliku od *messagebox* NAGova, ovaj NAG je napravljen na isti nacin kao i ostali dialozi u programu. Zasto nam je ovo bitno? Obicnim korisnicima i nije bitno ali ovo nama govori na koji nacin se generise ovaj NAG i kako mozemo da ga se resimo. Otvorimo program ...\Casovi\Cas2\NagWindow.exe u W32Dsamu. Posto je ovo dialog potrazicemo sve dialoge u ovom exe fajlu. Pogledajmo malo dole od pocetka disasemblovanog fajla i videcemo sledece:

```
++++++++++++++++++ DIALOG INFORMATION ++++++++++++++++++++
Number of Dialogs =    1 (decimal) <- WTF ???
Name: DialogID_0064, # of Controls=008, Caption:"½F", ClassName:""
    001 - ControlID:0000, Control Class:"EDIT" Control Text:""
    002 - ControlID:0000, Control Class:"EDIT" Control Text:""
    003 - ControlID:0000, Control Class:"BUTTON" Control Text:"&Check"
    004 - ControlID:0000, Control Class:"BUTTON" Control Text:"&Exit"
    005 - ControlID:0000, Control Class:"BUTTON" Control Text:"&?"
    006 - ControlID:0000, Control Class:"STATIC" Control Text:"Name:"
    007 - ControlID:0000, Control Class:"STATIC" Control Text:"Serial:"
    008 - ControlID:0001, Control Class:"" Control Text:""
```

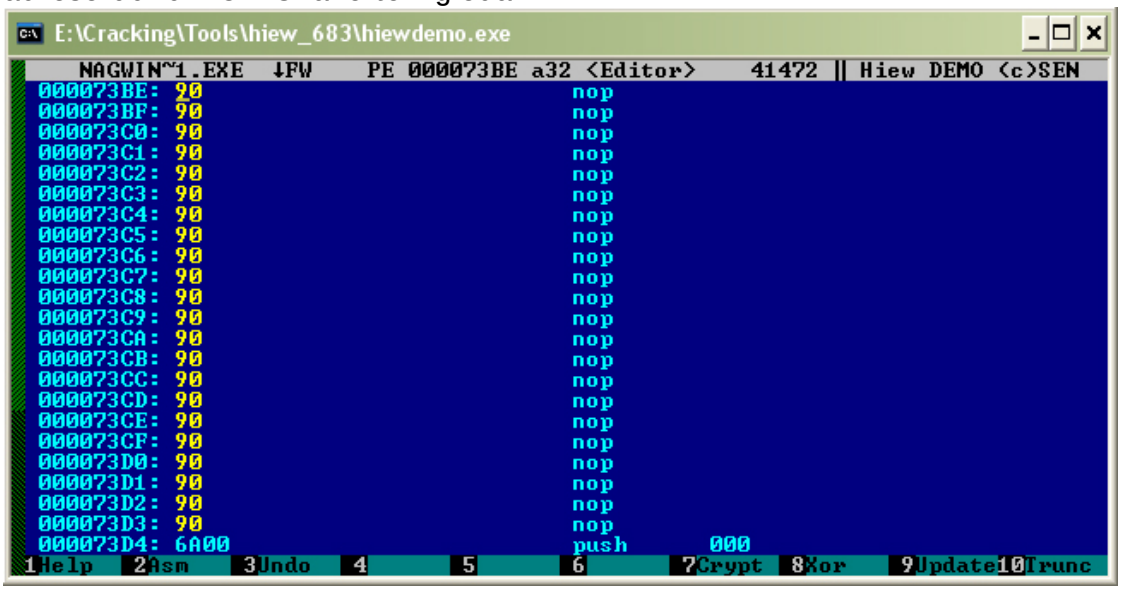
Ovde nesto nevalja! Dialog koji se ovde nalazi je onaj drugi dijalog koji se pojavljuje posto pritisnemo OK u NAG prozoru. Ali gde je onaj prvi, NAG window? Otkricemo to. Za sada samo zapamtite da je ime ovog dialoga (*DialogID_0064*) Posto W32Dsam dodaje prefix DialogID_ stvarno ime ovog dialoga je broj 64h u hex obliku a jednako je 100 u decimalnom. Ovo nam je bitno jer CALLu koji je zaduzen za prikazivanje ovog dialoga je potreban taj ID kako bih znao koji dialog da pokaze. Da bi smo saznali odakle se poziva ovaj dialog idemo gore u meniju na dugme DLG Ref (*Dialog References*) i duplim klikom na ime dialoga zavrsicemo ovde:

```
:00407FCF E8DCC5FFFF          Call 004045B0
:00407FD4 6A00                push 00000000
:00407FD6 68647E4000          push 00407E64
:00407FDB 6A00                push 00000000
:00407FDD 6A64                push 00000064  <- Ovde smo
:00407FDF FF354C984000        push dword ptr [0040984C]
:00407FE5 E8C6C5FFFF          Call 004045B0
:00407FEA E8D5B4FFFF          Call 004034C4
```

Ako prebrojimo parametre koji predhode prvom sledecem CALLu na adresi 00407FE5 videcemo koliko parametara ima funkcija za pozivanje dialoga. Zakljucilio smo da je potrebno 5 parametara za ovu funkciju a da je predzadnji odnosno drugi dialog ID. Posto se NAG pojavljuje pre pojavljivanja ovog dialoga zakljucujemo da se ista funkcija mora nalaziti i iznad ovoga gde smo sada, samo sa razlikom u id-u dialoga koji poziva. I bili smo u pravu.

```
:00407FBE 6A00                  push 00000000
:00407FC0 68647E4000            push 00407E64
:00407FC5 6A00                  push 00000000
:00407FC7 6A65                  push 00000065  <- Drugi dialog ID
:00407FC9 FF354C984000          push dword ptr [0040984C]
* Reference To: user32.DialogBoxParamA, Ord:0000h
|
:00407FCF E8DCC5FFFF            Call 004045B0
```

Isti broj parametara predhodi CALLu na adresi 00407FCF. Znaci sigurno se CALL na adresi 00407FCF koristi za prikazivanje NAGa. Ovi NAGovi se ukljanjaju na isti nacin kao i *message box* NAGovi. Znaci sve PUSH komande i CALL na kraju moraju biti NOPovane, a to znaci sve od adrese 00407FBE do adrese 00407FCF. Ovako to izgleda:



Ne dajte da vas brojevi sa strane zbune 73BE je tacan polozaj virtualne adrese 00407FBE koja odgovara stvarnoj 73BE samo u memoriji. Ovi stvarni polozaji bajtova ce vam se prikazati tek kada pritisnete F3 i udjete u Edit mod.

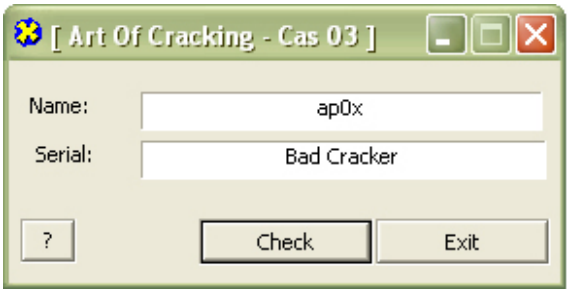
NAPOMENA: U vecini slucajeva se svi dijalozi pojave u W32Dsamu ali sada iako se desila neka greska u W32Dsamu uspeli smo da pronadjemo sve dialoge i da eliminisemo NAG ekran.

03 Cracking Serials

Sledece poglavlje govori o najcesce sretanom problemu pri reversingu. Veoma cesto se desava da je cela aplikacija ili da su neki njeni delovi zakljucani za koristenje i da se mogu otkljucati samo pravim seriskim brojem. Ovde ce biti govora o vise tipova provere seriskih brojeva i o nacinima resavanja ovih problema. Vazno je napomenuti da ce se od ovog poglavlja iskljucivo koristi najvazniji reverserski alat, OllyDBG. Nazalost ovo poglavlje je specificno jer se prilikom “pecanja” seriskih brojeva mora nadgledati radna memorija a ne sadrzaj dissasemblovanog fajla na disku. Zbog lakseg privikavanja na sve ovo prvo cu vam objasniti jedan primer na W32Dsamu.

The Serials – *Jumps*

Pored NAG ekrana jedna od prepreka vezanih za crackovanje je i registracija ili otkljucavanje odredjenih funkcija programa pomocu rutina za proveru seriskih brojeva. Ovo je veoma cesto sretan problem pri reversingu stoga se ovo poglavlje moze smatrati jednim od kljucnih. Prvi deo ovog poglavlja ce vas nauciti kako se ovakvi programi crackuju, drugi kako da nadjete pravi seriski broj za vase ime, a sledece poglavlje kako da napisete keygenerator za ovaj primer. Ovaj primer se nalazi u folderu ...\Casovi\Cas3\Serial.exe Za pocetak cemo startovati program i videcemo sta se desava. Ovo je obavezan korak koji nam omogucuje da skupimo sto je vise moguce informacija o "meti", da bi smo je lakse reversovali. Startujte "metu" i unesite kao ime ap0x i kao serial 111111, pritisnite Check. Pojavice se ovo:



Ono sto smo saznali iz ovog testa je da kada unesemo pogresan seriski broj, program izbaci poruku "Bad Cracker". Ovo ce pomoci prilikom potrage za mestom gde se proverava tacnost seriskog broja. Otvorite ovu "metu" u W32Dsamu i pronadjite string "Bad Cracker". Primetice te da se pored stringa "Bad Cracker" nalazi i ovo:

```
"About..."
"AMPM "
"Bad Cracker"
"Cracked ok"
"eeee"
"Enter name !!!"
"Error"
```

Ovo je jako zanimljivo posto kako izgleda mozda je poruka koja ce biti prikazana ako je seriski broj tacan "Cracked ok". Bez obzira na to mi cemo 2x kliknuti na "Bad Cracker" poruku i zavrsicemo ovde:

```
:00407DE9 E806BBFFFF            call 004038F4
:00407DEE 7517                  jne 00407E07
* Possible StringData Ref from Code Obj ->"Cracked ok"
|
:00407DF0 684C7E4000            push 00407E4C
:00407DF5 68B90B0000            push 00000BB9
```

```
:00407DFA A150984000            mov eax, dword ptr [00409850]
:00407DFF 50                    push eax
* Reference To: user32.SetDlgItemTextA, Ord:0000h
|
:00407E00 E8F3C7FFFF            Call 004045F8
:00407E05 EB15                  jmp 00407E1C
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00407DEE(C)
|
* Possible StringData Ref from Code Obj ->"Bad Cracker"
|
:00407E07 68587E4000            push 00407E58 <- Ovde smo
:00407E0C 68B90B0000            push 00000BB9
:00407E11 A150984000            mov eax, dword ptr [00409850]
:00407E16 50                    push eax
```

Obratimo paznju na ovaj red odmah iznad poruke o pogresnom seriskom broju:

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00407DEE(C)
```

Ovo znaci da postoji jedan uslovni (*zbog onog C, da stoji U to bi bio bezuslovni skok*) skok na adresi 00407DEE koji vodi na adresu 00407E07. Ako pogledamo sta se nalazi na toj adresi videcemo sledece:

```
:00407DEE 7517                  jne 00407E07
* Possible StringData Ref from Code Obj ->"Cracked ok"
|
:00407DF0 684C7E4000            push 00407E4C
```

Ovo znaci da ako nesto, a u ovom slucaju seriski broj, nije tacan skoci na poruku o pogresnom seriskom broju. Ako ovaj red izbrisemo (*citaj: NOPujemo*) onda ce program uvek prikazivati poruku o tacnom seriskom broju bez obzira na uneto ime ili seriski broj. To je jedan i ujedno i najlaksi nacin za resavanje ovog problema.

The Serials – *Fishing*

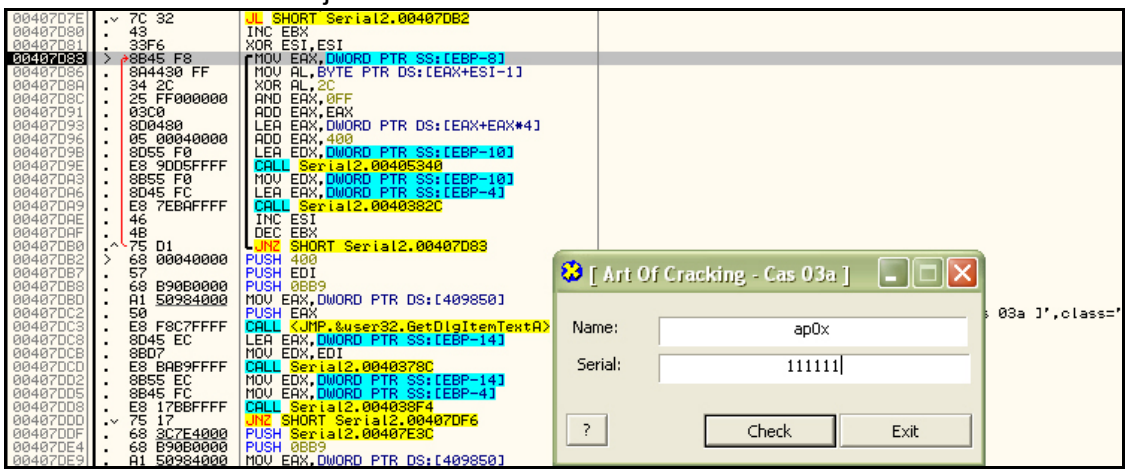
Prvi deo ovog poglavlja vas je naucio kako da program za bilo koji uneti seriski broj kaze da je ispravan, a ovaj drugi ce vas nauciti kako da nadjete pravi seriski broj za vase (*moje*) ime. Za ovo ce nam trebati malo drugaciji set alata koji cemo koristiti kako bih pronasli seriski broj. Jedini alat koji ce nam trebati je OllyDBG. Posto je ovo prvi put da se ovaj program pominje u ovoj knjizi trudicu se da veci broj stvari objasnim slikovno. Ucitajte program ...\Casovi\Cas3\Serial2.exe Olly. Videcete OEP (*prvu liniju*) programa:

```
00407FA8 > $  55              PUSH EBP
00407FA9   .  8BEC            MOV EBP,ESP
```

Sada samo treba da nadjemo gde se to proverava seriski broj. Posto smo to mesto vec nasli u prvom delu ustedecemo malo vremena jer vec znamo kako da ga nadjemo. Ekvivalent String Reference opcije iz W32Dsama je u Olly dume R u toolbaru. Da bi ste ucitali sve stringove iz fajla u R prozor kliknite desnim dugmetom na OEP,pa na **Search For -> All referenced text strings...** Tu u novo otvorenom R prozoru nadjite string “Bad Cracker”, i duplim klikom na njega zavrsicete ovde:

```
00407DF6  |> \68 487E4000   PUSH Serial2.00407E48    ; /Text = "Bad Cracker"
```

Ako odskrolujemo malo gore do 00407CF4 i dole do 00407E2C videcemo da je ovo Olly oznacio kao jednu celinu i da je 100% sigurno da se ovde negde proverava seriski broj. Pocnimo analizu od pocetka, od adrese 00407CF4. Selektujmo taj red i pritisnimo F2 da postavimo break-point (pauzu) na tu adresu. To znaci da kad program sa izvrsavanjem dodje do te adrese onda cemo mi imati puno kontrolu nad njegovim izvrsavanjem. Sada pritisnimo F9 da bi smo pokrenuli program. Kao ime u nasu “metu” unesite ap0x (*sa 0 – brojem a ne sa O slovom*) i kao seriski broj 111111. Pritisnite Check i program ce zastati na adresi 00407CF4. Polako pritiskamo F8 da bi smo se kretali red po red kroz program. Sve nam izgleda obicno i nevazno dok ne stignemo do adrese 00407D83. To jest do ove:



Ovo nam je zanimljivo jer seriski brojevi racunaju bas na ovaj naci. Tu je uvek u pitanju neka petlja koja se ponavlja za sva slova imena na osnovu koga se racuna seriski broj, naravno ako on direktno zavisi od imena. Procicemo 5 puta kroz ovu petlju sa F8 i primeticemo da se sa strane u registima (*gornjem desnom uglu*) pojavljuju neki brojevi:

```
EAX 0012FCAC
ECX 000005B8
EDX 00850CA8 ASCII "1464"
EBX 00000005
ESP 0012FC90
EBP 0012FCBC
ESI 00000000
EDI 00850888 ASCII "ap0x"
```

1 Prolaz

```
EAX 0012FCAC
ECX 00000702
EDX 00850CBC ASCII "1794"
EBX 00000004
ESP 0012FC90
EBP 0012FCBC
ESI 00000001
EDI 00850888 ASCII "ap0x"
```

2 Prolaz

```
EAX 0012FCAC
ECX 00000798
EDX 00850CA8 ASCII "1944"
EBX 00000003
ESP 0012FC90
EBP 0012FCBC
ESI 00000002
EDI 00850888 ASCII "ap0x"
```

3 Prolaz

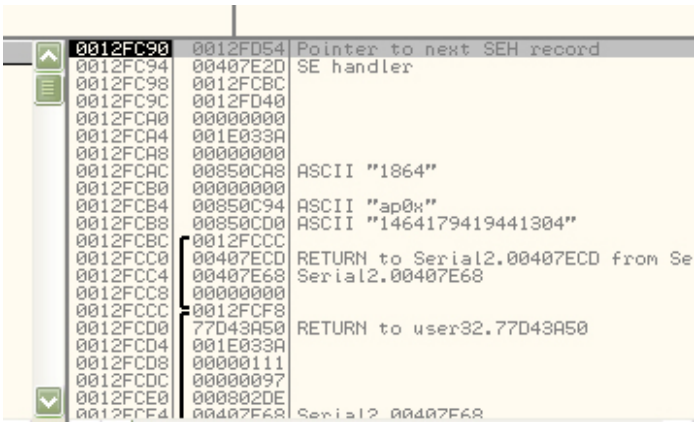
```
EAX 0012FCAC
ECX 00000518
EDX 00850CBC ASCII "1304"
EBX 00000002
ESP 0012FC90
EBP 0012FCBC
ESI 00000003
EDI 00850888 ASCII "ap0x"
```

4 Prolaz

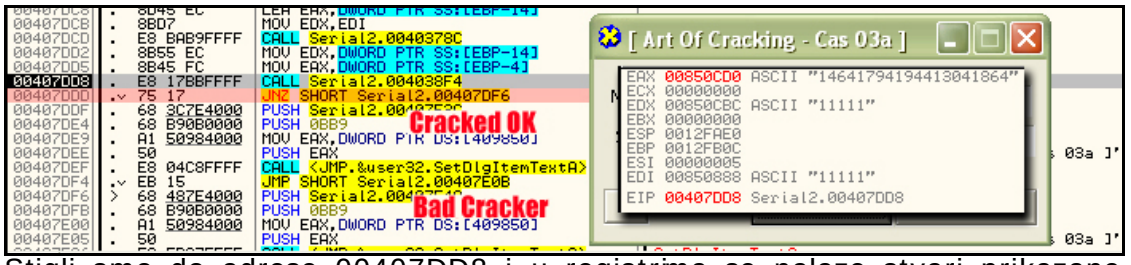
```
EAX 0012FCAC
ECX 00000748
EDX 00850CA8 ASCII "1864"
EBX 00000001
ESP 0012FC90
EBP 0012FCBC
ESI 00000004
EDI 00850888 ASCII "ap0x"
```

5 Prolaz

Ako pogledate u donjem desnom cosku videcete sta se desava sa ovim brojevima pod navodnicima.



Ovde na slici vidimo da se brojevi polako dodaju jedan na drugi i da se ispod stinga “ap0x” polako stvara neki broj. Ovaj broj moze biti bas seriski broj koji mi trazimo. Polako sa F8 prelazimo preko redova i posmatramo registre, gore desno, cekajuci da se pojavi seriski broj koji smo uneli (111111). Ovde se desava jako zanimljiva stvar. I nas uneti seriski broj i ovaj novi koji smo primetili da se polako stvara u onoj petlji pojaljuju se ovde:



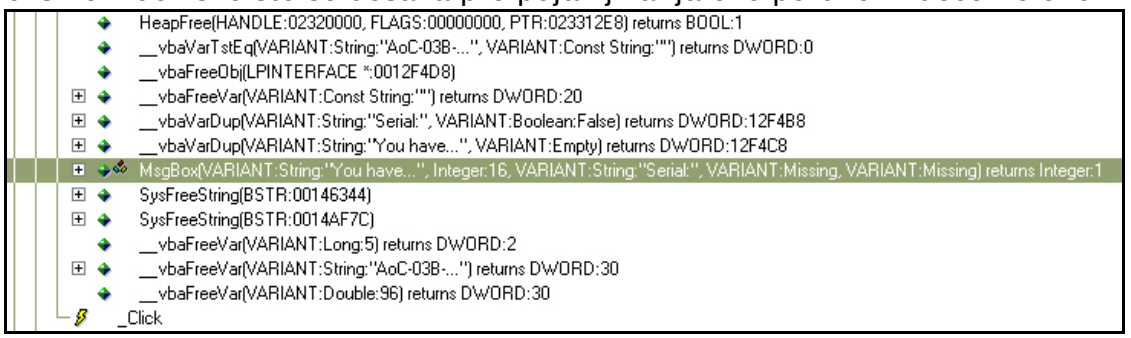
Stigli smo do adrese 00407DD8 i u registrima se nalaze stvari prikazane izvojene na ovoj slici. EAX sadrzi moguci pravi seriski broj a EDX nas uneti lazni seriski broj. Posto je posle izvrsavanja CALLa na adresi 00407DD8 odluceno da li je seriski broj ispravan ili ne, pa zakljucujemo da ovaj CALL sluzi za uporedjivanje unetog (111111) seriskog broja i ispravnog seriskog broja u EAXu. Sledeci (crveni) red 00407DDD proverava rezultat izvrsavanja CALLa. Ako su EAX i EDX jednaki program ce nastaviti dalje a ako su razliciti onda ce prikazati poruku o pogresnom seriskom broju. Ovo znaci da je pravi seriski broj za ime ap0x jednak 14641794194413041864. Ovde moram da napomenem da se seriski brojevi uvek proveravaju direktno ispred (*iznad*) poruke o pogresnom seriskom broj. Znaci ne mnogo iznad u kodu se moraju uporediti na neki nacin dve vrednosti, vrednost naseg unetog, laznog seriskog broja i ispravnog broja za uneto ime. Kao sto ste primetili “pecanje” seriskog broja se zasniva na namernom unosenju pogresnog seriskog broja u metu kako bi smo pronasli mesto gde se pogresan seriski broj proverava sa tacnim seriskim brojem. Kada pronadjemo ovo mesto preostaje nam samo da prepisemo ovaj tacan seriski broj i da ga unesemo u metu umesto naseg laznog seriskog broja.

Vezba:

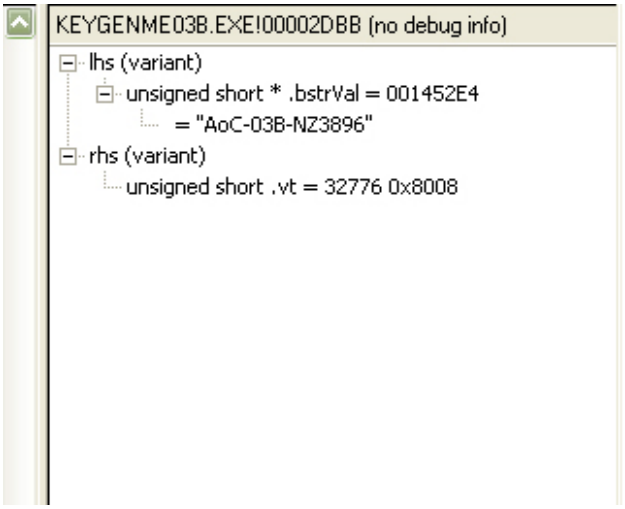
Ako zelite mozete proveriti znanje steceno o trazenju seriskog broja za uneto ime jednostavnim unosenjem nekog drugog prozivoljnog imena i trazenjem pravog seriskog broja za to ime.

The Serials – *Smart Check*

Treci deo ovog poglavlja ce vas nauciti kako da nadjete tacan seriski broj za bilo koje uneto ime uz pomoc Smart Checka. Ovaj program je specifican po svojoj upotrebi i koristi se za programe koji su pisani iskljucivo u Visual Basicu. Za potrebe ovog dela napravljen je specijan primer koji se nalazi ovde ...\Casovi\Cas3\keygenme03b.exe i njega cemo otvoriti pomocu SCa. Pritisnite F5 da bi ste startovali ovu "metu". Kada se ona pojavi u nju unesite kao ime ap0x a kao seriski broj 111111 i pritisnite Check. Pojavila se poruka o pogresnom seriskom broju. Kliknucete na OK i u SCu potrazicemo ovu poruku rucno (*moze i preko opcije find*). Pored zadnjeg reda [+]Click vidimo da imamo [+] i da ta grana moze da se rasiri. Uradimo to i na samom kraju ove grane videcemo jedno Click koje oznacava kraj te grane a odmah iznad njega MsgBox komandu sa tekstom o pogresnom seriskom broju. Selektovacemo nju i u meniju cemo kliknuti na ***View -> Show All events...*** da bi smo videli sve sto se desava pre pojavljivanja ove poruke. Videcemo ovo:



Ono sto mora da se desi pre pojavljivanja ovog MsgBoxa je neko poredjenje vrednosti. Obicno je u pitanju prvo poredjenje iznad samog MsgBoxa. Prvo poredjenje koje vidimo iznad MsgBoxa je _vbaVarTstEq koje poredi neku vrednost koja licni na tacan seriski broj. Posto se ne vidi ceo, selektovacemo



ga i desno ako se ne vidi ovaj deo ekrana, povucite ScrollBar levo i videcete ceo taj string u trecem redu. Dakle mozete i probati ovaj seriski broj sa imenom ap0x. Videcete da je seriski ispravan. Ovo je najjednostavniji primer na kojem se zasnivaju skoro svi programi pisani u Visual Basicu, tako da je seriske brojeve veoma lako naci. Ako zelite da vezbate mozete pronaci seriski u meti ...\Cas3\CrackMe1_bf.exe

The Serials – *Computer ID*

Do sada smo obradili programe koji generisu seriske brojeve na osnovu unetog imena. Ono sto preostaje u ovom opusu je generisanje seriskih brojeva na osnovu nekog hardwareskog parametra, to jest na osnovu computer IDa. Ono sto vecini korisnika nije jasno je da svaki program za sebe generise poseban computer ID, koji se razlikuje od programa do programa. Iako se seriski broj zasniva na nekom computer IDu sam princip generisanja je isti posto se seriski broj opet generise na osnovu neke konstante. Za potrebe ovog dela knjige sa interneta sam skinuo sledeci crackme ...\Casovi\Cas3\abexcm5.exe Posto je Abexov program pisan u TASMu jedini alat koji ce nam trebati je OllyDBG. Otvorite Olly i ucitajte ovaj fajl u njega. Kao sto primecujemo ovo je jako kratak program, pocinje na adresi 00401000 a zavrsava se na adresi 00401171. Takodje su ocigledne dve poruke koje se nalaze u programu, *"The serial you entered is not correct!"* i *"Yep, you entered a correct serial!"*. Primeticemo komandu RET 10 na adresi 00401069 koja oznacava da stvaran deo za proveru seriskog broja ne pocinje na adresi 00401056 nego na adresi 0040106C, stoga cemo postaviti break-point na tu adresu.

```
00401069  |. C2 1000          RET 10
0040106C  |>  6A 25           PUSH 25
0040106E  |.  68 24234000     PUSH abexcm5.00402324
00401073  |.  6A 68           PUSH 68
00401075  |.  FF75 08         PUSH DWORD PTR SS:[EBP+8]
00401078  |.  E8 F4000000     CALL <JMP.&USER32.GetDlgItemTextA>
```

Sa F9 cemo startovati program i bez ikakvog unosenja seriskog broja cemo pritisnuti Check. Normalno, program je zastao na break-pointu. Evo sta se desava... Prvo program ucitava nas uneti seriski broj u memoriju na adresi 00401078. Dalje se prosledjuju parametri funkciji koja vraca ime particije C:\ to jest takozvani volume label na adresi 00401099.

```
0040107D  |.  6A 00           PUSH 0
0040107F  |.  6A 00           PUSH 0
00401081  |.  68 C8204000     PUSH abexcm5.004020C8
00401086  |.  68 90214000     PUSH abexcm5.00402190
0040108B  |.  68 94214000     PUSH abexcm5.00402194
00401090  |.  6A 32           PUSH 32
00401092  |.  68 5C224000     PUSH abexcm5.0040225C
00401097  |.  6A 00           PUSH 0
00401099  |.  E8 B5000000     CALL <JMP.&KERNEL32.GetVolumeInformation>
```

Na adresi 004010A8 se poziva funkcija lstrcatA koja spaja dva stinga. Prvi string je volume label a drugi je uvek isti i jednak je 4562-ABEX.

```
0040109E  |.  68 F3234000     PUSH abexcm5.004023F3       ; /StringToAdd = "4562-ABEX"
004010A3  |.  68 5C224000     PUSH abexcm5.0040225C       ; |ConcatString = ""
004010A8  |.  E8 94000000     CALL <JMP.&KERNEL32.lstrcatA>; \lstrcatA
```

Dalje na adresi 004010AF pocinje loop koji se izvrsava dva puta i sluzi za menjanje prva cetiri slova ili broja iz volume labela.

```
004010AD  |.  B2 02           MOV DL,2
004010AF  |>  8305 5C224000>/ADD DWORD PTR DS:[40225C],1
004010B6  |.  8305 5D224000>|ADD DWORD PTR DS:[40225D],1
004010BD  |.  8305 5E224000>|ADD DWORD PTR DS:[40225E],1
004010C4  |.  8305 5F224000> |ADD DWORD PTR DS:[40225F],1
004010CB  |.  FECA            |DEC DL
004010CD  |.^ 75 E0           \JNZ SHORT abexcm5.004010AF
```

Idemo dalje i na adresi 004010D9 se ponovo spajaju dva stringa L2C-5781 i nista, stoga je rezultat uvek L2C-5781. Sa F8 se krecemo dalje i primecujemo da se ponovo spajaju dva stringa. Prvi je L2C-5781 a drugi je vec spojeni i promesani volume label na koji je “zalepljen” string 4562-ABEX, stoga je rezultat sledeci: L2C-5781volumelabel4562-ABEX, gde je volumelabel promesano ime particije C:\ Ostaje samo da se na adresi 004010F7 uporede uneti seriski broj i ovaj dobijeni seriski broj pomocu funkcije lstrcmpiA. Ova funkcija vraca 0 ako su uneti seriski i tacan seriski broj isti a 1 ako su razliciti.

```
004010CF  |.  68 FD234000     PUSH abexcm5.004023FD          ; /StringToAdd = "L2C-5781"
004010D4  |.  68 00204000     PUSH abexcm5.00402000          ; |ConcatString = ""
004010D9  |.  E8 63000000     CALL <JMP.&KERNEL32.lstrcatA> ; \lstrcatA
004010DE  |.  68 5C224000     PUSH abexcm5.0040225C          ; /StringToAdd = ""
004010E3  |.  68 00204000     PUSH abexcm5.00402000          ; |ConcatString = ""
004010E8  |.  E8 54000000     CALL <JMP.&KERNEL32.lstrcatA> ; \lstrcatA
004010ED  |.  68 24234000     PUSH abexcm5.00402324          ; /String2 = ""
004010F2  |.  68 00204000     PUSH abexcm5.00402000          ; |String1 = ""
004010F7  |.  E8 51000000     CALL <JMP.&KERNEL32.lstrcmpiA>; \lstrcmpiA
004010FC  |.  83F8 00         CMP EAX,0
004010FF  |.  74 16           JE SHORT abexcm5.00401117
```

Na osnovu toga se na adresama 004010FC i 004010FF odlucuje da li je seriski broj ispravan ili ne.

Ovo je samo jedan od mogucih primera kako se koriste hardware-ske komponente u generisanju seriskiog broja i kako se tako dobijeni seriski broj uporedjuje sa nasim unetim laznim seriskim brojem. Cesti su primeri da programi koriste karakteristicne kljuceve koji su validni samo za jedan kompjuter. Ovakvi kljucevi se nazivaju computer IDovi i zavise od hardwareskih i softwareskih komponenti samog sistema. Na osnovu ovog broja ili niza slova se generise unikatni seriski broj validan samo za masinu na kojoj je computer ID jednak spisku komponenti u odnosu na koje se racuna. U ovom slucaju da bi seriski broj radio na svim masinama moraju se napraviti keygeneratori koji ce sa uneti computer ID generisati tacan seriski broj. Naravno moguce je napraviti i keygenerator koji ce vam otkriti tacan computer ID za vas kompjuter u slucaju da sam program ne daje takve informacije o samom sebi. Oba postupka su ista i bice obradjena u odelju koji se bavi keygeneratorima.

The Serials – *API Fishing*

Ovo je drugi deo poglavlja o takozvanom “pecanju” seriskih brojeva. Ovaj deo knjige ce vas naucitit kako da iskoristite Windowsove API funkcije kako bi ste nasli mesto gde se racuna pravi seriski broj. Primer za ovaj deo knjige se nalazi u folderu Cas3 a zove se keygenme1.exe. Ovaj primer sam izabrao zato sto se u njemu ne nalaze klasicni stringovi koji bi vam pomogli da nadjete mesto gde se racuna i proverara seriski broj. Otvoricemo ovaj program uz pomoc Ollya i pocecemo sa potragom za pravim seriskim brojem. Pre nego sto pocnemo moracemo da znamo sve API funkcije pomocu kojih se citaju podaci iz polja za unos. Te API funkcije se nalaze u sledecoj tabeli:

Win16	Win32
GetWindowText	GetWindowTextA
GetDlgitemText	GetDlgitemTextA
GetDlgitemInt	GetDlgitemIntA

Sve API funkcije, koje poziva nasa meta se mogu videti u prozoru executable modules -> view names. Do ovoga cemo stici jednostavnim klikom na dugme E iz toolbara, selekcijom naseg exe fajla u novom prozoru i izabirom opcije View Names do koje stizemo desnim klikom na nas exe fajl. U prozoru view names cemo pronaci neke od API funkcija iz gornje tabele. Jedini API poziv koji cemo naci je GetWindowTextA pa cemo desnim klikom na njega postaviti break-point na svakoj referenci kao ovom APIu. Sada cemo se vratiti u nasu metu i kao ime i seriski broj unecemo: ap0x, 111111. Posle ovoga cemo pritisnuti OK i nacicemo se ovde:

```
00401074  |. 8B1D 9C404000   MOV EBX,DWORD PTR DS:[<&USER32.GetWindow>;
0040107A  |. 8D45 94         LEA EAX,DWORD PTR SS:[EBP-6C]
0040107D  |. 6A 64           PUSH 64                    ; /Count = 64 (100.)
```

Sada sa 100% sigurnoscu znamo da se negde ispod proverava seriski broj. Sa F8 cemo ispitati kod ispod nase trenutne pozicije.

```
004010AC  |. 50              PUSH EAX                          <- Ucitaj ime
004010AD  |. E8 CE020000     CALL keygenme.00401380            <- Vraca duzinu imena
...
004010CC  |> \8D85 30FFFFFF LEA EAX,DWORD PTR SS:[EBP-D0]     <- Ucitaj uneti seriski
...
004010D3  |. E8 A8020000     CALL keygenme.00401380            <- Vraca duzinu seriskog
...
00401104  |> /8A4C15 94      /MOV CL,BYTE PTR SS:[EBP+EDX-6C] <- Pocetak generisanja
00401108  |. 0FBEC1          |MOVSX EAX,CL                     <- tacnog seriskog broja
...
004011E1  |.^\7C E0          \JL SHORT keygenme.004011C3       <- Kraj generisanja
```

Posle ove petlje sledi sredjivanje rezultata u ASCII oblik pomocu wsprinfA funkcije. Na zadnoj adresi, 004011FE, cete primetiti da se u EAXu nalazi neki

```
004011E3  |> \52            PUSH EDX                          ; /<%lu>
004011E4  |. 52              PUSH EDX                          ; |<%X>
...
004011F1  |. FF15 B0404000  CALL DWORD PTR DS:[<&USER32.wsprintfA>] ;\wsprintfA
...
004011FE  |. 8D85 CCFEFFFF  LEA EAX,DWORD PTR SS:[EBP-134]     ; <- EAX sadrzi seriski
```

broj koji moze biti pravi seriski broj. Ako pogledate malo ispod videcete CALL koji poredi ovaj broj sa nasim unetim 111111, stoga je sadrzaj EAXa pravi seriski broj. Uspeli smo, tacan seriski je ED0C68403977013312.

The Serials – *VB & Olly*

Kao sto smo videli moguce je naci seriski broj sa specijalizovanim programom za VB programe koji se zove Smart Check. Sada cu vam pokazati kako se sa istom takvom lakocom mogu pronaci seriski brojevi uz pomoc Ollya. Otvorite primer keygenme03b.exe pomocu Ollya. Kada ovo uradite pogledajte imena koja se nalaze kao importi u ovom exe fajlu. Najcesce koriscene funkcije za racunanje seriskih brojeva su:

```
????????  .text    Import    MSVBVM60.__vbaCmpStr
00401010  .text    Import    MSVBVM60.__vbaLenBstr
004010C8  .text    Import    MSVBVM60.__vbaStrMove
00401030  .text    Import    MSVBVM60.__vbaVarCmpGe
00401090  .text    Import    MSVBVM60.__vbaVarCmpLe
0040105C  .text    Import    MSVBVM60.__vbaVarTstEq
```

Posto se skoro sve one nalaze u ovom exe fajlu problem predstavlja stvar izbora. Naravno da mozemo da postavimo break-point na sve ove importe ali cemo uraditi sledece. Prvo cemo u crackme uneti neko ime i neki seriski broj, a onda cemo postaviti break-point samo na importe koji sluze za poredjenje vrednosti. Posto u ovom exe fajlu nemamo vbaStrCmp postavicemo break-point on every reference na vbaVatTstEq, vbaVarCmpGe i vbaVarCmpLe. Kada uradimo ovo pritisnucemo Check u crackmeu i nacicemo se ovde:

```
00402CDC  . FF15 30104000          CALL DWORD PTR DS:[MSVBVM60.__vbaVarCmpGe ]
00402CE2  . 8D4D BC                LEA ECX,DWORD PTR SS:[EBP-44]
00402CE5  . 50                     PUSH EAX
```

Ovo nam ne govori mnogo pa cemo pritiskati F9 sve dok ne dodjemo do poslednjeg break-pointa koji se izvrsi bas pred prikazivanje poruke o pogresnom seriskom broju.

```
00402DBA  . FF15 5C104000          CALL DWORD PTR DS[MSVBVM60.__vbaVarTstEq]
00402DC0  . 8D4D B0                LEA ECX,DWORD PTR SS:[EBP-50]
00402DC3  . 8BF0                   MOV ESI,EAX
00402DC5  . FF15 E0104000          CALL DWORD PTR DS:[ MSVBVM60.__vbaFreeObj]
```

Ako pogledamo malo ispod videcemo sledece:

```
00402DFA  . C785 48FFFFFF>         MOV  DWORD  PTR  SS:[EBP-B8],keygenme.00402>;
UNICODE "Serial:"
...
00402E20  . C785 58FFFFFF>         MOV  DWORD  PTR  SS:[EBP-A8],keygenme.00402>;
UNICODE "You have entered correct serial!"
...
00402E6D  . C785 58FFFFFF>MOV DWORD PTR SS:[EBP-A8],keygenme.00402>; UNICODE
"You have entered WRONG serial!"
```

Ucicemo u CALL vbaVarTstEq sa F7 kako bi smo videli sta se to poredi. Kada dodjemo do adrese 66109845 videcemo u registrima sledece:

```
EAX 00000001
ECX 0012F344
EDX 0014B85C UNICODE "AoC-03B-NZ3896"
EBX 660FEA4C MSVBVM60.__vbaFreeVar
ESP 0012F3F4
EBP 0012F528
ESI 00C71854
EDI 00000000
EIP 66109845 MSVBVM60.66109845
```

Kao sto vidimo vrednosti sa kojom se poredi nas uneti seriski je AoC-03B-NZ3896 Na ovaj nacin se pronalaze sva moguca poredjenja laznog seriskog broja sa tacnom vrednoscu seriskog broja.

The Serials – *Patching*

Ponekad je potrebno naterati program da misli da je svaki uneti seriski broj ispravan. Ova tema je vec dotaknuta na samom pocetku poglavlja ali cemo ovde tu pricu prosiriti sa novim cinjenicama i olaksati sebi u mnogo cemu posao. Primer za ovaj deo poglavlja se nalazi u folderu Cas2 a zove se patchme.exe. Otvoricemo ovaj program pomocu Ollya i potrazicemo string koji se pojavljuje na ekranu kada unesemo pogresan seriski broj. Ta poruka je "Bad Cracker" - bez navodnika, a nacicemo je na standardan nacin pomocu string referenca. Poruka se nalazi ovde:

```
004087C8  |> \68 18884000    PUSH patchme.00408818          ; /Text = "Bad Cracker"
```

a ako pogledamo malo gore videcemo i poruku o ispravnom seriskom broju.

```
004087B1  |. 68 0C884000     PUSH patchme.0040880C          ; /Text = "Cracked ok"
```

Primeticemo jedan kondicioni skok iznad poruke o ispravnom seriskom broju.

```
004087A9  |. E8 7EFDFFFF     CALL patchme.0040852C
004087AE  |. 48              DEC EAX
004087AF  |. 75 17           JNZ SHORT patchme.004087C8
```

Tom skoku predhodi jedan CALL od kojeg direktno zavisi da li ce ovaj skok biti izvrsen, a ovo znaci da je ovo mesto na kome se proverava uneti seriski broj, stoga cemo postaviti jedan break-point na taj CALL i pokrenucemo program. U polja za unos cemo uneti ap0x kao ime a kao seriski broj 111111, pa cemo pritisnuti Check. Naravno program je zastao na nasem break-pointu. Sa F7 ucicemo u njega i nacemo se ovde:

```
0040852C  /$ 55              PUSH EBP
0040852D  |. 8BEC            MOV EBP,ESP
0040852F  |. 81C4 E4FEFFFF   ADD ESP,-11C
00408535  |. 53              PUSH EBX
```

Ono sto znamo o ASM komandama pomoce nam da resimo ovaj problem. Ako pogledamo adrese 004087AE i 004087AF videcemo da se EAX smanjuje za jedan, a ako je nesto manje ili jednako necemu onda se skace na poruku o pogresnom seriskom broju. Ovo nesto je zero flag. Dakle ako je EAX jednak minus 1 odnosno FFFFFFFF onda ce se skociti na poruku o pogresnom seriskom broju. Da bi smo ovo izbegli moramo da EAXu dodelimo takvu vrednost da kada se od nje oduzme jedan vrednost u EAXu bude veca od minus jedan. Naravno u EAX cemo smestiti najveci sledeci broj to jest jedan, tako da kad od njega oduzmemo jedan JNZ ne bude izvrsen jer je EAX posle oduzimanja jednak nula a ne minus jedan. Ovu izmenu koda cemo raditi unutar samog CALLa a ne pre CALLa jer ne znamo da li se ova funkcija poziva vise puta i odakle, tako da je najsigurnije da to uradimo u samom CALLu. Da bi smo uradili ovo sto smo zamislili selektovacemo adresu 0040852C i dulpim klikom izmenicemo njen sadrzaj u ovo:

```
0040852C    B8 01000000     MOV EAX,1
00408531    C3              RET
```

Izmenili smo ASM kod u ovo jer ovaj kod radi bas ono sto mi zelimo. On EAXu dodeljuje vrednost jedan i pomocu komande RET se vraca iz ovog CALLa, a samim tim ce svaka provera koja je prosledjena ovom CALLu uvek biti uspesna to jest za svaki uneti seriski broj i ime program ce pokazivati da je registrovan. Na isti nacin se rasava ovaj isti problem ako se umesto DEC EAX komande nalazi TEST EAX,EAX komanda. Ovaj primer se cesto srece u praksi a posledica je loseg nacina programiranja jer ovakav primer

predstavlja jednostavan poziv funkciji koja vraca vrednosti True ili False u zavisnosti da li je uneti seriski broj ispravan ili ne. Imajte na umu da se ovakva vrsta zastite, bez obzira koliko ona bila slozena unutar koda CALLa, resava za svega par minuta. Dakle resenje ovog problema shvatite tako sto nikada nemojte da pisete funkcije koje ce vracati vrednosti True ili False pri proveri seriskog broja, rutine za proveru pisite drugacije.
Iako izgleda da smo sa ovim primerom zavrsili to nije tacno. Ovaj program kao i mnogi dugi belezi podatke o registraciji negde u sistemu. Mi cemo pronaci mesto gde se belezi ova informacija i zaobicicemo proveru tacnosti ove informacije.
Kao sto sto smo vec videli, jednostavnim patchovanjem resavamo problem funkcija koje proveravaju tacnost nekog seriskog broja. Ono sto se sada pitamo je da li smo zaobisli vazne provere koje se mogu nalaziti unutar samog CALLa. Pogledacemo malo detaljnije CALL i videcemo sledece stvari:

1) On generise fajl TheArtOfCracking.key u koji se najverovatnije zapisuje registracija.
2) Ovaj fajl ce biti smesten u Windows direktorijum tako da kada se sledecu put crackme startuje, ovi podaci ce se ucitati i proveriti
3) Postoji jedan zanimljiv skok:

```
004085DE    /0F85 B8000000  JNZ patchme.0040869C
```

Ova skok nas vodi do ovde:

```
0040869C  |> \33DB          XOR EBX,EBX
0040869E  |>  33C0          XOR EAX,EAX
```

a sluzi za dodeljivanje vrednosti 0 EAXu i EBXu. Posto znamo da EAX na izlazu iz ovog CALL mora biti jednak jedan, ovo znaci da se gornji skok ne sme izvrsiti. Ono sto nas trenutno buni je to sto ako se ne izvrsi gornji skok primeticemo da ce se sledeca dva reda uvek izvrsiti:

```
00408695  |. BB 01000000    MOV EBX,1
0040869A  |. /EB 02         JMP SHORT patchme.0040869E
```

Hmmm, ovde imamo mali problem u EBX se smesta 1 a ne u EAX. Ako pogledamo odmah ispod prvog donjeg RETa videcemo odgovor na ovo pitanje:

```
004086C0   .  8BC3          MOV EAX,EBX
004086C2   .  5E            POP ESI
004086C3   .  5B            POP EBX
004086C4   .  8BE5          MOV ESP,EBP
004086C6   .  5D            POP EBP
004086C7   .  C3            RET
```

Aha na samom kraju se EAXu dodeljuje vrednost iz EBXa.

Ostaje nam samo da postaramo da se skok sa adrese 004085DE nikada ne izvrsi jer ce tada oba uslova biti ispunjena. I fajl ce biti snimljen i EAX ce uvek biti jednako jedan. Stoga cemo samo NOPovati JUMP na adresi 004085DE i fajl ce biti snimljen sa bilo kojim unetim imenom i seriskim brojem. Ako uradimo trajni patch na ovoj adresi videcemo da ako ponovo startujemo ovaj crackeme on ce uvek biti registrovan sa bilo kojim unetim imenom i sa bilo kojim unetim seriskim brojem.

The Serials – *KeyFile*

Cesto sretani problem prilikom reversinga su i takozvani keyfajlovi. Ovi fajlovi predstavljaju samo mesto gde su sacuvani podaci vezani za registraciju. Mi cemo uz pomoc Ollya i vaseg omiljeno Hex editora razbiti jednu ovakvu zastitu. Meta se zove CrackMe.exe a nalazi se u Cas3 folderu.

Najcesce se za citanje fajlova koristi API funkcija CreateFileA pa cemo postaviti break-point na nju. Ovo cemo uraditi na standardan nacin: U Ollyu cemo izabrati Executable modules -> View names -> CreateFileA -> Set breakpoint on every reference...

Posle ovoga cemo startovati program pomocu opcije Run, posle cega ce program zastati na sledecem mestu:

```
0040416A   .  E8 9DD0FFFF          CALL <JMP.&KERNEL32.CreateFileA>
```

Sa F8 cemo izvrsiti ovaj CALL i zavrsicemo ovde:

```
0040120C   $- FF25 04914200        JMP DWORD PTR DS:[<&KERNEL32.CreateFileA>
```

Uklonite ovaj break-point. Sada cemo pritiskati F8 onoliko puta koliko nam treba da se vratimo iz kernel32.dll. Pritiskajte F8 dok ne izvrsite prvu RET komandu. Posle cega cemo se vratiti ovde:

```
0040416F   > /83F8 FF              CMP EAX,-1
00404172   . /74 29                JE SHORT CrackMe.0040419D
```

Ovaj CMP proverava da li postoji fajl na disku koji sadrzi podatke o registraciji programa. Kako se zove taj fajl??? Ova informacija je sigurno morala da bude prosledjena gornjoj CALL CREATEFILEA funkciji, jer ovaj API mora da zna koji fajl treba da otvori. Zbog ovoga cemo postaviti break-point na prvu PUSH komandu koja se prosledjuje tom CALLu. Postavicemo break-point ovde:

```
0040415A   > \6A 00                PUSH 0                   ; /hTemplateFile = NULL
```

Pritisnite F9 kako bi ste nastavili sa izvrsavanjem programa. Sada nazad u crackmeu pritisnite dugme Try Again i program ce zastati na novo postavljenom break-pointu. Sada cemo izrsiti ovaj CALL sa F8 sve dok ne dodjemo do PUSH EAX komande. Kada dodjemo do nje sadrzaj EAXa ce biti ime fajla koji nam treba. Taj fajl je ctm_cm02.key.

Sada cemo napraviti jedan fajl sa sadrzajem jednog stringa, sa stringom ap0x (nula a ne O). Ovo mozete uraditi sa Notepadom ili sa nekim Hex editorom. Pitanje je samo gde ce se to ovaj fajl nalaziti ??? Odgovor je jednostavan: On ce se nalaziti u istom direktorijumu ili bi PUSH EAX funkcija sadrzala celu putanju do ctm_cm02.key fajla.

Sada cemo ponovo pritisnuti dugme Try again u crackmeu i program ce zastati na PUSH EAX break-pointu. Polako cemo izvrsavati sve redove koda sa F8 i posmatracemo sta se desava. Ovde:

```
0040416F   > /83F8 FF              CMP EAX,-1
00404172   . /74 29                JE SHORT CrackMe.0040419D
```

se sada ne izvrsava JE skok pa cemo se posle izvrsavanja donje RET komande naci ovde:

```
00426592   .  E8 4DC1FDFF          CALL CrackMe.004026E4
...
004265AA   .  E8 F9C0FDFF          CALL CrackMe.004026A8
```

kada dodjemo do ovog drugog CALL videcemo da EAX sada sadrzi broj 4 sto je duzina naseg stringa u key fajlu. Vidimo i koji red ispod da se EAX koristi za proveru da li je key fajl prazan ili ne.

```
004265B2  . 837D FC 00            CMP DWORD PTR SS:[EBP-4],0
...
004265B8  . BA 64674200           MOV EDX,CrackMe.00426764
...
004265CD  > \817D FC 00000>       CMP DWORD PTR SS:[EBP-4],10000
```

Kao sto vidimo u fajlu mora biti nesto zabelezeno a to nesto mora biti krace od 0x10000h karaktera. Prvi put se u registima pojavljuje adresa koja sadrzi nas uneti string na adresi 00426600.

```
00426600  . 8DB5 FCFFFEFF        LEA ESI,DWORD PTR SS:[EBP+FFFEFFFC]
```

Izvrsavajuci kod red po red dolazimo do sledece petlje:

```
00426616  > /8A1C16               MOV BL,BYTE PTR DS:[ESI+EDX]
00426619  . |84DB                 TEST BL,BL
0042661B  . |74 29                JE SHORT CrackMe.00426646
0042661D  . |E8 16000000          CALL CrackMe.00426638
00426622  . |52                   PUSH EDX
00426623  . |F7E3                 MUL EBX
00426625  . |5A                   POP EDX
00426626  . |35 326D5463          XOR EAX,63546D32
0042662B  . |FEC2                 INC DL
0042662D  . |39CA                 CMP EDX,ECX
0042662F  . |74 42                JE SHORT CrackMe.00426673
00426631  . |80FA FF              CMP DL,0FF
00426634  . |74 3D                JE SHORT CrackMe.00426673
00426636  .^\EB DE                JMP SHORT CrackMe.00426616
```

Analizom dolazimo do zakljucka da se na prvoj adresi ove petlje 00426616 u registar BL smesta hex vrednost svakog slova iz unetog stringa. Ovaj registar se koristi da bi se u EAX smestio neki broj. Mozda cak i pravi seriski broj ! Primeticete da se u ovom loopu nalazi jedan CALL. Ucicemo u njega da vidimo sta se tu desava:

```
00426638  /$ 57                   PUSH EDI
00426639  |. 8DBD F4FFFEFF        LEA EDI,DWORD PTR SS:[EBP+FFFEFFF4]
0042663F  |. 8B3F                 MOV EDI,DWORD PTR DS:[EDI]
00426641  |. 881C17               MOV BYTE PTR DS:[EDI+EDX],BL
00426644  |. 5F                   POP EDI
00426645  \. C3                   RET
```

Kao sto vidimo nista specijalno samo se slova iz stringa smestaju na neki duzi string. Posto skok:

```
0042661B  . /74 29                JE SHORT CrackMe.00426646
```

vodi van gornjeg loopa, postavicemo jedan break-point na adresu na koju on vodi. Postoji jos par skokova koji vode van tog loopa ali kako vidite ovde:

```
00426619  . 84DB                  TEST BL,BL
```

taj skok ce se izvrsiti kada se iskoriste sva slova iz unetog stringa. Takodje cemo postaviti break-point na adresu na koju vode ostala dva skoka:

```
00426634  . /74 3D                JE SHORT CrackMe.00426673
```

Prodjimo vise puta kroz ovaj loop i videcemo da se skok ispod TEST BL,BL nikada nece izvrsiti. Hmmm... ovde nesto nevalja, jer ako se ne izvrsi ovaj skok mi cemo zavrsiti na delu koji samo prikazuje poruku o pogresnom seriskom broju na ekran. Sledi pitanje: Kako da BL bude nula ???

Odgovor je jednostavan: U BL se smestaju hex vrednosti slova iz unetog stringa, jedan po jedan. JE skok ce se izvrsiti samo ako je BL jednak 0x00 stoga cemo pomocu Hex editora na kraj naseg stringa dodati 0x00 bajt. Posle ovoga cemo pritisnuti F9 kako bi smo nastavili sa izvrsavanje programa i ponovo cemo pritisnuti Try again u crackmeu. Kada zastanemo na PUSH 0 break-pointu pritisnucemo F9 2x kako bi smo dosli do ovde:

```
00426646  > \E8 EDFFFFFF          CALL CrackMe.00426638
```

Pritsnucemo F8 4x dok ne dodjemo do ovde:

```
0042664F  . 39D1                CMP ECX,EDX
```
Vrednosti koje se nalaze u ECXu i EDXu su:
```
ECX = 5
EDX = 9
```
Sta se ovde poredi ??? Duzina stringa ap0x + 0x00 sa 9. Ovo znaci da duzina stringa mora da bude 9. Sa Hex Editorom cemo dodati string 1234 na kraj key fajla. Ponovo cemo morati da pritisnemo F9 i da u crackmeu cemo kliknuti na Try again dugme. Ponovo cemo doci na adresu 00426646 i sa F8 cemo izvrsavati red po red sve dok ne dodjemo do ovde:
```
0042665C  > \3B0416             CMP EAX,DWORD PTR DS:[ESI+EDX]
0042665F  . 75 09               JNZ SHORT CrackMe.0042666A
00426661  . B8 00000000         MOV EAX,0
00426666  . 8907                MOV DWORD PTR DS:[EDI],EAX
00426668  . EB 10               JMP SHORT CrackMe.0042667A
0042666A  > B8 01000000         MOV EAX,1
0042666F  . 8907                MOV DWORD PTR DS:[EDI],EAX
00426671  . EB 07               JMP SHORT CrackMe.0042667A
```
Na prvom gornjem redu se poredi EAX = 69BB21A1 sa 34333231. Sta ovo znaci ??? Pogledajmo to ovako: 34 33 32 31 = 4 3 2 1, posto smo mi uneli kao string 1234 a ne 4321 vidimo da se brojevi okrecu. Ovo znaci da EAX sadrzi tacan seriski broj. EAX = 69BB21A1, samo morate da pazite kada ga unosite u fajl pomocu Hex Editora, ovaj broj se okrece pa cete uneti A1 21 BB 69.
Analizirajmo dalje... Kada su su ova dva broja jednaka EAX ce postati jednak 0 a ako nisu EAX ce postati jednak 1. Ovo se dole proverava i u zavisnosti od toga prikazuje se poruka ili o tacnom seriskom broju ili o pogresnom seriskom broju. Sada mozemo zatvoriti Olly i pomocu Hex editora sadrzaj fajla ctm_cm02.key promeniti u ovo:
```
61 70 30 78 00 A1 21 BB 69      ap0x..!.i
```
Sada mozete startovati crackme i videcete da je on registrovan. Dakle uspeli smo :)

Dodatna analiza:
Kao sto smo videli moramo da vodimo racuna o obliku key fajla. Ta forma izgleda ovako:
vase_ime 0x00 seriski duzine 4 bajta

Napomena:
Ako resavate problem sa kljucem nepoznatog oblika potrebno je da izvrsite detaljnu analizu dobijenih podataka. Bitno je sta se odakle cita i sa cime se poredi. Posto sam i ja prvi put reversovao ovaj primer ja sam to uradio ali je zbog duzine objasnjenja analiza problema svedena na minimalnu meru. Najbitnije je da posmatrate registre i da gledate gde se sta poredi.

04 Making KeyGens

U proslom poglavlju smo naucili kako da pronadjemo mesto na kome se proverava seriski broj i kako da saznamo gde se to racuna tacan seriski broj. U ovom poglavlju cemo nauciti kako da iskoristimo ovo znanje i da napravimo keygeneratore za razlicite programe. Za ovo poglavlje je preporucljivo znanje nekog od sledeca dva programska jezika. Morate znati ili Visual Basic ili Delphi. Posto su ova dva programska jezika relativno laka za ucenje i razumevanje samo oni ce biti obradjivani u knjizi. Naravno postoji i deo za one koji ne znaju ni jedan od ova dva programska jezika, a nalazi se na samom pocetku ovog poglavlja i on ce vas nauciti kako da izmenite kod programa i da pretvorite njega samog u svoj keygenerator.

KeyGen – *Ripping*

Poglavlje o pravljenju keygeneratora je podeljeno u vise delova. Prvi deo ce vas nauciti kako da izmenite kod u nekoj "meti" tako da ona sama postane svoj keygenerator, a ostali kako da u nekom drugom programskom jeziku napisete keygen."Meta" ce biti slicna kao na pocetku proslog poglavlja. Ucitajte program ...\Casovi\Cas4\Serial2.exe u Olly. Kao sto smo u proslom poglavlju utvrdili seriski broj se generise u petlji koja pocinje na adresi 00407D83. Postavite break-point odmah ispod ove petlje, postavite break-point na adresu 00407DB2. Sa F9 pokrenite program i unesite bilo koje ime. Zatim pritisnite Check i program ce zastati na adresi 00407DB2. Idite polako kroz program sa F8 sve dok ne dodjete do adrese 00407DD8 kada ce se u EAXu prikazati tacan seriski broj za uneto ime. Ako izvrsite CALL na toj adresi iz EAXa ce nestati tacan seriski broj, a to ne zelimo. Stoga selektujte taj CALL i pritisnite space da bi ste ga izmeniti. U novo otvoreni dijalog unesite NOP i pritisnite OK. Sada bi to trebalo da izgleda ovako:

```
00407DD2  .  8B55 EC      MOV EDX,DWORD PTR SS:[EBP-14]
00407DD5  .  8B45 FC      MOV EAX,DWORD PTR SS:[EBP-4]
00407DD8     90           NOP
00407DD9     90           NOP
00407DDA     90           NOP
00407DDB     90           NOP
00407DDC     90           NOP
00407DDD  v  75 17        JNZ SHORT Serial2.00407DF6
00407DDF     68 3C7E4000  PUSH Serial2.00407E3C          ASCII "Cracked ok"
```

Posto ni skok na adresi 00407DDD ne treba nikada da se izvrsi NOPovacemo i njega. Ako bi stvari ostavili kao sto sada jesu program bi stalno na ekranu prikazivao poruku o tacnom seriskom broju ("Cracked OK"), ali ne i tacan seriski broj. Zapamtite da u EAXu sada imate tacan seriski broj i da samo treba da ga prikazete na ekran. Iskoristicemo cinjenicu da se string "Cracked OK" prikazuje na ekranu i samo cemo PUSH komandu malo modifikovati tako da umesto tog stringa prikazuje sadrzaj iz EAXa. Ta izmena izgleda ovako:

```
00407DD8     90           NOP
00407DD9     90           NOP
00407DDA     90           NOP
00407DDB     90           NOP
00407DDC     90           NOP
00407DDD     90           NOP
00407DDE     90           NOP
00407DDF     50           PUSH EAX
00407DE0     90           NOP
00407DE1     90           NOP
00407DE2     90           NOP
00407DE3     90           NOP
00407DE4  .  68 B90B0000  PUSH 0BB9                     ControlID = BB9 (3001.)
00407DE9  .  A1 50984000  MOV EAX,DWORD PTR DS:[409850]
00407DEE  .  50           PUSH EAX                      hWnd => 000D02BA ('[ Art Of Cracking - Cas
00407DEF  .  E8 04C8FFFF  CALL <JMP.&user32.SetDlgItemTextA>  SetDlgItemTextA
00407DF4  .v EB 15        JMP SHORT Serial2.00407E0B
```

Umesto PUSH 00407E3C (*gde je ovaj hex broj adresa na kojoj se nalazi string "Cracked OK"*) stavicemo jednostavno PUSH EAX i program ce umesto stringa prikazivati sadrzaj EAXa na ekranu kada sa izvrsavanjem programa dodje do adrese 00407DEF to jest do windows api funkcije SetDlgItemTextA. Ako zelite da snimite ove promene kliknite desnim dugmetom na modifikovani deo koda i pritisnite ***Copy to executable -> All modifications -> Copy all*** i samo ga snimite pod nekim drugim imenom. Na ovaj jednostavan nacin mozemo dodavati ili modifikovati postojece funkcije u exe fajlu. Imajte na umu da ovo nije pravi keygenerator posto nismo saznali kako radi algoritam za generisanje pravog seriskog broja. Bez obzira na sve napravili smo program koji moze da prikaze tacan seriski broj za bilo koje uneto ime.

KeyGens & *OllyDBG*

A sada nesto malo komplikovanije, pokusacemo da napravimo keygenerator za nasu “metu” u dva programska jezika. Mislim da su ova dva programska jezika uobicajna i da njih zna najveci deo citalaca ove knjige, stoga su programski jezici koji ce biti obradjivani u knjizi biti Visual Basic i Delphi.“Meta” ce biti ista kao i u proslom poglavlju. Ucitajte program ...\Casovi\Cas4\Serial2.exe u Olly. Kao sto smo u proslom poglavlju utvrdili seriski broj se generise u petlji koja pocinje na adresi 00407D83. Ono sto cemo sada uraditi je sledece:

1) izvojicemo ovaj deo koda za analizu
2) pokusacemo da ponovo napisemo ovaj algoritam u nekom drugom programskom jeziku

Evo kako izgleda ta pelja izvojena:

00407D83	8B45 F8	MOV EAX,DWORD PTR SS:[EBP-8]
00407D86	8A4430 FF	MOV AL,BYTE PTR DS:[EAX+ESI-1]
00407D8A	34 2C	XOR AL,2C
00407D8C	25 FF000000	AND EAX,0FF
00407D91	03C0	ADD EAX,EAX
00407D93	8D0480	LEA EAX,DWORD PTR DS:[EAX+EAX*4]
00407D96	05 00040000	ADD EAX,400
00407D9B	8D55 F0	LEA EDX,DWORD PTR SS:[EBP-10]
00407D9E	E8 9DD5FFFF	CALL Serial2.00405340
00407DA3	8B55 F0	MOV EDX,DWORD PTR SS:[EBP-10]
00407DA6	8D45 FC	LEA EAX,DWORD PTR SS:[EBP-4]
00407DA9	E8 7EBAFFFF	CALL Serial2.0040382C
00407DAE	46	INC ESI
00407DAF	4B	DEC EBX
00407DB0	75 D1	JNZ SHORT Serial2.00407D83

Analizirajmo polako prolazak kroz ovu petlju. Treba da vam je ukljucen Olly da ste postavili break-point na adresu 00407D83, i da ste kao ime u “metu” uneli ap0x, seriski broj nije bitan.

Prvi od pet prolaz:

00407D83 Pri prvom prolazu kroz petlju u EAXu na adresi 00407D83 se nalazi broj 4, a posle izvrsenja koda na adresi 00407D83 nalazi se nase uneto ime to jest ap0x.

00407D86 Ovde se u AL stavlja nulti karakter stringa “ap0x”. Kao sto primecujete slovo a je prvi karakter. Sta je onda nulti karakter??? Nulti karakter je isti za sve stringove i iznosi 00h ili samo 0. Uvek je isti.

00407D8A Ovde se izvrsava obicna XOR operacija nad registrom AL sa vrednoscu 2Ch ili 44.

00407D8C Ovde se izvrsava logicko dodavanje 0FFh odnosno 255 na vrednosti EAX. Ovo je obicno svodjenje sa velikih hex brojeva na manje. Shvatite to ovako: ako je EAX 0085005C onda ce posle ovog logickog dodavanja rezultat biti 5C. Posto je posle izvrsenja ove komande EAX jednak registru AL ova komanda je nepotrebna i necemo je koristiti u keygenu.

00407D91 Na ovoj adresi se desava obino matematicko sabiranje vrednosti iz EAXa sa samom sobom.

00407D93 Ovde EAX dobija sledecu vrednost EAX = EAX + (EAX * 4), ovo je prosto i lako procitati samo treba obratiti paznju na matematicke operacije mnozenja.
00407D96 Opet operacija obicnog matematickog dodavanja 400h = 1024 na vrednost EAXa.
Sve do kraja nam je totalno nebitno jer kako vidimo samo se vrednost iz EAXa samo prebacuje iz brojeva u string i dodaje na novi prazan string. U sledecim prolazima desava se isto to samo se stringovi dodaju jedan na drugi, to jest vrednosti iz svakog prolaza se kao tekst dodaju jedna na drugu. Kao sto vidimo prve cetiri cifre seriskog broja su uvek iste jer se racunaju uvek za isto slovo, to jest za nulu. Ako izracunamo ovo videcemo da su prve cetiri cifre uvek 1464. Svaki sledeci prolaz se razlikuje samo u podatku koji se nalazi u Alu na adresi 00407D86, a on je uvek jednak ASCII kodu slova koje odgovara prolazu kroz petlju. Ako je u pitanju prolaz jedan, AL ce dobiti vrednost ASCII broja prvog slova iz unetog imena, i tako dalje za svako slovo. Nadam se da vam je ovo jasno i da nisam previse zakomplikovao ovo tumacenje koda. Keygenovanje samo po sebi nije jednostavno i zahteva mnogo vezbanja i iskustva, zato ne ocajavajte ako vam sve nije iz prve jasno. Trik kod pravljenja dobrih keygenova je razumevanje svakog reda vezanog za racunanje seriskog broja i njegovo rekreiranje u nekom programskom jeziku. Evo kako bi to izgledalo napisano u Visual Basicu.

```
Private Sub Command1_Click()
 User_name = Text1.Text
 Serial = “1464”
For i = 1 to Len(user_name)
 Al = Asc(Mid$(user_name,i,1))
 Al = Al xor 44
 Al = Al + Al
 Al = Al + (Al * 4)
 Al = Al + 1024
 Serial = Serial & Al
Next i
 Text2.Text = Serial
End Sub
```

Ovaj primer se nalazi vec gotov u folderu ...\Casovi\Cas4\Keygen source\VB\
A Evo kako bi to izgledalo napisano u Delphiu.

```
procedure TForm1.Button1Click(Sender: TObject);
var
 user_name,serial:string;
 al,i:integer;
begin
 serial := '1464';
 user_name := Edit1.Text;
 for i := 1 to Length(user_name) do begin
  al := Ord(user_name[i]);
  al := al xor 44;
  al := al + al;
  al := al + (al *4);
  al := al + 1024;
  serial := serial + IntToStr(al);
 end;
  Edit2.Text := serial;
end;
```

Ovaj primer se nalazi vec gotov u folderu ...\Casovi\Cas4\Keygen source\Delphi\

KeyGens & *Smart Check*

Do sada smo obradili keygenovanje programa preko Olly-a, a sada je na redu Smart Check. Za potrebe ovog dela skinuo sam primer sa interneta, on se nalazi ovde ...\Casovi\Cas3\abexcrackme2.exe, slobodno ga ucitajte u SC i pokrenite sa F5, kada se pojavi Abexov crackme unesite kao ime cracker a kao seriski 111111, pritisnite [Check] i izacice MessageBox 'Nope, the serial is wrong!'
Sada se u SCu pojavio [+]Click kao oznaka da smo mi kliknuli na dugme, duplim klikom na [+]Click pojavljuje se ovo:

```
<-><-><-><-><-><-><-><-><-><-><-><-><-><-><-><-><-><-><-><-><-><-><-><-><->
  Len(VARIANT:String:"cracker") returns LONG:1242252
  Len(VARIANT:String:"cracker") returns LONG:1242252
  Integer (1) --> Long (1)
  Mid(VARIANT:String:"cracker",long:1,VARIANT:Integer:1)
  Asc(String:"c") returns Integer:99
  Hex(VARIANT:Integer:199)
  Integer (2) --> Long (2)
  Mid(VARIANT:String:"cracker",long:1,VARIANT:Integer:1)
  Asc(String:"r") returns Integer: 114
  Hex(VARIANT:Integer:214)
  Integer (3) --> Long (3)
  Mid(VARIANT:String:"cracker",long:1,VARIANT:Integer:1)
  Asc(String:"a") returns Integer: 97
  Hex(VARIANT:Integer:197)
  Integer (4) --> Long (4)
  Mid(VARIANT:String:"cracker",long:1,VARIANT:Integer:1)
  Asc(String:"c") returns Integer: 99
  Hex(VARIANT:Integer:199)
<-><-><-><-><-><-><-><-><-><-><-><-><-><-><-><-><-><-><-><-><-><-><-><-><->
```

Na osnovu ovoga zakljucujemo da se seriski broj generise od prva 4 slova imena. Za svako slovo se izvaja ASCII broj od koga se racuna Hex broj :) Ali se za racunanje Hex broja dodaje josh 100 na ASCI vrednost svakog slova. To izgleda ovako:

```
serial = ..... Hex(Asc(Mid$(name,1,1)) + 100)
```

Ali sta se dalje radi sa seriskim??? Da li se Hex brojevi sabiraju, mnoze, dele, dodaju????

Obelezimo MsgBox i idemo na View -> Show All Events da bi smo imali detaljniji uvid u ono sto se desava. Sada vidimo ovo par redova iznad MsgBox-a:

```
<-><-><-><-><-><-><-><-><-><-><-><-><-><-><-><-><-><-><-><-><-><-><-><-><->
  _VbaVarMove(VARIANT:String:"111111",.....) returns ....          <- Nas lazni seriski broj
  _VbaFreeObj(...)
  _VbaVarTstEq(VARIANT:String:"C7D6C5C7",VARIANT:String:"111111") returns DWORD:0
<- Da li je C7D6C5C7 = 111111
  _VbaVarTstNe(VARIANT:String:"C7D6C5C7",VARIANT:String:"111111") returns DWORD:F
<- Da li C7D6C5C7 nije = 111111
  _VbaVarDup(...)
  _VbaVarDup(...)
  MsgBox(....)                                    <- Poruka o pogresno seriskom
<-><-><-><-><-><-><-><-><-><-><-><-><-><-><-><-><-><-><-><-><-><-><-><-><->
```

Analizirajmo ono sto vidimo. Nas lazni seriski broj se pojavljuje u memoriji, onda se uporedjuje sa nekom C7D6C5C7 vrednoscu i posle toga se pojavljuje poruka o netacnom seriskom broju. Znaci 100% je C7D6C5C7 bash taj seriski broj koji nama treba. Probajmo sada korisnicko ime cracker i seriski broj C7D6C5C7 i to je to, uspeli smo !!!! Program sada izbacuje poruku 'Yap, that`s the right key!' :) Posto znamo ovo od gore:

```
<-><-><-><-><-><-><-><-><-><-><-><-><-><-><-><-><-><-><-><-><-><-><-><-><->
  Len(VARIANT:String:"cracker") returns LONG:1242252
  Len(VARIANT:String:"cracker") returns LONG:1242252
  Integer (1) --> Long (1)
  Mid(VARIANT:String:"cracker",long:1,VARIANT:Integer:1)
  Asc(String:"c") returns Integer: 99
  Hex(VARIANT:Integer:199)                    <- C7
  Integer (2) --> Long (2)
  Mid(VARIANT:String:"cracker",long:1,VARIANT:Integer:1)
  Asc(String:"r") returns Integer: 114
  Hex(VARIANT:Integer:214)                    <- D6
  Integer (3) --> Long (3)
  Mid(VARIANT:String:"cracker",long:1,VARIANT:Integer:1)
  Asc(String:"a") returns Integer: 97
  Hex(VARIANT:Integer:197)                    <- C5
  Integer (4) --> Long (4)
  Mid(VARIANT:String:"cracker",long:1,VARIANT:Integer:1)
  Asc(String:"c") returns Integer: 99
  Hex(VARIANT:Integer:199)                    <- C7
<-><-><-><-><-><-><-><-><-><-><-><-><-><-><-><-><-><-><-><-><-><-><-><-><->
```

bice nam lako da napravimo keygen. Posto znamo da je seriski za ime cracker - C7D6C5C7, vise je nego ocigledno kako to program generishe seriski broj. On se racuna od prva 4 slova imena iz kojih se racuna Ascii kod za svako slovo, koji se koristi za izracunavanje Hex vrednosti. Primeticemo da je Ascii od "c" jednak 99 a da program racuna Hex(199) umesto Hex(99), isto tako za sva 4 slova se na Ascii vrednost slova dodaje 100 i od toga se racuna Hex vrednost od koje se obicnim dodavanje jedne vrednosti na drugu tkz. "lepljenjem" dobija konacni seriski broj. C7 D6 C5 C7 = C7D6C5C7.

Evo kako bi keygen izgledao u Visual Basicu.

```
ime = Text1.Text
If Len(ime) < 4 Then
 MsgBox "Unesite ime duze od 4 karaktera"
Else
 For i = 1 To 4
  serial = serial & Hex(Asc(Mid$(ime, i, 1)) + 100)
 Next
 Text2.text = serial
End If
```

A evo kako bi izgledao u Delphiu.

```
serial := '';
ime := Edit1.Text;
If Length(ime) < 4 Then begin
 messagedlg('Unesite ime duze od 4 karaktera',mtError,[mbOK],0);
end
 else
begin
 For i := 1 To 4 do begin
  serial := serial + IntToHex(Ord(name[i]) + 100);
 end;
 Edit2.text := serial;
end;
```

05 CD Checking...

Ovo poglavlje ce detaljno teoriski i prakticno obraditi osnovna pitanja vezana za proveru CD. Ovakve provere se najcesce nalaze u igricama i ne dozvoljavaju vam da ih startujete bez prisustva CDa u uredjaju. Ako ste ovaj problem do sada resavali raznim simulatorima virtualnih uredjaja ovo je pravo stivo za vas. Ne samo da cete ustedeti na prostoru nego cete nauciti i nesto novo vezano za reversni inzenjering.

CD Checking - *Examples*

Prvo cu vas upoznati sa osnovnom teorijom provere CDova. Treba da shvatite da su CD promene jako slicne uklanjanju NAG ekrana. U 90% slucajeva se radi o jednostavnim porukama koje program izbacuje ako u CD-ROMu ne nadje odgovorajuci CD. Ove poruke mozete veoma lako presresti i izmeniti program tako da se ova poruka nikada ne prikaze. Ovo cete moci uraditi samo sa programima koji nisu pakovani nekim pakerom ili nisu zasticeni nekim programom koji zabranjuje kopiranje CDova (*citaj: SafeDisc, StarForce3 i slicno*). Da bi smo odredili da li je neka igra zasticena najlakse je da otvorimo glavni ExE fajl igrice u PeIDu i vidimo da li je pakovan ili ne. Ali ne brinite postoji veliki broj novih i starih igara koji nemaju ovakvu zastitu i dobre su za praksu (*citaj: Quake 3, Worms World Party, GTA: Vice City,...*), a za one zasticene postoje programi kao sto su UnSafeDisc i slicni koji ce vam "pomoci" da odpakujete ExE fajl i sredite ga normalno kao da nije ni bio pakovan. Ove osnovne zastite koje zahtevaju prisustvo CDa u CD-Romu izgledaju manje ili vise ovako u ASMu:

```
CALL check_for_cd
TEST EAX,EAX
JE no_cd_inserted
MOV EAX,1
...
```

Ovde se poziva zamisljena adresa check_for_cd koja proverava da li postoji neki fajl na CDu, ako taj fajl ne postoji onda je pogresan ili nikakav CD ubacen. U sledecoj liniji se proverava EAX sa samim sobom sto ce za rezultat imati da zero flag (*vrednost koja moze biti samo 0 ili 1, a vrednost joj se uvek i samo dodeljuje posle nekog poredjenja tipa TEST ili CMP*) bude postavljen kao true (1) ili false (0), a ako je zero flag jednak nula onda ce program skociti na zamisljenu adresu no_cd_inserted i prikazace poruku o pogresnom CDu. Naravno ako je CD ubacen onda ce zero flag biti jednak jedan i igrica ce se startovati. Ovo problem mozemo resiti jednostavnim NOPovanjem JE no_cd_inserted skoka i onda ce za program stalno biti ubacen CD. Ali ne dajte se zavarati da ce igrica stvarno raditi bez CDa ako joj je nesto od fajlova stvarno potrebno sa CDa da bi radila.

Recimo da nas interesuje sta se nalazu u CALLu check_for_cd. Tu bi smo mogli da vidimo nesto slicno ovome:

```
check_for_cd:
   ...
   MOV ECX,03        {neke ASM funkcije, koje su nama nebitne}
   MOV EBX,401234 {neke ASM funkcije, koje su nama nebitne}
   ...
   TEST EAX,EAX
   JE good_cd
   XOR EAX,EAX
   RET
   ...
good_cd:
   MOV EAX,1
   RET
```

Ovde tri tackice naravno znace neodredjeni i proizvoljni broj linija koda. Prve dve komande MOV nisu ni bitne tu su samo da predoce neki kod. Ono sto je bitno je neka TEST provera, recimo postojanje odredjenog fajla na CDu. Ako fajl postoji onda ce se JE skok izvrsiti i EAX ce sadrzati broj 1, a ono JE no_cd_inserted se nece izvrsiti. A ako taj fajl ne postoji onda ce se izvrsiti XOR EAX,EAX sto je jednako komandi MOV EAX,0 i onda ce se onaj skok JE no_cd_inserted izvrsiti i prikazace se poruka o pogresnom CDu.

Najcesce se u CALLu koji sluzi za proveru CDa mogu naci i funkcije to jest CALLovi koji sluze za proveru tipa drajva. To jest da li je drajv koji se posmatra CD-ROM, FLOPPY, HARD-DISC, DVD,... Za ovo se koristi windowsa api funkcija GetDriveTypeA. Ona glasi ovako:

```
GetDriveType Lib "kernel32" Alias "GetDriveTypeA" (ByVal nDrive As String) As Long
```

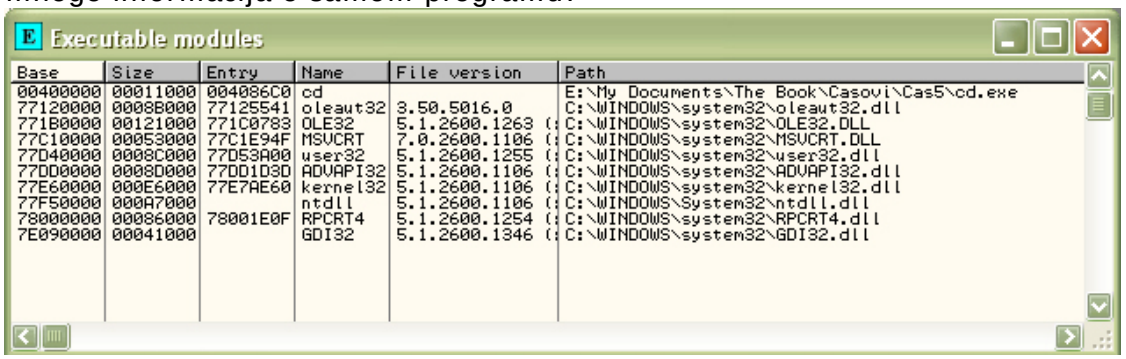
U ASMu bi ovo izgledalo odpilike ovako:

```
check_for_cd:
    ...
    MOV ECX,03       {neke ASM funkcije, koje su nama nebitne}
    MOV EBX,401234 {neke ASM funkcije, koje su nama nebitne}
    ...
    ...
    PUSH drive_letter
    CALL GetDriveTypeA
    CMP EAX,00000005
    JNE not_CD_drive
    ...
not_CD_drive:
    XOR EAX,EAX
    RET
```

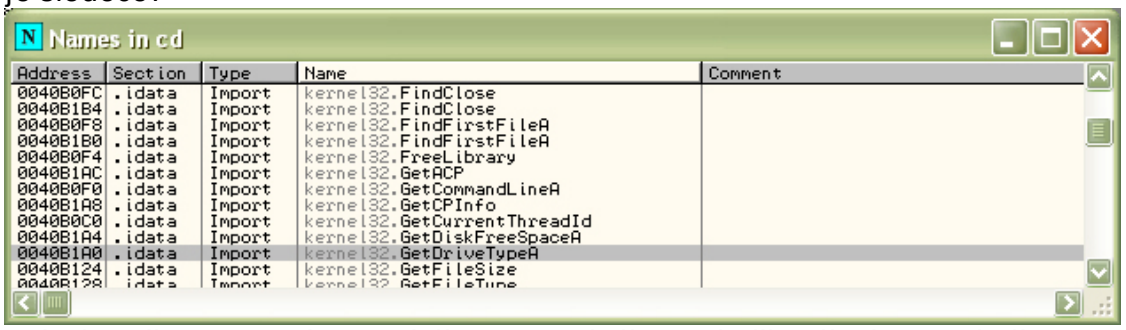
Ovde se desava sledece. Prosledjuju se potrebni parametri funkciji GetDriveTypeA, ona se poziva i vraca rezultat u EAX. Ako u EAXu nije broj 5 onda skoci na deo koji brise EAX (EAX = 0) i vrati se iz ovog CALLa. Primecujete da je identifikacioni broj CD uredjaja 5. Ovo JNE se takodje moze srediti jednim NOPom. Ova funkcija nam moze mnogo pomoci pri trazenju mesta gde se proverava tip uredjaja a samim tim i da li je pravi CD ubacen. Kao sto vidite postoji vise nacina na koji mozemo srediti CD proveru, ali najjednostavniji se sredjivanje skoka odmah posle CALL check_for_cd funkije. Ovo su samo neki teoriski primeri iz moje glave. CD provera uopste ne mora a u vecini slucajeva i ne izgleda ovako, ali najcesce postoji veoma mnogo slicnosti izmedju ovoga sto sam napisao i situacija u “stvarnom svetu”.

CD Checking - *CrackMe*

Posle teorije stize praksa. Za potrebe ovog dela knjige je napisan program koji se nalazi ...\Casovi\Cas5\cd.exe ovde. Ovo je jedan odlican primer posto tacno prikazuje kako bi radila neka CD zastita u praksi. Program ce prvo proveriti sve CD / CD-RW uredjaje u kompjuteru i na svakom od njih potrazice neki fajl, ako taj fajl postoji i ima sadrzinu koju treba da ima onda je u pitanju pravi CD. Naravno ovo je najjednostavniji primer koji se moze zaobici i virtualnim CDom ali mi to ne zelimo, zar ne ? Startujte program i videcemo da izgleda veoma obicno. Ako pritisnemo dugme :: Run Game :: pojavice se prozor o ne unetom CD u CD uredjaj. Ovo cemo da “popravimo”. Ako se secate teoriskog dela ovog poglavlja ovo ce biti izuzteno lako. Ucitajmo ovu “metu” u OllyDBG. Mogli bi smo da nadjemo mesto gde se stvara ovaj dialog ali pokusacemo nesto drugo ovaj put. Znamo da program sigurno mora da koristi Windows API funkciju GetDriveTypeA kako bi odredio koji od uredjaja u kompjuteru je CD-ROM. Sve sto treba da uradimo je da nadjemo gde se poziva ova funkcija. Sreca nasa sto koristimo Olly jer je trazanje ovakvih API poziva izuzetno lako. Pritisnimo dugme E iz gornjeg toolbara ili ALT + E na tastaturi. Ovaj prozor koji trenutno vidimo daje nam mnogo informacija o samom programu:

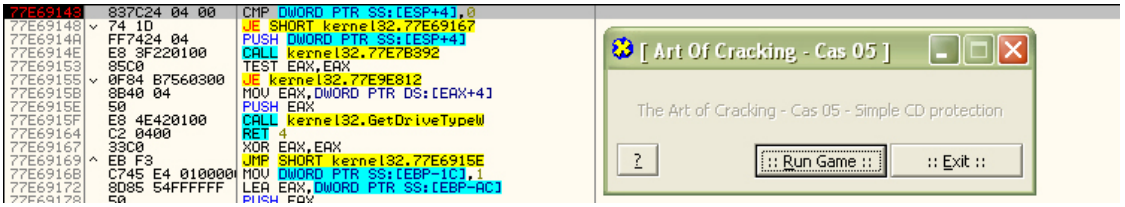


U ovom prozoru mozemo videti koji su sve fajlovi u direktnoj upotrebi od strane ovog programa, naravno Olly ce prikazati samo odredjene programske module (.vxd,.ocx,.dll i sl.) koji sadrze odredjeni ASM kod koji ovaj program poziva. Ono sto mi zelimo da saznamo je koje API pozive vrsi ovaj program. Kliknucemo desnim dugmetom misa na prvi red (on je ujedno i putanja do fajla pa i sam taj fajl) i kliknucemo na View Names. Ono sto se sada pojaljuje je sledece:



Spisak svih API poziva podeljen po sekcijama i dll fajlovima u kojima se nalaze. Da bi ste lakse nasli API koji nama treba mozete da kucate na

tastaruri njegovo ime. Sada kada smo ga nasli i selektovali postavicemo break-point na njega, pritisnite desno dugme na tom APIu i selektuje Toggle breakpoint on import, sto znaci da ce program stati kada program prvi put pozove ovaj API. Sada pokrenite program i pritisnite ::Run Game:: i program je zastao ovde:



Ako pogledate ovo cudno mesto gde smo sada nista vam nece biti jasno. Videcete samo da EAX sadrzi string A: i da ovaj deo koda najverovatnije sluzi sa proveru uredjaja A: Ali ovde nam je nesto cudno, pogledajte te adrese sa strane 77??????, kao da se nalazimo u nekom DLLu ? Ako pogledate ime prozora Ollya videcete da u njemu pise [CPU main thread, module kernel32] sto znaci da mi i jesmo u DLLu i to u kernel32.dll Skinimo ovaj break-point sa 77E69143 (*moze se razlikovati na vasem kompjuteru*) i pritiskajuci F8 sve dok se ne izvrsi i RET 4 komanda. Sada smo se iz kernel32.dll-a vratili u nasu metu. Nalazimo se u jednom malo duzoj petlji odmah ispod:

```
00408403  |. 50              |PUSH EAX                          ; /RootPathName
00408404  |. E8 93C7FFFF     |CALL JMP.&kernel32.GetDriveTypeA  ; \GetDriveTypeA
```

a odmah iznad

```
00408409  |. 83F8 05         |CMP EAX,5 <- Ovde smo
0040840C  |. 0F85 98000000   |JNZ cd.004084AA
00408412  |. 8D85 20FEFFFF   |LEA EAX,DWORD PTR SS:[EBP-1E0]
00408418  |. B9 2C854000     |MOV ECX,cd.0040852C               ; ASCII "\main.xxx"
0040841D  |. 8B55 F8         |MOV EDX,DWORD PTR SS:[EBP-8]
00408420  |. E8 FBB9FFFF     |CALL cd.00403E20
00408425  |. 8B85 20FEFFFF   |MOV EAX,DWORD PTR SS:[EBP-1E0]
0040842B  |. E8 E4D5FFFF     |CALL cd.00405A14
00408430  |. 84C0            |TEST AL,AL
00408432  |. 74 76           |JE SHORT cd.004084AA
```

Ono sto iz ovoga moze da se zakljuci a sto znamo iz teorije je da ako se EAX poredi sa 5, a sve ovo se desava ispod nekog poziva GetDriveTypeA APIa to znaci da se provera da li je drive CD-ROM ili ne. Ako se u EAXu ne nalazi 5 nego neka druga vrednost program ce skociti na adresu 004084AA, to jest ovde:

```
004084AA  |> \43             |INC EBX
004084AB  |. 83FB 5B         |CMP EBX,5B
004084AE  |.^ 0F85 08FFFFFF  \JNZ cd.004083BC
```

Primecujemo da se ovde EBX poredi sa 5Bh ili sa 91 decimalno, ako je broj veci ili jednak program ce nastaviti sa izvrsavanje i izacice iz ove petlje. Ovo znaci da ce se pretraziti svi uredjaji u sistemu od A: do Z: (90 je ASCII kod za Z) u potrazi za CD-ROMom. Ok sada smo saznali dosta o ovoj zastiti postavimo break-point na 00408409 da saznamo sta se dalje desava posle provere CMP EAX,5. Ako pritisnemo F8 program ce izvrsiti ovu komandu i precice na izvrsavanje sledece. Ono sto ne zelimo da se desi je da se izvrsi skok na kraj petlje pa cemo NOPovati adresu 0040840C. Izvrsavamo kod sa F8 sve dok ne stignemo do adrese 0040842B kada nam sadrzaj EAXa izgleda jako zanimljivo. EAX je sada jednak “A:\main.xxx”. Sta ovo znaci ? Ovo znaci da program ili proverava postojanje fajla “A:\main.xxx” na “CDu” ili

pokusava da otvori taj fajl. Ako fajl nepostoji ili ne moze biti otvoren onda ce se skok na adresi 00408432 izvrsiti i program ce proveriti neki drugi uredjaj. Pogledajmo sa bi se to izvrsilo kada bi fajl postojao :

```
00408434  |. 8D85 1CFEFFFF   |LEA EAX,DWORD PTR SS:[EBP-1E4]
0040843A  |. B9 2C854000     |MOV ECX,cd.0040852C            ; ASCII "\main.xxx"
0040843F  |. 8B55 F8         |MOV EDX,DWORD PTR SS:[EBP-8]
00408442  |. E8 D9B9FFFF     |CALL cd.00403E20
00408447  |. 8B95 1CFEFFFF   |MOV EDX,DWORD PTR SS:[EBP-1E4]
0040844D  |. 8D85 2CFEFFFF   |LEA EAX,DWORD PTR SS:[EBP-1D4]
00408453  |. E8 0CA5FFFF     |CALL cd.00402964
00408458  |. 8D85 2CFEFFFF   |LEA EAX,DWORD PTR SS:[EBP-1D4]
0040845E  |. E8 9DA2FFFF     |CALL cd.00402700
00408463  |. E8 B0A1FFFF     |CALL cd.00402618
00408468  |. 8D55 FC         |LEA EDX,DWORD PTR SS:[EBP-4]
0040846B  |. 8D85 2CFEFFFF   |LEA EAX,DWORD PTR SS:[EBP-1D4]
00408471  |. E8 86A7FFFF     |CALL cd.00402BFC
00408476  |. 8D85 2CFEFFFF   |LEA EAX,DWORD PTR SS:[EBP-1D4]
0040847C  |. E8 E7A7FFFF     |CALL cd.00402C68
00408481  |. E8 92A1FFFF     |CALL cd.00402618
00408486  |. 8B45 FC         |MOV EAX,DWORD PTR SS:[EBP-4]
00408489  |. BA 40854000     |MOV EDX,cd.00408540            ; ASCII "CD_main_xxx"
0040848E  |. E8 85BAFFFF     |CALL cd.00403F18
00408493  |. 75 05           |JNZ SHORT cd.0040849A
00408495  |. BE 01000000     |MOV ESI,1
0040849A  |  8D85 2CFEFFFF   |LEA EAX,DWORD PTR SS:[EBP-1D4]
004084A0  |. E8 7BA5FFFF     |CALL cd.00402A20
004084A5  |. E8 6EA1FFFF     |CALL cd.00402618
```

Ovo i ne izgleda kao nesto posebno, nema nikakvih poredjenja i uslova sem jednog skoka na adresi 00408493 koji kada se ne izvrsi ESI postaje jednak 1. Hmmm ovo bi mozda moglo biti zanimljivo??? Ako pogledamo odmah ispod ove petlje videcemo ovo:

```
004084B4  |. 85F6            TEST ESI,ESI
004084B6  |. 75 18           JNZ SHORT cd.004084D0
```

A ako se ovaj skok izvrsi vodi nas pravo na poruku o ubacenom CDu u CD-ROM i do “startovanja igre”. Ako pogledamo pre same petlje videcemo da je pocetno stanje ESIa jednako 0.

```
004083B5  |. 33F6            XOR ESI,ESI
004083B7  |. BB 41000000     MOV EBX,41
```

Analizom ove petlje dosli smo do zakljucka da ce program ispitati sve uradjaje u kompjuteru od A: do Z: u potrazi za CD koji na sebi ima fajl “main.xxx”, i ako je ovo sve ispunjeno i taj fajl sadrzi nesto sto se u njemu trazi (najverovatnije string “CD_main_xxx”), ako je ovo sve ispunjeno onda ce ESI biti jednak 1 kao znak da se CD sa igricom nalazi u sistemu. Ovaj problem mozemo resiti na razlicite nacine.

1) Mozemo promeniti skok na adresi 004084B6 u JMP tako da se igra uvek startuje (*ovo je ujedno i nalaksi nacin*)
2) Mozemo umesto JNE (jnz) na adresi 004084AE uneti MOV ESI,1 tako da je ESI pre same provere (TEST ESI,ESI) sigurno jednak 1
3) Mozemo izmeniti adresu 004083B2 i uneti MOV ESI,1 umesto postojeceg koda i tako cemo pre same petlje ESIu dodeliti vrednost 1 i izbrisati ono XOR ESI,ESI sto ESIu dodeljuje vrednost 0.
4) Postoji mnogo nacina, vi kao vezbu uz ovo poglavlje smislite neki koji ja nisam ovde nabrojao.

06 Code Hacking...

Prvi deo ovog poglavlja ce vas nauciti kako da uzmete neki deo tudjeg koda i stavite ga u svoj. Ovo ce biti veoma interesantno za Delphi programere koji su se suocili sa problemom reversovanja veoma dugackog algoritma i zeleli bi da ubrzaju proces rekodiranja tudjeg koda. Ova tehnika vam moze dobro doci kod pravljenja keygeneratora ali i kod drugih stvari.

Drugi deo ovog poglavlja ce vas nauciti kako da ubacite svoj kod u tudji program. Kada da naterate bilo koji program da se ponasa kako vi zelite, da ubacite svoje poruke, dialoge u programe.

Delphi and ASM

Kao veoma obican i lak primer sam za ovaj sesti cas pripremio Bangalyev CrackMe #2. On kao i ostali primeri koje ja nisam napisao mogu se preuzeti sa adrese www.crackmes.de Ako “propustimo” ovaj primer kroz PeID videcemo da je on pisan u TASMu / MASMu. Ok otvoricemo Olly da vidimo sta se desava unutar ovog programa. Predpostavicu da znate da pronadjete mesto gde se generise seriski broj pa cemo odmah preci na stvar. Znaci imamo ovaj deo koda koji nam je interesantan i ne zelimo da ponovo pisemo ovu rutinu:

```
00401304  |. B8 01000000     MOV EAX,1
00401309  |> 8B15 38304000  /MOV EDX,DWORD PTR DS:[403038]
0040130F  |. 8A90 37304000  |MOV DL,BYTE PTR DS:[EAX+403037]
00401315  |. 81E2 FF000000  |AND EDX,0FF
0040131B  |. 8BDA           |MOV EBX,EDX
0040131D  |. 0FAFDA         |IMUL EBX,EDX
00401320  |. 03F3           |ADD ESI,EBX
00401322  |. 8BDA           |MOV EBX,EDX
00401324  |. D1FB           |SAR EBX,1
00401326  |. 03F3           |ADD ESI,EBX
00401328  |. 2BF2           |SUB ESI,EDX
0040132A  |. 40             |INC EAX
0040132B  |. 49             |DEC ECX
0040132C  |.^ 75 DB         \JNZ SHORT Key-Crac.00401309
0040132E  |. 56             PUSH ESI
```

Mozemo da postavimo break-point na adresu 00401309 i da prodjemo par puta kroz ovaj loop da vidimo sta se to desava ovde. Videcemo da se u prva tri reda u EDX stavlja prvo,drugo,... slovo iz imena, u ostalim se vrse neke racunske operacije i ovaj loop se ponavlja vise puta, odnosno onoliko puta koliko ime ima slova. Ono sto vidimo je da se svi registi sem ESI resetuju to jest da dobijaju neke nove vrednosti. Ovo ujedno znaci da ce na kraju loopa ESI sadrzati pravi seriski broj. Ako ovo odavde ne mozete da zapazite imate i deo koda malo ispod koji vam upravo ovo govori:

```
0040133A  |. 3BC6           CMP EAX,ESI
0040133C  |. 75 15          JNZ SHORT Key-Crac.00401353
```

Ono sto cemo mi uraditi je da cemo iskoristiti ovaj ASM kod da napravimo Delphi keygenerator, ali to necemo radi prevodjenjem ovog ASM koda u Delphi nego malom modifikacijom ovog ASM koda i njegovim ubacivanjem u Delphi program. Sve od adrese 0040131B pa do adrese 00401328 cemo ostaviti kako jeste, dok cemo ostatak prilagoditi. Znaci ovo cemo malo prilagoditi:

```
MOV EBX,EDX
IMUL EBX,EDX
ADD ESI,EBX
MOV EBX,EDX
SAR EBX,1
ADD ESI,EBX
SUB ESI,EDX
```

Ono sto moramo da uradimo je da pre izvrsavanja ovoga u EDX stavimo vrednost ASCII koda nekog slova iz imena, a na kraju da ESI sacuvamo u neku promenjivu.Znaci kod cemo izmeniti u ovo:

```
asm              {Pocetak ASM bloka}
 XOR ESI,ESI     {Dodali smo ovu ASM komandu}
 MOV ESI,serial{U delphiu i unutar ASM bloka mozemo koristiti sve promenjive }
 MOV EBX,slovo{koje su definisane unutar procedure koju koristimo }
 XOR EBX,EBX
 MOV EBX,EDX
 IMUL EBX,EDX
 ADD ESI,EBX
 MOV EBX,EDX
 SAR EBX,1
 ADD ESI,EBX
 SUB ESI,EDX
 MOV serial,ESI  {ESI registar brise po zavrsetku bloka pa ga moramo sacuvati u promenjivoj}
end;             {Kraj ASM bloka}
```

Ono sto smo ovde dodali je brisanje ESIa i dodeljivanje vrednosti promenjive serial ESIu, a na kraju dodeljivanje vrednosti ESIa promenjivoj serial. Posto ovo treba da se ponavlja za svako slovo iz imena, to ce u Delphiu izgledati ovako:

```
procedure TForm1.Button1Click(Sender: TObject);
 var
 slovo,i,serial:longint;
 ime:string;
begin
 serial := 0;
 ime := Edit1.Text;
for i := 1 to Length(ime) do begin
 slovo := Ord(ime[i]);
asm
  XOR ESI,ESI
  MOV ESI,serial
  MOV EDX,slovo
  XOR EBX,EBX
  MOV EBX,EDX
  IMUL EBX,EDX
  ADD ESI,EBX
  MOV EBX,EDX
  SAR EBX,1
  ADD ESI,EBX
  SUB ESI,EDX
  MOV serial,ESI
 end;
end;
  messagedlg('Vas seriski broj je: ' + IntToStr(serial) + ' !', mtInformation,[mbOK],0);
end;
```

Ovo je jako jednostavan primer ali vam pokazuje kako mozete iskoristite deo tudjeg koda da napravite u ovom slucaju keygenerator. U folderu ...\Casovi\Cas6\keygen-source\ se vec nalazi ovaj gotov primer i source code za njega.

NAPOMENA: Ovo poglavlje je zamisljeno da bude od velike koristi Delphi programerima koji se bave reversnim inzinjeringom i zeleli bi da reversuju neke algoritme bez njihovog rekodiranja u Delphi sintaksu, ostalima ovaj deo poglavlja nece biti od velike vaznosti pa ga mozete preskociti.

Adding functions #1

Ovo poglavlje ce vas nauciti kako da dodate najobicniji NAG ekran u neki program i da se program posle toga moze startovati :) Pogledajte primer CRC32.exe koji se nalazi u folderu Cas06. Primeticete da je to jedan obican CRC32 calculator i da ako ga skeniramo sa PeIDom videcemo da je pisan u Visual C++. Ova informacija je jako bitna jer se ova tehnika dodavanja funkcija moze primeniti samo na Delphi, C++ i ASM programe.

Ono sto cemo mi u ovom poglavlju uraditi je dodavanje obicnog NAG ekrana pre samog pocetka programa. Ovaj NAG ce biti obican Message Box koji ce govoriti da smo mi dodali funkciju u program. Otvoricemo program pomocu Ollya i zapamticemo kako izgleda OEP. Trebace nam samo prvih par linija koda pa cemo ih iskopitari u Notepad. Trebaju nam ove linije:

```
00401580 > $  55              PUSH EBP
00401581  .  8BEC             MOV EBP,ESP
00401583  .  6A FF            PUSH -1
00401585  .  68 00404000      PUSH CRC32.00404000
0040158A  .  68 88264000      PUSH CRC32.00402688
```

Zasto smo uzeli samo ove linije??? Zato sto cemo zameniti ovaj OEP sa nekom nasom komandom ali posle izvrsavanja naseg koda ovaj OEP moramo i da vratimo nazad pa smo ga backupovali.

Prvo treba da nadjemo mesto gde cemo ubaciti nas NAG ekran. To mesto uvek trazimo tamo gde ima puno praznih, neiskoriscenih 00 ili 90 bajtova. Posto ce nam za ubacivanje NAGa i vracanje OEPa trebati oko 50 bajtova onda cemo mesto za ubacivanje koda traziti na kraju fajla gde po pravilu ima mnogo slobodnih 00 bajtova. Kao sto vidimo od adrese 00403356 pa do 00403FFF ima vise nego dovoljno mesta za ovo:

```
00403356    00                DB 00
...
00403FFF    00                DB 00
```

Tacnije ovde ima 3241 bajta slobodnog prostora. Pre ubacivanja samog MessageBoxA moramo da znamo da li se u fajlu nalazi API poziv ka MessageBoxA apiju i ako se nalazi koji su mu parametri. Mozete slobodno da odete u module i proverite da se API MessageBoxA stvarno nalazi u fajlu i da se API moze koristiti. Sama API funkcija ima sledece parametre:

```
MessageBoxA(hwnd, Text, Naslov, MB_TIPMESSAGEBOXA);
```

gde su:

- hwd; vlasnik messageboxa, moze biti nula
- Text; Tekst koji ce pisati unutar messageboxa
- Naslova; Naslov prozora messageboxa
- Tip; Da li je messagebox informacija ili upozorenje ili....

A ovo bi u ASMu izgledalo ovako:

```
PUSH Tip
PUSH Naslov
PUSH Text
PUSH 0
CALL MessageBoxA
```

Pre nego sto unesemo ovaj CALL negde u slobodnom prostoru prvo moramo da unesemo sve tekstove koji se pojavljuju u messageboxu. Ovo radimo zato sto se u PUSH komandama za naslov i text ne nalaze ASCII vrednosti nego

adrese do ASCII vrednosti koje se koriste. Selektovacemo adresu 00403358 i pritisnucemo desno dugme -> Binary -> Edit... Ako vam je otkaceno Keep Size iskljucite ga. Pa cemo uneti sledeci tekst bez navodnika “Cracked:”. Kada pritisnemo OK program ce pokazati ovo:

```
00403358    43              INC EBX
00403359    72 61           JB SHORT CRC32.004033BC
0040335B    636B 65         ARPL WORD PTR DS:[EBX+65],BP
0040335E    64:3A00         CMP AL,BYTE PTR FS:[EAX]
```

Posto ovo nije pravo stanje stvari pritisnucemo CTRL+A da bi smo analizirali fajl i da bi smo videli pravo stanje stvari. Ono sto se pojavilo je ovo:

```
00403358  .  43 72 61 63 6>ASCII "Cracked:",0
```

I to je ono sto treba ovde da se nalazi. Predpostavljam da vas interesuje sta znaci ova nula iza Stringa Cracked:. Ona predstavlja kraj stringa, to jest prvo se ispisuju sve ASCII vrednosti slova iz stringa pa se onda taj string zatvara sa 0x00 bajtom. Ovo C++ koderi sigurno znaju. Dalje cemo selektovati adresu 00403361 i na isti nacin cemo uneti string “The art of cracking was here!!!”, takodje bez navodnika. Sada imamo ovo:

```
00403358  .  43 72 61 63 6>ASCII "Cracked:",0
00403361     00         DB 00 <- Ostavio sam nulu pre unosenja drugog stringa, vi nemorate
00403362  .  54 68  65 20 6>ASCII "The art of Crack"
00403372  .  69 6E 67 20 7>ASCII "ing was here!!!",0
```

Odmacemo se malo od ovog teksta i unecemo NAG pocevsi od adrese 00403384. Selektujte tu adresu pritisnite Space da bi ste presli u Assembly mod i unesite sledece:

```
PUSH 40 <- Zato sto je 40 Hex vrednost za MB_ICONINFORMATION, a moze da se unese i PUSH
0, sasvim je svejedno, rec je samo o ikoni dialoga.
```

Pritisnite Assemble i ne gasenjem Assembly prozora samo unesite novi red:

```
PUSH 00403358  <- jer se na adresi 00403358 nalazi naslov messageboxa
```

Pritisnite Assemble i ne gasenjem Assembly prozora samo unesite novi red:

```
PUSH 00403362  <- jer se na adresi 00403362 nalazi tekst messageboxa
```

Pritisnite Assemble i ne gasenjem Assembly prozora samo unesite novi red:

```
PUSH 0  <- jer je hwnd jednak 0
```

Pritisnite Assemble pa Cancel jer nam treba malo pomoci pri unosenju sledeced reda. Znate da sledeca komanda koja sledi je CALL MessageBoxA ali nam za njeno pozivanje treba tacna adresa APIa MessageBoxA pa cemo skociti do prozura Modules (ALT+E) i otici na desno dugme na fajl CRC32.exe i selektujte View Names. U prozoru koji se sada pojavio potrazite MessageBoxA. Videcete da se u tom redu nalazi i adresa na kojoj se nalazi sam API. Dakle vidimo ovo:

```
Names in CRC32, item 34
 Address=004081B4
 Section=.idata
 Type=Import  (Known)
 Name=USER32.MessageBoxA
```

A adresa APIa je 004081B4. Posto znamo podatak koji nam je potreban vraticemo se na adresu 00403390 i unecemo sledece:

```
CALL DWORD[004081B4] <- Pozovi adresu na kojoj se nalazi API
```

I sada smo uspesno ubacili NAG u program. Sada preostaje samo da prepravimo OEP tako da se umesto prve linije koda izvrsava ovaj red koji prikazuje NAG screen. Ovo se moze uraditi na dva nacina.

NACIN 1:

Prvi nacin je izuzetno lak i ne zahteva nikakve dodatne alate sem Ollya. Posto smo backupovali OEP u Notepad znamo tacno kako on izgleda i mozemo da izmeniti i posle vratiti. Dakle u Notepadu imamo ovo:

```
00401580 > $  55            PUSH EBP
00401581  .  8BEC           MOV EBP,ESP
00401583  .  6A FF          PUSH -1
00401585  .  68 00404000    PUSH CRC32.00404000
0040158A  .  68 88264000    PUSH CRC32.00402688
```

Oticemo na adresu 00401580 i izmeniti je tako da ona vodi direktno do adrese 00403384 gde pocinje prikazivanje NAGa. Izmenicemo adresu 00401580 u jedan obican skok:

```
00401580 >   /E9 FF1D0000   JMP CRC32.00403384
00401585  .|68 00404000     PUSH CRC32.00404000
0040158A  .|68 88264000     PUSH CRC32.00402688
```

Dakle uneli smo jmp 00403384 ali smo na racun toga izgubili par redova OEPa koje moramo negde nadoknaditi. Izgubljeni redovi su:

```
00401580 > $  55            PUSH EBP
00401581  .  8BEC           MOV EBP,ESP
00401583  .  6A FF          PUSH -1
```

Pa cemo njih uneti odmah ispod NAGa koji pocinje na adresi 00403384. Oticemo dole i na adresi 00403398 cemo uneti delove koji nedostaju.

```
00403398  .  55             PUSH EBP
00403399  .  8BEC           MOV EBP,ESP
0040339B  .  6A FF          PUSH -1
```

i dodacemo jedan skok koji ce vratiti program na prvu neizmenjenu adresu od gore, na 00401585.

```
0040339D  .^\E9 E3E1FFFF    JMP CRC32.00401585
```

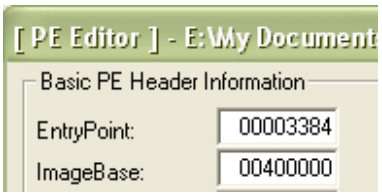
I sa time smo zavrsili sa dodavanjem obicnog NAG ekrana u program. Sada ostaje samo da snimimo promene i pokrenemo snimljeni fajl. Fajl crc32_added.exe pokazuje kako dodani NAG treba da izgleda.

NACIN 2:

Prvi nacin je izuzetno lak ali drugi nacin moze biti jos laksi ali zahteva upotrebu dva alata: Ollya i LordPe-a. Kada smo uneli NAG treba odmah ispod NAGa da dodamo skok koji ce nas vratiti na OEP posto se posle izvrsavanja NAGa program vraca na pocetak. Dodajte red:

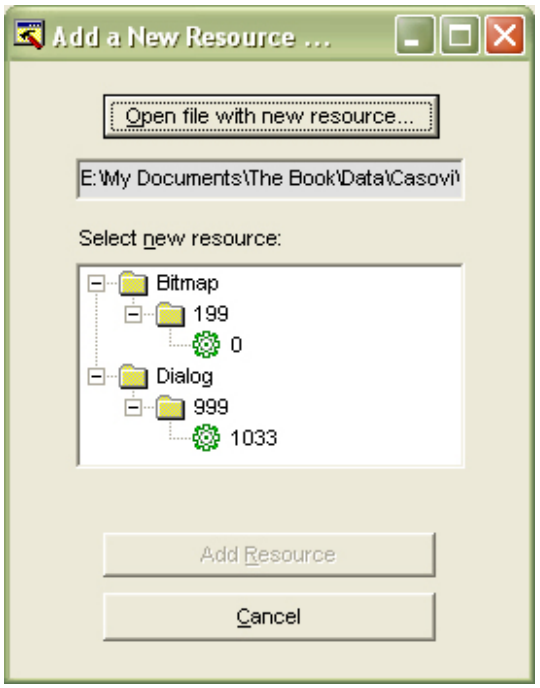
```
00403398  .^\E9 E3E1FFFF    JMP CRC32.00401580
```

Ovo radimo zato sto cemo promeniti OEP programa sa 00401580 na 00403384 tako da ce se prvo izvrsiti NAG pa tek onda sve ostalo. Kada smo dodali i ovaj red snimimo promene. Da bi smo promenili OEP fajla otvoricemo novo-snimljeni fajl pomocu LordPe-a. Kliknite na PE-Editor dugme i izaberite novo-snimljeni fajl i u njemu cemo izmeniti OEP u novu adresu 00403384 – ImageBase 00400000 = 00003384. Ovo treba da izgleda kao na slici. Posle promena mozemo da pritisnemo Save, da zatvorimo LordPE i da startujemo fajl sa izmenjenim OEPom. Videcemo da smo uspeli i da program radi bez ikakvih problema.



Adding functions #2

Prvi deo poglavlja je objasnio dodavanje MessageBoxa u program. U drugom delu ovog poglavlja cu vam pokazati kako da dodate Dialog u neki program i kako da ga nacinite 100% funkcionalnim. Zvuci li zanimljivo??? Pa da pocnemo. Otvoricemo metu original.exe a nalazi se u folderu Cas6, pomocu ResHackera. Zasto??? Zato sto moramo da dodamo nove resurse u sam exe fajl a to cemo najlakse uraditi pomocu ResHackera. Sada vidite zasto sam rekao da ResHacker ima jos mnogo korisnih funkcija. Kao sto vidimo na slici pored izabracemo iz glavnog menija Action -> Add new resource i kada se pojavi dialog sa slike izabracemo fajl addonDialog.res i dodacemo jedan po jedan oba resursa koja se nalaze u tom fajlu. Ne ja ovde nemam nameru da objasnim kako se prave .res fajlovi. Za ovo cete sami morati da pregledate Microsoftovu dokumentaciju o Resource Compileru. Posle ovog dodavanja moramo da mapiramo, to jest da zapisemo sve IDove koji se nalaze u novom dodatom Dialogu. ID samog dialoga je 999 (3E7h), dugme ? ima ID 3007 (BBFh), dugme Visit Web page ID 3005 (BBDh), a dugme Exit ID 3006 (BBEh). Ovo nam je potrebno kako bi smo povezali sve ove dugmice sa funkcijama koje ce raditi svoj posao. Sada mozemo da snimimo promene u ovom fajlu, snimicemo ovaj fajl kao AddOn.exe. Mozemo da zavrsimo sa ResHackerom i da otvorimo novo-snimljeni fajl sa Ollyjem. Pre nego sto pocnemo sa dodavanjem funkcija u fajl prvo moramo da odlucimo gde cemo ih dodati i kako. Posto moramo da prikazemo novi dialog na ekranu bekapovacemo sledece linije ASM koda u recimo Notepad, jer ce nam kasnije trebati:




```
00408722  . 6A 00                PUSH 0                          ; /lParam = NULL
00408724  . 68 C8854000          PUSH AddOn.004085C8             ; |DlgProc =
00408729  . 6A 00                PUSH 0                          ; |hOwner = NULL
0040872B  . 6A 64                PUSH 64                         ; |pTemplate = 64
0040872D  . FF35 4CA84000        PUSH DWORD PTR DS:[40A84C]      ; |hInst = 0
00408733  . E8 C4C4FFFF          CALL DialogBoxParamA            ; \DialogBoxParamA
```

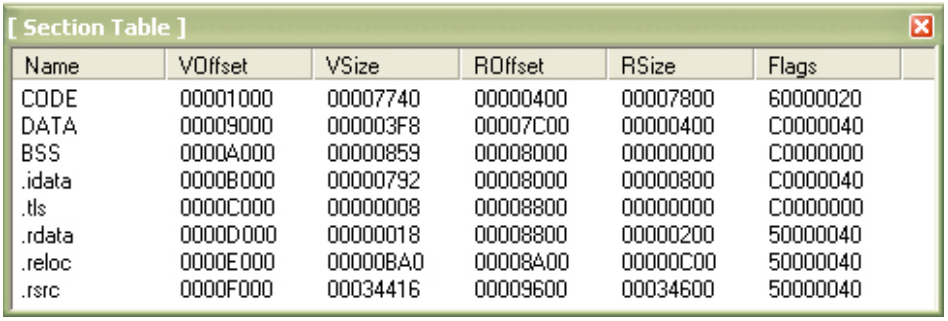
dalje treba da backupujemo poruku koja ce se pojaviti kada pritisnemo dugme ?. Dakle treba da backupujemo ovo:

```
00408611  |. 6A 40                PUSH 40                         ; /Style = MB_OK
00408613  |. A1 90924000          MOV EAX,DWORD PTR DS:[409290]   ; |
00408618  |. 50                   PUSH EAX                        ; |Title => "About..."
00408619  |. A1 94924000          MOV EAX,DWORD PTR DS:[409294]   ; |
0040861E  |. 50                   PUSH EAX                        ; |Text => "U “
0040861F  |. 53                   PUSH EBX                        ; |hOwner
00408620  |. E8 07C6FFFF          CALL MessageBoxA                ; \MessageBoxA
```

Ovo radimo jer cemo cemo umesto pritiska na dugme ? uraditi da se pojavi nas novi ubaceni dialog. Kada se pojavi novi dijalog onda cemo ubaciti ovu poruku. Dakle plan izgleda ovako:

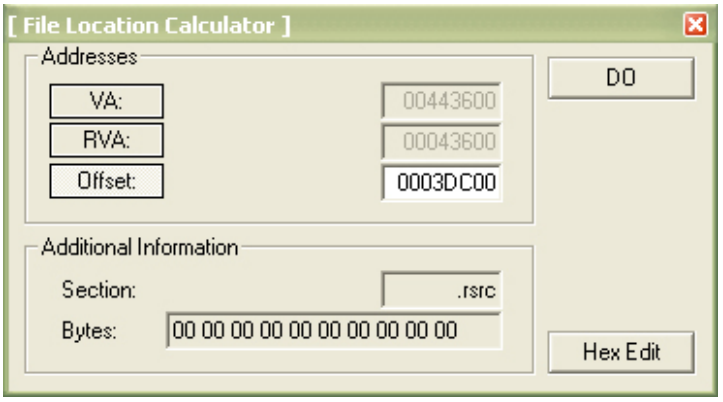
1) Dodati dialog pomocu ResHackera
2) Osloboditi mesto za kod koji cemo ubaciti
3) Napisati kod koji ce se kaciti na message loop za obradu poruka
4) Napisati kod za svako dugme novog dialoga
5) Dodati kod za vracanje iz dodatog koda u message loop

Ufff... ovo je mnogo posla a imamo i jedan veliki problem. Moramo prvo da oslobodimo dovoljno mesta da bi smo mogli da ubacimo novi kod koji ce uraditi sve sto smo zamislili. Za ovo ce nam trebati jedan dobar Hex Editor i LordPe, ja sam koristio staru verziju Hex WorkShop jer mi je jedina bila pri ruci, ali vi mozete koristiti bilo koji Hex Editor. Prvo cemo otvoriti AddOn.exe sa LordPe-om i nacicemo sve sekcije.



Name	VOffset	VSize	ROffset	RSize	Flags
CODE	00001000	00007740	00000400	00007800	60000020
DATA	00009000	000003F8	00007C00	00000400	C0000040
BSS	0000A000	00000859	00008000	00000000	C0000000
.idata	0000B000	00000792	00008000	00000800	C0000040
.tls	0000C000	00000008	00008800	00000000	C0000000
.rdata	0000D000	00000018	00008800	00000200	50000040
.reloc	0000E000	00000BA0	00008A00	00000C00	50000040
.rsrc	0000F000	00034416	00009600	00034600	50000040

Uobicajno je da prosirimo poslednju sekciju za odredjeni broj bajtova. Dakle otvoricemo isti ovaj fajl pomocu Hex Editora, oticemo na poslednji bajt i dodacemo jos 1000 0x00 bajtova. Pocetni bajt ce biti 3DC00 a poslednji 3DEE0. Kada ovo uradimo vraticemo se u LordPe i izmenicemo fizicku i virtualnu velicinu .rsrc sekcije sa 00034416 na 000348B0 jer smo dodali 1000 bajtova. Moracemo jos i da promenimo karakteristike (*flags*) same sekcije tako da bi smo mogli i da ubacimo kod u nju na F00000E0. Sada moramo da sracunamo gde se to nalazi virtualna adresa 3DC00. Za ovo cemo iskoristiti FLC opciju iz LordPe-a.



Ova RVA adresu je 00443600 a ovo predstavlja prvu RVA adresu od 1000 novih dodatih bajtova. Ovde cemo dodavati nas kod. Sada pocinje prava zabava :)
Prvo cemo dodati sve ASCII vrednosti koje su nam potrebne. Selektovacemo adresu 00443600 i pritisnucemo desno dugme i izabracemo Binary -> Edit... u novo otvorenom prozoru cemo u ASCII polju uneti http://www.google.com a primeticete da Keep size checkbox nesme da bude otkacen.
Drugi string cemo ubaciti na adresi 00443616, tako da cemo selektovati 00443615 i izmenicemo drugi bajt i tri iza njega u string “open” - bez navodnika. Zasto ovako razmacinjemo dva stringa??? Zato sto se svaki string mora zavrsavati sa po jednim 0x00 bajtom.
Dalje cemo presresti loop koji je zaduzen za prevodjenje poruka i njihovu interpretaciju. Ovaj loop se najlakse nalazi tako sto cemo naci mesto gde se nalaze WM_nesto komande. Taj CALL pocinje ovde 004085C8, a vec je definisan kada se pojavio default dialog na ekranu. Pogledajte ovu adresu:

```
00408724  .  68 C8854000     PUSH original.004085C8          ; |DlgProc = original.004085C8
```

NOPovacemo sve od 00408611 pa do 00408624 jer nam taj dao koda ne treba. Na mesto prikazivanja poruke kada se pritisne dugme ? cemo ubaciti pojavljivanje novog dialoga i obradu poruka. Selektovacemo adresu 00408611 i na to mesto cemo ubaciti sledeci kod:

```
JMP 00443620
```

a ovo ce nas odvesti do novog koda koji cemo dodati na toj adresi.Na adresi 00443620 dodajte kod:

```
CMP ESI,0BBF
JNZ 00443643
PUSH 40                                 {
MOV EAX,DWORD PTR DS:[409290]
PUSH EAX
MOV EAX,DWORD PTR DS:[409294]                   Kod za prikazivanje MsgBoxA
PUSH EAX
PUSH EBX
CALL 00404C2C                           }
JMP 00408625
NOP
NOP
```

Sta se ovde desava ??? Prvo se ESI (sadrzace ID objekta koji je pritisnut) uporedjuje sa 0BBF, to jest proveravamo da li je pritisnuto drugo (*novo*) dugme ?, a ako jeste onda cemo prikazati backupovani messagebox sa starog *(00408611)* ? Posle ovoga se proverava ako ovo nije tacno program ce skociti na neku dalju adresu, a ako je ESI 0BBF onda ce se prikazati MessageBox na ekranu. Ako se pokaze MessageBox na ekranu onda cemo se posle prikazivanja MessageBoxa vratiti u loop za proveru poruka pomocu JMPa. Dalje cemo selektovati adresu 00443643 i unecemo sledece:

```
CMP ESI,0BBE
JE 00408374
```

Ovde poredimo ESI sa dugmetom za gasenje programa, EXIT. Ako je pritisnuto ovo dugme program ce otici na mesto iznad message loopa ko je je zaduzeno za gasenje programa. Ovo smo ranije odredili da se nalazi na adresi 00408374. Dalje cemo selektovati adresu 00443655 i unecemo:

```
CMP ESI,0BBD
JNZ 00443673
PUSH 1                                  {
PUSH 0
PUSH 0
```

```
PUSH 443600                              Kod za otvaravnje web-strane
PUSH 443616
PUSH 1
CALL 00408388                            }
JMP 00408625
```

Prvo cemo uporediti ESI sa 0BBD, to jest sa dugmetom Visit Web site. Ako je pritisnuto ovo dugme onda cemo ga posetiti pomocu ShellExecuteA funkcije. Napominjem da adresu na kojoj se nalazi ova funkcija u fajlu morate naci u View Names opciji u modulima. I poslednje, ali i najvaznije je da se postaramo da se novi dialog i pojavi na ekranu. Selektovacemo adresu 00443673 i unecemo sledece:

```
CMP ESI,0BBA
JNZ 00408625
PUSH 0                                   {
PUSH 4085C8
PUSH 0
PUSH 3E7                                 Kod za prikazivanje dialoga
PUSH DWORD PTR DS:[40A84C]
CALL 00404BFC                            }
JMP 00408625
```

Ovde smo prvo uporedili ESI sa 0BBA, odnosno sa starim dugmetom ?, ovo nas je podsetilo da ovakav CMP nismo NOPovali gore pa cemo se posle vratiti na to. Ako je pritisnuto staro dugme ? onda cemo prikazati dialog na ekran. Ovo je samo copy / paste backupovanog koda sa 00408722 koji sluzi za prikazivanje prvog dialoga na ekran. Posle ovoga se jednostavno vracamo u gornji loop za proveru WM poruka sa obicnom JMP komandom. Sada cemo snimiti nase promene, selektovacemo kod od 00443600 do 0044369D (*jer smo taj deo koda dodali*) i pritisnucemo desno dugme -> Copy to executable -> Section i snimicemo ovaj fajl kao Addon2.exe, pa cemo otovriti ovaj novi fajl pomocu Ollya.

Sada cemo se vratiti da povezemo stari loop sa novim loopom poruka. NOPovacemo sve od 00408611 pa do 00408624 jer nam taj dao koda ne treba. Na njegovo mesto cemo ubaciti pojavljivanje novog dialoga i obradu poruka. Selektovacemo adresu 00408611 i na to mesto cemo ubaciti sledeci kod:

```
JMP 00443620
```

Posle ovoga nam ostaje samo da ubijemo staru proveru za dugme ?, BBA. Zato cemo NOPovati adrese:

```
00408609  |> \81FE BA0B0000        CMP ESI,0BBA
0040860F  |. 75 14                 JNZ SHORT AddOn2.00408625
```

Sada kada smo sve zavrsili mozemo da snimimo promene pritiskom na desno duge -> Copy to executable -> All modifications -> Copy All -> desno -> Save file... i mozemo probati fajl. Ako ste nesto propustili ili niste dobro uradili program ce vam se srusiti, dobro uradjen primer imate u istom folderu pod imenom AddOn3.exe, pa mozete videti kako to treba da izgleda.

NAPOMENA: Ovo je izuzetno teska tehnika i zahteva zaista dosta prakse. Zbog ovoga ne ocajavajte ako ne racumete sve sada ili nemozete da naterate program da se startuje. Posle kad sakupite dosta reverserskog iskustva vratite se na ovo poglavlje.

Adding functions #3

Prvi deo poglavlja je objasnio dodavanje MessageBoxa u program, drugi delo ovog poglavlja sam vam pokazao kako da dodate Dialog u neki program, a u trecem cu vam pokazati kako da dodate API koji vam je potreban za reversing u IAT tabelu bilo kog programa.

Ovaj deo posla, ubacivanje u IAT, je do skora bio jako tezak i zahtevao jer ili rucnu modifikaciju PE Sekcija, dodavanje nove sekcije, pravljenje validnog IAT headera, ili ubacivanje API poziva preko LoadLiberaryA funkcije i trazenje APIa po njegovom ordinalnom broju. Iako su oba nacina veoma korisna ja cu vam ovde objasniti kako se ovo veoma lako radi pomocu samo jednog alata: IIDKinga.

Ovaj alat se koristi izuzetno jednostavno:

1) Selektujte aplikaciju kojoj dodajete API
2) Ako zelite da ubacite dodatan kod u .exe unesite dodatnu velicinu sekcije (u ovom slucaju 1000)
3) Unesite ime .dll fajla
4) Unesite ime API poziva
5) Pritisnute + da dodate API i tog .dll-a
6) Pritisnite Add them da bi ste snimili fajl

Imajte na umu da mozete dodavati API pozive iz samo jednog .dll fajla. Ako zelite da dodate APIje iz drugog .dll-a morate prvo da snimite promene i da ponovo otvorite isti fajl da bi ste dodali jos jednu sekciju sa API pozivima u njega. Posle pritiska na dugme Add them !! IIDKing ce napraviti .txt fajla sa adresama do svih dodatih API poziva. IIDKing program mozete naci u folderu Cas6.

07 Getting caught

Ovo poglavlje sadrzi osnove detekcije debuggera i osnovne trikove detekcije modifikacije ili pokusaja modifikacije memorije programa kao i programa samog. Ove tehnike ce biti opisane detaljno tako da ce koristi od ove oblasti imati i software developeri i reverseri. Prvi da unaprede zastitu na svom software-u a drugi da saznaju kako je jednostavno ukloniti primitivne zastite.

SoftICE detection

Evo jednog primera za Delphi programere. Ovaj deo koda je zasebna funkcija koja se koristi za detekciju aktivnog SoftICEa. Ovo je standardna detekcija SoftICEa i kao ovakva se nalazi u 90% programa koji koriste takozvanu Anti-SoftICE funkciju.

```
function SoftIce95: boolean;
var hfile: Thandle;
begin
  result:=false;
  hFile:=CreateFileA('\\.\SICE', GENERIC_READ or GENERIC_WRITE, FILE_SHARE_READ or
FILE_SHARE_WRITE, nil, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, 0);
  if(hfile<>INVALID_HANDLE_VALUE) then
  begin
    CloseHandle(hfile);
    result:=true;
  end;
end;
```

Prisustvo debuggera mozete proveriti uz pomoc malog Delphi programa koji se nalazi ovde ...\Casovi\Cas7\Debugger check.exe. Posto znamo kako se zastita implementira u neki program stoga znamo kako i da zaobidjemo ovu proveru. Zastita koju u ovom primeru hocemo da zaobidjemo je provera aktivnog SoftICEa i SoftICEa na NTu. Neznatna je razlika izmedju ove dve provere i one uvek u programima idu zajedno. Jedina razlika izmedju NT i ostalih windowsa je u tome sto je prvi parametar za CreateFileA API \\.\NTICE umesto \\.\SICE. Posto su ovde u pitanju dva string parametra koja se prosledjuju API funkciji stoga se one sigurno u fajlu nalaze kao stringovi i mogu se naci uz pomoc String Reference prozora u W32Dsamu ili pomocu odgovarajuce komande u Olly-u. Dakle provericemo ovu teoriju u praksi. Otvorite ovu “metu” u Olly-u, pronadjite pomenute stringove i postavite break-pointove na njih. Sa F9 startujte program i pritisnite SoftICE NT dugme. Ono sto vidite je sledeca

```
004523C4  /$  53              PUSH EBX
004523C5  |.  33DB            XOR EBX,EBX
004523C7  |.  6A 00           PUSH 0                      ; /hTemplateFile = NULL
004523C9  |.  68 80000000     PUSH 80                   ; |Attributes = NORMAL
004523CE  |.  6A 03           PUSH 3                      ; |Mode = OPEN_EXISTING
004523D0  |.  6A 00           PUSH 0                      ; |pSecurity = NULL
004523D2  |.  6A 03           PUSH 3                      ; |ShareMode = ...
004523D4  |.  68 000000C0     PUSH C0000000             ; |Access = ...
004523D9  |.  68 F4234500     PUSH Debugger.004523F4 ; |FileName = "\\.\NTSICE"
004523DE  |.  E8 4543FBFF     CALL CreateFileA           ; \CreateFileA
004523E3  |.  83F8 FF         CMP EAX,-1
004523E6  |.  74 08           JE SHORT Debugger.004523F0
004523E8  |.  50              PUSH EAX                    ; /hObject
004523E9  |.  E8 1A43FBFF     CALL CloseHandle            ; \CloseHandle
004523EE  |.  B3 01           MOV BL,1
004523F0  |>  8BC3            MOV EAX,EBX
004523F2  |.  5B              POP EBX
004523F3  \.  C3              RET
```

funkcija koja proverava da li je SoiftICE aktivan. Ono sto vidimo je jednostavna funkcija koja vraca 1 (true) ako je SICE detektovan ili 0 (false)

ako SICE nije detektovan. Reversing ove funkcije mozete raditi ili na ulazu u nju ili na izlazu iz nje, a mozete je raditi i na sredini. Bez nekog detaljisanja (*dobar deo ovoga je vec vise puta objasnjen u knjizi*) precicu na moguca resenja ovog problema.

1) Ako zelimo mozemo da promenimo sam pocetak:

```
004523C4  /$ 53            PUSH EBX
004523C5  |. 33DB          XOR EBX,EBX
004523C7  |. 5B            POP EBX
004523C8  |. C3            RET
```

2) Mozemo da izmenimo i samo jedan skok:

```
0045232F  |. 83F8 FF       CMP EAX,-1
00452332  |. EB 08         JMP SHORT Debugger.0045233C
00452334  |. 50            PUSH EAX                              ; /hObject
00452335  |. E8 CE43FBFF   CALL<JMP.&kernel32.CloseHandle>       ; \CloseHandle
0045233A  |. B3 01         MOV BL,1
```

3) Ili mozemo da resetujemo EBX na samom kraju:

```
0045233C    33DB           XOR EBX,EBX
0045233E  |. 5B            POP EBX
0045233F  \. C3            RET
```

Resenja je mnogo, mozete i sami smisliti neko. Moze se desiti da je string koji trazimo (\\.\SICE) maskiran ili enkriptovan u fajlu i da ne mozemo da ga nadjemo. U tom slucaju mozemo ovo mesto naci pomocu API funkcije CreateFileA koja sluzi za pravljenje “fajla” \\.\SICE. API funkcije mozemo naci tako sto cemo otvoriti Executables Module window (ALT + E) i tu u glavnom exe fajlu pronaci nasu API funkciju pod opcijom View Names (CTRL + N). Ono na sta morate da pazite kada stavljate kondicionalni, importni ili obican break-point na API funkciju je da ce program uvek zastati u dll fajlu u kome se nalazi ta API funkcija (kernel32.dll, user32.dll itd.). Ono sto moramo da uradimo da bi smo se vratili u exe koji poziva tu API funkciju je da izvrsimo kod do prve sledece RET funkcije, posle cega cemo se vratiti u ExE kod odakle je pozvana ta funkcija.

Iako je ovo standardan nacin detekcije SICEa ali postoje i druge metode koje se zasnivaju na detekciji aktivnih procesa i otvorenih prozora. Za ovo se koriste druge API funkcije. Neke od njih mogu biti FindWindowA, FindWindowExA, Process32First, Process32Next, MessageBoxA (*prikaz poruke o aktivnom debuggeru*), itd. Ove zastite zaobilazite zavisno od slucaja do slucaja ali najcesce se resavaju pomocu poruke koju o aktivnom debuggeru koju programeri najcesce ostavljaju u programu. Ovo znaci da nam skoro u 90% preostalih slucajeva programeri preko obicnog message boxa prikazuju poruku da program nece biti startovan jer je neki od debuggera aktivan.

NAPOMENA: Primeticete sam u prvom primeru reversinga SICE funcije izmenio 3 i 4 liniju u POP EBX i RET. Ovo RET je logicno jer ono sto se desava je jednostavno vracanje iz CALLa a ono sto moramo uraditi pre vracanje je da vratimo ulazne paramerte funkcije pomocu komande POP (*ulazni parametri se ne moraju uvek nalaziti pre CALLa nego se mogu naci i unutar samog CALLa na njegovom pocetku*). Posto je jedini ulazni parametar EBX vracamo samo njega. Nema preterane potrebe da ovo pamtite, ovo se moze jednostavno prepisati sa kraja CALLa ali je bitno da imate na umu prilikom menjanja koda.

Windows check debugger API

Evo jos jednog primera za Delphi programere. Ovaj deo koda je zasebna funkcija koja se koristi za detekciju bilo kog aktivnog debuggera. Ova funkcija ce detektovati da li je program startovan uz pomoc nekog debuggera ili ne. Ovo je bar koliko je meni poznato nedokumentovana API funkcija. Slucajno sam naisao na nju prilikom reversovanja programa koji je bio pakovan i zasticen AsProtectom. Posle dosta razmatranja dosao sam do zakljucka da AsProtect koristi windowsovu API funkciju IsDebuggerPresent koja se nalazi u kernel32.dll-u koja nema ulaznih parametara a vraca kao rezultat 1 ili 0 u zavisnosti da li je debugger detektovan. Kada se sve ovo sroci u jednu Delphi funkciju to izgleda upravo ovako:

```
function IsDebuggerPresent: Integer; stdcall; external 'kernel32.dll' name
'IsDebuggerPresent';
```

Kao sto je za SICE objasnjeno reversing ove funkcije se radi veoma jednostavno. Ako otvorite isti primer koji samo koristili u proslom delu poglavlja. Ako postavimo break-point na IsDebuggerPresent pokrenemo program i pritisnemo dugme Test Debugger API zavrsicemo u kernel32.dll-u a posle izvrsenja RET komande nacicemo se ovde:

```
0045229A    8BC0            MOV EAX,EAX
0045229C  . E8 F3FFFFFF     CALL IsDebuggerPresent  <- Ovde smo
004522A1  . 48              DEC EAX
004522A2  . 75 16           JNZ SHORT Debugger.004522BA
```

Ako pogledamo sadrzaj EAXa videcemo da on sadrzi broj 1 jer smo program pokrenuli uz pomoc debuggera (Ollya). Ovo mozemo promeniti na veoma lak nacin tako da program uvek “misli” da nije startovan pomocu debugger.

```
0045229A  .  8BC0           MOV EAX,EAX
0045229C  .  B8 00000000    MOV EAX,0
004522A1  . 48              DEC EAX
004522A2  . 75 16           JNZ SHORT Debugger.004522BA
```

Sta se ovde desava ? Umesto poziva funkciji IsDebuggerPresent mi cemo samo postaviti da je EAX uvek jednak nula (MOV EAX,0 ili XOR EAX,EAX isto je). Zasto nula ? Zato sto imamo kondicioni skok JNZ na adresi 004522A2 koji ce se izvrsiti samo ako je EAX jednak FFFFFFFF (-1) a pre toga imamo jednu DEC EAX komandu (EAX = EAX – 1) pa stoga pre ove dve komande EAX uvek mora biti jednak nula.

Ovo je jedna zaista obicna i doista jednostavna zastita koja se lako implementira a jos lakse zaobilazi. Ali i bez obzira na to moze se sresti i u komercijalnim zastitama (*AsProtect*) pa je korisno znati i ovaj API.

Memory modification check

Ovo je jedna retko (*skoro nikada*) koriscena tehnika. Ova tehnika je jako korisna kao zastita od raznih vrsta loadera, memory patchera ili bilo kakve manipulacije memorijom. Ja licno nisam sreo ovu tehniku u "metama" ali nije na odmet znati kako dodatno zastititi vasu aplikaciju. Posebno napisana aplikacija koja se nalazi ovde ...\Casovi\Cas7\Memory Checker.exe. Malo uputstvo pre nego sto pocnemo sa reversingom ove aplikacije. U samom programu cete videti 00 koje ce se posle par sekundi promeniti u broj 1727. Ovaj broj predstavlja checksum vrednost zasticenjog dela koda. Svakih par sekundi odredjeni deo koda (*memorije*) se proverava i ako mu je checksum razlicit od 1727 onda ce se umesto ove poruke pojaviti poruka o modifikaciji memorije. Ova poruka ce se pojaviti samo ako modifikacija memorije ne sadrzi ni jednu NOP komandu. Ako je deo memorije NOPovan program ce se sam zatvoriti. Posmatrani deo memorije je segment od 20 bajtova koji sluzi za prikazivanje NAG poruke svakih 20 sekundi. Ovaj deo koda odnosno memorije moramo da promenimo tako da se ova NAG poruka ne pojavljuje a da program idalje misli da je sve uredu sa sadrzajem memorije. Ovaj zadatak iako izgleda veoma komplikovan nije toliko tezak.

Uredu, pa da pocnemo. Otvorite ovu metu pomocu Ollya. Ono sto mozete pokusate je da nadjete gde se to pojavljuje poruka o menjanju memorije. Da bi smo saznali kako ova poruka izgleda moramo da prvo ubijemo onu NAG poruku. Nadjimo je i postavimo jednostavnu RET komandu na adrese 004507F4 i 004507F5 (*zapamtite, ne sme biti ni jedna NOP komanda*).

```
004507F4    6A 40           PUSH 40 <- Izmeniti u RET
004507F6    68 08084500     PUSH Memory_C.00450808          ; ASCII "Error"
004507FB    68 10084500     PUSH Memory_C.00450810          ; ASCII "This NAG will
be shown every 20 seconds, KILL IT !!!"
00450800    6A 00           PUSH 0
00450802    E8 45FEFFFF     CALL <JMP.&user32.MessageBoxA>
00450807  . C3              RET
```

Program ce posle par sekundi izbaciti poruku Error – Modified. Sada mozemo da potrazimo ovaj string u fajlu. Nacemo ga ovde:

```
Text strings referenced in Memory_C:CODE, item 1825
 Address=004507E0
 Disassembly=ASCII "Error - Modified"
```

Duplim klikom na ovaj red zavrsicemo ovde:

```
004507D8  . FFFFFFFF        DD FFFFFFFF
004507DC  . 10000000        DD 00000010
004507E0  . 45 72 72 6F 7>ASCII "Error - Modified"
004507F0  . 00              ASCII 0
004507F1    00              DB 00
```

Posto ovo ne predstavlja nikakav kod nego samo referencu ka stringu (*ovo se koristi tako sto se kada zelimo da se ovaj string pojavi na ekranu funkciji zaduzenoj za pojavljivanje ne prosledjujemo string nego adresu na kojoj se on nalazi*), selektovacemo adresu 004507E0 i postavicemo *desno dugme -> Breakpoint -> Memory, on access* breakpoint na nju. Posle malo cekanja zavrsicemo ovde usled izvrsavanje breakpointa:

```
00404687  |. 8B1F           MOV EBX,DWORD PTR DS:[EDI]
00404689  |. 38D9           CMP CL,BL
0040468B  |. 75 41          JNZ SHORT Memory_C.004046CE
```

Ono sto sada moramo da uradimo je da izvrsimo sav kod dok se ne vratimo u deo koda koji je pristupio ovoj memoriji. Sto u principu znaci da treba (u

ovom slucaju) da se vratimo iz 2 CALLa to jest da izvrsimo dve RET komande. Kako ovo znamo??? Ne znamo, nego se jednostavno vracamo iz jednog po jednog CALLa sa F8 dok nedodjemo do koda koji izgleda kao funkcija za prikazivanje ovog stringa na ekranu. Za ovo je potrebno malo vezbe. Kada se konacno vratimo iz drugog CALLa videcemo ovo:

```
0045077E     /74 13            JE SHORT Memory_C.00450793
00450780  |. |8B45 FC          MOV EAX,DWORD PTR SS:[EBP-4]
00450783  |. |8B80 04030000    MOV EAX,DWORD PTR DS:[EAX+304]
00450789  |. |BA E0074500      MOV EDX,Memory_C.004507E0              ; ASCII "Error -
Modified"
0045078E  |. |E8 C1F1FDFF      CALL Memory_C.0042F954
00450793  |> \33C0             XOR EAX,EAX
00450795  |.  5A               POP EDX
```

Ovo vec izgleda kao deo koda koji sluzi za prikazivanje poruke na ekranu. Ono sto treba da uradimo da se ova poruka nikada ne prikaze na ekranu je da skok na adresi 0045077E promenimo iz JE u JMP i to je to. Progam ce uspesno umesto poruke o modifikaciji memorije prikazivati poruku o novom memory checksumu. Sada mozemo snimiti ove promene pritiskom na *desno dugme -> Copy to executable -> All modifications -> Save...*

Napomena:

Ovaj problem moze biti resen i mnogo jednostavnije ako znate koji je to Win32API zaduzen za citanje memorije. Taj API je ReadProcessMemory.

Vezba:

Ova nasa modifikacija ce raditi samo u slucaju da smo promenili NAG pomocu RET komande. Probajte isto ovo samo umesto RET komande unesite RET pa NOP. Naravno program ce se zatvoriti. Na vama je da otkrijete gde se vrsi provera da li je neka adresa NOPovana i da je zaobidjete.

Resenje:

Promenite skok na adresi 0045070C iz JNZ (*JNE*) u JMP.

Reversing CRC32 checks

Jedna od veoma cesto koriscenih anti-cracking tehnika je provera CRC checksuma nekog fajla. Sta je to CRC??? CRC je tehnika koja omogucava proveru sadrzine nekog fajla. Ona jednostavno uzme neki fajl, napravi njegovu "sliku" i to pretvori u osmocifreni hexadecimalni broj koji predstavlja unikatnu "sliku" posmatranog fajla. Posto CRC zavisi od sadrzine fajla to znaci da razliciti fajlovi imaju razlicite CRCove. CRC je broj koji se nalazi u rasponu od 00000000 pa do FFFFFFFF. Ovo ujedno znaci da ako promenimo neki fajl patchovanjem promenicemo mu i CRC pa ce sam program jednostavnom proverom znati da je modifikovan. Imajte na umu da ako je ova zastita dobro implementirana u program bice vam potrebno, izuzetno mnogo vremena i zivaca da pronadjete gde se to tacno nalazi CRC provera u programu koji reversujete. Pimer koji sam ja za ovu priliku napravio je veoma lak i necete imati nikakvih problema da ga reversujete. Primer se nalazi ovde: ...\Casovi\Cas7\CRC\crc32.exe, a sadrzi i pomocni fajl crc32table.txt koji ustvari cuva enkriptovani CRC32 checksum crc32.exe fajla. Primecujete da se u txt fajlu cuva CRC checksum koji program koristi da ga uporedi sa svojim CRCom i pitate se zasto je to ovako??? Zasto program ne cuva svoj CRC u samom sebi??? Pa odgovor je veoma jednostavan: Ako bi program cuvao svoj CRC u samom sebi CRC bi se uvek menjao, ako kompajlujemo program koji cuva u sebi jedan CRC, zbog tog broja CRC bi se promenio, a ako bi taj broj promenili u novi CRC, onda bi se CRC opet promenio pa se zbog toga CRC cuva u posebnom fajlu ili u Registryu. Ako startujete ovaj primer videcete da se pojavljuje standardan NAG ekran pre samog startovanja i zatvaranja aplikacije. Ovo je ono sto moramo da promenimo u ovom primeru je da ubijemo ovaj NAG. Ovo je veoma lako i to, ako ste citali knjigu lepo sa razumevanjem, morate da znate da uradite sami. Kada ovo uradite snimite fajl kao crc32.exe a stari reimenujte u crc32_1.exe. Ako sada startujete crc32.exe fajl prikazace se poruka o pogresnom CRCu. Mozete i nju naci i promeniti. Ovo je isto lako:

```
00451104  |.  E8 DB3AFBFF     CALL crc32.00404BE4
00451109  |.  74 40           JE SHORT crc32.0045114B
0045110B  |.  6A 40           PUSH 40; MB_OK|MB_ICONASTERISK|MB_APPLMODAL
0045110D  |.  68 C0114500     PUSH crc32.004511C0              ; |Title = "Error"
```

samo treba promeniti skok na adresi 00451109 iz JE u JMP i to je to. Kao sto rekoh ovo je veoma laka "zastita", dok ce vas lepo implementirana CRC provera posteno namuciti. Pogledajmo malo ovu CRC zastitu koja pocinje na adresi 00450ED0. Jednostavnim pregledom stringova videcemo da program otvara fajl crc32table.txt i kopira crc32.exe u bak.bak da bi najverovatnije na novom bak.bak fajlu proverio CRC. Ovo je zanimljivo posto kako vidimo program koristi specificno ime crc32.exe a ne neku promenljivu u slucaju da se ime exe fajla promeni. I ovo mozemo da iskoristimo. Iskopirajmo crc32.exe u novi fajl new.exe i u new.exe fajlu samo ubimo NAG. Ako sada startujemo new.exe videcemo da nema NAGa ali da nema ni poruke o pogresnom CRCu jer se CRC32 proverava na kopiji crc32.exe fajla a ne na samom sebi, u ovom slucaju new.exe-u. Naravno ako u praksi nadjete ovako glupu zastitu posaljite mi email programera koji ju je napisao da ga castim pivom :) A posto sam ovako glupu zastitu napisao ja sam, mislim da cu

morati sam sebe da castim pivom :) Bilo kako bilo, ova CRC zastita definitivno zasluzuje nagradu sa +Fravia sajta:

"Da da znam, zasluzio sam nagradu posteno :)" – Ap0x

Salu na stranu ono (*pametno*) sto ce ovaj primer da vas nauci je kako da dekriptujete onaj crc32table.txt fajl i tako razbijete CRC zastitu. Otvorite originalni crc32.exe fajl u Ollyu postavite obican break-point na adresu 00450ED0. Sa F9 startujte program i sacekajte da se Olly zaustavi na ovom break-pointu. Sa F8 polako analizirajte kod. Na adresi 00450ED8 imamo stvarno dugacak loop (*cak 41h puta se ponavlja*) koji ne radi nista bitno za nas. Idemo dalje... Kada stignemo do adrese 00450F28 videcemo da se u EAX smesta cela putanja do crc32table.txt fajla. Ovo moze biti zanimljivo. Sa F8 idemo dalje kroz kod. Nista bitno se ne desava sve dok ne dodjemo do adrese 00451037 kada program kopira samog sebe u bak.bak fajl. Nista bito za ekripciju, pa nastavljamo dalje. Stigli smo do 004510C9 i sada imamo neki loop koji se izvrsava nekoliko puta. Ovaj loop mora biti jako zanimljiv posto se nalazi odmah iznad poruke o pogresnom CRCu. Da vidimo sta se desava u njemu.

```
004510CE  |> /8D45 FC       /LEA EAX,DWORD PTR SS:[EBP-4]
004510D1  |. |E8 1A3CFBFF   |CALL crc32.00404CF0
004510D6  |. |8B55 FC       |MOV EDX,DWORD PTR SS:[EBP-4]
004510D9  |. |8A541A FF     |MOV DL,BYTE PTR DS:[EDX+EBX-1]
004510DD  |. |80F2 2C       |XOR DL,2C
004510E0  |. |885418 FF     |MOV BYTE PTR DS:[EAX+EBX-1],DL
004510E4  |. |43            |INC EBX
004510E5  |. |4E            |DEC ESI
004510E6  |.^\75 E6         \JNZ SHORT crc32.004510CE
```

Prodjimo par puta kroz njega i videcemo da se uzima jedno po jedno slovo iz nekog stringa, mozda onog koji se nalazi u samom crc32table.txt fajlu, i da se njegova ASCII vrednost xoruje sa 2Ch ili sa 44 decimalno. Kada se ovaj loop zavrsi, to jest kada stignemo do adrese 004510E8 onda cemo u EAXu imati string **EAX 008B213C ASCII "7CC85525"** koji odgovara dekriptovanom CRCu iz fajla crc32table.txt. Ako je taj CRC jednak CRCu fajla bak.bak onda fajl nije modifikovan. Ono sto nas interesuje je kako da reversujemo enkripciju pa da dobijemo drugi enkriptovani CRC, za izmenjeni NAG free program, koji bi smestili u crc32table.txt fajl. Ovo mozemo resiti na dva nacina.

Prvi nacin:

Predpostavimo za trenutak da neznamo kako program generise enkriptovani string koji sadrzi CRC vrednost crc32.exe fajla. Ono sto cemo da uradimo je da otvorimo crc32table.txt fajl i da iz njega iskopiramo onaj string. Jedino potrebno podesavanje Notepada je da promenite font u Terminal. **[Hint:** *Format -> Font -> Terminal -> Regular***]**, ono sto nam je jos potrebno je ASCII tablica koju mozete preuzeti sa www.asciitable.com (*opciono ali veoma korisno*) i spremni smo da pocnemo.

Prvo sto treba da uradimo je da da sve simbole iz crc32table.txt pretvorimo u ASCII kodove. U ovom slucaju je ovo jako tesko izvesti, jer su slova prilicno ne standardna, pa cemo napraviti program koji ce to uraditi za nas. Kod Visual Basic programa koji to radi bi izgledao bas ovako:

```
Text2.Text = ""
for i = 1 to Len(Text1.Text)
Text2.Text = Text2.Text & Asc(Mid$(Text1.Text,i,1))
if i < Len(Text1.Text) then Text2.Text = Text2.Text + ","
next i
```

Text2 sadrzi rezultat a Text1 sadrzi string za koji se racuna ASCII. Gotov primer se nalazi u istom folderu pod imenom ascii.exe. Otvorimo ovaj program i u njega unesimo string iz fajla crc32table.txt. Ono sto ce program pokazati je:

27,111,111,20,25,25,30,25

Sada treba patchujemo program crc32.exe i snimimo patchovan program kao new.exe. Patchovacemo 00450E88 adresu u ovo:

```
00450E88    C3        RET
00450E89    90        NOP
```

Reimenujte originalni crc32.exe u crc32_1.exe, a new.exe u crc32.exe. Sada cemo pomocu getCRC32.exe dobiti CRC vrednost fajla crc32.exe (*new.exe*) fajla, dobicemo novi CRC koji iznosi 0D2541F9 (*zapamtite CRC mora imati 8 cifara*). Sada cemo i ovaj CRC pretvoriti u ASCII pomocu ascii.exe-a. Dobicemo ovo:

48,68,50,53,52,49,70,57

Pored ovog nam je poteban i ACII kod originalnog crc32.exe fajla, koji se trenutno zove crc32_1.exe. Posto znamo da ovaj CRC iznosi 7CC85525, i njega cemo pretvooriti u ASCII. To izgleda ovako:

55,67,67,56,53,53,50,53

Sada treba da otkrijemo kako treba enkriptujemo novu CRC vrednost - 0D2541F9. Za ovo su nam potrebna dva podatka. ASCII enkriptovanog CRCa i ASCII samog CRCa. Ovi podaci su:

27,111,111,20,25,25,30,25 i 55,67,67,56,53,53,50,53

Ono sto je bitno da se svaka prosta a i komplikovana enkripcija zasniva na obicnom XOR-ovanju. Ovo znaci da je ASCII kod jednog karaktera enkriptovan (xorovan) nekom vrednoscu i tako je dobijena nova ASCII vrednosti. Mi treba da saznamo kojom vrednoscu je XORovan svaki karakter originalnog CRCa - 7CC85525 kako bi bio dobijen enkriptovani string koji se nalazi u crc32table.txt fajlu. Ovo cemo dobiti reversnim xorovanjem, to jest uporedjivanjem enkriptovanog stringa i neenkriptovanog originalnog CRCa crc32.exe fajla. Znaci **27 xor ?? = 55**, **111 xor ?? = 67**,... Ono sto je bitno kod xor funkcije je da je ona revezibilna to jest da je **x xor y = z** ali i da je **z xor x = y i z xor y = x,** i tako cemo otkriti vrednost ??. Znaci **55 xor 27 = 44, 67 xor 111 = 44**,... i tako smo dobili magicnu xor vrednost za svako slovo CRCa. Sada treba samo da enkriptujemo novi CRC pomocu vrednosti 44, i da dobijenu ASCII vrednost pretvorenu u slova i simbole snimimo u crc32table.txt fajl. Ovo cemo uraditi pomocu novog programa koji cemo sami napisati. Evo kako to izgleda u Visual Basicu:

```
 Text2.Text = ""
for i = 1 to Len(Text1.Text)
 Text2.Text = Text2.Text & Chr(Asc(Mid$(Text1.Text,i,1)) xor 44)
next i
 open "crc32table.txt" for output as #1
  print #1, Text2.Text
 close #1
```

Gotov primer se nalazi u istom folderu pod imenom XORascii.exe. Otvorimo ovaj program, unesimo novi CRC u njega (*0D2541F9*) i program ce sam generisati novi crc32table.txt fajl. Ako probamo da startujemo crc32.exe sada videcemo da se poruka o modifikaciji ne menja, to jest da je jos tu. Mora da smo negde pogresili !!! Probajmo samo da umesto 0D2541F9 unesemo D2541F9 u XORascii.exe i startujmo program sada. Radi, znaci ipak je sve ok. Ova tehnika moze biti "malo" komplikovana ali je odlicna ako se svaki karakter XORuje drugacijim brojem, tako da ovu vrstu dekripcije nije lose znati. Istu ovakvu ekripciju passworda mozete naci i u programu Trillian, pa za vezbu mozete napraviti program koji dekriptuje njegov password.

Drugi nacin:

Drugi nacin se zasniva na reversovanju samog loopa za dekripciju enkriptovanog CRCa iz crc32table.txt fajla. Ovo je veoma lako (*lakse od prvog nacina*) i dacu vam odmah source za Visual Basic:

```
 Text2.Text = "" 'izlazni text box
for i = 1 to Len(Text1.Text) 'ulazni text box
 Text2.Text = Text2.Text & Chr(Asc(Mid$(Text1.Text,i,1)) xor 44)
next i
 open "crc32table.txt" for output as #1
  print #1, Text2.Text
 close #1
```

i za Delphi:

```
var
 wFile:TextFile;
 crc:string;
 i:integer;
begin
 crc := '';
for i := 1 to Length(Edit1.Text) do begin //ulazni text box
 crc := crc + Chr(ORD(Edit1.Text[i]) xor 44);
end;
 AssignFile(wFile,'crc32table.txt');
  writeln(wFile,crc);
 CloseFile(wFile);
end;
```

“Not Getting Caught :)” - *Exerecise*

Ako niste primetili pored primera ...\Casovi\Cas7\Debugger check.exe postoji i primer ...\Casovi\Cas7\Debugger Check2.exe. Ovaj drugi je napravljen da bude vezba za izbegavanje detekcije debuggera. Ako ste pazljivo citali prvi i drugi deo ovog poglavlja bez problema cete zaobici sve ove zastite i resicete uspesno i ovu vezbu. Pokusajte sami da resite ovaj problem a ako ne budete mogli vratite se ovde da utvrdite gradivo.

Resenje:

Ako citate ovaj deo predpostavljam da niste sami mogli da se oslobodite provere debuggera u primeru. Predpostavljam da ste krenuli logickim redosledom tako sto ste zapamtili poruku o detektovanom debuggeru i probali ste da je potrazite u fajlu. Ali, desilo se nesto jako cudno, nema takve niti bilo kakve slicne poruke u fajlu :) Ono sto sam namerno uradio za ovu vezbu je ekripcija stringova unutar samog fajla. To znaci da su stringovi idalje tu samo nisu isti u fajlu i u memoriji. Ovo znaci da se stringovi pre upotrebe dekriptuju u svoj pravi oblik. Naravno vi ne znate kakav sam ja algoritam koristio za enkripciju stringova tako da nikako ne mozete znati kako koja poruka izgleda enkriptovana. Ono sto cemo mi uraditi je veoma jednostavno, koristicemo drugi metod za pronalazak mesta gde se proverava da li je program otvoren uz pomoc debuggera. Trazicemo ova mesta pomocu API poziva. Otvoricemo ovaj program uz pomoc Ollya i postavicemo break-point na MessageBoxA API posto se pomocu njega prikazuje poruka o aktivnom debuggeru. Ovo se radi u Executable modulima (*ALT + E*), pod opcijom View Names (*CTRL + N*); Toggle break-point on import, ako ste zaboravili. Sada pokrenimo ovaj primer i sacekajmo da program dodje do break-pointa. Uklonimo ovaj break-point pritiskom na F2. Posto se nalazimo u dll fajlu sa F8 cemo izvrsavati kod polako dok ne dodjemo do RET komande i dok se i ona ne izvrsi. Sada, kada smo se vratili iz funkcije, nalazimo se na adresi 00450D7F. Ovo izgleda kao funkcija koja prikazuje poruku o prisustvu debuggera. Ovo cemo ukloniti jednostavnim postavljanjem obicne RET komande na adresi 00450D44. Resetujmo program sa CTRL + F2 i izmenimo adresu 00450D44 u RET (C3). Startujmo program i videcemo da se poruka opet pojavila. Hmmm. Resetujmo program opet, izmenimo adresu 00450D44 i postavimo opet break-point na MessageBoxA. Program je opet stao u nekom dll-u i posle izvrsavanja RET komande vraticemo se u deo koda koji poziva MessageBoxA API. Sada se nalazimo na adresi 00450847. Hmmm ovaj CALL izgleda isto kao onaj koji pocinje na adresi 00450D44. Da li postoje dve vrste provere ? Zapamtimo dve adrese 00450D44 i 0045080C posto su one pocetne adrese ovih CALLova. Resetujmo program sa CTRL + F2 i pomocu Go To adress opcije (*prvo dugme na levo pored dugmeta L u toolbaru*) oticemo na obe ove adrese. Kada odemo na prvu i selektujemo je videcemo detalje o njoj:

Local call from 00450DED

Ovo znaci da se ovaj CALL poziva samo sa adrese 00450DED. A kada odemo na adresu 0045080C videcemo sledece:

```
Local calls from 004509BA, 00450A13, 00450A45, 00450A67, 00450A89, 00450AAB,
00450ACD, 00450AEF, 00450B11
```

to jest da se on poziva sa vise adresa. Na svaku od ovih adresa mozemo da odemo pritiskom da desnuo dugme na ovoj liniji i selekcijom na koju to adresu zelimo da odemo. Ako odemo samo na neke od tih adresa bice nam jasno sta se tu desava. Oticemo samo na adrese 00450DED i na adresu 00450B11. Na adresi 00450DED imamo:

```
00450DC3  . 27 20 37 25 2 ASCII "' 7%%'"
00450DC9  . 30 62 24 2D 3 ASCII "0b$-7,&nb#22.+!#"
00450DD9  . 36 2B 2D 2C 6 ASCII "6+-,b5+..b',&l",0
00450DE8  . E8 87F9FFFF  CALL IsDebuggerPresent  ; [IsDebuggerPresent]
00450DED  . E8 52FFFFFF  CALL Debugger.00450D44
00450DF2  . C3           RET
```

Ovaj prvi kod prikazuje poziv poruci o detektovanom debuggeru odmah posle CALLa IsDebuggerPresent APIa. Ok sada znamo da program prisustvo debuggera proverava uz pomoc API funkcije. Ovo je prvi deo provere. Drugi deo provere se nalazi na adresi 00450B11. Na toj adresi imamo ovo:

```
00450B09  |. 50              PUSH EAX                 ; /Title
00450B0A  |. 6A 00           PUSH 0                   ; |Class = 0
00450B0C  |. E8 0F62FBFF     CALL FindWindowA         ; \FindWindowA
00450B11  |. E8 F6FCFFFF     CALL Debugger.0045080C
00450B16  |. 33C0            XOR EAX,EAX
```

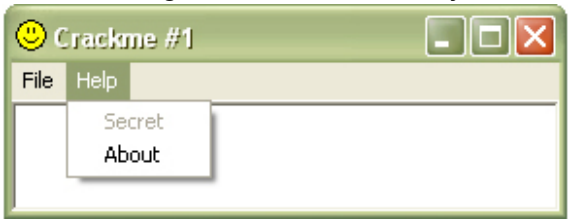
odakle zakljucujemo da je druga provera FindWindowA. Resenje obe ove provere je izuzetno lako. Mozemo jednostavno staviti RET komande na adresama 00450D44, 0045080C i problem je resen. Ako zelimo da vidimo koje to prozore ovaj program trazi mozemo jednostavno postaviti break-pointove na sve CALLove ka FindWindowA APIu. Imajte na umu da se break-pointi ne postavljaju na sam CALL FindWindowA nego na PUSH EAX pre toga posto ce se tada u EAXu nalaziti naslov (title) prozora koji se trazi.

08 Cracking it...

Ovo poglavlje sadrzi osnove ukljucivanja iskljucenih menija i dugmadi, pronalazenje izgubljenih passworda, pobedjivanje time-trial programa. Ova tehnika je podcenjena zbog postojanja velikog broja raznih programa koji ovo mogu uraditi za vas brzo i jednostavno. Nas jednostavno zanima kako se ovo radi rucno i kako bih to mogli izvesti u nedostatku alata poput ReSHackera.

ReEnable buttons - *ASM*

Ovaj deo poglavlja ce vas nauciti kako da ukljucite neke iskljucene opcije u programima. Ovo je narocito korisno ako je neki program zasticen bas na ovaj nacin. Na primer, moze se desiti u nekoj "meti" da je recimo Save opcija zatamljena i da se nemoze ukljuciti. Naravno vecina vas je naucila da ovakve probleme resava preko raznih ReSHackera i slicnih alata. Nemam ja nista protiv takvog pristupa ali morate i sami priznati da je skroz lame. Primer koji sam specijalno za potrebe ovog poglavlja "iskopao" na netu pristupa ovoj problematici na sasvim drugaciji nacin. Naime fajl crackme.exe koji se nalazi u folderu ..\Casovi\Cas8\ je specifican po tome sto ne koristi resurse nego sam programski pomocu Win32Api-a pravi elemente koji se nalaze u njemu. Kao sto vidite ovde ReSHacker ne pomaze :) Naravno NAG do sada sigurno znate da ubijete a ono sto je predmet razmatranja ovog poglavlja je kako ukljuciti iskljucen meni. Posto se ovaj "tajni" meni pravi preko API komandi iskoristicemo to da pronadjemo mesto odakle se poziva procedura koja pravi ovaj meni. Otvoricemo ovaj program u Ollyu i postrazicemo sve string reference ka stringu Secret. Evo spiska svih referenci koje se nalaze u tom crackmeu.




```
Text strings referenced in crackme:.text
Address   Disassembly                    Text string
0040104E  PUSH crackme.0040315C          ASCII "Secret Unrevealed!"
....
004010F2  PUSH crackme.00403018          ASCII "&Secret"
00401103  PUSH crackme.00403010          ASCII "&About"
00401112  PUSH crackme.00403008          ASCII "&Help"
....
004011D7  PUSH crackme.00403190          ASCII "Crackme #1"
004011F0  PUSH crackme.004031C0          ASCII "Error!"
004011F5  PUSH crackme.00403170          ASCII "Aishh..msgbox takleh create!"
```

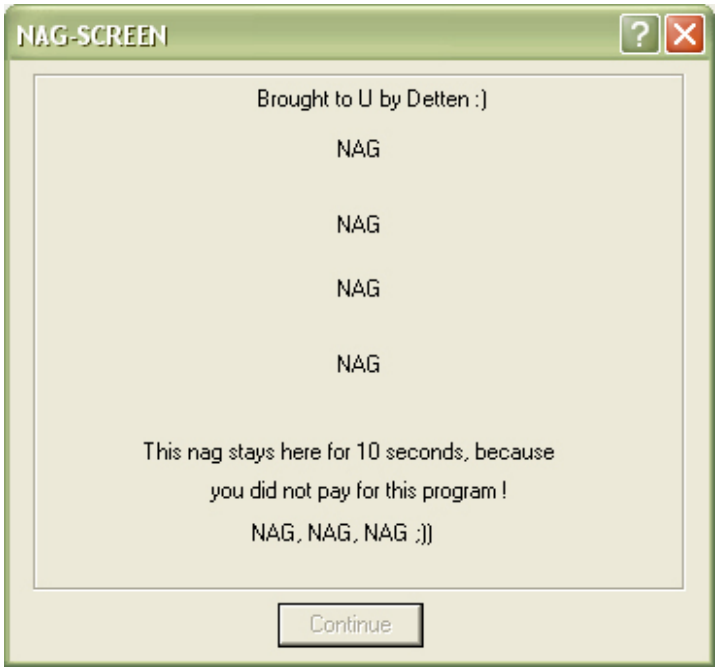
Kao sto vidimo samo zeleni red sadrzi string koji je istovetan onom koji vidimo u crackmeu. Ne dajte da vas zbuni ono & ispred stringa. Kada su meniji u pitanju ono & oznacava da je prvo slovo stringa podvuceno. Ako 2x kliknete na taj red docicete do ovde:

```
FFD7          CALL EDI
68 18304000   PUSH crackme.00403018
8BF8          MOV EDI,EAX
68 2B230000   PUSH 232B
6A 01         PUSH 1
57            PUSH EDI
FFD6          CALL ESI
68 10304000   PUSH crackme.00403010
68 2A230000   PUSH 232A
6A 00         PUSH 0
57            PUSH EDI
FFD6          CALL ESI

CreatePopupMenu
pItem = "&Secret"
ItemID = 232B (9003.)
Flags = MF_BYCOMMAND|MF_GRAYED|MF_STRING
hMenu
AppendMenuA
pItem = "&About"
ItemID = 232A (9002.)
Flags = MF_BYCOMMAND|MF_ENABLED|MF_STRING
hMenu
AppendMenuA
```

Kao sto primecujete osim razlike u prvoj PUSH komandi, koja nam je nebitna jer predstavlja samo ID menija, imamo razliku i u drugoj PUSH komandi. Kod Secret menija imamo PUSH 1 a on je iskljucen a kod About menija imamo PUSH 0 a on je ukljucen. Dolazimo do zakljucka da treba samo promenimo PUSH 1 u PUSH 0 da bi i Secret meni bio ukljucen. Kao sto vidite ovaj problem je bilo veoma lako resiti.

ReEnable buttons - *API*

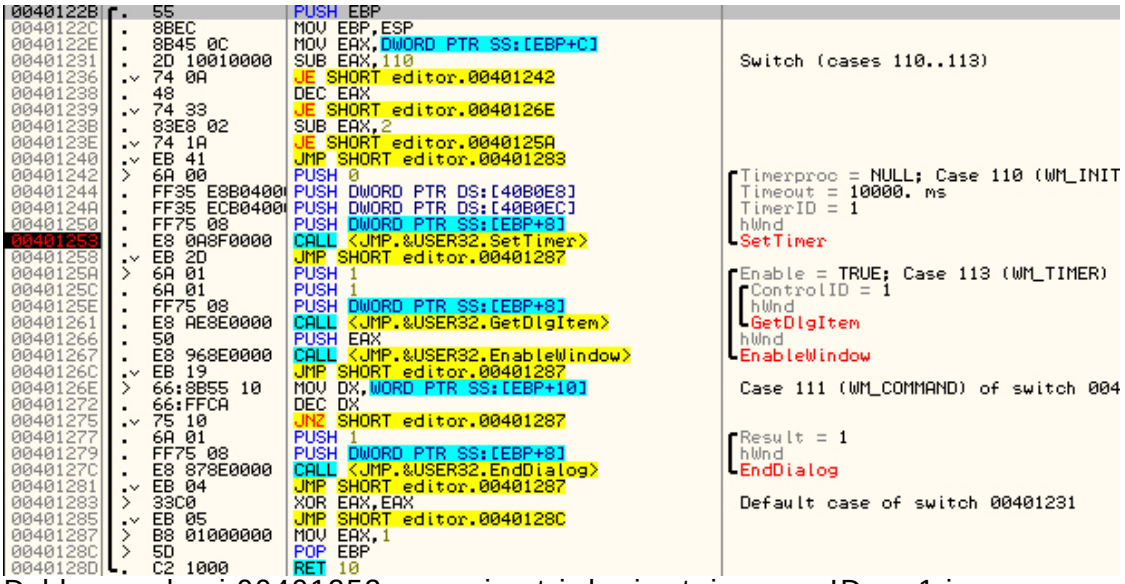
Ovaj deo poglavlja ce vas nauciti kako da pronadjete tacno mesto odakle se iskljucuju / ukljucuju opcije. Naravno posto je Microsoft-ova dokumentacija veoma obimna spisak svih raspolozivih API funkcija nije ni malo lako prelistati i naci ono sto vam treba. Naravno kao i svaki drugi veoma nestrpljivi cracker odlucio sam da batalim API reference i da nadjem neki prakticni primer kojim cu lako otkriti kako se to iskljucuju / ukljucuju dugmici, meniji, formovi i sl... Slucajno sam pri ruci imao savrsen primer za ovaj problem. Primer ...\Casovi\Cas8\editor.exe izgleda ovako:



Kao sto se vidi na ovoj slici u pitanju je jos jedan NAG ekran koji treba ukloniti. Jedina prednost ovog primera nad svim drugima je to sto on ima iskljuceno dugme koje se posle deset sekundi ukljuci propustajuci vas dalje u program. Ovo je bitna karakteristika koju cemo iskoristiti da bi smo nasli koja je to API funkcija zaduzena za ukljucivanje tog dugmeta. Da bi smo ovo uradili mozemo da koristimo dva nacina: laksi nacin i moj nacin :) Jedini problem sa laksim nacinom je da on predpostavlja da znate API funkcije koje se koriste za simuliranje tajmera. To jest da znamo kako to program napravi pauzu od 10 sekundi, a ako vam ovo nije poznato mozete uvek da koristite moj nacin :)

Nacin 1:

Laksi nacin pronalaska mesta odakle se poziva funkcija koja ukljucuje iskljuceno dugme je da postavimo break-point na svaku referencu ka API funkcijama koje sluze za pravljenje tajmera i / ili pauziranje programa. Ovo rade dve API funkcije: **kernel32.Sleep** i **user32.SetTimer** pa cemo pronaci koja se od ove dve koristi u ovom primeru. Preko Ollya pogledajte koji se to importi nalaze u module -> names prozoru. Od ove dve spomenute funkcije pronacicete samo API SetTimer. Naravno da se vreme moze pronaci i preko drugih APIa kao sto su GetLocalTime, GetSystemTime i slicnih ali je najbrze preko SetTimer APIa. Dakle postavimo break-point on every reference na ovaj API i onda startujmo program. Program ce zastati na adresi 00401253 odakle se poziva SetTimer API. Ako odskrolujemo malo gore i pregledamo ceo ovaj CALL videcemo kako se to startuje timer i mozda cemo naci API funkciju koja ukljucuje dugme. Ono sto vidimo prikazano je na sledecoj slici:



Dakle na adresi 00401253 se poziva t.j. kreira tajmer sa IDom 1 i vremenom ponavljanja od 10000 ms odnosno 10 sekundi. Ako analiziramo malo ova deo koda primeticemo da posto se kreira tajmer izvrsava se neki JMP skok koji preskace par API funkcija (*GetDlgItem, EnableWindow, EndDialog*). Primeticemo da se i iznad API funkcije za kreiranje tajmera nalazi par kondicionih JE skokova je jedan nekondicioni JMP skok. Mozemo da postavimo break-pointe na njih i da startujemo program sa F9 kada dodje do njih. Tada cemo primetiti da se tajmer kreira samo jednom a da se ostali skokovi ne izvrsavaju dok ne prodje 10 sekundi, kada se izvrsava skok na adresi 0040123E koji vodi do adrese 0040125A koja prvo vraca handle (*adresu kontrole koja se trazi u memoriji*) neke kontrole uz pomoc GetDlgItem API da bi se toj istoj kontroli poslala komanda Enable = True preko APIa EnableWindow. Izgleda da se API EnableWindow koristi za ukljucivanje iskljucenih dugmadi, menija i sl. I posle malo mucenja pronasli smo API funkciju koja se koristi za ukljucivanje dugmica. Ona glasi ovako:

EnableWindow(hwnd,1)

gde umesto hwnd ide handle dugmeta ili nekog drugog objekta u prozoru a umesto 1 ide 1 za ukljucivanje i 0 za iskljucivanje. Hwnd dugmeta mozemo dobiti pomocu druge API funkcije GetDlgItem koja se poziva ovako:

GetDlgItem(hwnd,ControlID)

gde je hwnd handle prozora u kojem se nalazi dugme a control ID je Id koji smo dodelili nasem dugmetu u .res fajlu, u ovom slucaju ControlID je 1 jer je tako Daten (*autor programa*) deklarisao ID u .res fajlu. Naravno ako ste za razliku od mene procitali Microsoft-ove API reference ovo ste znali i bez primene reversenog inzinjeringa. Ja sam ovo morao da naucim na malo tezi nacin :)

Nacin 2:

Ovo je malo tezi (*ili malo laksi???*) nacin od predhodnog i ja sam ga prvobitno upotrebio da pronadjem API koji nam treba. Otvorio sam ovaj crackme pomocu ResHackera da pronadjem ID (*jedinstveni identifikacioni broj*) pomocu

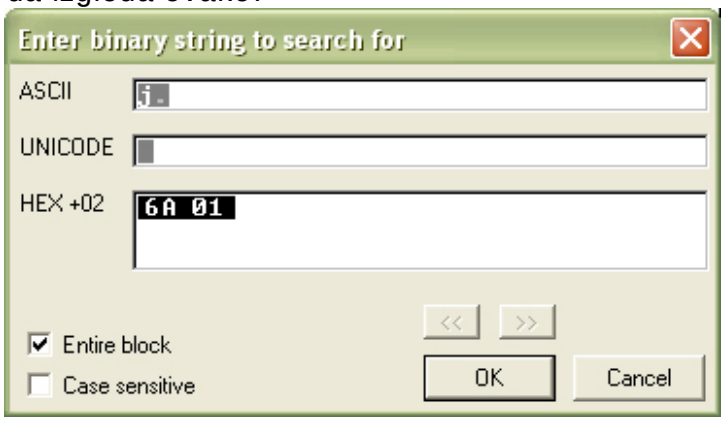
kojeg se program “obraca” ovoj kontroli. Kada otvorimo crackme pomocu ResHackera imacemo ovo:

```
900 DIALOG 0, 0, 240, 191
STYLE DS_MODALFRAME | DS_CONTEXTHELP | WS_POPUP | WS_VISIBLE | WS_CAPTION |
WS_SYSMENU
CAPTION "NAG-SCREEN"
LANGUAGE LANG_NEUTRAL, SUBLANG_NEUTRAL
FONT 8, "MS Sans Serif"
{
  CONTROL "&Continue", 1, BUTTON, BS_PUSHBUTTON | BS_CENTER | WS_CHILD |
WS_VISIBLE | WS_DISABLED | WS_TABSTOP, 92, 172, 50, 14
  CONTROL "NAG", 101, STATIC, SS_CENTER | WS_CHILD | WS_VISIBLE, 20, 48, 200, 10
  CONTROL "This nag stays here for 10 seconds, because", 102, STATIC, SS_CENTER |
WS_CHILD | WS_VISIBLE, 20, 120, 192, 8
  CONTROL "you did not pay for this program !", 103, STATIC, SS_CENTER | WS_CHILD |
WS_VISIBLE, 20, 132, 200, 9
  CONTROL "NAG, NAG, NAG ;))", 104, STATIC, SS_CENTER | WS_CHILD | WS_VISIBLE, 40,
146, 148, 13
...
}
```

iz cega smo saznali mnogo... Saznali smo da je ID cele forme koja se pojavljuje na ekranu 900 (*decimalno*) i saznali smo da je ID dugmeta Continue jednak 1. Ovo je bitno jer kada program na ASM nivou zeli da uradi nesto sa nekim objektom on prvo mora da zna na koji objekat se data komanda odnosi. Dakle ono sto je sigurno je da ce se kao jedan od parametara API funkciji koja ukljucuje dugme sigurno proslediti i ID dugmeta. To bi moglo da izgleda bas ovako:

```
PUSH ukljuci_dugme
PUSH Id_controle
PUSH neki_drugi_parametri
CALL UkljuciDugmeAPI
```

Kao sto vidimo moraju se koristiti PUSH komande, a PUSH 1 kako bi trebalo da izgleda prosledjivanje IDa dugmeta bi u HEX obliku izgledalo ovako 6A01. Sada samo treba da potrazimo ovaj binarni string u fajlu i videcemo odakle se to poziva API za ukljucivanje dugmeta. Pritisnite CTRL + B u CPU prozoru Ollya da nadjete ovaj string. Kada unesete 6A01 prozor koji se pojavio treba da izgleda ovako:



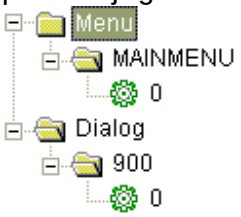
Pritiskom na dugme OK program ce nas odvesti na adresu 0040111B a ocigledno je da je to pogresna adresa jer CreateFileA sluzi za nesto drugo. Pritisnucemo onda CTRL+L (search again) i zavrsicemo na adresi 0040125A gde se nalazi par zanimljivih redova:

PUSH 1 // enable = true,

red PUSH 1 // ControlID i red CALL EnableWindow pa je ocigledno da se API EnableWindow koristi za ukljucivanje dugmeta a da su mu parametri handle kontrole i true ili false switch.

ReEnable buttons - *ResHacker*

Ova neverovatno lame vrsta “crackovanja” je veoma zastupljena kod pocetnika i onih koji jednostavno ne zele da se “zamaraju” kopanjem po ASM kodu. Daleko od toga da ReSHacker nije koristan i pravim crackerima i da se ne ustrucavaju da ga koriste. Naravno pravi crackeri upotrebljavaju ovaj izuzetan program za kreiranje RES fajlova za modifikaciju sopstvenih programa a ne za beznacajno ukljucivanje ukljucenih dugmica. Bez obzira na sve ovo naucicu vas kako da koristite ovaj program za resavanje NAG problema. Kada otvorite Datenov program (*pogledaj prosli deo poglavlja*) i preko njega otvorite nasu metu videcete sledece drvo u levoj strani ekrana:



Naravno jasno je sta sve ovo znaci... U .res sekciji .exe fajla postoji jedan Meni i jedan Dialog. Ako selektujemo ./Menu/MAINMENU/0 videmo glavni meni programa koji se pojavljuje kada se iskljuci NAG screen. Jedini dialog koji se nalazi u .res sekciji je bas taj NAG screen koji zelimo ili da uklonimo ili da dugme Continue u njemu uvek bude ukljuceno. Da bi smo ovo uradili otvoricemo ./Dialog/900/0 i pogledacemo sta se nalazi tamo:

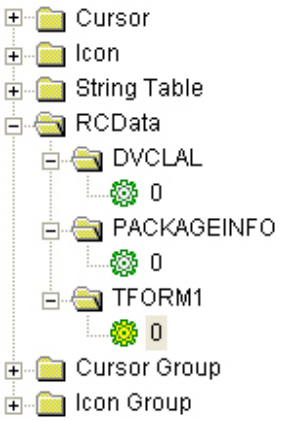
```
900 DIALOG 0, 0, 240, 191
STYLE DS_MODALFRAME | DS_CONTEXTHELP | WS_POPUP | WS_VISIBLE | WS_CAPTION |
WS_SYSMENU
CAPTION "NAG-SCREEN"
LANGUAGE LANG_NEUTRAL, SUBLANG_NEUTRAL
FONT 8, "MS Sans Serif"
{
  CONTROL "&Continue", 1, BUTTON, BS_PUSHBUTTON | BS_CENTER | WS_CHILD |
WS_VISIBLE | WS_DISABLED | WS_TABSTOP, 92, 172, 50, 14
  CONTROL "NAG", 101, STATIC, SS_CENTER | WS_CHILD | WS_VISIBLE, 20, 48, 200, 10
  CONTROL "This nag stays here for 10 seconds, because", 102, STATIC, SS_CENTER |
WS_CHILD | WS_VISIBLE, 20, 120, 192, 8
  CONTROL "you did not pay for this program !", 103, STATIC, SS_CENTER | WS_CHILD |
WS_VISIBLE, 20, 132, 200, 9
  CONTROL "NAG, NAG, NAG ;))", 104, STATIC, SS_CENTER | WS_CHILD | WS_VISIBLE, 40,
146, 148, 13
  CONTROL "Frame1", 105, STATIC, SS_ETCHEDFRAME | WS_CHILD | WS_VISIBLE, 8, 4, 224,
164
...
}
```

Ovo sto tu pise nam je skroz nebitno posto sve mozemo da odradimo pomocu samog programa ResHacker, bez imalo kucanja. Ako NAG dialog nije prikazan pritisnite Show Dialog dugme. Potom selektujete iskljuceno dugme, pritisnite desno dugme misa na njemu i selektujete Edit Control. U novo-otvorenom prozoru pronadjite WS_DISABLED i iskljucite ga. Pre nego sto snimite promene potrebno je da kliknete na OK a onda na Compile Script da bi program ucitao promene u memoriju. Sada mozete pritisnuti Save... Ovako ce dugme uvek biti ukljuceno jer program ne proverava stanje dugmeta pre nego sto mu posalje komandu da se ukljuci.

Sam NAG Dialog se moze ukloniti na par nacina: Mozete selektovati ./Dialog/900/0 i klikom na desno dugme ili reimenovati Dialog 900 u nesto drugo ili da jednostavno obrisati. Tako se NAG dialog nece nikada vise pojavljivati. **NAPOMENA:** Brisanje ili reimenovanje dialoga nije preporucljivo a u vecini slucajeve nece ni raditi kako treba pa ce se program srusiti.

ReEnable buttons -*ResHacker & Delphi*

Prica vezana za ResHacker se malo razlikuje kada je Delphi u pitanju. Delphi je specifican jer u novijim Delphi verzijama, Delphi ne pravi klasicne .res sekcije unutar exe fajla nego je ova sekcija specificna samo za njega. Kada otvorimo program pomocu ...\Casovi\Cas8\EnableMe.exe ResHackera videcemo u drvetu sa strane ovo:



Za Delphi je karakteristicno da svoje resurse smesta u posebnom obliku. Ono sto vidite na slici je putanja do glavne forme u kojoj se nalaze uskljuceni objekti. Kada selektujemo ovu putanju ./RCData/TFORM1/0 imacemo pregled svih objekata koji se nalaze u selektovanoj formi i odakle je moguce modifikovati sve objekte koji se nalaze u formi. Primera radi odskrolovacemo na kraj ovog prozora da bi smo na kraju pronasli tri iskljucena objekta. Prvi iskljuceni objekat je dugme, drugi iskljuceni objekat je EditBox a trece je glavni meni. Svaki objekat je definisan pocetkom krajem koji se oznacava kao END te kontrole. Izmedju pocetka i kraja se nalaze sve vrednosti koje taj objekat ima i koje se mogu odavde modifikovati.

```
object Button1: TButton
  Left = 8
  Top = 56
  Width = 177
  Height = 33
  Caption = 'The Art Of Cracking - Enable ME'
  Enabled = False
  TabOrder = 0
  OnClick = Button1Click
end
object Edit1: TEdit
  Left = 48
  Top = 24
  Width = 129
  Height = 21
  Enabled = False
  TabOrder = 1
  Text = 'Enable this'
end
object MainMenu1: TMainMenu
  Left = 88
  Top = 48
  object File1: TMenuItem
  Caption = 'File'
  Enabled = False
  object Exit1: TMenuItem
  Caption = 'Exit'
  OnClick = Exit1Click
end
```

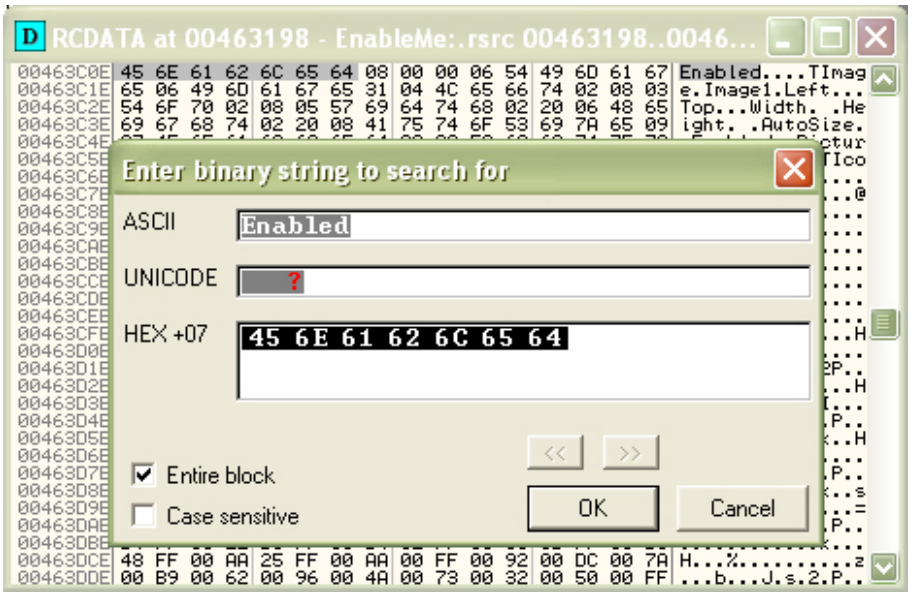
Ono sto je nama bitno za svaki objekat je vrednost svake kontrole koja se nalazi uz Enabled. Ako je Enabled = False onda je taj objekat iskljucen a ako se uz Enabled nalazi True umesto False onda ce objekat biti ukljucen. Kada izmenimo False u True za sve objekte koji nam trebaju onda cemo pritisnuti Compile Script dugme, a sa Save snimamo promene.

ReEnable buttons – *Olly & Delphi*

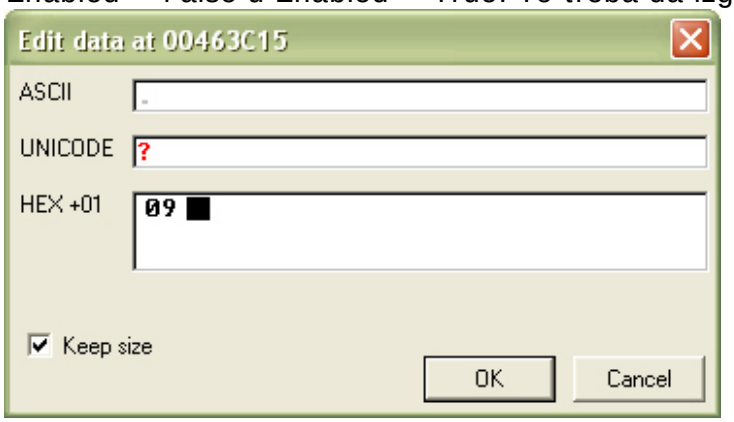
Videli smo da se format Delphi resursa dosta razlikuje od standardnog nacina zapisa resursa. U ovom delu poglavlja cu vas nauciti kako da iskoristite ovo da bez upotrebe ResHackera ukljucite iskljucene objekte. Naravno da se pitate zasto bi ste ovo uopste ucili ako su vam pri ruci alati kao sto su ResHacker, VBReformer i slicni. Naravno da ne morate da ucite ovu tehniku ali postoji veci broj prednosti ako patchujete programe ovako. Naime kada ukljucite neko dugme recimo pomocu ReSHackera on umesto da jednostavno promeni samo jedan bajt on promeni veci deo koda i zbog toga se promeni velicina fajla koji ste patchovali. Sada shvatate zasto je ovo lose. Ako koristite ResHacker necete moci da uporedite dva fajla jer su im velicine razlicite, a samim ti cete biti prinudjeni da ako distribuirate vase patcheve (*ne radite ovo sa komercijalnim shareware programima jer je nezakonito*) bicete prinudjeni da saljete cele .exe fajlove a ne male patchere na internet. Zato je najbolje koristiti ovaj prilicno 1337 (*elite*) nacin :) Otvorite program ...\Casovi\Cas8\EnableMe.exe pomocu Ollya. Predjite u Executable Modules Window (ALT + E) i tu pritisnite desno dugme na glavni exe fajl i izaberite View ALL Resources... Prozor koji ce se pojaviti odgovara onom na slici:

Address	Type	Name	Language	Size	Information
004606B8	CURSOR	1	0000 Language	00000134	
004607EC	CURSOR	2	0000 Language	00000134	
00460920	CURSOR	3	0000 Language	00000134	
00460A54	CURSOR	4	0000 Language	00000134	
00460B88	CURSOR	5	0000 Language	00000134	
00460CBC	CURSOR	6	0000 Language	00000134	
00460DF0	CURSOR	7	0000 Language	00000134	
00460F24	ICON	1	0409 English	000002E8	
0046120C	STRING	FF5	0000 Language	00000274	Menu '%s' is already being used by another form
00461480	STRING	FF6	0000 Language	000000FC	Enter
0046157C	STRING	FF7	0000 Language	000000F8	Information
00461674	STRING	FF8	0000 Language	00000358	Cannot change Visible in OnShow or OnHide
004619CC	STRING	FF9	0000 Language	000003E0	Icon image is not valid
00461DAC	STRING	FFA	0000 Language	00000390	Invalid property path
0046213C	STRING	FFB	0000 Language	000003D0	Friday
0046250C	STRING	FFC	0000 Language	000000F4	September
00462600	STRING	FFD	0000 Language	000000C4	May
004626C4	STRING	FFE	0000 Language	000002E0	Error creating variant array
004629A4	STRING	FFF	0000 Language	0000035C	Floating point underflow
00462D00	STRING	1000	0000 Language	000002B4	'%s' is not a valid integer value
00462FB4	RCDATA	DVCLAL	0000 Language	00000010	
00462FC4	RCDATA	PACKAGEINFO	0000 Language	000001D4	
00463198	RCDATA	TFORM1	0000 Language	000014EF	
00464688	GROUP_CURSOR	7FF9	0000 Language	00000014	
0046469C	GROUP_CURSOR	7FFA	0000 Language	00000014	
004646B0	GROUP_CURSOR	7FFB	0000 Language	00000014	
004646C4	GROUP_CURSOR	7FFC	0000 Language	00000014	
004646D8	GROUP_CURSOR	7FFD	0000 Language	00000014	
004646EC	GROUP_CURSOR	7FFE	0000 Language	00000014	
00464700	GROUP_CURSOR	7FFF	0000 Language	00000014	
00464714	GROUP_ICON	MAINICON	0409 English	00000014	

Od svih ovih resursa interesuje nas samo sadrzaj glavnog Forma, forma 1 i zato cemo klikutni desnim dugmetom na nju i izvrsicemo Dump kao na slici iznad. U novo otvorenom prozoru moramo da potrazimo objekte koje treba da ukljucimo. Imajte na umu da se ovo moze raditi i preko obicnog Hex editora samo ja preferiram da sve zavrsavam preko Ollya koji se u ovom slucaju ponasa bas kao obican Hex Editor. Ono sto cemo traziti u ovom delu koda kao obican binary string je rec Enabled jer je ona specificna za Delphi objekte i oznacava da li je neki objekat ukljucen ili ne. Pritisnite desno dugme -> Search for -> Binary string i u polje ASCII unesite string Enabled koji cemo traziti. Klikom na OK Olly ce nas odvesti do prvog pojavljivanja stringa Enabled to jest do adrese 00463C0E a to vidimo i na slici koja se nalazi ispod.



Ako pogledate malo iznad selektovanog teksta videcete na koji objekat se odnosi nadjeni string Enabled. Taj objekat je **TLabel.Label1** na kojem pise sledeci tekst ***Enable this label two***, a kao sto smo videli pri startovanju mete ovaj tekst je iskljucen i treba ga ukljuciti. Kako ovo radimo ??? Selektovacemo prvi broj koji se nalazi iza nadjenog teksta Enabled (*prvu tacku ako posmatrate tekst Enabled a ne hex vrednosti*), dakle selektovacemo 08 i pritisnucemo desno dugme -> binary -> edit ili samo CTRL + E. Ovaj hex broj 08 treba da promenimo u 09 da bi smo promenili stanje ovog objekta iz Enabled = False u Enabled = True. To treba da izgleda bas ovako:



Posle pritiska na OK dugme program ce patchovati selektovani bajt u 09 a mi cemo pritiskom na CTRL+L nastaviti potragu za iskljucenim objektima. Da li je objekat iskljucen ili ne mozete prepoznati jednostavnim pogledom na HEX broj iza stringa Enabled. Ako je taj broj 08 onda je objekat iskljucen a ako je broj 09 onda je objekat ukljucen. Izmenite sve iskljucene objekte u ukljucene u ovom primeru. Ako startujete program sa F9 videcete da su svi dugmici ukljuceni i da program pritiskom na dugme The Art Of Cracking – Enable Me govori da je uspesno crackovan. Jedini problem nastaje kada zelite da snimite ove promene. Iz nekog razloga ovako izmenjeni podaci se ne mogu snimiti. Ovo i nije neki problem, jednostavno zapisite adrese na kojima se nalaze 08 bajtovi i izmenite ih u CPU prozoru na isti nacin kao i ovde. Iz CPU prozora je dozvoljeno snimanje klikom desno dugme -> Copy to executable -> All modifications -> Copy All -> Save

ReEnable buttons – *Olly & VB*

Videli smo da Delphi ima svoj nacin zapisivanja resursa, a ono sto sledi iz toga je da i drugi programi mogu imati nestandardne nacine zapisivanja. Visual Basic je takodje jedan od programa sa nestandardim nacinom zapisivanja resursa. Postoje programi koji mogu da urade slicne stvari, npr. VBReformer, ali nacin koji cu vam ja ovde predstaviti je 1337 :)

Posto postupak pronalaska tacnog mesta u fajlu gde se to definise da li je dugme ili neki drugi objekat ukljucen ili iskljucen zahteva da imate instaliran Visual Basic alat, ja sam vec uradio uporedjivanje dva fajla sa iskljucenim i ukljucenim dugmicima i menijima. Prilikom toga dosao sam do zakljucka da ako imamo recimo meni koji treba ukljucimo treba da pronadjemo naziv menija u fajlu pri vrhu samog fajla, to jest prvi od vrha i da izmenimo treci bajt iza ovog stringa. Ako je meni ukljucen onda je taj bajt FF a ako je iskljucen onda je taj bajt 00. Evo kako to izgleda:

```
00401386  . 4C 65 76 65 6>ASCII "Level02",0
0040138E    05          DB 05
0040138F    00          DB 00 <- Ovo je treci bajt jer se racuna i nula iza Level02
00401390    FF          DB FF <- Taj bajt je 00 sto znaci da je meni iskljucen
```

Ista prica se ponavlja i sa dugmicima samo sto se pozicija menja. To jest kod dugmica nije u pitanju treci nego dvanaesti bajt. Evo kako to izgleda:

```
00401303  . 45 6E 61 62 6>ASCII "Enable Me  - Lev"
00401313  . 65 6C 30 31 0>ASCII "el01",0
00401318    04          DB 04
00401319    78          DB 78                       ; CHAR 'x'
0040131A    00          DB 00
0040131B    78          DB 78                       ; CHAR 'x'
0040131C    00          DB 00
0040131D    4F          DB 4F                       ; CHAR 'O'
0040131E    0B          DB 0B
0040131F    67          DB 67                       ; CHAR 'g'
00401320    02          DB 02
00401321    08          DB 08
00401322    00          DB 00 <- Ovo je dvanaesti bajt jer se racuna i nula iza Level012
00401323    11          DB 11 <- Taj bajt je 00 sto znaci da je dugme iskljuceno
00401324    00          DB 00
00401325    00          DB 00
00401326    FF          DB FF
00401327    03          DB 03
```

Naravno u pitanju je obicno kompajlovanje VB programa a ne PCODE. Ovo je zapazanje vezano samo za obicne VB programe, i vazi samo za dugmice i menije. Sto se tice ostalih elemenata koji se mogu naci u VB programima, to cete morati sami da ispitate ali ce logika (*FF je jednako true, a 00 false*) sigurno biti ispunjena. Objasnio sam vam kako treba da pronadjete bajtove koje treba da izmenite kako bi ukljucili menije i dugmice. Za vezbu uradite program ...\Casovi\Cas8\Crackme08.exe

Resenje:
0040138F promeniti 00 u FF
00401322 promeniti 00 u FF
004013C6 promeniti 00 u FF

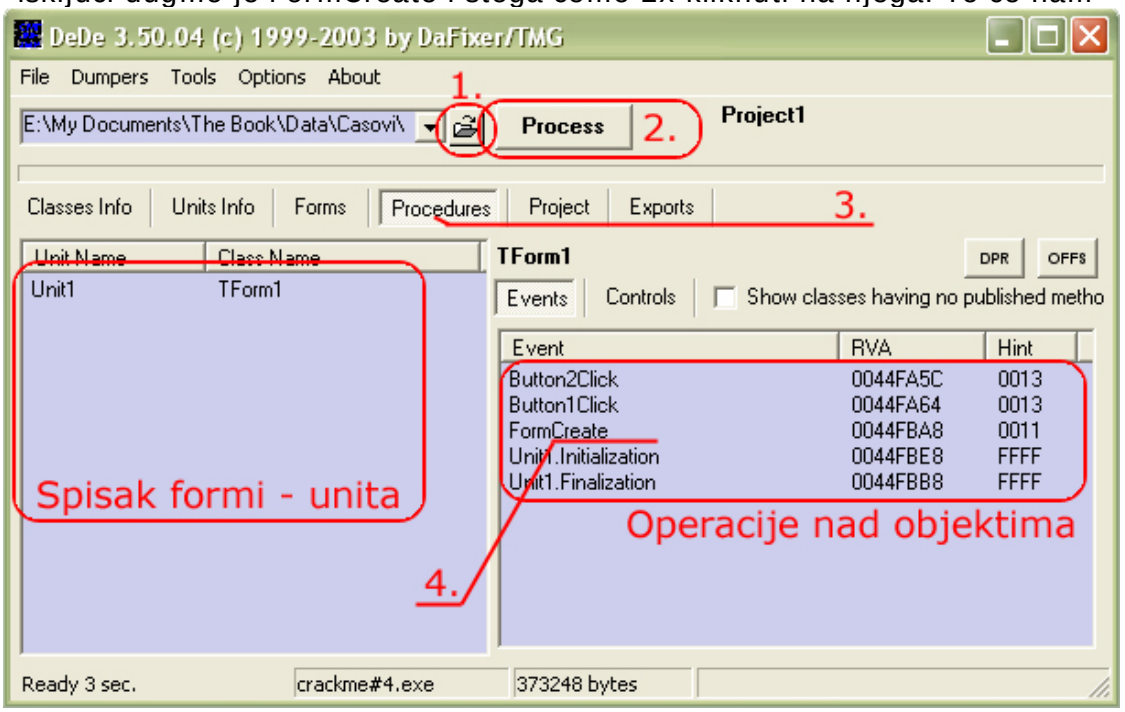
ReEnable buttons – *DeDe & Delphi*

Vec sam pomenuo da postoje specificni programi koji se bave analizom posebnih kompajlera. Jedan od takvih kompajlera je i Delphi a specializovani program koji se bavi samo Delphiem i njegovim podverzijama je DeDe, program koji je napisao DaFixer. Meta u kojoj je iskljuceno dugme preko neke procedure (*tipa Object.Enabled := False;*) se nalazi u folderu Cas12 a zove se crackme#4.exe i nju cemo izmeniti tako da dugme check bude uvek ukljuceno. Mozemo prvo da pogledamo resurse u programu da bi smo zakljucili kako je ovo dugme iskljuceno:

```
object Button1: TButton
  Left = 176
  Top = 9
  Width = 81
  Height = 24
  Caption = 'Check'
  TabOrder = 0
  OnClick = Button1Click
end
```

kao sto se vidi iz ResHackera dugme je stalno ukljuceno tako da se njegovo iskljucivanje sigurno radi preko procedure u obliku Button1.Enabled := False; Preko DeDe-a cemo pronaci gde se ovakva procedura izvrsava. Otvoricemo DeDe i ucitacemo metu u njega pomocu komande Open a njenu analizu cemo izvrsiti pomocu komande process.

Posle ovoga DeDe ce vas pitati da li da izvrsi standardno VCL prepoznavanje, na sta cete odgovoriti potvrdno. Kada se zavrsi ovo ispitivanje izaberite opciju procedures i u njoj cete videti spisak svih operacija nad objektima. Posto ih ima malo vidi se da jedini koji moze da iskljuci dugme je FormCreate i stoga cemo 2x kliknuti na njega. To ce nam



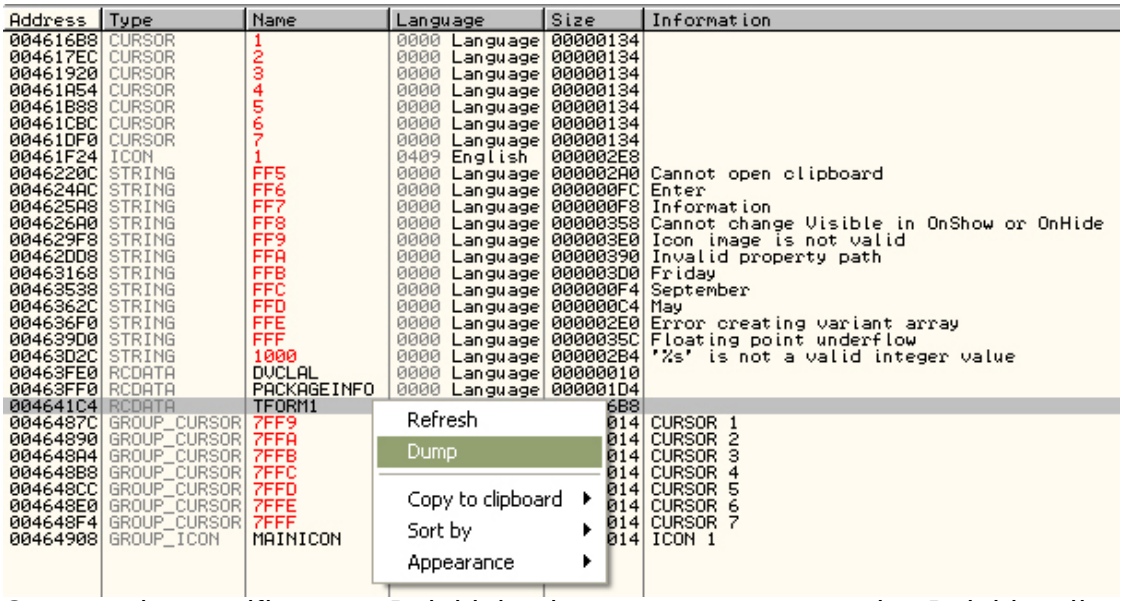
prikazati ovo

```
* Reference to method TButton.SetEnabled(Boolean)
0044FBB2  FF5164          call  dword ptr [ecx+$64]
0044FBB5  C3              ret
```

a ovo znaci da se CALL na adresi 0044FBB2 koristi za isklucivanje dugmeta. Da bi dugme stalno bilo ukljuceno potrebno je samo NOPovati ovaj CALL.

Passwords – *Olly & Delphi*

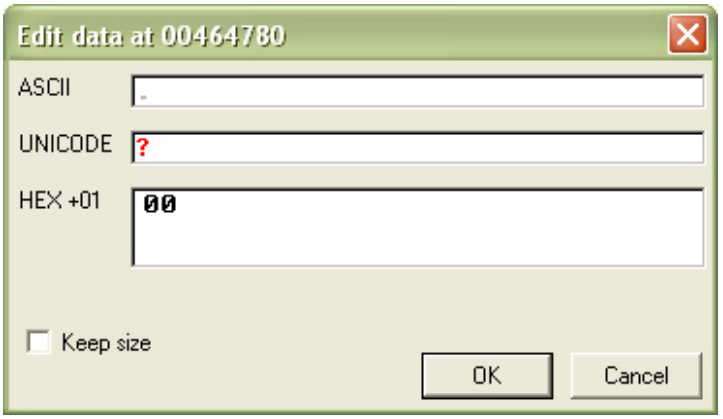
Videli smo da Delphi ima svoj nacin zapisivanja resursa, a ono sto nas sada interesuje je kako da provalimo u password polja i vidimo sta se nalazi zapisano unutra. Primer za ovo se nalazi u folderu Cas8 a zove se pwdMe.exe. Ovaj fajl cemo otvoriti pomocu Ollya. Vec smo naucili kako da pronadjemo resurse u Delphi exe fajlu pa to necu ponavljati. Dumpovacemo jedinu formu, TFORM1.



Ono sto je specificno za Delphi je da unutar resursa svako Delphi polje, odnosno Edit ima parametar PasswordChar koji oznacava koji je to karakter sa kojim ce biti zamenjeno svako uneto slovo u taj Edit. Ovaj karakter je najcesce zvezdica *. Stoga cemo pronaci PasswordChar u dumpu koji ce se pojaviti. Taj string cemo naci ovde:

00464772 50 61 73 73 77 6F 72 64 43 68 61 72 06 01 2A 08 PasswordChar..*.

Ako ste nekada koristili Delphi videli ste da je Default karakter za polje Edit ako ono nije Password tipa #0 ili 00 hex. Stoga cemo samo zameniti nasu zvezdicu 2A hex sa 00 i pretvoricemo polje za unos iz Password polja u obicno polje. Evo kako to izgleda posle patchovanja tog bajta:



Passwords – *Olly & VB*

Visual Basic kao uvek je prica za sebe pa ce i uklanjanje zvezdica sa password polja u VB programima biti prica za sebe. Meta za ovaj deo knjige se nalazi u folderu Cas8 a zove se pwdMe2.exe. Otvorite ovu metu pomocu Ollya. Sada pocinju muke, treba pronaci kako se zove forma u kojoj se nalazi, nepoznati textbox sa passwordom i kako se konacno sklanjaju zvezdice. Oticicemo u string reference i tamo cemo pronaci sto vise informacija o samoj meti. Videcemo koji se to objekti pojavljuju u exe fajlu. To su:

```
004013F8  DD pwdMe2.00401410          ASCII "Form1"
00401814  DD pwdMe2.00401474          ASCII "Form"
0040183C  DD pwdMe2.00401500          ASCII "Command1"
00401864  DD pwdMe2.0040151C          ASCII "Text2"
0040188C  DD pwdMe2.00401524          ASCII "Text1"
```

Znaci imamo samo jednu formu sa dva Text boxa i jednim dugmetom. Posto se objektima dodeljuju vrednosti na samom pocetku fajla kliknucemo na sadrzaj jednog od textboxova iz istog forma. Kliknucemo 2x na ovo:

```
Text strings referenced in pwdMe2:.text, item 7
 Address=00401297
 Disassembly=ASCII "CrackeR",0
```

jer ne znamo sta je sadrzaj password polja a ono se nalazi dovoljno blizu. Ovo iznad nam izgleda mnogo sumljivo:

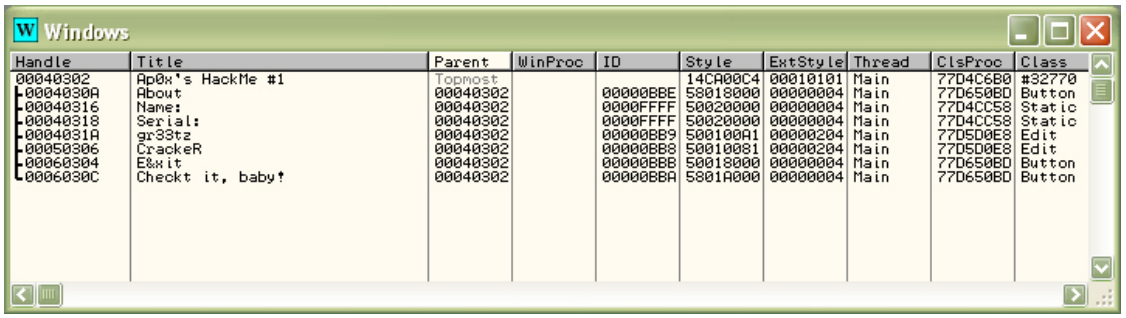
```
00401260  > \1D 010B0B00          SBB EAX,0B0B01
00401265  . 70 77                 JO SHORT pwdMe2.004012DE
00401267  . 64:55                 PUSH EBP                    ; Superfluous prefix
00401269  . 6E                    OUTS DX,BYTE PTR ES:[EDI]   ; I/O command
0040126A  . 48                    DEC EAX
0040126B  . 696465 4D 650>        IMUL ESP,DWORD PTR SS:[EBP+4D],1120065
00401273  . 0022                  ADD BYTE PTR DS:[EDX],AH
00401275  . 0100                  ADD DWORD PTR DS:[EAX],EAX
00401277  . 2A00                  SUB AL,BYTE PTR DS:[EAX]
00401279  . 2803                  SUB BYTE PTR DS:[EBX],AL
0040127B  . FF03                  INC DWORD PTR DS:[EBX]
0040127D  . 26:0000               ADD BYTE PTR ES:[EAX],AL
00401280  . 0001                  ADD BYTE PTR DS:[ECX],AL
00401282  . 05 00546578           ADD EAX,78655400
00401287  . 74 31                 JE SHORT pwdMe2.004012BA
00401289  . 0002                  ADD BYTE PTR DS:[EDX],AL
0040128B  . 04 78                 ADD AL,78
0040128D  . 00F0                  ADD AL,DH
0040128F  . 0007                  ADD BYTE PTR DS:[EDI],AL
```

Da li je ovo podatak koji nam treba ??? Ako pretrazimo ovaj kod nacicemo tacno jedan 2A bajt. Ovo bi u ASCIIu znacilo *, tako da to moze biti bas ono sto trazimo. Selektovacemo adresu 401277 i klikucemo na desno dugme -> Binary -> Edit. I izmenicemo ovaj bajt 2A u 00 vodeci se onim sto smo naucili od Delphia. Startocacemo program sa F9 i videcemo da u password polju sada pise: pwdUnHideMe. Dakle uspeli smo provalili smo u VB passworde. Naravno postoji laksi nacin da nadjemo ovaj bajt pomocu nekog Hex editora cemo samo potraziti 2A bajtove i pogledacemo gde se on nalazi. Ako se nalazi blizu nekog textboxa moguce je da je on bas taj bajt koji trazimo. Ako postoji vise ovakvih polja moracete malo da pogadjate na koji se textbox odnosi 2A bajt i da li on uopste pripada nekom textboxu.

Passwords – *Olly & ASM*

Za razliku od Delphia, ASM, C++ programi, imaju specijalan nacin zapisivanja to jest koriscenja ovog password polja. Ako otvorite neku metu sa ResHackerom videcete da polje ima atribut ES_PASSWORD, koji mozemo da uklonimo i polje ce postati vidljivo, to jest nece biti vise password tipa nego ce biti obicno. Ovo moze u nekim slucajevima da pomogne (*ako program koristi resurse i nema CRC32 proveru*) ali nacin koji cu vam ja sada pokazati vazi za svaki programski jezik i omoguca vam da bez ikakvih password unhider programa procitate sta se to nalazi unutar nekog password polja. Meta za ovaj deo knjige se nalazi u folderu Cas8 a zove se pwdMe3.exe, i nju cemo otvoriti pomocu Ollya.
Pokrenite nasu metu pomocu Ollya, to jest pritisnite F9. Pojavice se nasa meta u svom punom sjaju.

Treba da shvatimo da iako se preko nepoznatog teksta nalaze zvezdice on je jos tu i stoga mozemo saznati sta pise u tom polju. Imamo srece sto u samom Ollyu postoji opcija koja nam moze prikazati sve trenutno aktivne prozore debugovane aplikacije, kao i sadrzaj svakog polja za unos. Ova opcija se nalazi u toolbaru i obelezena je sa W. Kada pritisnemo to dugme videcemo sledece:



Vidimo da se u aktivnom prozoru nalaze samo dva Edit dugmeta i da su njihovi sadrzaji: CrackerR i gr33tz. Posto vidimo samo CrackeR znaci da je sadrzaj password polja gr33tz, i ovo je nas trazeni password. Ovo je zgodan i najbrzi nacin da pronadjete skriveni password u bilo kom programu. Izgleda da password polja i nisu bas toliko sigurna.

Time-Trial

Postoji veliki broj programa pisan u svim programskim jezicima cije je koriscenje ograniceno na jedan, a najcesce veoma kratak, vremenski period. Jedna takva aplikacija se nalazi u folderu Cas08 a zove se timetrial.exe.

Primeticete da se aplikacija moze koristiti samo tri dana ili se moze startovati samo nekoliko puta. Ovo cemo resiti tako da aplikacija nikada ne istekne. Upalite Olly i ucitajte ovu aplikaciju. Videcete niz API poziva: CreateFileA, MessageBoxA, ReadFileA, GetSystemTime.... Bez nekakve velike mudrosti vidi se da program otvara fajl DATA.DET, a ako on nepostoji prikazuje poruku o tome na ekran. Ako fajl postoji iz njega se cita 50 bajtova, posle cega se poziva API GetSystemTime koji vraca vreme na vasem kompjuteru. Procicemo kroz sve ove API pozive sa F8 sve dok se ne nadjemo ovde:

```
00401081  |.  803B 00                CMP BYTE PTR DS:[EBX],0
```

Primeticemo da se JNZ skok nece izvrsiti pa cemo sigurno doci u sledeci loop:

```
0040108B  |> /66:8B81 E4304>         /MOV AX,WORD PTR DS:[ECX+4030E4]
00401092  |. |66:35 6969              |XOR AX,6969
00401096  |. |66:8981 AB304>         |MOV WORD PTR DS:[ECX+4030AB],AX
0040109D  |. |83C1 02                |ADD ECX,2
004010A0  |> |83F9 08                |CMP ECX,8
004010A3  |.^\76 E6                  \JBE SHORT timetria.0040108B
```

Posto ja nemam zelju da prolazim kroz njega postavicu break-point na prvu adresu ispod loopa (004010A5) i pritisnucu F9. Ali objasnicu vam sta se ovde desava: Program jednostavno proverava kada je fajl snimljen, ako je to 2001 prva godina onda ce samo skinuti jedno otvaranje programa i update-ovace datum koji se nalazi na .DET fajlu. Naravno sve ovo ce biti snimljeno negde dole, ovde gore u loopu se samo proverava datum. Zbog ovoga ce se samo prvi put ici u ovaj loop, a svaki sledeci put ne. Posto nas ovaj prvi put i ne zanima onda cemo pritisnuti F9 da bi smo startovali program i updateovali datum .DET fajla. Primeticemo da nam je program smanjio broj startovanja programa za jedan. Restartovacemo Olly i icemo sa F8 kroz kod sve dok nedodjemo do:

```
00401084  |. /75 22                  JNZ SHORT timetria.004010A8
```

primetite da se sada ovaj skok izvrsava i da cemo se naci ovde:

```
004010A8  |> \8B0D AB304000          MOV ECX,DWORD PTR DS:[4030AB]
004010AE  |. 81F1 69696969           XOR ECX,69696969
004010B4  |. A1 E4304000             MOV EAX,DWORD PTR DS:[4030E4]
004010B9  |. 3BC1                    CMP EAX,ECX
004010BB  |. 0F85 85000000           JNZ timetria.00401146
004010C1  |. 66:8B0D B1304>          MOV CX,WORD PTR DS:[4030B1]
004010C8  |. 66:81F1 6969            XOR CX,6969
004010CD  |. 66:A1 EA30400>          MOV AX,WORD PTR DS:[4030EA]
004010D3  |. 66:2BC1                 SUB AX,CX
004010D6  |. 66:83F8 03              CMP AX,3
004010DA  |. 77 6A                   JA SHORT timetria.00401146
004010DC  |. 2805 00304000           SUB BYTE PTR DS:[403000],AL
004010E2  |>  A0 B5304000            MOV AL,BYTE PTR DS:[4030B5]
004010E7  |. 34 69                   XOR AL,69
004010E9  |. 3C 00                   CMP AL,0
```

Kao sto vidimo neke vrednosti se racunaju i porede. Ako bilo koji od ovih uslova nije ispunjen skace se ovde:

```
00401146  |> \6A 30                PUSH 30
00401148  |. 68 97304000           PUSH timetria.00403097; |Title = "Too Bad ;)"
0040114D  |. 68 76304000           PUSH timetria.00403076 ;|Text = "Sorry, this crackme"
00401152  |. 6A 00                 PUSH 0                 ; |hOwner = NULL
00401154  |. E8 A1000000           CALL <MessageBoxA>     ; \MessageBoxA
```

Naravno moguce je promeniti skokove tako da se ovo nikada ne izvrsi ali posto postoje programi koji proveravaju da li su im modifikovani odredjeni skokovi mi cemo naterati ovaj program da snimi .DET fajl tako da program nikada ne istekne. Prvo imamo ovu proveru:

```
0040 10A8 |> \8B0D AB304000        MOV ECX,DWORD PTR DS:[4030AB]
004010AE  |. 81F1 69696969         XOR ECX,69696969
004010B4  |. A1 E4304000           MOV EAX,DWORD PTR DS:[4030E4]
004010B9  |. 3BC1                  CMP EAX,ECX
004010BB  |. 0F85 85000000         JNZ timetria.00401146
```

Posto se porede EAX i ECX u ECX cemo staviti vrednost iz EAXa jer ce tako ova dva broja uvek biti jednaka. Promenicemo gornji kod u ovo:

```
004010A8  |> \8B0D E4304000        MOV ECX,DWORD PTR DS:[4030E4]
004010AE  |. 90                    NOP
004010AF  |. 90                    NOP
004010B0  |. 90                    NOP
004010B1  |. 90                    NOP
004010B2  |. 90                    NOP
004010B3  |. 90                    NOP
004010B4  |. A1 E4304000           MOV EAX,DWORD PTR DS:[4030E4]
004010B9  |. 3BC1                  CMP EAX,ECX
004010BB  |. 0F85 85000000         JNZ timetria.00401146
```

Primetite da sam program modifikovao tako da ce se u EAX i ECX smestati vrednosti sa iste adrese 004030E4. Primeticete i da sam obrisao (NOPovao) onu XOR ECX komandu, jer bi ona poremetila vrednost ECXa. Idemo na sledeci niz provera:

```
004010C1  |. 66:8B0D B1304>        MOV CX,WORD PTR DS:[4030B1]
004010C8  |. 66:81F1 6969          XOR CX,6969
004010CD  |. 66:A1 EA30400>        MOV AX,WORD PTR DS:[4030EA]
004010D3  |. 66:2BC1               SUB AX,CX
004010D6  |. 66:83F8 03            CMP AX,3
004010DA  |. 77 6A                 JA SHORT timetria.00401146
```

Ovde je ocigledno da se nesto racuna pa se AX poredi sa 3... Hmmm sa brojem dana koji nam je dozvoljen da koristimo program. Ovo cemo izmeniti u ovo:

```
004010C1  |. 66:8B0D EA304>        MOV CX,WORD PTR DS:[4030EA]
004010C8  |. 90                    NOP
004010C9  |. 90                    NOP
004010CA  |. 90                    NOP
004010CB  |. 90                    NOP
004010CC  |. 90                    NOP
004010CD  |. 66:A1 EA30400>        MOV AX,WORD PTR DS:[4030EA]
004010D3  |. 66:2BC1               SUB AX,CX
004010D6  |. 66:83F8 03            CMP AX,3
004010DA  |. 77 6A                 JA SHORT timetria.00401146
```

Primeticete da smo kao i gore CXu i AXu dodelili iste vrednosti, da smo izbrisali XOR. Ovo je zgodno jer ako pogledate skok ispod videcete da ce se on izvrsiti samo ako je AX vece od 3. Posto AX i CX imaju iste vrednosti posle oduzimanja SUB AX,CX vrednost koja ce se nalaziti u AX je 0, pa stoga nikada nece biti veca od 3. Idemo dalje i stizemo do poslednje provere:

```
004010DC  |. 2805 00304000         SUB BYTE PTR DS:[403000],AL
004010E2  |>  A0 B5304000          MOV AL,BYTE PTR DS:[4030B5]
004010E7  |. 34 69                 XOR AL,69
```

```
004010E9  |.  3C 00                 CMP AL,0
004010EB  |. /74 59                 JE SHORT timetria.00401146
```

A ovde se sigurno proverava da li je broj preostalih startovanja programa nula ili ne. Stoga cemo ovu proveru promeniti u ovo:

```
004010DC  |.  2805 00304000         SUB BYTE PTR DS:[403000],AL
004010E2  |>  B0 90                 MOV AL,90
004010E4  |.  90                    NOP
004010E5  |.  90                    NOP
004010E6  |.  90                    NOP
004010E7  |.  34 69                 XOR AL,69
004010E9  |.  3C 00                 CMP AL,0
004010EB  |.  74 59                 JE SHORT timetria.00401146
```

Posto se AL xoruje sa 69, jedina vrednost koja ce posle xorovanja dati nulu je ako se u AL nalazi bas 69. Dakle 69 xor 69 jednako 0. Ovo cemo resiti jednostavno tako sto cemo umesto MOV AL, BYTE PTR DS:[4030B5] ubaciti MOV AL,90. Broj 90 je proizvoljan i mozete uzete bilo koji broj koji je veci od 69. Naravno da smo mogli i ovde da obrisemo XOR ali nema potrebe, ovako je lakse. Ako pogledamo dole sta ce se izvrsiti ako se svi ovi skokovi ne izvrse, a nece posto smo ih modifikovali tako da nikada nece:

```
004010ED  |.  FEC8                 DEC AL
004010EF  |.  A2 01304000          MOV BYTE PTR DS:[403001],AL
004010F4  |.  34 69                XOR AL,69
004010F6  |.  A2 B5304000          MOV BYTE PTR DS:[4030B5],AL
004010FB  |.  6A 00                PUSH 0                          ; /Origin = FILE_BEGIN
004010FD  |.  6A 00                PUSH 0                          ; |pOffsetHi = NULL
004010FF  |.  6A 00                PUSH 0                          ; |OffsetLo = 0
00401101  |.  FF35 14314000        PUSH DWORD PTR DS:[403114]      ; |hFile = NULL
00401107  |.  E8 18010000  CALL <JMP.&KERNEL32.SetFilePointer>     ; \SetFilePointer
0040110C  |.  6A 00                PUSH 0                          ; /pOverlapped = NULL
0040110E  |.  68 E0304000          PUSH timetria.004030E0
00401113  |.  6A 0B                PUSH 0B                         ; |nBytesToWrite = B
00401115  |.  68 AB304000          PUSH timetria.004030AB          ; |Buffer = 004030AB
0040111A  |.  FF35 14314000        PUSH DWORD PTR DS:[403114]      ; |hFile = NULL
00401120  |.  E8 05010000          CALL <WriteFile>                ; \WriteFile
```

Videcemo da se u fajl DATA.DET snimaju novi podaci o vremenu i broju pokusaja. Odnosno samo o broju startovanja programa jer se poslednje vreme startovanja programa cita iz DATA.DET file-timea. Kada snimimo sve ove promene mozemo da startujemo crackme i videcemo sledece:

Mozemo restartovati ovaj crackme koliko puta zelimo, mozemo menjati datum a rezultat ce uvek biti isti: “Crackme zarobljen u vremenu :)”. Uspesno smo modifikovali program, a ako ste negde pogresili mozete iskoristiti backup fajl DATA.DET.bak da bi ste vezbali na ovom primeru.

09 Decrypt me...

Neke lake dekrpicije naizgled kompleksnih problema. Ovde cete nauciti kako da brzo, lako i jednostavno razbijete lake enkripcije. Ovo poglavlje takodje sadrzi i deo posvecen bruteforcerima. Taj deo poglavlja nije ni malo lak i moracete da se potrudite kako bi ste stekli kompletnu sliku vezanu za ovu vrstu programa, ali se nadam da cete sve razumeti.

Simple Encryption

Ako ste citali deo poglavlja sedam o CRC proverama primeticete da sam se dotakao jednostavne enkripcije koja se moze sresti u nekim programima. Kao primer sam tamo naveo enripciju koju koristi Trillian pa cemo mi upravo na ovom programu nauciti osnove kriptografije. Naravno ovo je obicno XORovanje i jedva se moze nazvati kriptografijom ali od neceg mora da se pocne. Pre nego sto se damo u resavanje problema osvrnucemo se malo ka istoriji, ka prvim primenama kriptografije. Dakle, prva (*meni poznata*) upotreba kriptografije se moze sresti za vreme vladavine Cezara. Kako bih zastitio poruke koje su nosili kuriri Cezar je primenjivao osnovnu kriptografiju. Kljuc za njeno resavanje je bio jednostavno pomeranje slova za tri. Dakle ako je slovo A onda bi ovako enkriptovano slovo bilo C, i tako dalje. Naravno pri dekripciji rimljani bi radili suprotan postupak, to jest pomerali slova po tri ali unazad. Ova mala prica o istoriji je tu da vam samo predoci osnovu kako to kriptografija radi. Bez obzira koji se algoritam koristi uvek morate imati tri osnovne jedinice: tekst koji se ekriptuje, kljuc za enkripciju i algoritam koji vrsi enkriptovanje uz pomoc kljuca. Algoritmi se mogu razlikovati i tako kao danas najpopularnije komplikovane algoritme poznajemo: RC4, RC5, AES, DES, BlowFish, Gost, TEA,.... Nemojte se zanositi da cete nauciti kako se razbijaju ove slozene ekripcije. Ove ekripcije se ne mogu razbiti bez odgovarajuceg kjuca. Da bi ste pronasli pravi kljuc preostaje vam samo da radite bruteforce ali to moze da potraje dosta, dosta dugo i na najjacim kompjuterima. Enkripcija koja se nalazi unutar jednog Trillian password fajla je veoma prosta i nju cemo razbiti u ovom poglavlju. Ako otvorimo msn.ini fajl koji se nalazi u Cas9 folderu videcemo ovo:

```
[msn]
auto reconnect=1
save passwords=1
idle time=15
show buddy status=1
port=1863
server=messenger.hotmail.com
last msn=ap0x@hotmail.com
connect num=10
connect sec=60
save status=1
ft port=6891
[profile 0]
name=ap0x@hotmail.com
password=C214B2F00CB0ECAA48
display name=Ap0x
auto connect=0
status=1
```

Ono sto nas interesuje je enkriptovan password. Ono sto ja znam a vi ne je koji sam ja to password uneo. Uneo sam 123456789 kao password sto ce nam omoguciti da provalimo nacin na koji se password 123456789 ekriptuje u C214B2F00CB0ECAA48. Ono sto moramo da primetimo je da je duzina neekriptovanog passworda 9 a enkriptovanog 18. Ovo moze da znaci da svakom broju iz neekriptovanog passworda odgovaraju dva iz enkriptovanog passworda. Ako rastavimo enkriptovani password na cinioce imacemo ovo:

```
C2 14 B2 F0 0C B0 EC AA 48
```

Dakle sada mozemo da uporedjujemo enkriptovane brojeve sa neenkriptovanim zakljucicemo da je ASCII od 1 XOR neki broj jednako C2 u hex formatu ili 194 u decimalnom (*za pretvaranje koristite windows calculator*). Dakle da bi smo otkrili koji je to broj sa kojim se ASCII horuje moramo da znamo: 1) ASCII vrednost svakog karaktera iz neenkriptovanog passworda i 2) decimalne vrednosti hex parova iz enkriptovanog passworda. Za prvi uslov pogledajte ASCII tablicu koja se nalazi na www.asciitable.com a za drugi koristite Windows Calculator. Jednacine koje mi moramo da resimo su:

49 XOR x = 194, x = ?
50 XOR y = 020, y = ?
51 XOR z = 178, z = ?
....

Ova jednacina se resava veoma lako. x = 49 XOR 194; y = 50 XOR 20; z = 51 XOR 178,.... Ovo je jednako posto je XOR funkcija revezibilna. Tablica sa Hex i x,y,z,... vrednostima:

ASCII	XOR	REZULTAT	HEX
49	243	194	C2
50	38	20	14
51	129	178	B2
52	196	240	F0
53	57	12	0C
54	134	176	B0
55	219	236	EC
56	146	170	AA
57	113	72	48

Kao sto se vidi iz tablice dobili smo vrednosti sa kojima se XORuju ASCIIi svih slova iz neenkriptovanog passworda. Koristeci ovu semu sada mozemo dekriptovati bilo koji password koji je Trillian enkriptovao ili mozemo cak napisati i program koji ce raditi dekripciju enkriptovanog passworda. Program koji dekriptuje Trillian password bi izgledao ovako:

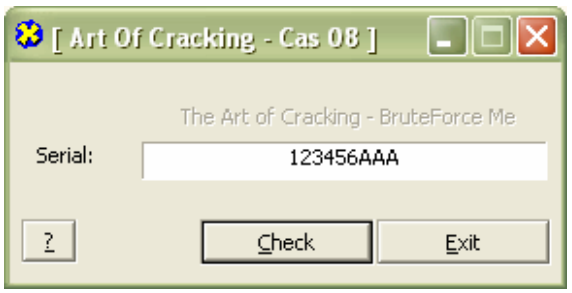
```
Dim table(1 To 9) As Integer
 table(1) = 243
 table(2) = 38
 table(3) = 129
 table(4) = 196
 table(5) = 57
 table(6) = 134
 table(7) = 219
 table(8) = 146
 table(9) = 113
 x = 0
 dec = ""
 enc = Text1.Text                'Ovo je prvi TextBox koji sadrzi enkriptovani password
                                 'U njega zalepite enkriptovani password bez razmaka
For i = 1 To Len(enc) Step 2
 x = x + 1
 dec = dec & Chr(Val("&H" + Mid$(enc, i, 2)) Xor table(x))
Next i
 Text2.Text = dec                'Ovo je drugi TextBox koji sluzi za ispisivanje dekriptovanog
                                 'passworda
```

Ovaj program mozete probati na drugom yahoo.ini fajlu koji se nalazi u istom direktorijumu.

Bruteforce

Pre nego sto sam napisao ovo poglavlje morao sam da napravim prvo algoritam koji bih morao da objasnim u ovom poglavlju. Od velikog broja mogucnosti odlucio sam se za onu najtezu za crackovanje, odlucio sam se za slucaj kada je jedini parametar od kojeg zavisi da li ce program biti uspesno crackovan sam seriski broj. Kao takav seriski broj se u ovom slucaju ne racuna uz pomoc nekog hardware IDa ili nekog unetog parametra. Jedini potreban i dovoljan parametar samom algoritmu je seriski broj. Pogodnost ovakve vrste algoritma je to sto postoji beskonacan broj tacnih resenja, a samo resenje necete naci u programu. Zbog ove cinjenice potrebno je napisati program koji ce pronaci sva ili samo neka moguca resenja.
Otvorite program BruteForceMe.exe koji se nalazi u folderu Cas9 pomocu Ollya. Namerno sam ubacio u program da on pokaze string Cracked OK kada ste pronasli tacan seriski broj. Ovo sam uradio da bih vama olaksao posao trazenja mesta gde se to racuna seriski broj. Videcete da CALL koji racuna seriski broj pocinje na adresi 00407D40. Analizirajte malo kod sami kako bih stekli sto vise informacija o algoritmu.

Sigurno sto primetili petlju koja pocinje na adresi 00407DE3. Postavicemo break-point na tu adresu i startovacemo program sa F9. U sam crackme unesite sledece. Zasto smo uneli bas ovo??? Veoma jednostavno ako pritisnete dugme? videcete da sam vam ostavio malu pomoc kako bi trebalo da izgleda seriski broj. Ovo nije bas 100% isti format zapisa ali redosled brojeva i slova nema veze. Tako sam napravio ovaj crackme. Ako pritisnemo dugme Check program ce zastati na adresi 00407DE3 i mi cemo se kretati kroz ovaj loop red po red sa F8 sve dok ne saznamo sta to ovaj loop radi. Evo tog loopa izvojenog i analiziranog:




```
MOV EAX,DWORD PTR SS:[EBP-C]          Prebaci u EAX uneti seriski broj
MOV DL,BYTE PTR DS:[EAX+EBX-1]        U DL stavi 1,2,3,... slovo seriskog
MOV EAX,EDX                           Prebaci ga u EAX, to jes u HEX
CMP AL,40                             Uporedi EAX sa 40 Hex
JBE SHORT BruteFor.00407E01           Ako je EAX <= 40 Hex onda preskoci...
MOV ECX,DWORD PTR SS:[EBP-C]          #
CMP AL,5B                             Uporedi HEX vrednost slova sa 5B
JNB SHORT BruteFor.00407E01           Ako nije manje ili jednako 5B preskoci..
MOV EAX,DWORD PTR SS:[EBP-C]          #
XOR EAX,EAX                           # Obrisi EAX
MOV AL,DL                             Prebaci slovo iz AL u DL
ADD DWORD PTR SS:[EBP-14],EAX         Saberi sadrzaj EBP-14 (pocetno 1) sa EAXom
MOV EAX,DWORD PTR SS:[EBP-C]          Prebaci u EAX ceo seriski broj...
MOV AL,BYTE PTR DS:[EAX+EBX-1]        Prebaci u AL slovo 1,2,3...
CMP AL,2F                             Uporedi AL sa 2F HEX...
JBE SHORT BruteFor.00407E2D           Ako je AL <= 2F Hex onda preskoci...
MOV EDX,DWORD PTR SS:[EBP-C]          #
CMP AL,3B                             Prebaci u AL slovo 1,2,3,4...
JNB SHORT BruteFor.00407E2D           Ako nije manje ili jednoko onda skoci na...
LEA EAX,DWORD PTR SS:[EBP-18]         #
MOV EDX,DWORD PTR SS:[EBP-C]          #
MOV DL,BYTE PTR DS:[EDX+EBX-1]        U DL prebaci slovo 1,2,3,...
CALL BruteFor.004037B8                #
MOV EDX,DWORD PTR SS:[EBP-18]         #
LEA EAX,DWORD PTR SS:[EBP-4]          Posle CALLa se brojevi dodaju jedan na drugi
CALL BruteFor.00403874                dole desno u HEX dump prozoru
INC EBX                               Povecaj brojac ciklusa
DEC ESI                               #
JNZ SHORT BruteFor.00407DE3           Loopuj
```

Sa leve strane se nalazi kod iz loopa a sa desne se nalaze objasnjenja ASM komandi iz loopa. Mislim da je svim jasno sta se desava u ovom loopu. Izdvajaju se brojevi i slova. Brojevi se dodaju jedan na drugi kao string a slova se pretvaraju u ASCII vrednosti i sabiraju (*add komanda*). Pocetna vrednost na koju se dodaju ASCII vrednosti je jedan.

Odmah posle ovog loopa na adresi 00407E31 pocinje novi segment koda. Mozete slobodno proci sa F8 kroz ovaj deo koda dok ne dodjete do adrese 00407F18. Izmedju 00407E31 i adrese 00407F18 se brojevi grupisu u parove. Znaci ako je uneti seriski broj 123456 onda ce postojati tri para brojeve i to su 12, 34, 56. Ako pogledamo ovaj deo koda odmah posle loopa:

```
00407E39  |. 83F8 06          CMP EAX,6
00407E3C  |. 0F85 1D010000    JNZ BruteFor.00407F5F
```

videcemo da se brojevna duzina (*123456*) poredi sa brojem 6. Ovo znaci da u samom seriskom broju moraju postojati 6 cifara. Mozemo i da analiziramo posebno sta se sve to desava izmedju 00407E31 i 00407F18 ali nema potrebe jer je mnogo lakse shvatiti ovaj algoritam na ovom delu koda:

```
00407F20  |. 8BC8             MOV ECX,EAX
00407F22  |. 8BC6             MOV EAX,ESI
00407F24  |. F7EB             IMUL EBX
00407F26  |. F7E9             IMUL ECX
00407F28  |. 99               CDQ
00407F29  |. F77D EC          IDIV DWORD PTR SS:[EBP-14]
00407F2C  |. 83FA 11          CMP EDX,11
00407F2F  |. 75 17            JNZ SHORT BruteFor.00407F48
```

Posle izvrsavanja komandi MOV ECX,EAX i MOV EAX,ESI videcemo sta se to nalazi u registrima EAX,ECX,ESI jer su nam ovi registri potrebni za IMUL komande ispod. Dakle posle izvrsavanja adrese 00407F22 imacemo u registrima sledece vrednosti:

```
EAX 0000000C                    <- Decimalno 12
ECX 00000038                    <- Decimalno 56
EDX 0012FC5C
EBX 00000022                    <- Decimalno 34
ESP 0012FC74
EBP 0012FCBC
ESI 0000000C
EDI 00860888 ASCII "123456AAA"
```

U tim registrima se nalaze oni parovi brojeva samo u hex obliku. Posle mnozenja, to jest posle izvrsavanja adrese 00407F28 imacemo u EAXu vrednost 5940h ili 22848 decimalno. A ako pogledamo proizvod 12 * 34 * 56 = 22848 i tako se dobija vrednost u EAXu. Na sledecoj adresi ovaj broj iz EAXa se deli sa vrednoscu adrese EBP-14 a ona je jednaka zbiru ASCII vrednosti svih slova iz imena plus jedan. Posle deljenja EAX ce biti jednak 74h ili 116 decimalno. Ovo nam govori da nije upitanju obicno deljenje nego je upitanju ostatak pri celebrojnom deljenju. Na sledecoj adresi se ovaj ostatak poredi sa 11h ili 17 decimalno. Ako je ostatak jednak 17 onda je seriski broj ispravan a ako je ostatak veci ili manji onda je seriski broj pogresan.

Posto postoji neogranicen broj tacnih seriskih brojeva mi moramo da napravimo program koji ce pronaci tacne vrednosti za uneta slova. Ova tehnika se moze poistovetiti sa butefoceingom. Algoritam cemo napraviti u VBu. Pre samog pravljenja ovog bruteforcera moramo da znamo sledece:

1) Da se seriski deli na tri dvocifrena broja i na slova
2) Da se tri dvocifrena broja medjusobno mnoze

3) Da se proizvod mnozenja deli sa zbirom ASCII vrednosti slova
4) Da se proverava da li je ostatak deljenja jednak 17

Na osnovu ove cetiri tacke mozemo da konstruisemo sledeci algoritam:

```
List1.Clear
 nic = 1                                    'Osnova za dodavanje je jedan
For i = 1 To Len(Text1.Text)
 nic = nic + Asc(Mid$(Text1.Text, i, 1))    'Dodavanje ASCII vrednosti slova
Next i
For y = 10 To 99                            'Posto se mnoze tri dvocifrena broja
 For x = 10 To 99                           'Moramo imati tri petlje od 10 do 99
  For i = 10 To 99
   If (i * x * y) Mod nic = 17 Then         'Proverimo da li je ostatak deljenja 17
    List1.AddItem x & i & y & Text1.Text    'Ako jeste seriski broj je ispravan
   End If
  Next i
 Next x
Next y
```

Imajte na umu da su ovo samo moguci seriski brojevi, sto znaci da nece svi generisani seriski brojevi pomocu ovog algoritma raditi. Ja sam vec napravio bruteforcer po ovom algoritmu i on se moze naci u istom folderu kao i sam crackme. Ako u bruteforcer unesete neki veliki string sa razmacima izmedju slova postoji velika verovatnoca da seriski broj nece raditi. Stoga se drzite unosenja do 4-5 slova, gde slova mogu samo velika.

Ovaj deo poglavlja je samo uvod u tematiku bruteforceovanja sto bi trebali da shvatite ovo kako bi ostatak ovog poglavlja lakse savladali. Iako je algoritam koji se crackuje lak za reversing on je sasvim dovoljan da shvatite kada se, zasto se i kako se to prave bruteforceri.

Bruteforce the encryption

Ovaj deo poglavlja spaja dve jako bitne celine u jednu. Ovde ce biti detaljno objasnjene tehnike koriscenja bruteforsinga u razbijanju enkripcija. Morate da shvatite da kada je upitanju reversing slozenih ili relativno slozenih enkripcija nije moguce naci pravi nacin za njihovo dekriptovanje kao sto je to bio slucaj u predhodnom delu poglavlja. Jedini nacin da se pronadje pravi password za enkriptovani fajl je da proverimo sve oblike koje seriski broj moze da ima i da dekriptujemo fajl sa njima. Ovo moze biti komplikovano ako pokusavamo da dekriptujemo ceo fajl, da bi smo ubrzali ovaj proces mi cemo dekriptovati samo prvih sest bajtova fajla. Ako su svih sest bajtova ispravni onda je velika verovatnoca da ce ceo fajl biti dekriptovan kako treba. Ja sam delimicno olaksao posao dekriptovanja tako sto sam koristio password duzine 6 karaktera, gde su svi karakteri velika slova. Iako se koriste samo velika slova broj kombinacija slova koje mogu predstavljati tacan seriski broj iznosi 244,140,625 seriskih brojeva duzine sest karaktera. Ovaj neverovatan broj postaje samo veci ako se koriste mala slova i brojevi zajedno sa velikim slovima. Naravno da nalazenje passworda rucno moze da potraje dosta dugo vremena pa se zbog toga pisu specijalizovane aplikacije koje nam olaksavaju posao proveravajucu sve moguce kombinacije slova i brojeva.

Za potrebe ovog poglavlja je napisan poseban algoritam koji encryptuje odredjeni fajl pomocu nekog passworda i nekog algoritma za enkriptovanje. Primer enkripcije se vidi na slici ispod. Naravno password sa kojim je enkriptovan fajl encrypted.exe nije AAAAAB nego neki drugi koji mi treba da otkrijemo. Pre samog pisanja decryptera-bruteforcera moracemo prvo da vidimo kako to algoritam za enkripciju radi da bi smo mogli na osnovu toga da napisemo odgovarajuci bruteforcer. Ne brinite algoritam nije komplikovan ali je dovoljno tezak da se fajl ne moze dekriptovati bez bruteforcera. Otvorimo program encryptFile.exe pomocu Ollya. Posto program izbaci string encrypting done!!! kada se zavrsi enkriptovanje mozemo lako da nadjemo CALL koji je zaduzen za enkripciju. Taj CALL pocinje na adresi 00453A64. Kao parametre u program unesite original.exe, new.exe i PSWORD. Postavite break-point na pocetak CALLa za racunanje i pritisnite Encrypt File dugme. Olly je zastao na break-pointu i sada treba da se krecemo kroz kod sa F8 kako bi videli sta se to desava u ovom CALLu. Sve ovo nam je totalno nebitno dok ne stignemo do ovde:


```
00453BD5  |. BB 01000000    MOV EBX,1
00453BDA  |. 8B07           MOV EAX,DWORD PTR DS:[EDI]
00453BDC  |. 3347 04        XOR EAX,DWORD PTR DS:[EDI+4]
00453BDF  |. 3347 08        XOR EAX,DWORD PTR DS:[EDI+8]
00453BE2  |. 3347 0C        XOR EAX,DWORD PTR DS:[EDI+C]
00453BE5  |. 3347 10        XOR EAX,DWORD PTR DS:[EDI+10]
00453BE8  |. 33C6           XOR EAX,ESI
00453BEA  |. 8BF0           MOV ESI,EAX
```

Posto znamo da se prilikom enkripcije koriste XOR komande ovaj deo koda mora biti vazan za ceo proces enkripcije. Primeticemo da su sadrzaji adresa EDI, EDI+4, EDI+8, EDI+C, EDI + 10, ESI Hex vrednosti ASCII kodova slova iz passworda kojim enkriptujemo fajl. Iz ovoga zakljucujemo da je:

EAX = Pass[1] xor Pass[2] xor Pass[3] xor Pass[4] xor Pass[5] xor Pass[6]

gde su Pass[1..6] ASCII vrednosti 1..6 slova iz passworda. Posle ovog xorovanja vrednost iz EAX se prebacuje u ESI. Ispod ovoga sledi petlja iz koje zakljucujemo sledece:

```
00453C01  |.  33C0            |XOR EAX,EAX
00453C03  |.  8A45 FF         |MOV AL,BYTE PTR SS:[EBP-1]
00453C06  |.  33C6            |XOR EAX,ESI
00453C08  |.  33449F FC       |XOR EAX,DWORD PTR DS:[EDI+EBX*4-4]
00453C0C  |.  43              |INC EBX
00453C0D  |.  83FB 06         |CMP EBX,6
00453C10  |. /7E 05           |JLE SHORT encryptF.00453C17
00453C12  |. |BB 01000000     |MOV EBX,1
```

- Na adresi 00453C03 se u AL smesta jedan po jedan karakter iz fajla original.exe (MZ....)
- Na sledecoj adresi se EAX xoruje sa vrednoscu iz ESIa
- Ispod ovoga se EAX horuje sa ASCII vrednoscu slova prolaza. Primera radi ako je ovo prvi prolaz od sest mogucih (jer toliko ima slova u passwordu) EAX ce biti xorovan sa ASCII vrednoscu prvog slova iz passworda.
- EBX se povecava za jedan i on predstavlja brojac prolaza
- Ako je prolaz veci od 6 prolaz (EBX) ce biti resetovan na 1
- Posle se EAX pretvara u ASCII slovo koje se zapisuje u novi fajl

Ovaj loop ce se ponavljati 4096 puta jer toliko bajtova ima originalni fajl. Kao sto se vidi enkripcija se vrsi na svakih 6 bajtova a onda se postupak ponavlja. To jest prvi bajt se xoruje sa ESIjem i sa ASCII vrednoscu prvog slova, drugi bajt sa drugim, a kada brojac slova predje sest, to jest dodje do sedam onda se on resetuje i krece ponovo od jedan. Tako ce sedmi bajt biti xorovan sa ESIjem i sa ASCII vrednosu prvog karaktera. Na osnovu detalja koje smo saznali mozemo napraviti deo algoritma za bruteforceing. Pre pisanja algoritma moramo da znamo unapred kako cemo ga napisati. Najjednostavniji nacin je da dekriptujemo samo 6 prvih bajtova iz originalnog fajla i da ih uporedimo sa bajtovima iz neenkriptovanog fajla. Zasto uzimamo prvih sest bajtova ??? Prosto posto se enkripcija ponavlja svakih sest bajtova, prvi i sedmi bajt ce biti enkriptovani na isti nacin.

Kada konstruisemo algoritam prvo moramo da definisemo tablicu slova koja ce biti koristena da se od njih napravi password. Ova tablica je velika 26 karaktera i sadrzi sva velika slova od A-Z. Posto imamo sest karaktera koji cine password moramo imati i sest petlji od 1 do 26 koje se ponavljaju jedna unutar druge. Zasto ovo radimo ??? Jer se kombinacije koje prave password krecu od AAAAAA pa do ZZZZZZ menjajuci pri tome zadnje slovo 26 puta, a kada zadnje slovo dodje do 27 onda se krece ispocetka i umesto 27 slova koristi se 1 slovo, slovo A, dok se drugo A od kraja povecava za jedan i postaje B. Tada ce kombinacija izgledati ovako AAAABA.

Ono sto mi moramo da proverimo je da li je xorovana vrednost svih slova iz passworda (ESI) xorovana slovom passworda (Pass[i]) koje se

odredjuje pomocu brojaca (I) i xorovana ASCII vrednoscu prvog bajta iz enkriptovanog fajla jednaka 4D hex ili M u ASCII obliku. Ako je ovo ispunjeno provericemo da li je drugi bajt jednak 5A hex ili Z u ASCIIu. Sta sve ovo znaci ??? Ovo znaci da cemo prepisati prvih 6 bajtova iz originalnog fajla (*4D5A90000300*) i da cemo jednacinu postaviti ovako:

Da li je 10h xor ESI xor ASCII(A) = 4D
Da li je 03h xor ESI xor ASCII(A) = 5A
Da li je C0h xor ESI xor ASCII(A) = 90
Da li je 4Dh xor ESI xor ASCII(A) = 00
Da li je 5Fh xor ESI xor ASCII(A) = 03
Da li je 5Ah xor ESI xor ASCII(A) = 00

gde je 10h prvi bajt iz enkriptovanog fajla (prvih 6 bajtova iz enkriptovanog fajla encrypted.exe *1003C04D5F5A*), ESI sadrzaj ESI registra, a ASCII(A) je ASCII vrednost prvog slova iz predpostavljenog passworda. Predpostavljeni password za prvi prolaz je AAAAAA, za drugi prolaz je AAAAAB, za treci je AAAAAC, itd... sve dok se ne dodje do ZZZZZZ. Ovo radimo ovako zato sto pokusavamo da dekriptujemo bajtove iz enkriptovanog fajla pomocu xorovanja enkriptovanog bajta sa vrednoscu ESIa i vrednoscu slova prolaza, ako je dobijena vrednost jednaka neenkriptovanoj onda moguce da je predpostavljeni password tacan. Da bi sve ovo bilo jasnije pogledajte sledecu tabelu:

Password	ESI	Prolaz	Slovo / ASCII	Enkriptovano	Rez.	Original
ABCDEF	7	1	A / 65 / 41h	10h / 16	86	77
ABCDEF	7	2	B / 66 / 42h	03h / 03	70	90
ABCDEF	7	3	C / 67 / 43h	C0h / 192	132	144
ABCDEF	7	4	D / 68 / 44h	4Dh / 77	14	0
ABCDEF	7	5	E / 69 / 45h	5Fh / 95	29	3
ABCDEF	7	6	F / 70 / 46h	5Ah / 90	27	0

- Kolona ESI se dobija ASCII(A) xor ASCII(B) xor ASCII(C) xor.... ASCII(F)
- Prolaz je brojac koji se koristi za odabiranje slova iz passworda
- Kolona enkriptovano sadrzi prvih 6 bajtova iz encrypted.exe fajla
- Rezultat je: kolona ESI xor ASCII iz istog reda xor Enkriptovano
  Primer: 7 xor 65 xor 16 = 86
- Posto 86 nije jednako originalnom bajtu 77 (4Dh) ovaj predpostavljeni password nije tacan pa se ostali prolazi ne moraju ni proveravati. Precicemo samo na sledeci password ABCDEG i pokusacemo sa njim.

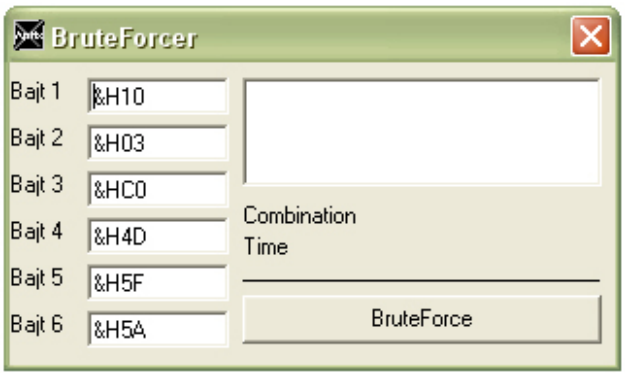
Sada znamo sve sto nam treba kako bi smo napisali algoritam u nekom programskom jeziku. Ono sto moramo da napravimo je da napisemo algoritam koji ce genirisati sve moguce passworde (od AAAAAA-ZZZZZZ) i proveravati da li je predpostavljen seriski broj tacan ili ne. Provera se vrsi bas onako kako je opisano u tablici iznad. Ja sam napravio jedan jako dobar primer u VB kako se to moze napisati jedan bruteforcer. Ovaj primer naravno nije optimizovan i posle par sekundi bruteforceovanja program ce prikazivati da je “zakucao” ali nije, on u pozadini radi svoj posao sto brze moze. Dajte mu vremena i on ce zavrsiti svoj posao, kad tad :) Ovo moze mnogo da

potraje ali bruteforceovanje je bas takvo, ipak postoji “samo” 244,140,625 kombinacija koje treba proveriti.

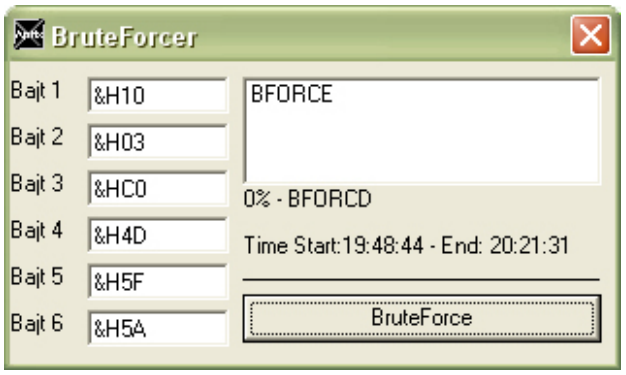
Algoritam:

```
Dim table(1 To 26) As String
For i = 65 To 90
 table(i - 64) = Chr(i)
Next i
 Label3.Caption = "Time Start: " & Time$
 Me.Refresh
For i1 = 1 To 26
 For i2 = 1 To 26
  For i3 = 1 To 26
   For i4 = 1 To 26
    For i5 = 1 To 26
     For i6 = 1 To 26
       sm = Asc(table(i1)) Xor Asc(table(i2)) Xor Asc(table(i3)) Xor Asc(table(i4)) Xor
Asc(table(i5)) Xor Asc(table(i6))
      my = (Val(Text1.Text) Xor sm Xor Asc(table(i1)))
     If my = 77 Then
      my = (Val(Text2.Text) Xor sm Xor Asc(table(i2)))
      If my = 90 Then
       my = (Val(Text3.Text) Xor sm Xor Asc(table(i3)))
       If my = 144 Then
        my = (Val(Text4.Text) Xor sm Xor Asc(table(i4)))
        If my = 0 Then
         my = (Val(Text5.Text) Xor sm Xor Asc(table(i5)))
        If my = 3 Then
         my = (Val(Text6.Text) Xor sm Xor Asc(table(i6)))
        If my = 0 Then
          List1.AddItem table(i1) & table(i2) & table(i3) & table(i4) & table(i5) & table(i6)
          GoTo 2
         End If
        End If
       End If
      End If
     End If
    End If
     Label2.Caption = table(i1) & table(i2) & table(i3) & table(i4) & table(i5) & table(i6)
     Form1.Caption = "BruteForcer - " & Label2.Caption
    Next i6
   Next i5
  Next i4
 Next i3
Next i2
Next i1
2
Form1.Caption = "BruteForcer"
Label3.Caption = Label3.Caption & " - End: " & Time$
```

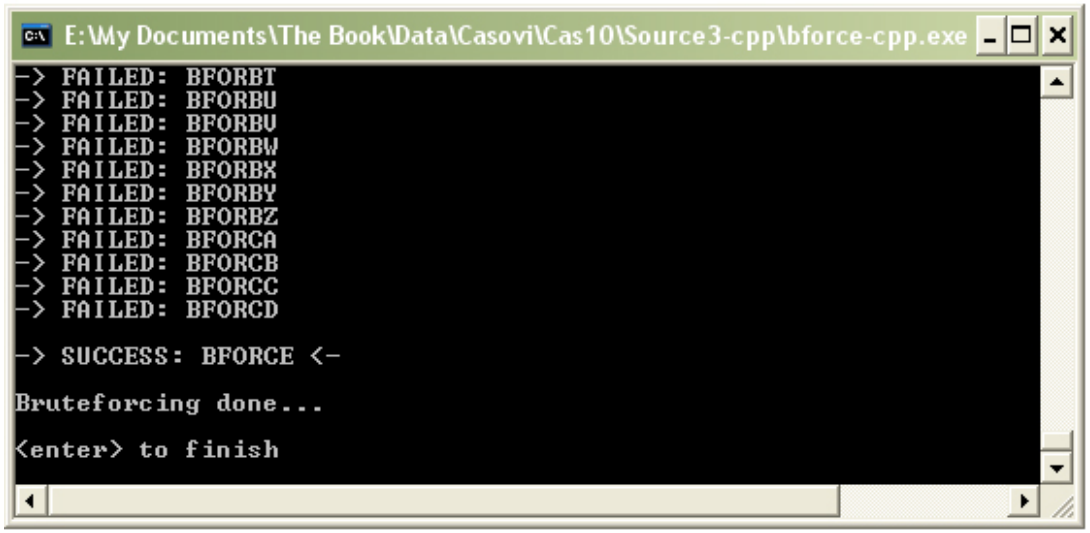
Objasnjenje:



Tekstualna polja od 1 do 6 se koriste za unosenje prvih sest bajtova iz enkriptovanog fajla. Ako koristite HEX vrednosti ne zaboravite da uh unosite kao &H pa tek onda HEX vrednost bajta. Pritiskom na dugme BruteForce algoritam od gore se izvrsava i krece se sa razbijanjem passworda. Source se nalazi u folderu ...\ Cas9\Source2\bforce



Slobodno startujete algoritam (*bforce-vb.exe*) da bi ste pronasli pravi password za enkriptovani fajl encrypted.exe... I tako posle 40 minuta na ekranu ce se pojaviti konacan bruteforce rezulatat. Password sa kojim je enkriptovan fajl encrypted.exe je BFORCE. Naravno ako fajl original.exe zelite da enkriptujete nekim drugim passwordom mozete po uciniti pomocu programa encryptFile.exe. Da bi bruteforceovali taj novi password morate da ispunite prvih sest polja sa vrednostima prvih sest bajtova iz novog enkriptovanog fajla. Kako bih se resili ovog VBu tipicnog zakucavanja napisacemo isti algoritam u C++, source tog algoritma se nalazi u folderu ...\Casovi\Cas9\Source3-cpp a kompajlovani bruteforcer se zove bforce-cpp.exe. Posle malo kraceg vremena i ovaj C++ bruteforcer pokazuje tacan rezultat:

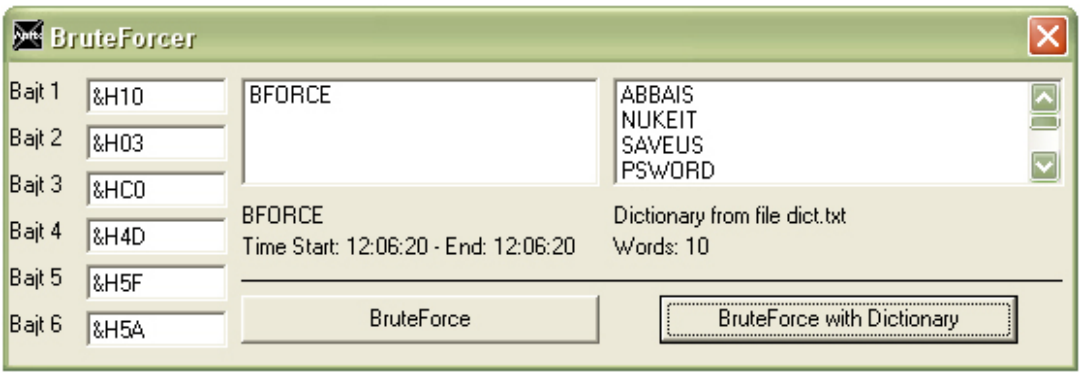


Konacno kad smo nasli tacan password startujmo encryptFile.exe i u njega redom unesimo encrypted.exe, new.exe, BFORCE ispunivshi sva tri polja sa po jednim parametrom. Kada se zavrsi dekriptovanje mozemo startovati new.exe koji predstavlja dekriptovani encrypted.exe fajl!!!

Ova tehnika se mozda na prvi pogled cini komplikovanom ali je ona u nekim slucajevima jedina koja se moze upotrebiti kada je upitanju crackovanje nekih passwordom zasticenjih fajla. Naime ova tehnika se mora koristiti kada je zasticeni fajl enkriptovan nekim passwordom. Ove slucajeve srecemo kod arhivera kao sto su WinZIP (od verzije 9.0 koristi AES enkripciju), WinRAR (AES-128 bit), WinACE i tako dalje... Postoje mnogo programi koji vec mogu da vrse bruteforce na sve standardne algoritme kao sto su AES, DES, RC4,... ali ako je upitanju neki specifican algoritam sami cete morati da napisete bruteforcer za njega.

Bruteforce with dictionary

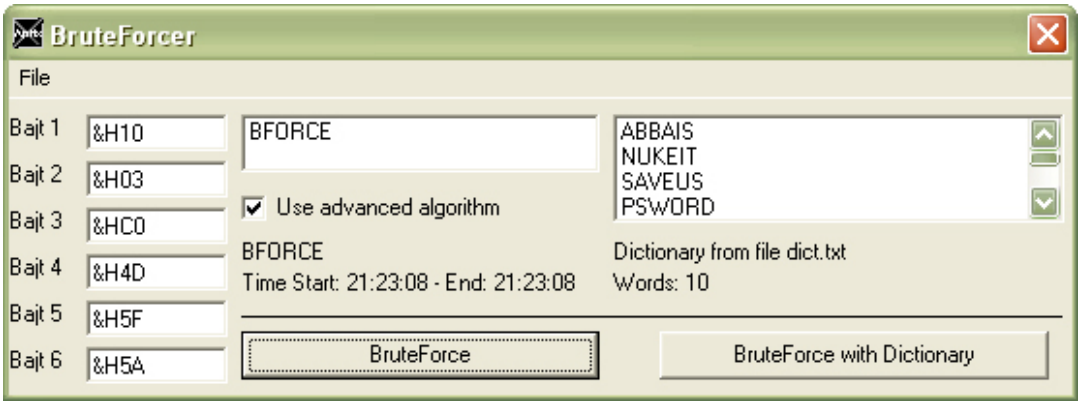
Ovo nije posebna vrsta bruteforce-ovanja nego je sam jedna njegova grana. Cilj ove tehnike je da se smanji vreme koje je potrebno da se pronadje tacan password. Naravno ova tehnika ima svoje prednosti i mane. Najbitnija prednost je smanjenje vremena koje je potrebno da se pronadje pravi password. Nazalost ova tehnika ima mnogo vise mana nego prednosti. Najveca mana je to sto moramo sami da napravimo recnik koji bi nas program koristio, a takodje je problem to sto mi ne mozemo znati sve moguce kombinacije koje korisnik moze upotrebiti kao password. Mozemo samo da predpostavimo neke najcesce koriscene passworde i da se nadamo da je korisnik koristio neki od njih. Primer upotrebe recnika u bruteforce-ovanju mozete naci ovde ...\Cas9\Source2\bforce with dict\



Kao sto vidite iskoristio sam obe opcije za bruteforceovanje i napravio jednu lepu i funkcionalnu celinu. Program ima dve mogucnosti: da vrsi klasicno bruteforceovanje ili da vrsi bruteforceovanje pomocu recnika. Primeticete da je vreme koje je potrebno da se pronadje pravi password svedeno na samo jednu sekundu, ali da je velicina samog recnika smanjena na samo deset mogucih kombinacija. Ovo znaci da brze ne znaci uvek i bolje, klasican nacin bruteforceovanja bi nasao password za veoma dug vremensi period ali bih ga sigurno nasao ako on ispunjava uslove algoritma koji se koristi za dekriptovanje, dok kod recnika to nije slucaj. Brze ili bolje odlucite sami...

Advanced bruteforceing

Prica o ovom bruteforce algoritmu nije gotova. Naime postoji jos nesto sto mozemo da uradimo kako bi smo ubrzali trazenje tacnog passworda. Za ovo nam nece trebati ni velike baze podataka kao recnici, ni selektivno trazenje u raznim ospezima. Jednostavno cemo iskoristiti matematiku samog algoritma tako da napisemo bruteforcer koji ce pronaci tacan password za samo jednu sekundu bez obzira na prvo slovo samog passworda. Ne dajte se plasiti matematikom kao nekim velikim zlom, shvatite to tako da cracking bez osnova matematike ne ide. Naravno za cracking vam nece trebati integrali, ili determinate ali ce vam trebati osnove kombinatorike i sistema verovatnoce. Ovaj deo poglavlja ce detaljno objasniti upotrebu matematike u pravljenu bruteforcera.



Kao sto znamo vec smo napravili alogritam koji ce sigurno naci pravi password za enkriptovani fajl samo sto ce broj mogucih kombinacija biti ogroman, pa ce samim tim i vreme za dekriptovanje biti dugacko. Recimo da zelimo da izbrojimo sve moguce kombinacije koje password moze imati. Za ovo ce nam biti potrebno malo poznavanje kombinatorike.
Dakle uslovi zadatka su da imamo sest slova od A do Z, i da se slova u passwordu mogu ponavljati. Posto ce prva kombinacija izgledati AAAAAA a zadnja ZZZZZZ, zakljucujemo da se na prvom mestu moze nalaziti bilo koje od 25 slova, isto tako i na drugom,... sestom. Znaci broj mogucih kombinacija koje moze imati password je:
25 * 25 * 25 * 25 * 25 * 25 = 25^6 = 244,140,625
Dakle broj mogucih kombinacija iznosi nesto preko 244 miliona. Naravno ovo smo mogli i do izbrojimo samo bi to potrajalo. Naravno mi smo napravili algoritam koji ce probati svako od ovih resenja i ispitati da li je tacno, ali ovo ce potrajati a mi zelimo da smanjimo vreme ispitivanja na sto je manje mogucu meru. Zbog ovoga cemo pokusati nasu srecu sa matematikom i sa algoritmom za enkriptovanje.
Algoritam za bruteforceing smo pisali jer tacan password zavisi od vise parametara. Tacan password zavisi od svakog slova passworda ali i od redosleda slova. Mozda se tokom citanja ovog poglavlja pomislili da iskoristimo istu tehniku kao kod dekrioptovanja Trillian passworda, ali onda

ste sigurno zaboravili na registar ESI. Da vas podsetim registar ESI se dobija tako sto se xoruju sve ASCII vrednosti iz predpostavljenog passworda. Zato nismo mogli da iskoristimo tehniku koju smo upotrebili za Trillian, jer se svaki bajt iz fajla enkriptuje ne samo sa slovom iz passworda nego i sa ESI vrednoscu. Naravno sama ESI vrednost se razlikuje za svaki password, ali ne i za svaki redosled slova :) Ovo znaci da ce ESI biti isti za AAABBB i za BBBAAA jer jer xor funkcija revezibilna to jest A xor A xor A xor B xor B xor B = 3 ali je i B xor B xor B xor A xor A xor A = 3 tako da cemo ovo iskoristiti.
Pogledajmo malo sta ta xor funkcija radi.

```
65 xor 65 = 0
0 xor 65 = 65
65 xor 66 = 3
3 xor 66 = 65
65 xor 66 = 3
```

Dakle iz ovoga zakljucujemo da ako imamo sest razlicitih slova bez obzira na to koje je slovo na kom mestu rezultat xorovanja to jest ESI se krece u vrednostima od 0 pa do 91. Zasto ??? Zato sto ako se na kraju xoruju dva ista slova onda ce rezultat biti minimalan to jest 0, a ako su slova najudaljenija jedna od drugog onda je maksimum 90 xor 1 to jest 91. Ovo je jako zanimljivo jer sada ne moramo racunati ESI jer znamo u kojim se on granicama krece. Naravno ovo moze da se sredi i preko integrala posto znamo granice kojima je odredjen ESI ali necemo to tako. Imamo kompjuter ispred sebe pa cemo ga maksimalno iskoristiti.
Moracemo samo malo da promenimo princip razmisljanja kako bi smo ubrzali onaj nas algoritam za bruteforceovanje. Dakle vec smo dosli do formule:
ESI xor ASCII(SLOVO1) xor BAJT1 = 4D
i tako dalje za svako sledece slovo i sledeci bajt. Ono sto cemo mi iskoristiti je cinjenica da ESI moze da ima vrednosti od 0 do 91. Zbog ovoga vise ne moramo da racunamo sam ESI nego cemo ga predpostaviti u intervalu od 0 do 100. Dakle dekriptovacemo samo prvo slovo po formuli:
ESI xor ASCII(SLOVO1) xor BAJT1 = 4D
gde je ESI od 0 do 100 a SLOVO1 je takodje predpostavljena vrednosti od A do Z. To bi u Visual Basicu izgledalo ovako:

```
For j = 0 To 100
 For i = 65 To 90
   chk = i Xor j Xor tab2(1)
  If chk = tab1(1) Then
     ???????????
  End If
 Next i
Next j
```

Ovim smo definisali kod sve do onih upitnika. Sada treba da napravimo algoritam koji ce dekriptovati i sva ostala slova iz passworda. Imajte na umu da smo vec odredili ESI kao promenjivu j i da se ona u predelu upitnika nece promeniti. Ostatak algoritma je jednostavan samo treba da napravite jos jednu petlju koja ide od A do Z i da proverite da li je j xor SLOVOx (x je broj od 2 do 6) xor BAJTx jednako originalnom BAJTUx. Ovaj primer sam ja vec iskodirao i nalazi se u folderu ...\Cas9\Source2\bforce with dict - optimized\ Mozete slobodno analizirati ovaj kod da bi ste stekli kompletnu sliku o ovom naprednom pisanju algoritma.

10 Unpacking...

Ovo veoma bitno poglavlje reversnog inzinjeringa se bavi razbijanjem osnovnih zastita koje mozete naci na netu. Vecina ovih zastita sluzi za sprecavanje modifikacije koda kod manji deo ovih programa sluzi za smanjenj velicine kompajlovanih exe i dll fajlova.

Unpacking anything...

Sta su to pakeri??? Pakeri su specificni programi koji se koriste u cilju zastite ili smanjenja velicine originalnih exe i dll fajlova. Ovakvi fajlovi se ne mogu direktno debugovati i / ili modifikovati. Ono sto se mora prvo uraditi da bi se moglo pristupiti klasicnom reversingu je odpakivanje. Postoji veliki broj komercijalnih, ali i besplatnih, zastita i pakera tako da se nas posao dodatno komplikuje jer je potrebno znati odpakovati sve ove vrste zastita. Naravno to nije moguce uciniti jer se odpakivanje programa zasticenih ovom vrstom packera razlikuje od verzije do verzije pakera pa ce u ovoj knjizi biti opisani samo nacini odpakivanja najpopularnijih zastita koje se mogu naci na internetu.

Odpakivanje svih vrsta zastita i pakera je u osnovi veoma lako i svodi se na samo tri bitna koraka:

1) **Ubijanje Anti-Debugging algoritama**
2) **Pronalazenje OEPa i dum povanje**
3) **Popravka importa i optimizacija fajla**

Analiziracemo svaki od ova tri koraka:

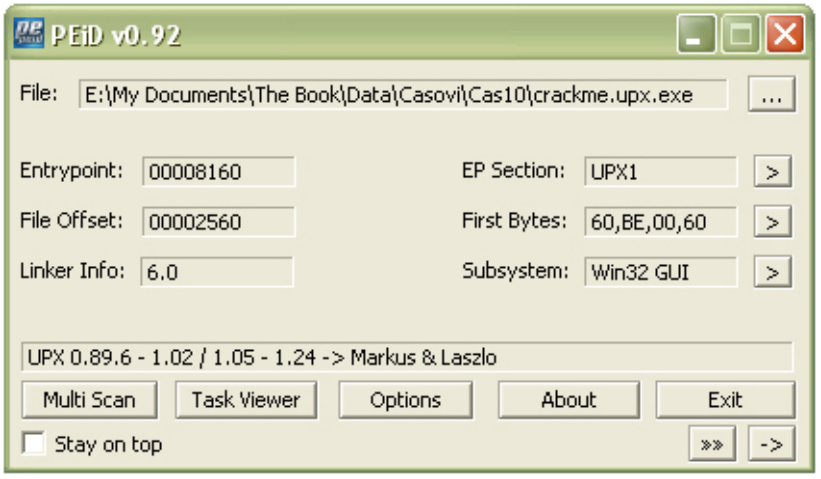
1) Prvi korak je vec detaljno objasnjen u poglavlju “Getting caught” i nema nikakve potrebe za dodatnim objasnjenjima u vezi detektovanja aktivanog debbugera
2) Sta je to OEP??? OEP je cesto koriscena skracenica za Original Entery Point sto je u prevodu prva linija koda koja ce se izvrsiti nakon ucitavanja fajla u memoriju. Sada shvatate zasto nam je ovo bitno. Ako je program pakovan i / ili zasticen nekom tehnikom onda ce se sve sem koda packera nalaziti kompresovano i / ili enkriptovano u fajlu. Nas cilj je da pustimo sam algoritam da se odpakuje u memoriju tako da ne krene sa izvrsavanjem zapakovanog programa. To jest da se linija koda koja se nalazi na OEPu ne izvrsi. Ovaj podatak nam je bitan jer mi pokusavamo da vratimo fajl u ono stanje kakvo je bilo pre pakovanja nekim packerom.
3) Sta su to importi??? Importi su tablice sa tacnim adresama do svih dll fajlova koje koristi neki program. Pri mapiranju ovih dll fajlova takodje se mapiraju i sve API funkcije i ostale export funkcije koje se nalaze u njima i koje program moze da poziva prilikom svog izvrsavanja.

Sada kada znamo sta su nam prioriteti prilikom svakog odpakivanja zasticenog fajla mozemo se posvetiti pojedinacnim packerima.
Za identifikovanje verzije i vrste pakera koristicemo program pod imenom PeID sa kojim ste se do sada, ako ste pazljivo citali ovu knjigu, sigurno morali sresti i upoznati.

UPX 0.89.6 - 1.02 / 1.05 - 1.24

Kao klasican primer packera, ali nikako i protektora, se navodi UPX. Ovaj odlican paker je najbolji i najlaksi primer kako se to radi unpacking zapakovanih aplikacija. Pre nego sto pocnemo sa objasnjenjem kako to packeri rade. Najjednostavnije je da razmisljate o pakovanim exe fajlovima kao da su u pitanju standardni SFX exe fajlovi. To jest razmisljajte o tome da imate arhivu, rar ili zip stvarno je nebitno, i da ste od nje napravili SFX exe fajl. Kada startujete SFX prvo ce se startovati kod SFXa koji ce odpakovati pakovani sadrzaj negde na disk. Isto se desava i sa exe pakerima kao sto je UPX samo sa razlikom sto se pakovani kod odpakuje direktno u memoriju a ne na hard disk.

Krenucemo sa odpakivanjem programa koji se nalazi u folderu Cas10 a zove se crackme.upx.exe. Prvo sto moramo uraditi je identifikacija pakera sa kojim je nasa meta pakovana. Stoga cemo pritisnuti desno dugme na taj fajl i selektovati Scan with PeID. Pojavice se prozor sa slike. Kao sto vidimo PeID



nam daje dosta jako korisnih detalja u vezi programa koji pokusavamo da odpakujemo. Sada znamo koja je glavna izvrsna sekcija (UPX1) znamo OEP (8160) znamo i sa kojom je to verzijom i kojim je pakerom zapakovana meta (UPX 0.89 - 1.24). Ove verzije packera oznacavaju da se princip odpakivanja nije promenio od verzije 0.89 i da bez ikakvih problema mozemo odpakovati bilo koju veziju UPXa primenjujuci isti metod. Otvoricemo metu pomocu Ollya i videcemo upozorenje da je upitanju zapakovani PE fajla pa da se zbog poga mora pazit gde i kako postavljamo break-pointe. Ignorisite ovo upozorenje posto se ovo kada su upitanju zapakovani fajlovi podrazumeva. Prve linije koda ili sam OEP izgledaju bas ovako:

```
00408160 > $  60              PUSHAD
00408161  .  BE 00604000      MOV ESI,crackme_.00406000
00408166  .  8DBE 00B0FFFF    LEA EDI,DWORD PTR DS:[ESI+FFFFB000]
0040816C  .  57               PUSH EDI
0040816D  .  83CD FF          OR EBP,FFFFFFFF
00408170  >  EB 10            JMP SHORT crackme_.00408182
```

I ovaj sablon je isti za svaku verziju UPXa po cemu se UPX i raspoznaje od drugih pakera. Odpakivanje UPXa je veoma lako, postoji veoma mnogo nacina da se odpakuje program pakovan program UPXom ali ja cu vas nauciti najlaksi i najbrzi nacin kako da dodjete do samog OEPa i tu uradite memory dump. Odpakivanje UPXovanih programa se svodi na jednostavno

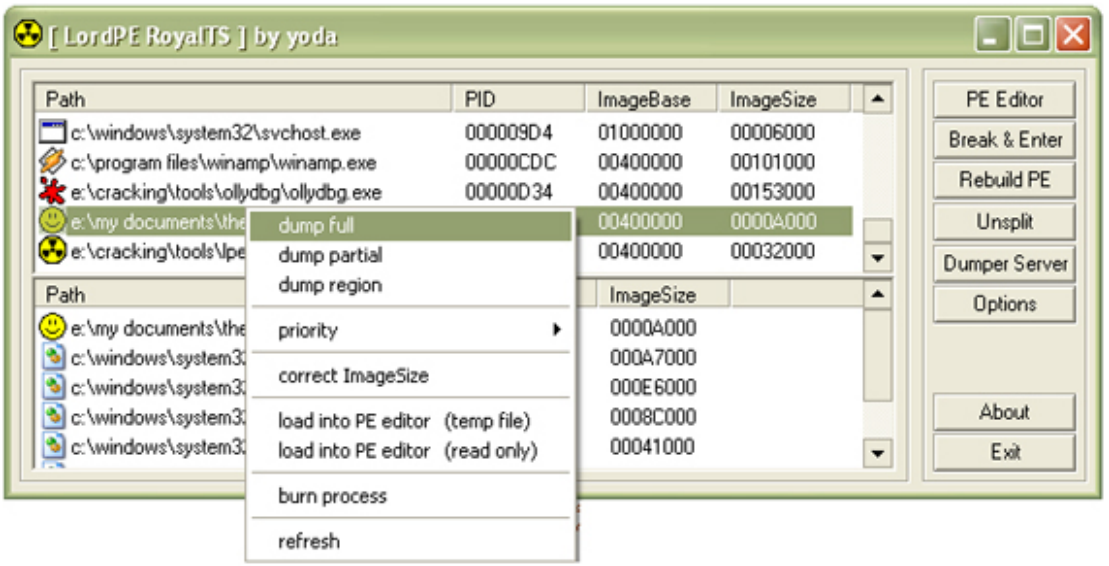
skrolovanje nanize dok ne dodjete do zadnje komande iz koje se nalaze samo neiskosceni 0x00 bajtovi. Ti par poslednjih redova izgledaju ovako:

```
004082A8  > \FF96 54850000 CALL DWORD PTR DS:[ESI+8554]
004082AE  > 61             POPAD
004082AF  .- E9 0C90FFFF   JMP crackme_.004012C0
```

Ako ste u nekim drugim tutorijalima citali kako treba traziti sledecu POPAD komandu i to je tacno isto jer posle POPAD komande nalazi se samo jos jedna komanda koja kada se izvrsi vodi nas pravo na OEP. Posto je upitanju bezuslovni skok (JMP) sama adresa ka kojoj on vodi je adresa do OEPa. Ovo znaci da se OEP nalazi na adresi 004012C0. Ako odete na adresu 004012C0 videcete da se tamo ne nalazi nista, to jest nalazi se samo gomila praznih 0x00 bajtova. Ovo se desava zato sto se algoritam za odpakivanje nije izvrsio odpakivanje zapakovanog dela fajla u memoriju i stoga originalni OEP jos ne postoji. Ono sto moramo da uradimo je da nateramo algoritam za odpakivanje da se izvrsi u podpunosti da bi smo onda mogli da uradimo dump memorije. Da bi smo ovo uradili postavicemo break-point na adresu na kojoj se nalazi jump koji vodi direktno do OEPa, to jest na adresu 004082AF. Sa F9 cemo startovati program i Olly ce zastati na adresi na kojoj smo postavili break-point. Potrebno je jos samo izvrsiti ovaj skok da bi smo se nasli na OEPu. Pritisnucemo F8 i nacicemo se na OEPu.

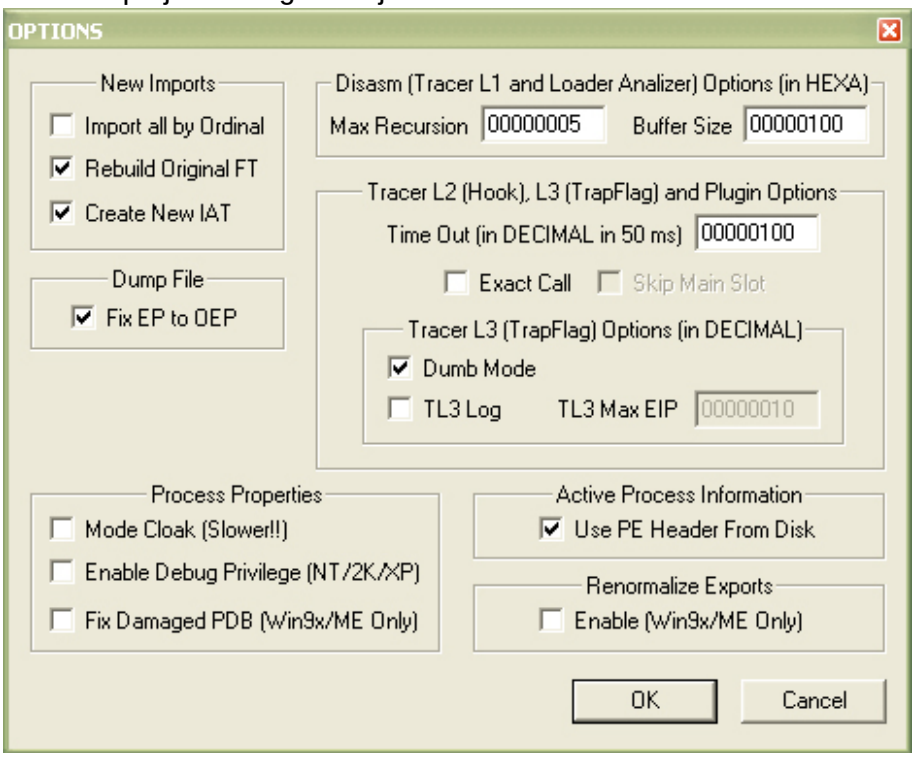
```
004012C0   55              PUSH EBP
004012C1   8BEC            MOV EBP,ESP
004012C3   6A FF           PUSH -1
004012C5   68 F8404000     PUSH crackme_.004040F8
004012CA   68 F41D4000     PUSH crackme_.00401DF4
004012CF   64:A1 00000000  MOV EAX,DWORD PTR FS:[0]
004012D5   50              PUSH EAX
```

Sada kada se nalazimo na OEPu treba da uradimo memory dump kako bi sacuvali memorisku sliku odpakovanog fajla na hard disk. Za ovo ce nam trebati LordPE. Startujte ga i iz liste procesa selektujte crackme.upx.exe fajl. Klikom na desno dugme na selektovani fajl izabracemo opciju dump full kao na slici.



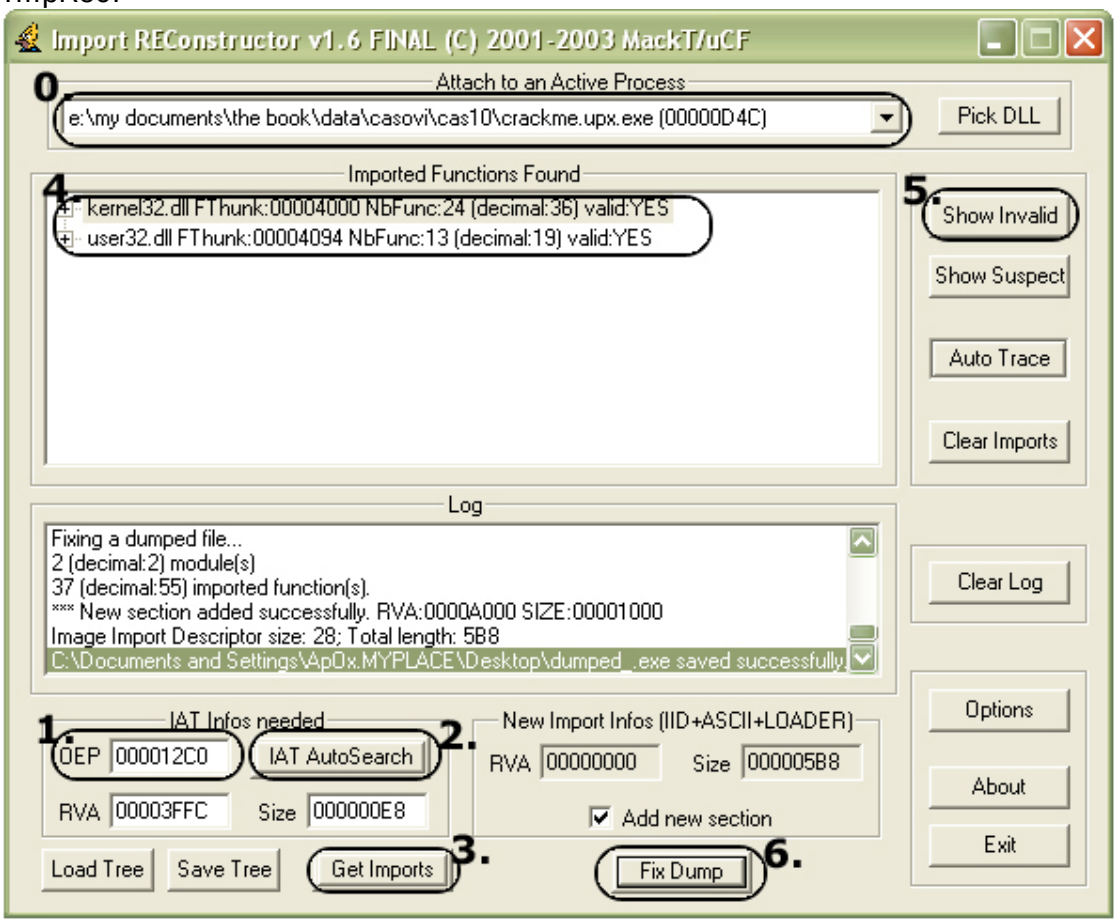
Kada smo snimili memorisku sliku nase mete bilo gde na disk mozemo da zatvorimo LordPE posto nam on nece vise trebati. Pitate zasto nisam koristio

Ollyev dump plugin ??? Odgovor je jednostavan: Sve vezije koje sam ikada testirao su imale neku gresku i 50% fajlova koje sam dumpovao imali su neku gresku, sto vise volim dobri stari full dump iz LordPEa pa preporucujem i vama da ga koristite.
Ako ste pomislili da je gotovo sa odpakivanjem prevarili ste se. Ostaje nam jos dosta posla kako bi smo naterali metu da se startuje. Zapamtili smo adresu na kojoj se nalazi pravi OEP: 004012C0, ona ce nam trebati kako bi smo ispravili OEP na pravu vrednost. Ali pre nego sto ispravimo OEP moracemo da popravimo importe posto je UPX ostavio samo par importa koji su njemu potrebni kako bi odpakovao zapakovani kod u memoriju. Da bi smo popravili import treba nam jedan drugi alat, treba na ImportReconstructer. Ne gaseci Olly i ne pomerajuci se linije OEPa startujete ImportREconstructer ili ImpRec. Pre upotrebe ImpReca moramo ga prvo pravilno konfigurisati. Podesite vase opcije da odgovaraju slici:



Kao sto primecujete podesio sam ImpRec tako da on sam prilikom popravljanja importa popravi i OEP sto ce nam samo sacuvati vreme odlaska u neki PE editor kako bi smo popravili OEP. Kada smo konfigurisali ImpRec selektovacemo iz liste aktivnih procesa nas crackme.upx.exe fajl i sacekacemo da se proces ucita. Kada se ovo desi videcemo da je vrednost OEPa ona stara 00008160 pa cemo je promeniti na novu kako bi ImpRec pronasao sve importe. Umesto stare vrednosti OEPa unecemo novu, unecemo 004012C0 - ImageBase (00400000) = 000012C0. Obratite paznju na ovo, a to je da se image base mora oduzeti od RVA vrednosti originalnog OEPa kako bi ImpRec trazio importe na pravom mestu. Kada ovo uradimo pritisnucemo dugme IATAutoSearch pa dugme GetImports. U gornjem delu

prozora ce se pojaviti svi .dll fajlovi koje ovaj crackme koristi i kao grane istih ce se pojaviti odgovarajuci API pozivi. Provericemo da li su svi pozivi tacni i da li je ImpRec pronasao sve API pozive. Kliknucemo na dugme Show Invalid. Posto se nista nije pojavilo zakljucujemo da su svi importi ispravni. Sada nam ostaje samo pa popravimo importe u fajlu koji smo dumpovali predhodno. Kliknucemo na dugme Fix Dump i selektovacemo fajl koji smo dumpovali sa LordPEom. ImpRec ce napraviti novi fajl koji ce sadrzati popravljene importe. Na sledecoj slici je detaljno objasnjeno kako se koristi ImpRec:



Ono sto je bitno a nije naglaseno na ovoj slici je da checkbox Add new section mora biti selektovan kako bi se importi dodali u novu sekciju PE fajla a ne prepisali preko starih. Ako pogledamo sekcije koje sada ima ovaj novi popravljeni fajl videcemo sledece:

```
UPX0    | 00005000 | 00001000 | 00005000 | 00001000 | E0000080
UPX1    | 00003000 | 00006000 | 00003000 | 00006000 | E0000040
.rsrc   | 00001000 | 00009000 | 00001000 | 00009000 | C0000040
.mackt  | 00001000 | 0000A000 | 00001000 | 0000A000 | E0000060
```

Videcemo da je dodata sekcija .mackt koja sadrzi sve ispravne importe. Pitate se zasto sam posvetio cetiri strane ovako jednostavnom pakeru ??? Razlog je jednostavan. Odpakivanje svih vrsta packera se zasniva na istim postupcima pa ostale packere necu objasnjavati ovako detaljno, podrazumevacu da sve ove osnovne stvari vec znate

UPX-Scrambler RC1.x

Vec je objasnjeno kako se odpakuje UPX. UPX-Scrambler predstavlja samo malu modifikaciju pakerskog koda kako se on nebi lako prepoznao i jos lakse odpakovao. Ako izuzmemo ovu cinjenicu videcemo da se i ovaj skremblovani UPX veoma lako odpakuje. Meta se nalazi u folderu Cas10 a zove se crc32.upx-scrambler.exe. Zeleo bih da napomenem da sam ovu aplikaciju stavio kao primer samo zato sto nisam imao UPX-Scrambler da zapakujem neki fajl, nadam se da se Ghosthunter nece ljutiti. Inace aplikacija je jako korisna ako zelite da proverite da li se CRC32 nekog fajla promenio.

Otvorite ovu metu pomocu Ollya i videcete sledece na OEPu:

```
0041544F > $ 90                 NOP
00415450  > 61                  POPAD
00415451  . BE 00E04000         MOV ESI,crc32_up.0040E000
00415456  . 8DBE 0030FFFF      LEA EDI,DWORD PTR DS:[ESI+FFFF3000]
0041545C  . 57                  PUSH EDI
0041545D  . 83CD FF             OR EBP,FFFFFFFF
00415460  . EB 10               JMP SHORT crc32_up.00415472
00415462  . EB 00               JMP SHORT crc32_up.00415464
00415464  >^ EB EA              JMP SHORT crc32_up.00415450
00415466  .^ EB E8              JMP SHORT crc32_up.00415450
00415468  > 8A06                MOV AL,BYTE PTR DS:[ESI]
0041546A  . 46                  INC ESI
0041546B  . 8807                MOV BYTE PTR DS:[EDI],AL
```

Ovo ni malo ne podseca na UPX, ali slicnost postoji. Ako pogledate dole na sam kraj algoritma za odpakivanje videcete da se nalazi par komandi slicnih UPXu. Ti zadnji redovi izgledaju ovako:

```
0041559E  > \60                 PUSHAD
0041559F  .- E9 1C69FFFF        JMP crc32_up.0040BEC0
004155A4    BC554100            DD crc32_up.004155BC
004155A8    C4554100            DD crc32_up.004155C4
004155AC    CC                  INT3
004155AD    D4                  DB D4
004155AE    40                  DB 40                   ; CHAR '@'
004155AF    00                  DB 00
```

Primeticemo slicnost sa UPXom samo sto se u UPX pre zadnjeg skoka nalazi POPAD a ovde se nalazi PUSHAD komanda. Postavite break-point na JMP to jest na adresu 0041559F, pritisnite F9 da startujete program i program ce zastati na adresi 0041559F. Sada ostaje samo da pritisnemo F8 i da izvrsimo i ovaj skok kako bi smo se nasli na OEPu. Posle izvrsenja ovog skoka nalazimo se ovde:

```
0040BEC0   55                   PUSH EBP                ; USER32.77D40000
0040BEC1   8BEC                 MOV EBP,ESP
0040BEC3   83C4 F4              ADD ESP,-0C
0040BEC6   B8 38BE4000          MOV EAX,crc32_up.0040BE38
```

Ovo je pravi OEP i tu treba da uradimo memory dump. Posle dumpa treba da uradimo jos popravljanje importa sa ImpRecom i uspesno smo zavrsili sa odpakivanjem UPX-Scramblera.

UPX Protector 1.0x

Jos jedna u seriji modifikacija UPX kompresora. Meta koju cemo odpakivati je NFO builder a nalazi se ovde ...\Cas10\NFO-Builder.2000.rar kao sto vidite ovo je izuzetno koristan NFO editor stoga spajamo lepo i korisno. Mislim da mi momci iz FMWa nece zameriti sto se sluzim njihovom aplikacijom da objasnim odpakivanje UPX Protectora.

Otvorite metu pomocu Ollya i pogledajte sta se nalazi na OEPu. Kao sto primecujute OEP je malo neobican:

```
0044E91E  . FF00            INC DWORD PTR DS:[EAX]
0044E920  > 60              PUSHAD
0044E921  . BE 00204300     MOV ESI,NFO_Buil.00432000
0044E926  . 8DBE 00F0FCFF   LEA EDI,DWORD PTR DS:[ESI+FFFCF000]
0044E92C  . 57              PUSH EDI
0044E92D  . 83CD FF         OR EBP,FFFFFFFF
0044E930  . EB 10           JMP SHORT NFO_Buil.0044E942
0044E932 > $^ EB EC         JMP SHORT NFO_Buil.0044E920
```

ali bez obzira na sve i ovaj UPX “Protector” se odpakuje na isti nacin kako i klasicna UPX i kao UPX Scrambler. Mislim da je ovaj protector napisan samo zato da se aplikacije pakovane UPXom ne bi odpakivale preko -d opcije u UPXu. Za one koji to ne znaju UPXovana aplikacija se moze odpakovati preko samog pakera koji se moze preuzeti sa http://upx.sourceforge.net. Sve sto treba da uradite je da startujete upx.exe sa parametrom upx -d imefajla.exe i on ce se odpakovati za nekoliko sekundi. Naravno UPX nemoze da odpakuje programe “zasticene” UPX Protectorom i UPX Scramblerom. Ovo moramo da uradimo rucno.

Jednostavno cemo odskrolovati na sam kraj algoritma za odpakivanje i videcemo sledece:

```
0044EA71  > \61             POPAD
0044EA72  . C3              RET
0044EA73  > 61              POPAD
0044EA74  . EB 06           JMP SHORT NFO_Buil.0044EA7C
0044EA76    AD              DB AD
0044EA77    FA              DB FA
0044EA78    EA              DB EA
0044EA79    00              DB 00
0044EA7A  . 00EA            ADD DL,CH
0044EA7C  >- E9 2B52FCFF    JMP NFO_Buil.00413CAC
```

Videcemo da se ispod prve POPAD komande nalazi RET, sto nam je totalno nebitno, a ispod druge skok koji vodi na drugi skok 0044EA7C, koji dalje vodi na sam OEP. Vidimo da cak iako je ovaj algoritam malo promenjen osnove odpakivanja ostaju iste. Dakle sve sto treba da uradimo je da postavimo break-point na zadnju jump komandu, da pritisnemo F9 da bi smo dosli do samog break-pointa i da pritisnemo F8 1x kako bi smo se nasli na OEPu. I evo kako to OEP izgleda:

```
00413CAC   55               PUSH EBP
00413CAD   8BEC             MOV EBP,ESP
00413CAF   6A FF            PUSH -1
00413CB1   68 D8694100      PUSH NFO_Buil.004169D8
```

Posle ovoga ostaje samo da uradimo memory dump i popravku importa na isti nacin kao da ja upitanju obican UPX.

UPXShit 0.06

I poslednji paker u seriji UPX modifikacija za ovu knjigu (*obecavam :)*) je UPXShit autora snaker-a. Meta koja je pakovana ovom vrstom packera je sam PeID pa cemo odpakivati njega. Verzija PeIDa koju cu ja odpakivati je verzija 0.92. Ucitacemo metu u Olly i prve linije koda ce izgledati ovako:

```
0045E3E2 > $  B8 CCE34500    MOV EAX,PEiD.0045E3CC          <- Packer OEP
0045E3E7  .  B9 15000000     MOV ECX,15
0045E3EC  >  803408 7F       XOR BYTE PTR DS:[EAX+ECX],7F
0045E3F0  .^ E2 FA           LOOPD SHORT PEiD.0045E3EC
0045E3F2  .^ E9 D6FFFFFF     JMP PEiD.0045E3CD
```

Mozete probati da izvrsite sve ove linije koda sa F8, sto vam ja ne preporucujem jer LOOPD komanda oznacava loop koji ce se odigrati mnogo puta. Kao sto vidimo ovaj loop sluzi za xorovanje memorije bajtom 7F. Da ne bi smo izvrsavali LOOPD komandu postavicemo break-point na adresu 0045E3F2 i pritisnucemo F9 da dodjemo do njega. Sa F8 cemo izvrsiti taj skok i zavrsicemo ovde:

```
0045E3CD  > /B8 B7E34500    MOV EAX,PEiD.0045E3B7
0045E3D2   |B9              DB B9
0045E3D3   |15              DB 15
0045E3D4   |00              DB 00
0045E3D5   |00              DB 00
0045E3D6   |00              DB 00
0045E3D7   |80              DB 80
0045E3D8  .|34 08 7F        ASCII "4.."
```

Kao sto vidimo Olly nije stigao da analizira ovaj deo koda pa cemo ga naterati da to uradi pritiskom na CTRL+A. Posle analize taj deo koda sada izgleda ovako:

```
0045E3CD  > /B8 B7E34500   MOV EAX,PEiD.0045E3B7
0045E3D2  . |B9 15000000   MOV ECX,15
0045E3D7  > |803408 7F     XOR BYTE PTR DS:[EAX+ECX],7F
0045E3DB  .^|E2 FA         LOOPD SHORT PEiD.0045E3D7
0045E3DD  .^|E9 D6FFFFFF   JMP PEiD.0045E3B8
```

Ponovicemo isti postupak od malopre i postavicemo break-point na 0045E3DD, pa cemo pritisnuti F9 da dodjemo do bp-a a onda F8 da izvrsimo skok. I opet zavrsavamo na delu koda koji nije analiziran od strane Ollya. Posle pritiska na CTRL+A pojavljuje se ovo:

```
0045E3B8  > /B8 A2E34500   MOV EAX,PEiD.0045E3A2
0045E3BD  . |B9 15000000   MOV ECX,15
0045E3C2  > |803408 7F     XOR BYTE PTR DS:[EAX+ECX],7F
0045E3C6  .^|E2 FA         LOOPD SHORT PEiD.0045E3C2
0045E3C8  .^|E9 D6FFFFFF   JMP PEiD.0045E3A3
```

Ponovicemo isti postupak od malopre i postavicemo break-point na 0045E3C8, pa cemo pritisnuti F9 da dodjemo do bp-a a onda F8 da izvrsimo skok. I opet zavrsavamo na delu koda koji nije analiziran od strane Ollya. Posle pritiska na CTRL+A pojavljuje se ovo:

```
0045E3A3  ? B8 8DE34500   MOV EAX,PEiD.0045E38D
0045E3A8  . B9 15000000   MOV ECX,15
0045E3AD  >  803408 7F    XOR BYTE PTR DS:[EAX+ECX],7F
0045E3B1  .^ E2 FA        LOOPD SHORT PEiD.0045E3AD
0045E3B3  .^ E9 D6FFFFFF  JMP PEiD.0045E38E
```

Ponovicemo isti postupak od malopre i postavicemo break-point na 0045E3B3, pa cemo pritisnuti F9 da dodjemo do bp-a a onda F8 da izvrsimo

skok. I opet zavrsavamo na delu koda koji nije analiziran od strane Ollya. Posle pritiska na CTRL+A pojavljuje se ovo:

```
0045E38E  > /B8 78E34500    MOV EAX,PEiD.0045E378
0045E393  ? |B9 15000000    MOV ECX,15
0045E398  . |803408 7F      XOR BYTE PTR DS:[EAX+ECX],7F
0045E39C  ?^|E2 FA          LOOPD SHORT PEiD.0045E398
0045E39E  ?^|E9 D6FFFFFF    JMP PEiD.0045E379
```

Ponovicemo isti postupak od malopre i postavicemo break-point na 0045E39E, pa cemo pritisnuti F9 da dodjemo do bp-a a onda F8 da izvrsimo skok. I opet zavrsavamo na delu koda koji nije analiziran od strane Ollya. Posle pritiska na CTRL+A pojavljuje se ovo:

```
0045E379  > /B8 63E34500    MOV EAX,PEiD.0045E363
0045E37E  . |B9 15000000    MOV ECX,15
0045E383  > |803408 7F      XOR BYTE PTR DS:[EAX+ECX],7F
0045E387  .^|E2 FA          LOOPD SHORT PEiD.0045E383
0045E389  .^|E9 D6FFFFFF    JMP PEiD.0045E364
```

Ponovicemo isti postupak od malopre i postavicemo break-point na 0045E389, pa cemo pritisnuti F9 da dodjemo do bp-a a onda F8 da izvrsimo skok. I opet zavrsavamo na delu koda koji nije analiziran od strane Ollya. Posle pritiska na CTRL+A pojavljuje se ovo:

```
0045E364  > /B8 4EE34500    MOV EAX,PEiD.0045E34E
0045E369  . |B9 15000000    MOV ECX,15
0045E36E  > |803408 7F      XOR BYTE PTR DS:[EAX+ECX],7F
0045E372  .^|E2 FA          LOOPD SHORT PEiD.0045E36E
0045E374  .^|E9 D6FFFFFF    JMP PEiD.0045E34F
```

Ponovicemo isti postupak od malopre i postavicemo break-point na 0045E374, pa cemo pritisnuti F9 da dodjemo do bp-a a onda F8 da izvrsimo skok. I opet zavrsavamo na delu koda koji nije analiziran od strane Ollya. Posle pritiska na CTRL+A pojavljuje se ovo:

```
0045E34F  > /B8 39E34500    MOV EAX,PEiD.0045E339
0045E354  . |B9 15000000    MOV ECX,15
0045E359  > |803408 7F      XOR BYTE PTR DS:[EAX+ECX],7F
0045E35D  .^|E2 FA          LOOPD SHORT PEiD.0045E359
0045E35F  .^|E9 D6FFFFFF    JMP PEiD.0045E33A
```

Ponovicemo isti postupak od malopre i postavicemo break-point na 0045E35F, pa cemo pritisnuti F9 da dodjemo do bp-a a onda F8 da izvrsimo skok. I opet zavrsavamo na delu koda koji nije analiziran od strane Ollya. Posle pritiska na CTRL+A pojavljuje se ovo:

```
0045E33A  > /B8 24E34500    MOV EAX,PEiD.0045E324
0045E33F  . |B9 15000000    MOV ECX,15
0045E344  > |803408 7F      XOR BYTE PTR DS:[EAX+ECX],7F
0045E348  .^|E2 FA          LOOPD SHORT PEiD.0045E344
0045E34A  .^|E9 D6FFFFFF    JMP PEiD.0045E325
```

Ponovicemo isti postupak od malopre i postavicemo break-point na 0045E34A, pa cemo pritisnuti F9 da dodjemo do bp-a a onda F8 da izvrsimo skok. I opet zavrsavamo na delu koda koji nije analiziran od strane Ollya. Posle pritiska na CTRL+A pojavljuje se ovo:

```
0045E325  ? B8 0FE34500     MOV EAX,PEiD.0045E30F
0045E32A  . B9 15000000     MOV ECX,15
0045E32F  >  803408 7F      XOR BYTE PTR DS:[EAX+ECX],7F
0045E333  .^ E2 FA          LOOPD SHORT PEiD.0045E32F
0045E335  .^ E9 D6FFFFFF    JMP PEiD.0045E310
```

Ponovicemo isti postupak od malopre i postavicemo break-point na 0045E335, pa cemo pritisnuti F9 da dodjemo do bp-a a onda F8 da izvrsimo skok. I opet zavrsavamo na delu koda koji nije analiziran od strane Ollya. Posle pritiska na CTRL+A pojavljuje se ovo:

```
0045E310  > /B8 FAE24500    MOV EAX,PEiD.0045E2FA
0045E315  . |B9 15000000    MOV ECX,15
0045E31A  > |803408 7F      XOR BYTE PTR DS:[EAX+ECX],7F
0045E31E  .^|E2 FA          LOOPD SHORT PEiD.0045E31A
0045E320  .^|E9 D6FFFFFF    JMP PEiD.0045E2FB
```

Ponovicemo isti postupak od malopre i postavicemo break-point na 0045E320, pa cemo pritisnuti F9 da dodjemo do bp-a a onda F8 da izvrsimo skok. I opet zavrsavamo na delu koda koji nije analiziran od strane Ollya. Posle pritiska na CTRL+A pojavljuje se ovo:

```
0045E2FB  > /B8 E5E24500    MOV EAX,PEiD.0045E2E5
0045E300  . |B9 15000000    MOV ECX,15
0045E305  > |803408 7F      XOR BYTE PTR DS:[EAX+ECX],7F
0045E309  .^|E2 FA          LOOPD SHORT PEiD.0045E305
0045E30B  .^|E9 D6FFFFFF    JMP PEiD.0045E2E6
```

Ponovicemo isti postupak od malopre i postavicemo break-point na 0045E30B, pa cemo pritisnuti F9 da dodjemo do bp-a a onda F8 da izvrsimo skok. I opet zavrsavamo na delu koda koji nije analiziran od strane Ollya. Posle pritiska na CTRL+A pojavljuje se ovo:

```
0045E2E6  > /B8 D0E24500    MOV EAX,PEiD.0045E2D0
0045E2EB  . |B9 15000000    MOV ECX,15
0045E2F0  > |803408 7F      XOR BYTE PTR DS:[EAX+ECX],7F
0045E2F4  .^|E2 FA          LOOPD SHORT PEiD.0045E2F0
0045E2F6  .^|E9 D6FFFFFF     JMP PEiD.0045E2D1
```

Ponovicemo isti postupak od malopre i postavicemo break-point na 0045E2F6, pa cemo pritisnuti F9 da dodjemo do bp-a a onda F8 da izvrsimo skok. I opet zavrsavamo na delu koda koji nije analiziran od strane Ollya. Posle pritiska na CTRL+A pojavljuje se ovo:

```
0045E2D1  ? B8 5FE14500     MOV EAX,PEiD.0045E15F
0045E2D6  ? B9 71010000     MOV ECX,171
0045E2DB   >  803408 7F      XOR BYTE PTR DS:[EAX+ECX],7F
0045E2DF  .^ E2 FA          LOOPD SHORT PEiD.0045E2DB
0045E2E1  .^ E9 7AFEFFFF    JMP PEiD.0045E160
```

Ponovicemo isti postupak od malopre i postavicemo break-point na 0045E2E1, pa cemo pritisnuti F9 da dodjemo do bp-a a onda F8 da izvrsimo skok. I opet zavrsavamo na delu koda koji nije analiziran od strane Ollya. Posle pritiska na CTRL+A pojavljuje se ovo:

```
0045E160  > /60             PUSHAD
0045E161  . |BE 00D04300    MOV ESI,PEiD.0043D000
0045E166  . |8DBE 0040FCFF  LEA EDI,DWORD PTR DS:[ESI+FFFC4000]
0045E16C  . |57             PUSH EDI
0045E16D  . |83CD FF        OR EBP,FFFFFFFF
```

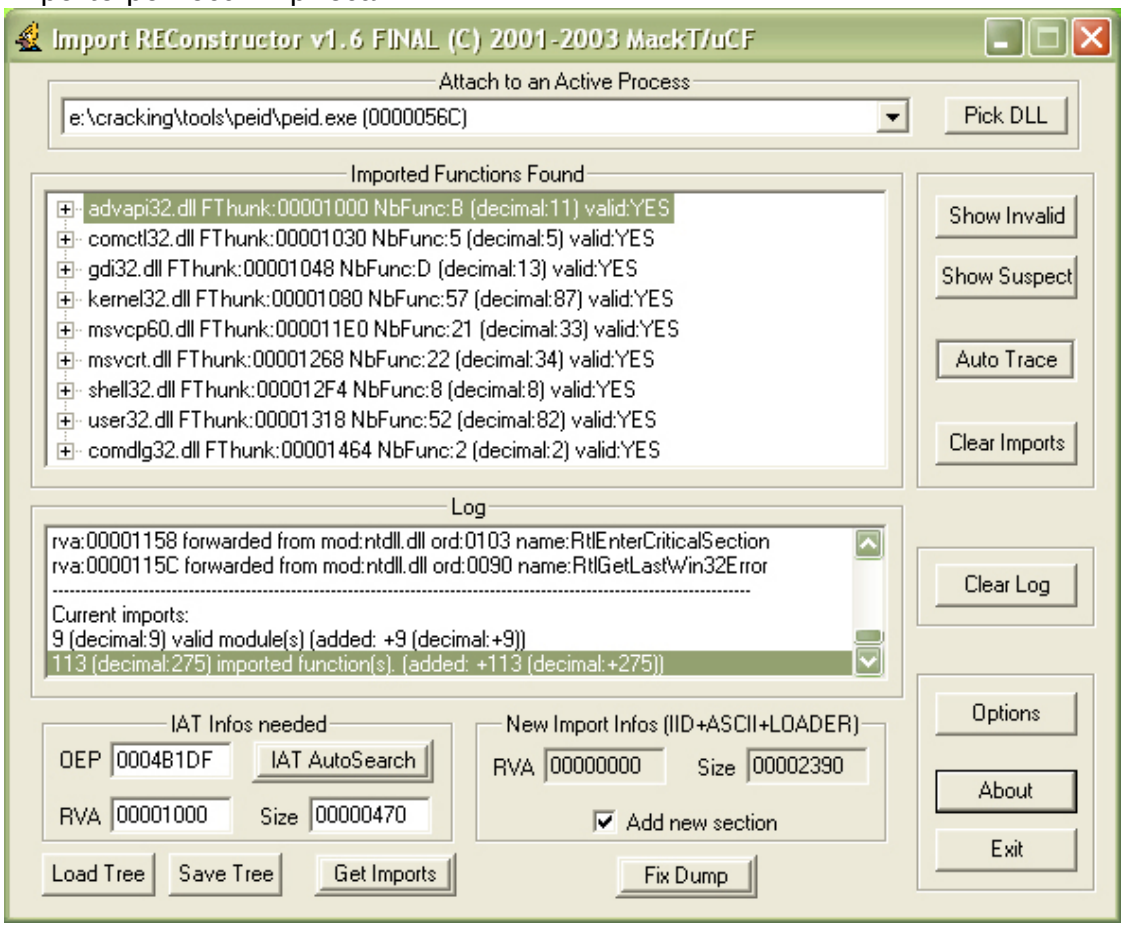
Izvinjavam se sto je malo vise od dve cele strane knjige copy-paste ali tako se odpakuje i zeleo sam da budem siguran da cete ispratiti sve ove jumpove kako treba. Ono sto sada vidimo nas podseca na originalni UPX kod, ali nije bas sasvim. Bez obzira na sve bas kao i obican UPX, UPXShit ima jedan obican JMP skok koji vodi pravo na OEP odmah ispod prve POPAD komande. Taj deo koda izgleda ovako:

```
0045E2BC  . /74 07          JE SHORT PEiD.0045E2C5
0045E2BE  . |8903            MOV DWORD PTR DS:[EBX],EAX
0045E2C0  . |83C3 04        ADD EBX,4
0045E2C3  .^|EB D8          JMP SHORT PEiD.0045E29D
0045E2C5  > \FF96 6CE90500 CALL DWORD PTR DS:[ESI+5E96C]
0045E2CB  >  61             POPAD
0045E2CC  .^ E9 0ECFFEFF    JMP PEiD.0044B1DF
```

pa cemo samo postaviti jedan break-point na 0045E2CC, pritisnucemo F9 da dodjemo do njega, i sa F8 dolazimo na OEP koji ce posle analize sa CTRL + A izgledati ovako:

```
0044B1DF  /> /55            PUSH EBP                <- Pravi OEP
0044B1E0  |. |8BEC           MOV EBP,ESP
0044B1E2  |. |6A FF          PUSH -1
0044B1E4  |. |68 702B4200    PUSH PEiD.00422B70
0044B1E9  |. |68 AAB14400    PUSH PEiD.0044B1AA
0044B1EE  |. |64:A1 0000000> MOV EAX,DWORD PTR FS:[0]
0044B1F4  |. |50             PUSH EAX
```

Sada nam ostaje samo da uradimo dump preko LordPEa i da popravimo importe pomocu ImpReca.



Ovo je najbolja modifikacija UPXa koju sam do sada video, stoga moram da priznam da je snaker odradio fenomenalan posao dodajuci enkripciju samog koda za odpakivanje UPX sekcije. Ideja je odlicna ali je odpakivanje bez obzira na sto je malo duze idalje lako izvodljivo.

FSG 1.33 - 2.0

FSG takodje sluzi samo za smanjenje velicine konacnog kompajlovanog exe ili dll fajla. On, kao ni UPX, u sebi ne sadrzi kod za zastitu od odpakivanja ili debuggovanja sa Ollyem ili bilo kojim drugim debuggerom. Pre nego sto pocnemo sa odpakivanjem mete pakovane sa FSGom objasnicu par stvari vezanih za kod koji odpakuje program u memoriju. Bez obzira na vrstu i verziju pakera sledeca stvar mora uvek biti ispunjena:

Posle izvrsavanja koda za odpakivanje program ce nekako skociti na virtualnu adresu odpakovanog koda. Ovo skakanje moze biti uradjeno samo na dva nacina, preko komande RET u slucaju da se kod za odpakuvanje ponasa kao CALL i preko komande JMP (i varijanti) kada se posle odpakivanja direktno skace na OEP.

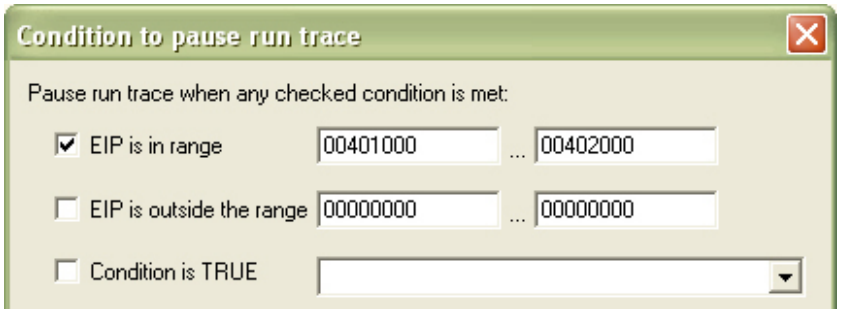
Ovo je najbitnije pravilo sto se tice odpakivanja zapakovanih programa i uvek mora biti ispunjeno sto cemo mi iskoristiti u nasu korist. Meta pakovana sa FSGom se nalazi u folderu Cas10 a zove se crackme.fsg.exe i nju cemo odpakovati uz pomoc Ollya. Kada otvorite ovu metu pomocu Ollya videcete sledece komande na mestu OEPa:

```
00405DC5 >  BE A4014000      MOV ESI,crackme_.004001A4
00405DCA    AD               LODS DWORD PTR DS:[ESI]
00405DCB    93               XCHG EAX,EBX
00405DCC    AD               LODS DWORD PTR DS:[ESI]
00405DCD    97               XCHG EAX,EDI
00405DCE    AD               LODS DWORD PTR DS:[ESI]
```

Ocigledno je da je ovo kod koji pripada pakeru i da to nije originalni nepakovani kod. Kao i kod UPX se na kraju koda za odpakivanje nalazi gomila 0x00 bajtova. Deo koda od packer OEPa pa do 00405E8C predstavlja deo koda koji sluzi direktno i samo za odpakivanje. U ovom delu koda cemo pokusati da pronadjemo skok koji bi vodio do adrese koja ne pripada ovom delu adrese, od 00405DC5 do 00405E8C. Svi skokovi pripadaju ovom delu adresa osim jednog koji se nalazi na adresi 00405E66.

```
00405E66  - 0F84 26B5FFFF   JE crackme_.00401392
```

Ovaj skok moze voditi direktno do OEPa. Ovo mozemo proveriti tako sto cemo staviti break-point na tu adresu i pusticemo je da se izvrsi. Posto je ce ovom malo duze trajati jer se ovaj skok neizvrsava vise puta ali se preko njega prelazi vise puta pa zbog toga nije pozeljno traziti OEP na ovaj nacin. Mi cemo iskoristiti cinjenicu da se u registru EIP uvek nalazi adresa koja ce sledeca izvrsiti. Dakle ako je EIP = 00401392 sledeca adresa koja ce se izvrsiti je 00401392 to jest pravi OEP. Za ovo cemo iskoristiti mogucnot koju nam pruza Olly: Tracing. Pritisnite CTRL+T, ispunite prozor kao na slici i pritisnite OK. Sada nam ostaje samo da porenemo Trace pritiskom na



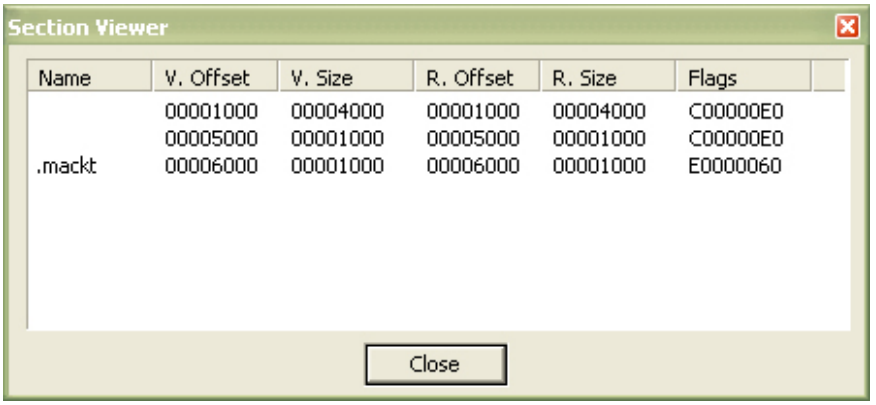
CTRL + 12 ili na Debug -> Trace over u gornjem meniju. Posle nekoliko sekundi Olly ce se zaustaviti ovde:

```
00401392    55          DB 55                    ; CHAR 'U'
00401393    8B          DB 8B
00401394    EC          DB EC
00401395    83          DB 83
00401396    EC          DB EC
00401397    44          DB 44                    ; CHAR 'D'
00401398    56          DB 56                    ; CHAR 'V'
00401399    FF          DB FF
0040139A    15          DB 15
0040139B    14          DB 14
```

Ovo sigurno ne izgleda kao OEP programa. Gde li smo to pogresili ??? Odgovor je: Nigde. Ovo je verovali ili ne pravi OEP je bas na adresi 00401392. Ovaj deo koda Olly nije stigao da analizira pa izgleda ovako kako izgleda. Pritisnite CTRL + A da bi ste analizirali kod i umesto gornjeg junka pojavice se pravi OEP.

```
00401392  /. 55          PUSH EBP                 ; USER32.77D40000
00401393  |. 8BEC        MOV EBP,ESP
00401395  |. 83EC 44     SUB ESP,44
00401398  |. 56          PUSH ESI
```

Ostaje nam samo da uradimo dump na adresi 00401392 i da popravimo importe sa ImpRecom kao sto je objasnjeno za UPX. Ono sto sam primetio prilikom odpakivanja FSGa je da on ne deklarise imena za svoje sekcije. Ovo i nije bitno jer PE fajl moze da radi sasvim ok i bez imena sekcija. Ovo se vidi u svim section viewerima. Jedan takav imamo u PeIDu a do njega se dolazi klikom na dugme > pored prve EP Sekcije. U slucaju FSGa vidimo ovo:



NAPOMENA:

Odpakivanje FSG verzija 1.3 do 2.0 se ne razlikuje u postupku pa odpakivanje aplikacije pakovane sa FSGom 2.0 necu prikazati u ovoj knjizi. Vi za vezbu mozete odpakovati fajl unpackme15.fsg20.exe.

ASPack 1.x - 2.x

Ovaj paker se nije mnogo menjao kroz svoje verzije kao ni UPX, dakle ako mozete da odpakujete ovaj primer mozete da odpakujete bilo koju verziju ASpacka. Odpakujemo fajl damn_contest.aspack.exe koji se nalazi u folderu Cas10. Za razliku od proslih pakera u ovom cemo koristiti Ollydump plugin jer ovde skracuje posao odpakivanja i popravljanja importa. Otvorimo Olly i ucitajmo damn_contest.exe u njega File -> Open. U CPU Windowu [ALT + C] vidimo prve redove fajla:

```
00411000 > 60              PUSHAD
00411001   E8 00000000     CALL damn_con.00411006
00411006   5D              POP EBP
00411007   81ED 0A4A4400   SUB EBP,444A0A                ; UNICODE "ymunds"
0041100D   BB 044A4400     MOV EBX,444A04
```

Ovo nije orginalni kod DAMN tryme-a nego je kod ASpacka koji se prvo izvrsava i sluzi za odpakivanje ASpackovanog fajla u memoriju. Pritisnite F8 da bi ste dosli do linije 00411001 odnosno da bi ste presli preko PUSHAD comande.

```
EAX 00000000
ECX 0006FFB0
EDX 7FFE0304
EBX 7FFDF000
ESP 0006FFA4            <- Crven
EBP 0006FFF0
ESI 00000000
EDI 00000000
EIP 00411001 damn_con.00411001
```

Pogledajmo FPU registere. Sve je isto samo se ESP promenio. Desno dugme na njega i klik na Follow in Dump. Dole u Hex dumpu ce se pojaviti ovo:

```
0006FFA4  00 00 00 00 00 00 00 00  ........
0006FFAC  F0 FF 06 00 C4 FF 06 00  ðÿ ..Äÿ ..
0006FFB4  00 F0 FD 7F 04 03 FE 7F  .ðý• þ•
0006FFBC  B0 FF 06 00 00 00 00 00  °ÿ ......
```

Sada selektujte prva cetiri para nula. Desno dugme na selektovano pa Breakpoint -> Hardware, on access -> Dword. Postavili smo breakpoint i program ce stati kad dodje do selektovane linije. Pritisnimo jedno F9 da startujemo program. Zavrshili smo ovde:

```
00411559  /75 08          JNZ SHORT damn_con.00411563      <- Ovde smo
0041155B  |B8 01000000    MOV EAX,1
00411560  |C2 0C00        RETN 0C
00411563  \50             PUSH EAX
00411564   C3             RETN
00411565   55             PUSH EBP
00411566   8BEC           MOV EBP,ESP
```

Pritisnite F8 2x i pustite RETN comandu da se izvrshi. Ona nas vodi direktno do prve linije programa pre pakovanja sa ASpackom t.j. vodi nas do OEPa. Vidimo nesto neobicno:

```
00401000    6A        DB 6A                    ; CHAR 'j'
00401001    00        DB 00
00401002    E8        DB E8
00401003    28        DB 28                    ; CHAR '('
```

itd itd, ali verujte mi na rec to je OEP, tu smo. Pritisnite CTRL + A da analizirate fajl i umesto ovoga gore pojavice se:

```
00401000  . 6A 00                PUSH 0
00401002  . E8 28040000          CALL damn_con.0040142F
00401007  . BF 0B234000          MOV EDI,damn_con.0040230B
```

```
0040100C  . 85C0                TEST EAX,EAX
0040100E  . 74 1E               JE SHORT damn_con.0040102E
00401010  . A3 95234000         MOV DWORD PTR DS:[402395],EAX
00401015  . 8307 01             ADD DWORD PTR DS:[EDI],1
00401018  . 33C0                XOR EAX,EAX
0040101A  . 50                  PUSH EAX                 ; /lParam => NULL
0040101B  . 68 45104000         PUSH damn_con.00401045
00401020  . 50                  PUSH EAX                 ; |hOwner => NULL
00401021  . 6A 73               PUSH 73                  ; |pTemplate = 73
00401023  . FF35 95234000       PUSH DWORD PTR DS:[402395]
00401029  . E8 B3030000         CALL damn_con.004013E1
0040102E  > 6A 00               PUSH 0                   ; /ExitCode = 0
00401030  . E8 F4030000         CALL damn_con.00401429   ; \ExitProcess
```

Sta sam vam rekao na OEPu smo. Sada preostaje samo da izvrshimo dump programa i to je to. Idemo na Plugins -> OllyDump -> Dump debuged proccess. Default podesavanja su OK pritisnite Dump i snimite fajl. Zatvorite Olly i probajte da startujete novi fajl i on radi !!!!!! Ok uspeli smo. Da bi smo proverili da li je sve OK otvorimo dumpovan fajl u Olly-u. Idemo na Desno dugme -> Search for -> All referenced strings. Vidimo ovo:

```
Text strings referenced in dmp:CODE
Address   Disassembly                     Text string
00401000  PUSH 0                          (Initial CPU selection)
0040108D  PUSH dmp.00402023               ASCII "-=[ ABOUT ]=-"
00401092  PUSH dmp.00402031               ASCII "You are just trying to solve DAMN's Official
joinig Contest. Made by tHE EGOiSTE/DAMN. At
first make a keygen for this simple keycheck
routine, then try to crack this program. The
LOCKED - Button should show an UNLOCKED-sign
and if "...
00401107  PUSH dmp.0040227D               ASCII "-=[ YEAH! ]=-"
0040110C  PUSH dmp.0040220E               ASCII "You got it! Thank you for registering!"
00401124  PUSH dmp.00402353               ASCII "                                      "
00401149  PUSH dmp.00402321               ASCII "                                      "
0040127E  PUSH dmp.00402317               ASCII "About"
004012CE  PUSH dmp.0040228B               ASCII "-=[ CHECK ]=-"
00401301  MOV EDI,dmp.00402353            ASCII "                                      "
```

Ovo izgleda ok. Sada idemo na ALT + E da proverimo da li su svi imorti OK. Desno dugme na ime .exe fajla pa na View names. Tabela izgleda ovako:

```
Names in dmp
Address   Section   Type   ( Name                    Comment
004030B0  .idata    Import ( user32.AppendMenuA
004030B4  .idata    Import ( user32.DeleteMenu
004030F8  .idata    Import ( GDI32.DeleteObject
004030B8  .idata    Import ( user32.DialogBoxParamA
004030BC  .idata    Import ( user32.DrawMenuBar
004030C0  .idata    Import ( user32.EnableWindow
004030C4  .idata    Import ( user32.EndDialog
004030EC  .idata    Import ( kernel32.ExitProcess
004030C8  .idata    Import ( user32.GetDlgItem
004030CC  .idata    Import ( user32.GetDlgItemTextA
004030F0  .idata    Import ( kernel32.GetModuleHandleA
004030D0  .idata    Import ( user32.GetSystemMenu
004030D4  .idata    Import ( user32.LoadBitmapA
004030D8  .idata    Import ( user32.LoadIconA
004030DC  .idata    Import ( user32.MessageBoxA
00401000  CODE      Export   <ModuleEntryPoint>
004030E0  .idata    Import ( user32.SendDlgItemMessageA
004030E4  .idata    Import ( user32.SendMessageA
004030AC  .idata    Import ( user32.SetWindowTextA
004030A8  .idata    Import ( user32.wsprintfA
```

Sve izgleda OK. Znaci sve je OK t.j. raspakovali smo fajl kako treba !!! Po imortima se zakljucuje da je ova .exe fajl pisan u ASMu.

PEtite 2.2

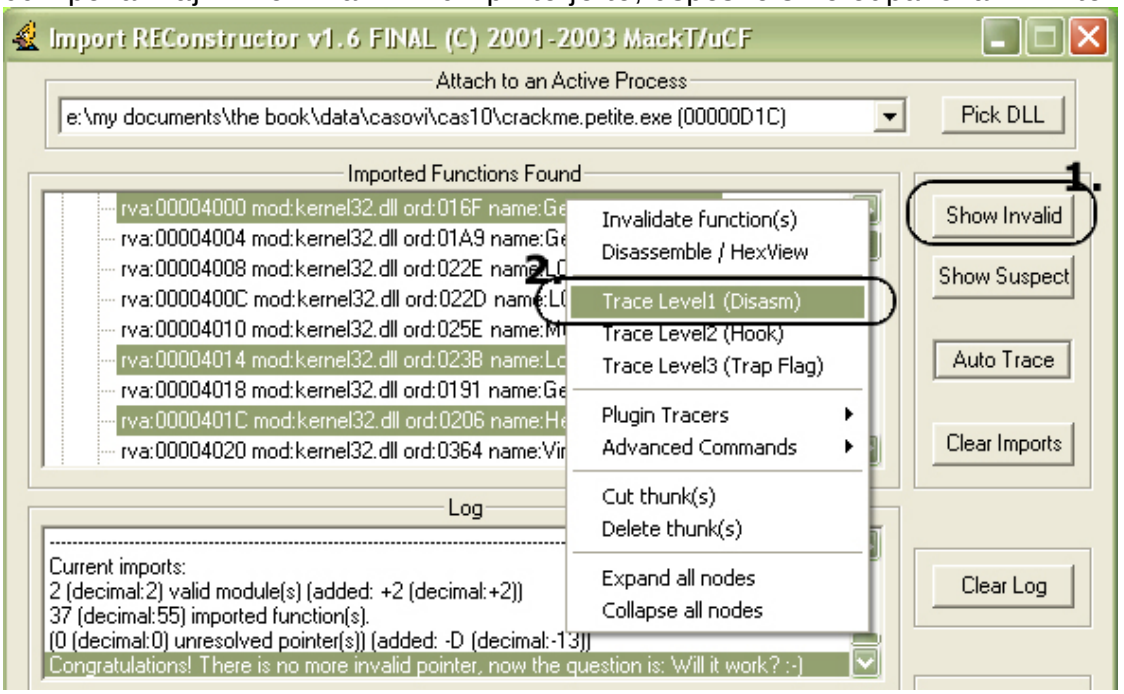
Do sada smo rucno trazili OEPe, ali cemo za odpakivanje PETitea koristiti Ollyev plugin OllyScript kako bi brze i lakse nasli sam OEP. Meta koju cemo koristiti se nalazi u folderu Cas10 a zove se crackme.petite.exe a olly skripta koja ce nam trebati se nalazi u fajlu ...\ Download\OllyDBG scripts.rar a zove se petite2.2.txt.
Kada imamo sve sto nam treba mozemo da pocnemo sa odpakivanjem mete. Ucitajte metu u Olly. Ako Olly bude izbacivao neke poruke, pritisnite Yes. Sada samo treba da izvrsimo petite2.2.txt pomocu Plugins -> OllyScript -> Run Script... Pritisnemo OK tri puta i nalazimo se na OEPu. Brzo zar ne ?? Ono sto vidimo je ovo:

```
004012C0     55          DB 55                          ; This is the entry point
004012C1     8B          DB 8B
```

Naravno ovo je sigurno OEP samo treba da ga analiziramo pomocu Ollya. Pritiskom na CTRL + A vidimo ovo:

```
004012C0  /. 55             PUSH EBP                    ; This is the entry point
004012C1  |. 8BEC           MOV EBP,ESP
```

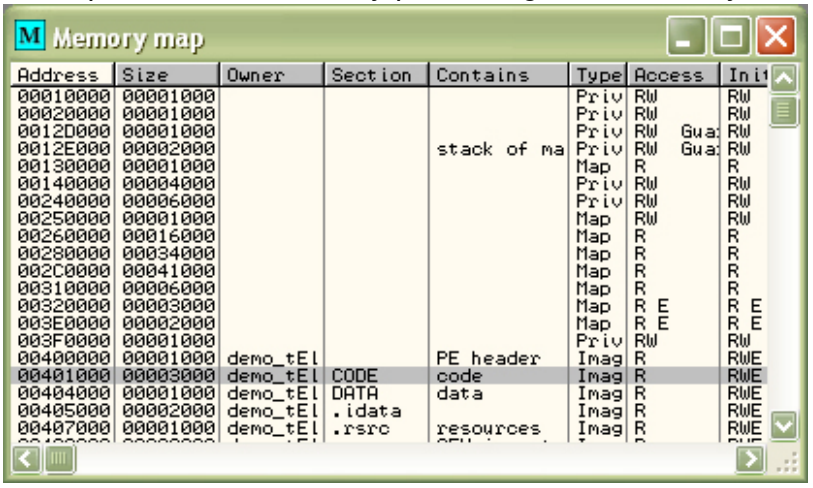
Sada treba da uradimo standardan memory dump i da popravimo importe. Dump cemo uraditi preko LordPEa, uradicemo full dump. Sve do sada je bilo lako, i ostatak je lak samo morate da naucite par novih trikova. Otvoricemo ImpRec i ucitacemo crackme.petite.exe u njega. Moramo da promenimo OEP podesavanja kako bi smo dobili tabelu importa za korektan OEP. Promenicemo OEP na adresu 000012C0, pritisnucemo IATAutoSearch, pa GetImports. Kao sto vidimo nisu svi importi tacni, za oba .dll fajla nam fali po nekoliko importa. Kliknucemo na dugme Show Invalid, pa cemo onda pronaci sve netacne importe desnim klikom na selektovane importe i selekcijom Trace Level1 dobicemo sve importe koji nam fale. Ostaje samo da popravimo dumpovan fajl klikom na Fix Dump i to je to, uspesno smo odpakovali PETite.



tElock 0.80

Moram da priznam da sam fajl sa ovom zastitom slucajno nasao medju svojom kolekcijom targeta. Naime ovaj fajl se nalazio zajedno sa programom pod imenom DZA patcher crackerske grupe TNT! Posto nisam nikada pre odpakivao ni jedan program pakovan sa tElocom odlucio sam da probam. Naravno posto nigde nisam nasao tutorijal kako bih odpakovao ovu zastitu (*a nisam ni trazio posto pri ruci nisam imao internet pa sam samo prekopao svoj hard disk*) odlucio sam da ovo uradim onako muski, na nevidjeno. Meta za ovaj deo poglavlja ce biti fajl demo.tElock.exe, ovaj fajl cemo ucitati u Olly i pokusacemo da ga odpakujemo. Imajte na umu da sam se ja mucio oko ovoga nesto malo manje od pola sata i da cu neke stvari ovde izgledati mozda malo nelogine ali su tu kako bi ubrzale proces odpakivanja, stoga ih shvatite kao apriori, to jest kao cinjenice koje su takve i ne dokazuju se.

Posle vecanja kako da odpakujem ovu metu odlucio sam da je najlakse da postavim jedan memoriski break point na CODE sekciju fajla i da odatle krenem sa reversingom. Da bi smo ovo uradili moramo da odemo prozor modules ALT+M i da desnim klikom na sekciju CODE postavimo Set Memory breakpoint on access. Taj prozor izgleda kao ovaj sa slike dole. Ja sam vec



vise puta startovao ovaj program pa znam da ce vam on praviti gomily execptiona i da necete moci da ih sve zaobidjete pritiskom na CTRL+F9. Zato cemo pritiskati CTRL+F9 onoliko puta koliko je potrebno da se predju svi ovi exceptioni i da se dodje do prvog memoriskog break-pointa. Ja sam pritisnuo CTRL+F9 cak 10x kako bi strigao do prvog memoriskog break-pointa. Posto sam ja ovaj kod vise puta analizirao otkrio sam da on sluzi za odpakivanje koda u memoriju. Sada se mozemo vratiti u prozor Memory map i iskljuciti break-point na CODE sekciji. Kada se vratimo nazad u CPU prozor videcemo gde smo to stali sa izvrsavanjem koda.

```
0040868F   AC              LODS BYTE PTR DS:[ESI]
00408690   F6D8            NEG AL
00408692   04 01           ADD AL,1
00408694   8AD2            MOV DL,DL
```

Ovaj deo koda nam je totalno nerazumljiv ali posle dosta (*stvarno dosta*) traceovanja dosao sam do zakljucka da i ovaj deo koda sluzi za odpakivanje u memoriju. Da bi smo preskocili ceo ovaj kod za odpakivanje skrolovacemo dole do adrese 004087BA na kojoj se nalazi jedna JMP EAX komanda. Na nju cemo postaviti obican break-point. Ako vas Olly nesto pita odgovorite mu sa YES. Sada cemo pritisnuti F9 da bi smo dosli do tog break-pointa.

Pritisnucemo F9 jos 2x kada stignemo do tog break-pointa. Zasto ??? Zato sto ce se dole u Hex Dumpu (donji levi deo prozora) prikazivati odpakovani bajtovi. Kada i treci put pritisnemo F9 u Hex Dumpu cemo imati ovo:

```
00404000  55 4E 52 45 47 49 53 54  UNREGIST
00404008  45 52 45 44 21 21 21 21  ERED!!!!
00404010  21 21 21 21 21 21 21 21  !!!!!!!!
00404018  21 00 54 68 69 73 20 69  !.This i
00404020  73 20 74 68 65 20 55 4E  s the UN
00404028  52 45 47 49 53 54 45 52  REGISTER
00404030  45 44 20 76 65 72 73 69  ED versi
00404038  6F 6E 2C 79 6F 75 20 6D  on,you m
00404040  75 73 74 20 50 41 59 21  ust PAY!
00404048  00 00 00 00 00 00 00 00  ........
```

a ovo izgleda kao odpakovani kod. Mozete da pritisnete F9 i cetvrti put ali ce se program startovati i mi necemo stici do naseg OEPa. Verujte mi probao sam vise puta ovo. Zato cemo pritisnuti F8 sto ce nas odvesti do sledece adrese:

```
00408975   61              POPAD
00408976   56              PUSH ESI
00408977   57              PUSH EDI
```

Polako sa F8 cemo vrsiti trace nadole sve dok nestignemo do adrese:

```
004089AD  ^ \7F D3         JG SHORT demo_tEl.00408982
```

Ovo je jedan jakooo dugacak loop u verujte mi ne zelite da ga izvrsavate celog, stoga cemo postaviti break-point odmah ispod ove adrese, postavicemo break-point na:

```
004089AF   5F              POP EDI
```

Idalje traceujemo na dole sa F8 sve dok ne dodjemo do adrese:

```
00408BBF   FF95 889C4000   CALL DWORD PTR SS:[EBP+409C88]
```

kada se u registrima pojavljuje ime user32.dll fajla. Posle detaljne analize shvatio sam da ovaj deo koda sluzi za odpakivanje i rekonstrukciju IATa (*import tabele*) stoga sam traceovao do zadnje adrese u ovom loopu koji sluzi odpakivanje IATa. Drzacemo F8 sve dok ne dodjemo do ovoga:

```
00408E49   8803            MOV BYTE PTR DS:[EBX],AL
00408E4B   43              INC EBX
00408E4C   3803            CMP BYTE PTR DS:[EBX],AL
00408E4E  ^ 75 F9          JNZ SHORT demo_tEl.00408E49
00408E50   8385 1A9D4000 0>ADD DWORD PTR SS:[EBP+409D1A],4
00408E57  ^ E9 5DFEFFFF    JMP demo_tEl.00408CB9
```

Kada dodjemo do ovoga takodje cemo imati loop koji ubacuje API pozive u IAT tabelu. Postavicemo stoga break-point na 00408E57 i pritisnucemo F9 da dodjemo do njega. Sa F8 se ponovo vracamo na pocetak ovog loopa za odpakivanje IATa. Pritisnucemo F9 10x sto je ekvivalentno tome da smo odpakovali u memoriju deset API poziva. Zasto bas 10x ??? Pa mozete sa F8 traceovati ceo ovaj loop 10x i videcete da cete stalno dolaziti do break-pointa na adresi 00408E57. Sada cemo sa F8 preci jos 1x ovaj deo koda za odpakivanje IATa i ponovo cemo stici od 00408E57. Drzacemo sada ponovo F8 kako bi odpakovali sledecu API funkciju, ali... Izgleda da nema vise API funkcija jer smo sada stigli do jednog drugog loopa koji kako vidimo prepravlja memoriju. Ovaj loop se nalazi ovde:

```
00408EFB   AC              LODS BYTE PTR DS:[ESI]
00408EFC   3206            XOR AL,BYTE PTR DS:[ESI]
00408EFE   AA              STOS BYTE PTR ES:[EDI]
00408EFF  ^ E2 FA          LOOPD SHORT demo_tEl.00408EFB
```

Posto je i ovo veoma dugacak loop postavicemo break-point na CALL koji se nalazi odmah ispod LOOPD komande, postavicemo break-point na adresu:

```
00408F01   E8 2A3E3E3E       CALL 3E7ECD30
```
i pritisnucemo F9 da dodjemo do te adrese. Posto ovaj CALL izgleda zanimljivo pritisnucemo F7 da udjemo u njega i nacemo se ovde:
```
00408F1A   FFB5 DE9D4000     PUSH DWORD PTR SS:[EBP+409DDE]    ; kernel32.77E60000
00408F20   FF95 849C4000     CALL DWORD PTR SS:[EBP+409C84]
00408F26   40                INC EAX
```
Sada cemo polako ici sa F8 sve dok ne dodjemo do adrese:
```
00408F32   E8 0C000000       CALL demo_tEl.00408F43
```
U ovaj CALL moramo uci sa F7 jer ako pokusamo da ga predjemo sa F8 program ce se startovati i mi cemo izgubiti nas OEP. Prebacili smo se malo nize i sada se nalazimo ovde:
```
00408F43   FFB5 DE9D4000     PUSH DWORD PTR SS:[EBP+409DDE]   ; kernel32.77E60000
00408F49   FF95 849C4000     CALL DWORD PTR SS:[EBP+409C84]
00408F4F   40                INC EAX
00408F50   48                DEC EAX
```
Sa F8 cemo izvrsiti sve dok ne dodjemo do adrese:
```
00408F63   E8 11000000       CALL demo_tEl.00408F79
```
kada cemo pritisnuti F7 da udjemo u CALL, a onda cemo pritiskati F8 sve dok ne dodjemo do:
```
00408FF9   0000              ADD BYTE PTR DS:[EAX],AL
00408FFB   0000              ADD BYTE PTR DS:[EAX],AL
00408FFD   F3:AA             REP STOS BYTE PTR ES:[EDI]
00408FFF   66:AB             STOS WORD PTR ES:[EDI]          <- Ovde
00409001   EB 02             JMP SHORT demo_tEl.00409005
```
Kada cemo pritisnuti F8 2x i pusticemo da se izvrsi skok 00409001 kao i sledeci JMP skok na adresi:
```
00409001  /EB 02             JMP SHORT demo_tEl.00409005
00409003  |CD 20             INT 20
00409005  \61                POPAD
00409006  - FF6424 D0        JMP DWORD PTR SS:[ESP-30]         ; demo_tEl.00401000
0040900A   98                CWDE
0040900B   F8                CLC
0040900C   73 02             JNB SHORT demo_tEl.00409010
```
Posle izvrsetka ovog skoka nalazimo se tacno ovde:
```
00401000    6A               DB 6A                    ;  CHAR 'j'
00401001    00               DB 00
00401002    68               DB 68                    ;  CHAR 'h'
00401003    1C               DB 1C
00401004    1040 00          ADC BYTE PTR DS:[EAX],AL
```
A to je verovali ili ne OEP. Ako pritisnete CTRL+A videcete i kako on izgleda. On izgleda bas ovako:
```
00401000  . 6A 00            PUSH 0                    ; /lParam = NULL
00401002  . 68 1C104000      PUSH demo_.0040101C ; |DlgProc = demo_tEl.0040101C
00401007  . 6A 00            PUSH 0                    ; |hOwner = NULL
00401009  . 6A 01            PUSH 1                    ; |pTemplate = 1
```
Sada mozemo uraditi dump (*uradite ga preko LordPEa jer Ollydmp nece da uradi posao kako treba, ako vam KAV prijavi virus ignorisite ga jer virusa nema*) i popravicemo importe pomocu ImpReca (*OEP=00001000*). I to je to, odpakovan je i dobri stari tElock. Ako vam nesto nije jasno u vezi odpakivanja tElocka (a *sigurno vam nesto nije jasno*), tipa zasto smo usli u ovaj CALL a ne u ovaj. Odgovor je: Probao sam da predjem preko svih CALLova sa F8, kada bi se program startovao posle nekog CALLa zakljucio sam da u njega treba da udjem sa F7 i tako redom. Reversing nije egzaktna nauka, morate pasti mnogo puta kako bi ste ustali, i stali cvrsto na svoje noge.

PeCompact 2.22

Moram da priznam da sam svojevremeno mislio da je PeCompact zastita bolja nego sto jeste. Sada sam konacno odlucio da se posvetim rucnom odpakivanju aplikacije zasticene PeCompactom 2.22. Posto sam sa interneta ranije skinuo sam trial PeCompact pakera primetio sam da se mogu birati moduli sa kojim ce se pakovati exe fajlovi. Default podesavalje ukljucuje samo jedan dll i on se zove pecompact_ffss.dll ali se meta pakovana PeCompactom odpakuje isto kao da imamo i ukljucen dll pecompact_crc32.dll fajl. Meta za ovaj deo knjige je pakovana sa trial verzijom PeCompacta mada licno mislim da nema veze da li je trial ili ne, moram ovo da naglasim ako razlike ipak postoje. Sama meta se zove crackme.pecompact22.exe a nalazi se u folderu Cas10. Ovu metu cemo otvoriti pomocu Ollya, i na OEPu cemo imati sledece:

```
004012C0 > $ B8 D07D4000   MOV EAX,crackme_.00407DD0
004012C5   . 50            PUSH EAX
004012C6   . 64:FF35 00000> PUSH DWORD PTR FS:[0]
004012CD   . 64:8925 00000> MOV DWORD PTR FS:[0],ESP
004012D4   . 33C0          XOR EAX,EAX
004012D6   . 8908          MOV DWORD PTR DS:[EAX],ECX
```

Ovaj “OEP” izgleda dosta cudno. Metu cemo odpakovati na nacin na koji sam je ja odpakovao. Precicemo deo OEPa sa F8 sve dok ne dodjemo do adrese 004012D6 kada ce Olly prikazati Access violation. Ovo cemo iskoristiti u nasu korist. Pritisnucemo CTRL+F7 kako bi smo stigli do reda iznad kojeg se desio Access violation. To se desilo upravo ovde:

```
77FB4DB3  |. 8B1C24        MOV EBX,DWORD PTR SS:[ESP]
77FB4DB6  |. 51            PUSH ECX
77FB4DB7  |. 53            PUSH EBX
```

Vidimo da se ne nalazimo u kodu crackmea nego se nalazimo u kodu ntdll.dll fajla. Sa F8 cemo se kretati kroz kod sve dok ne dodjemo do ove adrese:

```
77FB4DC6  |. E8 480BFCFF    CALL ntdll.ZwContinue
```

tada cemo pritisnuti F7 da udjemo u ovaj CALL jer ce nas to vratiti na mesto iznad kojeg se desio Access violation. Sada se nalazimo ovde

```
77F75913 >/$ B8 20000000   MOV EAX,20
77F75918  |. BA 0003FE7F    MOV EDX,7FFE0300
77F7591D  |. FFD2          CALL EDX
77F7591F  \. C2 0800        RET 8
```

ali smo idalje u ntdll.dll fajlu, pa cemo zato sa F8 izvrsiti sve komande do ukljucujuci i RET 8 komandu. Tada cemo se naci ovde:

```
00407DF3   B8 796D40F0     MOV EAX,F0406D79
00407DF8   64:8F05 0000000>POP DWORD PTR FS:[0]
```

ovo je vec deo koda PeCompacta koji sluzi za odpakivanje fajla u memoriju. Ovaj deo koda cemo izvrsiti sa F8 sve dok ne dodjemo do adrese

```
00407E94   5D              POP EBP
00407E95  - FFE0           JMP EAX                 ; crackme_.<ModuleEntryPoint>
```

i kao sto vidimo JMP EAX vodi do ModuleEnteryPoint-a to jest do samog OEPa pa cemo i ovaj skok izvrsiti sa F8 i nacemo se na OEPu, koji posle analize sa CTRL+A izgleda ovako:

```
004012C0 >/$ 55            PUSH EBP
004012C1  |. 8BEC          MOV EBP,ESP
004012C3  |. 6A FF         PUSH -1
```

Sada ostaje samo da dumpujemo memoriju sa LordPEo i da popravimo importe sa ImpRecom, posle cega smo uspesno odpakovali PeCompact.

PeCompact 1.40

Sada sam 100% siguran da se pitate: “Zasto je napisao prvo kako se odpakuje PeCompact 2.22 a tek onda kako PeCompact 1.40”? Naravno imam ja svoje razloge zasto je to ovako uradjeno. Jedan od glavnih razloga je taj sto je mnogo lakse odpakovati novu 2.22 verziju nego staru 1.40. Ne verujete mi??? Videcemo :) Meta koja je pakovana sa 1.40 verzijom PeCompacta se zove Notepad.pecompact140.exe. Pre nego sto otvorimo metu pomocu Ollya prvo cu vam pokazati jedan jako interesantan trik sa PeIDom. Naime PeID ima jedan veoma interesantan plugin koji nam omogucava da unapred odredimo OEP mete. Dakle otvoricemo metu pomocu PeIDa koji ce nam prvo reci da je meta pakovana sa: PECompact 1.40 - 1.45 -> Jeremy Collake a ako kliknemo na dugme -> koje se nalazi u donjem desnom uglu i izaberemo Plugins -> Generic OEP finder, posle cega ce PeID prikazati poruku da se OEP nalazi na adresi 004010CC. Posle ovoga mozemo da zatvorimo PeID i da ucitamo metu u OllyDBG.
Pakerov OEP izgleda ovako:

```
0040AC5E > /EB 06           JMP SHORT Notepad_.0040AC66
0040AC60  |68 CC100000      PUSH 10CC
0040AC65  |C3              RET
0040AC66  \9C              PUSHFD
```

Sada vidite razliku izmedju verzija. Btw da li ste primetili parametar koji ima PUSH komanda. Da, da znam PUSH 10CC, to je PUSH OEP ;) Bez obzira na ovo, pritiskacemo F8 sve dok ne dodjemo do CALL na adresi 0040AC68 u koji moramo da udjemo. Taj CALL izgleda bas ovako:

```
0040AC68   E8 02000000      CALL Notepad_.0040AC6F
```

stoga cemo pritisnuti F7 da udjemo u ovaj CALL kada dodjemo do njega, to nas vodi ovde:

```
0040AC6F   8BC4             MOV EAX,ESP
0040AC71   83C0 04          ADD EAX,4
0040AC74   93               XCHG EAX,EBX
0040AC75   8BE3             MOV ESP,EBX
```

Posto se dole nalazi jedna RET komanda a izmedju nema nikakvih skokova izvrsicemo sve komande sa F8, ukljucujuci i RET komandu, posle cega smo ovde:

```
0040D1C3   BD 53310000      MOV EBP,3153
0040D1C8   57               PUSH EDI
0040D1C9   5E               POP ESI
0040D1CA   83C6 42          ADD ESI,42
0040D1CD   81C7 53110000    ADD EDI,1153
0040D1D3   56               PUSH ESI
```

dalje i nema neke velike mudrosti jednostavno drzimo F8 sve dok ne dodjemo do adrese:

```
0040D350   57               PUSH EDI
0040D351   8BBD D7A44000    MOV EDI,DWORD PTR SS:[EBP+40A4D7]
0040D357   03BD A6A04000    ADD EDI,DWORD PTR SS:[EBP+40A0A6]
0040D35D   8B8D DBA44000    MOV ECX,DWORD PTR SS:[EBP+40A4DB]
```

malo ispod toga imamo dosta loopova koji se izvrsavaju dosta puta, ti loopovi izgledaju ovako:

```
0040D36E  /74 72            JE SHORT Notepad_.0040D3E2
0040D370  |78 70            JS SHORT Notepad_.0040D3E2
0040D372  |66:8B07          MOV AX,WORD PTR DS:[EDI]
0040D375  |2C E8            SUB AL,0E8
0040D377  |3C 01            CMP AL,1
```

```
0040D379  |76 38          JBE SHORT Notepad_.0040D3B3
0040D37B  |66:3D 1725     CMP AX,2517
0040D37F  |74 51          JE SHORT Notepad_.0040D3D2
0040D381  |3C 27          CMP AL,27
0040D383  |75 0A          JNZ SHORT Notepad_.0040D38F
0040D385  |80FC 80        CMP AH,80
0040D388  |72 05          JB SHORT Notepad_.0040D38F
0040D38A  |80FC 8F        CMP AH,8F
0040D38D  |76 05          JBE SHORT Notepad_.0040D394
0040D38F  |47             INC EDI
0040D390  |43             INC EBX
0040D391  ^|EB DA          JMP SHORT Notepad_.0040D36D
```

Da bi smo presli sva ova ponavljanja pronacicemo negde dole zadnji skok koji ce nas vratiti negde gore u kod. Taj skok se nalazi ovde:

```
0040D3E0  ^ \EB 87          JMP SHORT Notepad_.0040D369
```

Posto zelimo da preskocimo sve ove loopove jednostavno cemo postaviti break-point na adresu odmah ispod ovog skoka, postavicemo ga na adresu 0040D3E2 i pritisnucemo F9 da dodjemo do njega. Dalje cemo nastaviti sa izvrsavanjem koda sa F8 sve dok ne stignemo do adrese 0040D48F.

```
0040D488   9D               POPFD
0040D489   50               PUSH EAX
0040D48A   68 CC104000      PUSH Notepad_.004010CC
0040D48F   C2 0400          RET 4
```

Kada se izvrsi i ova RET 4 komanda mi cemo se naci ovde:

```
004010CC    55              DB 55                   ; CHAR 'U'
004010CD    8B              DB 8B
004010CE    EC              DB EC
```

Kao sto nam je PeID rekao na adresi 004010CC se nalazi pravi OEP. Posle analize ovog koda sa CTRL+A cemo videti da je ovo zaista pravi OEP,

```
004010CC  /. 55             PUSH EBP
004010CD  |. 8BEC           MOV EBP,ESP
004010CF  |. 83EC 44        SUB ESP,44
004010D2  |. 56             PUSH ESI
```

i da ovde mozemo da uradimo dump pomocu LordPEa. Posle dumpa nam ostaje samo da popravimo importe pomocu ImpReca na standardan nacin.

Videli smo da PeID bez greske moze da nadje OEP u PeCompactovanim fajlovima i da se bez brime mozemo pouzdati u njegovu procenu OEPa. Ova informacija nam moze posluziti da napravimo neku skriptu koja bi za nas odpakovala program pakovan PeCompactom ili je mozemo koristiti da se proverima da li smo mi na pravom mestu i da li je to pravi OEP. Kao sto vidite mnogo je lakse odpakovati noviju nego stariju verziju PeCompacta, mislim lako je odpakovati i jednu i drugu verziju samo nam za ovu novu verziju treba manje posla.

PE Pack 1.0

Prelistavajuci neke tudje keygeneratore koje imam na svom hard disku naisao sam na jedan zanimljiv keygen za CDRLabel 4.1 koji su napravili momci iz CORE crackerske grupe. Meni licno sam keygen nije bio zanimljiv nego mi je bila zanimljiva zastita sa kojom je pakovan ovaj keygen. Upitanju je PE Pack koji nikada ranije nisam runo odpakovao pa sam odlucio da se oprobam i sa ovim pakerom. Molim vas da imate na umu da mene, a ne bi trebalo ni vas, ne interesuje sta ova meta radi, nego me interesuje kako bih ja to mogao da je odpakujem. Meta se nalazi u folderu Cas10 a zove se crcdl41.pepack10.exe. Ovu metu cemo otvoriti pomocu Ollya i pogledacemo sta se nalazi na OEPu.

```
00401212 > $ /74 00          JE SHORT cr-cdl41.00401214
00401214  >-\E9 E74D0000     JMP cr-cdl41.00406000
```

Dosta cudno, ali nema veze, sa F8 cemo izvrsiti oba skoka i nacemo se ovde:

```
00406000   60                PUSHAD
00406001   E8 00000000       CALL cr-cdl41.00406006
00406006   5D                POP EBP
00406007   83ED 06           SUB EBP,6
```

Sa F8 cemo izvrsavati red po red sve dok ne dodjemo do jednog dugackog loopa:

```
004060D4   /73 38            JNB SHORT cr-cdl41.0040610E
004060D6   |48               DEC EAX
004060D7   |74 35            JE SHORT cr-cdl41.0040610E
004060D9   |78 33            JS SHORT cr-cdl41.0040610E
004060DB   |66:8B1C39        MOV BX,WORD PTR DS:[ECX+EDI]
004060DF   |80FB E8          CMP BL,0E8
004060E2   |74 0F            JE SHORT cr-cdl41.004060F3
004060E4   |80FB E9          CMP BL,0E9
004060E7   |74 0A            JE SHORT cr-cdl41.004060F3
004060E9   |66:81FB FF25     CMP BX,25FF
004060EE   |74 0F            JE SHORT cr-cdl41.004060FF
004060F0   |41               INC ECX
004060F1   ^|EB E3           JMP SHORT cr-cdl41.004060D6
004060F3   |294C39 01        SUB DWORD PTR DS:[ECX+EDI+1],ECX
004060F7   |83C1 05          ADD ECX,5
004060FA   |83E8 04          SUB EAX,4
004060FD   ^|EB D7           JMP SHORT cr-cdl41.004060D6
004060FF   |295439 02        SUB DWORD PTR DS:[ECX+EDI+2],EDX
00406103   |83C1 06          ADD ECX,6
00406106   |83EA 04          SUB EDX,4
00406109   |83E8 05          SUB EAX,5
0040610C   ^|EB C8           JMP SHORT cr-cdl41.004060D6
0040610E   \C685 D3000000 F>MOV BYTE PTR SS:[EBP+D3],0F8
00406115   5B                POP EBX
00406116   5A                POP EDX
00406117   5E                POP ESI
00406118   ^ E9 76FFFFFF     JMP cr-cdl41.00406093
0040611D   6A 04             PUSH 4
```

Da ne bi smo izvrsavali ovaj dugacak loop postavicemo break point na adresu 0040611D, odnosno na PUSH 4 komandu jer cemo tu sigurno stici odmah posle izvrsavanja ovog loopa to jest posle odpakivanja u memoriju. Sa F8 cemo polako izvrsavati kod sve dok ne dodjemo do adrese:

```
004061C2   0385 47050000     ADD EAX,DWORD PTR SS:[EBP+547]
004061C8   52                PUSH EDX                    ; cr-cdl41.00404000
```

kada ce se u EAXu pojaviti user32.dll string. Zaci da je na red doslo odpakivanje IATa u memoriju pa cemo potraziti loop koji ce to uraditi. On se nalazi ovde:

```
00406252   83C7 04          ADD EDI,4
00406255  ^ EB BD           JMP SHORT cr-cdl41.00406214
```

to jest ovo je poslednja adresa loopa koji sluzi za odpakivanje API poziva u memoriju, dok se odmah ispod nalazi drugi skok:

```
00406257   83C2 14          ADD EDX,14
0040625A  ^ E9 55FFFFFF     JMP cr-cdl41.004061B4
```

koji sluzi za odpakivanje imena svih dll fajlova koji sadrze odpakovane API pozive. Zbog ovog cemo postaviti dva break-pointa na adrese:

```
004061C2   0385 47050000    ADD EAX,DWORD PTR SS:[EBP+547]
004061C8   52               PUSH EDX                    ; cr-cdl41.00404014
```

da bi smo videli imena dll fajlova koja se nalaze u IATu. Treba nam jos jedan break-point za slucaj da pritisnemo F9 jednom vise puta nego sto treba. Zbog ovoga cemo postaviti break-point na adresu:

```
0040625F   8B85 57050000    MOV EAX,DWORD PTR SS:[EBP+557] <- Ovde
00406265   0385 8B050000    ADD EAX,DWORD PTR SS:[EBP+58B]
0040626B   894424 1C        MOV DWORD PTR SS:[ESP+1C],EAX
0040626F   61               POPAD
00406270   FFE0             JMP EAX
```

jer ce se posle ova dva loopa nastaviti sa izvrsavanjem unpacking koda. Sada cemo pritiskati F9 i belezicemo imena dll fajlova koja ce se pojavljivati u EAXu na adresi 004061C8. Ti dll fajlovi su:

```
user32.dll
msvcrt.dll
kernel32.dll
```

i kada sledeci put pritisnemo F9 zastacemo na adresi 0040625F koja predstavlja nas safe break-point. Ovo znaci da se API pozivi nalaze samo u tri dll fajla. Ovo smo mogli da uradimo i sa imenima API poziva, ali za to nema potrebe. Bitni su nam samo dll fajlovi posle kada budemo popravljali IAT pomocu ImpReca. Sa F8 cemo izvrsiti par redova sve dok ne dodjemo do adrese:

```
0040626F   61               POPAD
00406270  - FFE0            JMP EAX                     ; cr-cdl41.004010D0
```

Navikli smo da se na OEP ide pomocu skokova i RET komandi, tako da skok koji se izvrsava odmah posle POPAD funkcije moze voditi pravo na OEP. Izvrsicemo i njega sa F8 i videcemo gde ce nas to odvesti. Docicemo ovde:

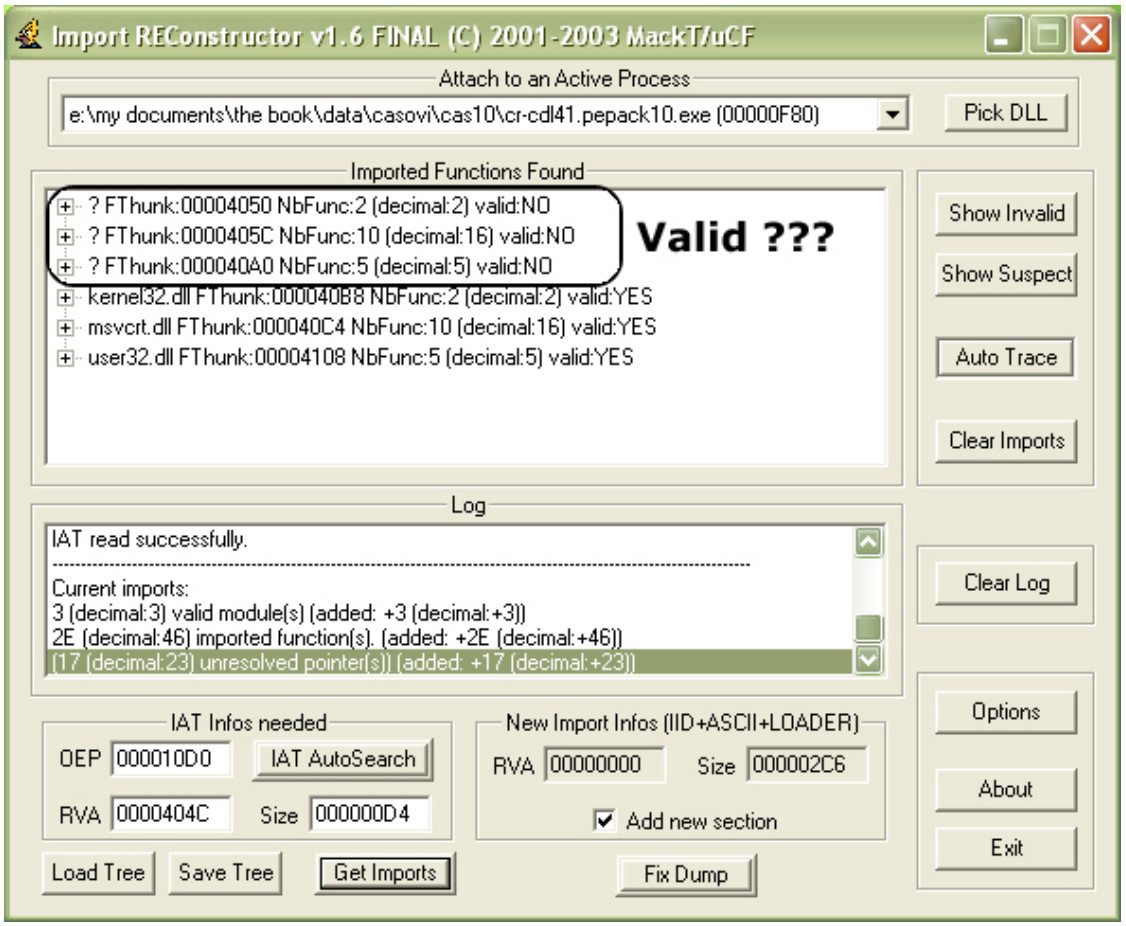
```
004010D0    55              DB 55                       ;  CHAR 'U'
004010D1    8B              DB 8B
004010D2    EC              DB EC
004010D3    6A              DB 6A                       ;  CHAR 'j'
004010D4    FF              DB FF
```

Ovo naravno lici na OEP i posle analize koda pritiskom na CTRL+F2 i zakljucujemo da je ovo pravi OEP jer izgleda bas ovako:

```
004010D0  . 55              PUSH EBP
004010D1  . 8BEC            MOV EBP,ESP
004010D3  . 6A FF           PUSH -1
004010D5  . 68 00204000     PUSH cr-cdl41.00402000
```

Dakle adresa na kojoj mozemo da izvrsimo memory dump je 004010D0. Dump cemo uraditi na standardan nacin pomocu LordPEa a rekonstrukciju importa cemo uraditi pomocu ImpReca. Sada ce na scenu stupiti ono sto je bilo potrebno da zapisemo. Sada ce nam trebati imena dll fajlova iz kojih se citaju importi. Ovo smo uradili zato sto ce se desiti da ImpRec nece moci da nadje sve dll fajlove iz kojih se citaju API pozivi pa cemo mi morati rucno da

modifikujemo i / ili obrisemo neispravne API pozive. Ne brinite ovo ce biti izuzetno lako posto znamo imena dll fajlova iz kojih se citaju API pozivi. Otvoricemo ImpRec i ucitacemo proces cr-cdl41.pepack10.exe u njega. Promenicemo OEP na pravu vredno, promenicemo ga na 000010D0, pritisnucemo IAT AutoSearch i dobicemo importe kao na slici ispod.



Ono sto je bitno da se vidi je da se svi dll fajlovi koje smo pronasli prilikom odpakivanja mete vec uneti i da je ovaj visak koji je na slici zaokruzen stvarno visak i da se moze odseci iz nove IAT tablice. Ovo cemo uraditi klikom na dugme Show Invalid, pa cemo kliknuti desnim dugmetom na selektovane API pozive i izabracemo opciju Cut thunk(s). Kada ostanu samo ispravni API pozivi iz ona tri pronadjena dll fajla mocemo da uradimo popravku importa u dumpovanom fajlu pomocu opcije Fix Dump. I to je to uspeli smo uspesno da odpakujemo i PE Pack.

ASProtect 1.2 / 1.2c

Dugo vremena sam bio plasen od strane drugih (doduse manje iskusnih) crackera kako je najvece zlo koje stoji omah uz Armadillo, protector pod imenom ASProtect. Danas ja pokusavam da dokazem suprotno. Iako je verzija koju cemo mi reversovati pomalo zastarela, jer je u opticaju verzija 1.3x, idalje je dobar primer kako se to reversuje ASProtect. Ne shatite ovaj deo knjige olako, jer idalje postoji mnogo komercijalnih programa pakovanih bas ovom verzijom programa. Postoje neke prednosti kod reversinga ove verzije ASProtecta, a to su: nema previse teskih importa, nema "ukradenih bajtova" i sto je najlepse od svega ima exceptiona koji nam pomazu da pronadjemo pravi OEP. Meta se zove crackme.asprotect.exe a nalazi se u folderu Cas10. Ucitacemo je u Olly i na pakerovom OEPu cemo imati sledece:

```
00401000 >/$ 68 01704000   PUSH crackme_.00407001
00401005  \. C3            RET
```

Zanimljivo ili ne, ali mi se necemo bakcati sa ovim. Kao sto sam rekao Olly ce nam pomoci preko exceptiona da nadjemo OEP. Pritisnite F9 i nacicete se ovde:

```
0087009D   3100            XOR DWORD PTR DS:[EAX],EAX
0087009F   EB 01           JMP SHORT 008700A2
008700A1   68 648F0500     PUSH 58F64
008700A6   0000            ADD BYTE PTR DS:[EAX],AL
```

Sada cemo pritiskati CTRL+F9 da bi smo presli preko ovih exceptiona. Ovo cemo uraditi onoliko puta koliko je potrebno da se sama zapakovana meta ne startuje. To jest docicemo do dela koda gde kada bi pritisnuli CTRL+F9 jos jednom NAG ekran iz crackmea bi se pojavio. Neki to rade brojanjem ali ovde nema potrebe za brojanjem jer je to mesto isto za sve programe i izgleda ovako:

```
00882FCC   FE02            INC BYTE PTR DS:[EDX]
00882FCE  ^EB E8           JMP SHORT 00882FB8
00882FD0   E8 0A000000     CALL 00882FDF
```

Znaci pritiskacemo CTRL+F9 sve dok ne dodjemo do ovog mesta u kodu. Sada cemo umesto CTRL+F9 pritisnuti CTRL+F8 da bi smo presli preko ovog exceptiona za jednu liniju koda i kao poruceno zavrsavamo ovde:

```
77FB4DB3  |. 8B1C24          MOV EBX,DWORD PTR SS:[ESP]
77FB4DB6  |. 51              PUSH ECX
77FB4DB7  |. 53              PUSH EBX
77FB4DB8  |. E8 ACBDFAFF     CALL ntdll.77F60B69
77FB4DBD  |. 0AC0            OR AL,AL
```

Da, da zahvaljujuci Windowsu XP zavrsavamo u ntdll.dll-u. Ovo je dobra stvar jer cemo odatle izaci tacno ispod exceptiona. Pritiskajte F8 sve dok ne dodjete do prvog CALL, a onda pritisnite F7 da udjete u njega. Idalje smo u istom dll-u pa cemo se kretati sa F8 sve dok ne dodjemo do ovog CALLa:

```
77F60BEA  . FF75 0C        PUSH DWORD PTR SS:[EBP+C]
77F60BED  . 53             PUSH EBX
77F60BEE  . 56             PUSH ESI
77F60BEF  . E8 528F0100    CALL ntdll.77F79B46              <- Ucemo u ovaj CALL (F7)
77F60BF4  . F605 4A32FC77> TEST BYTE PTR DS:[77FC324A],80
77F60BFB  . 8BF8           MOV EDI,EAX
77F60BFD  . 0F85 896F0200  JNZ ntdll.77F87B8C
```

u koji cemo uci sa F7. A to nas vodi dublje u sam dll fajl. Sada smo ovde:

```
77F79B46  $ BA B89BF777    MOV EDX,ntdll.77F79BB8
77F79B4B  . EB 07          JMP SHORT ntdll.77F79B54
77F79B4D  /$ BA DF9BF777   MOV EDX,ntdll.77F79BDF
```

```
77F79B52  |. 8D09            LEA ECX,DWORD PTR DS:[ECX]
77F79B54  |> 53              PUSH EBX
77F79B55  |. 56              PUSH ESI
77F79B56  |. 57              PUSH EDI
77F79B57  |. 33C0            XOR EAX,EAX
77F79B59  |. 33DB            XOR EBX,EBX
77F79B5B  |. 33F6            XOR ESI,ESI
77F79B5D  |. 33FF            XOR EDI,EDI
77F79B5F  |. FF7424 20       PUSH DWORD PTR SS:[ESP+20]          ; /Arg5
77F79B63  |. FF7424 20       PUSH DWORD PTR SS:[ESP+20]          ; |Arg4
77F79B67  |. FF7424 20       PUSH DWORD PTR SS:[ESP+20]          ; |Arg3
77F79B6B  |. FF7424 20       PUSH DWORD PTR SS:[ESP+20]          ; |Arg2
77F79B6F  |. FF7424 20       PUSH DWORD PTR SS:[ESP+20]          ; |Arg1
77F79B73  |. E8 06000000     CALL ntdll.77F79B7E                 ; \ntdll.77F79B7E
77F79B78  |. 5F              POP EDI
77F79B79  |. 5E              POP ESI
77F79B7A  |. 5B              POP EBX
77F79B7B  \. C2 1400         RET 14
....
77F79BA2  |. FFD1            CALL ECX
77F79BA4  |. 64:8B25 00000> MOV ESP,DWORD PTR FS:[0]
77F79BAB  |. 64:8F05 00000> POP DWORD PTR FS:[0]
77F79BB2  |. 8BE5            MOV ESP,EBP
77F79BB4  |. 5D              POP EBP
77F79BB5  \. C2 1400         RET 14
```

Sada cemo se kretati sa F8 preko svih ostalih ASM komandi sem CALLova u koje cemo ulaziti sa F7. Zadnji CALL u koji cemo ovde uci je CALL ECX posle cega stizemo ovde:

```
00882FBE   8B6424 08        MOV ESP,DWORD PTR SS:[ESP+8]
0088 2FC2   EB 0C           JMP SHORT 00882FD0
00882FC4   2BD2             SUB EDX,EDX
00882FC6   64:FF32          PUSH DWORD PTR FS:[EDX]
```

ovde se ne desava nista bitno po odpakivanje pa cemo sa F8 ici kroz kod sve dok ne dodjemo do skoka koji bi nas vratio daleko gore u kod. Taj skok se nalazi ovde:

```
0088307D   /E3 03           JECXZ SHORT 00883082
0088307F   |59              POP ECX
00883080  ^|EB C8           JMP SHORT 0088304A            <- Ovaj skok
00883082   \59              POP ECX
00883083    1BC3            SBB EAX,EBX
00883085    F8              CLC
00883086    61              POPAD
00883087    F8              CLC
00883088    03C6            ADD EAX,ESI
0088308A    C3              RET
```

Zato cemo postaviti break-point na prvu komandu ispod ovog skoka, na POP ECX, i pritisnucemo F9 da dodjemo do nje. Docicemo do prve sledece RET komande pritiskajuci F8 i izvrsicemo i nju sa F8 da bi smo se nasli ovde:

```
00882F9E   5B               POP EBX                     ; crackme_.00400000
00882F9F   58               POP EAX
00882FA0    05 6618F45A     ADD EAX,5AF41866
00882FA5    5C              POP ESP
00882FA6    0BC9            OR ECX,ECX
00882FA8  ^ 74 E3           E SHORT 00882F8D
00882FAA    8901            MOV DWORD PTR DS:[ECX],EAX
00882FAC    03C3            ADD EAX,EBX
00882FAE    894424 1C       MOV DWORD PTR SS:[ESP+1C],EAX
00882FB2    61              POPAD
00882FB3    FFE0            JMP EAX
```

Da li i vama ona komanda JMP EAX izgleda zanimljivo :) Sa F8 cemo izvrsiti sve komande do skoka pa i sam skok, JMP EAX i nacemo se ovde:

```
004012C0    55              DB 55                       ;  CHAR 'U'
```

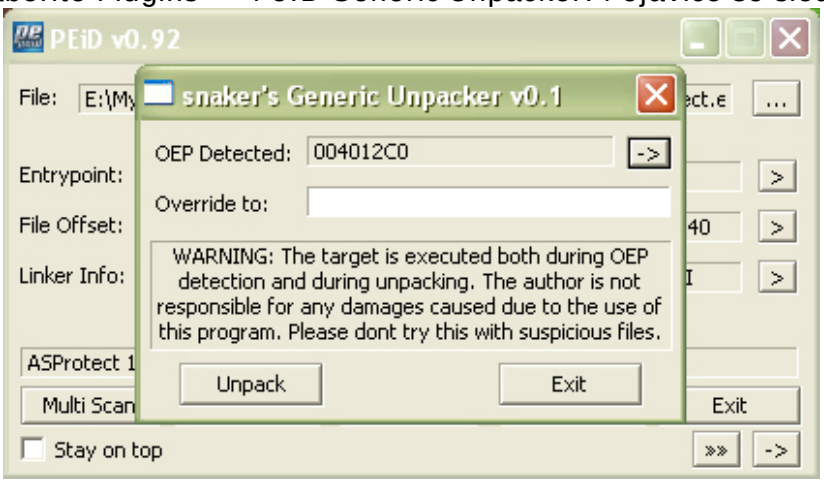
```
004012C1    8B              DB 8B
004012C2    EC              DB EC
004012C3    6A              DB 6A                   ; CHAR 'j'
004012C4    FF              DB FF
004012C5    68              DB 68                   ; CHAR 'h'
```

Verovali ili ne ovo je nas pravi OEP. Pritisnite CTRL+A da bi Olly analizirao ovaj deo koda i pojavice se sledece:

```
004012C0  /. 55             PUSH EBP
004012C1  |. 8BEC           MOV EBP,ESP
004012C3  |. 6A FF          PUSH -1
004012C5  |. 68 F8404000    PUSH crackme_.004040F8
```

Nasli smo mesto na kome cemo uraditi dump pomocu LordPEa. Zapamtite samo adresu na kojoj se nalazi OEP posto ce nam trebati, ta adresa je 004012C0 ili samo 000012C0 ako oduzmemo image base PE sekcije. Sada nam ostaje samo da pronadjemo sve importe i da ih zalepimo za dumpovan fajl. Da bi smo ovo uradili ucitacemo ovaj proces u ImpRec. Promenicemo OEP na kome ce ImpRec traziti importe na 000012C0 i pritisnucemo IAT AutoSearch pa Get Imports. Sta je ovo ??? Nemamo ni jedan validan API poziv, pa cak ni jedan validan dll. Ne nismo mi nista uradili pogresno nego samo treba da potrazimo importe. Kliknimo na dugme Show Invalid, pa cemo kliknuti desnim dugmetom da nepoznate importe i selektovacemo Trace Level1 sto ce nam vratiti neke API pozive, ali ne i sve. Ponovo cemo kliknuti na dume Show invalid pa cemo i taj poslednji API poziv naci na sledeci nacin. Kliknucemo desnim dugmetom na njega i izabradjemo Plugin Tracers -> ASProtect 1.22 i tako cemo pronaci i taj zadnji API poziv. Sada mozemo da pritisnemo dugme Fix dump i da popravimo dumpovan fajl. To je sve sto treba da uradimo kako bi smo uspesno odpakovali ASProtect 1.2x.

Opisani nacin nije jedini sa kojim se moze odpakovati ASProtectom zasticena aplikacija. Ova verzija ASProtecta se takodje moze odpakovati i pomocu PeIDa, i to mnogo brze nego rucno :) Otvorite PeID i skenirajte nasu metu pomocu njega, zatim selektujte dugme -> koje se nalazi u donjem desnom cosku i izaberite Plugins -> PeID Generic Unpacker. Pojavice se sledece:



Kada se pojavi ovaj prozor pritisnite dugme -> da detektujete OEP i pritisnite Unpack. Kada se zavrsi sa dumpovanje PeID ce vas pitati da li da popravi importe, a vi odgovorite sa Yes. Posle nekoliko sekundi imacete odpakovani program u istom direktorijumu kao i originalni fajl. Brzo i lako, ali lame :)

Neolite 2.0

Evo kako se odpakuje jedan od najlaksih pakera koje sam ikada video. Meta se nalazi u folderu Cas10 a zove se Artemis.Neolite20.exe. Sam OEP izgleda ovako:

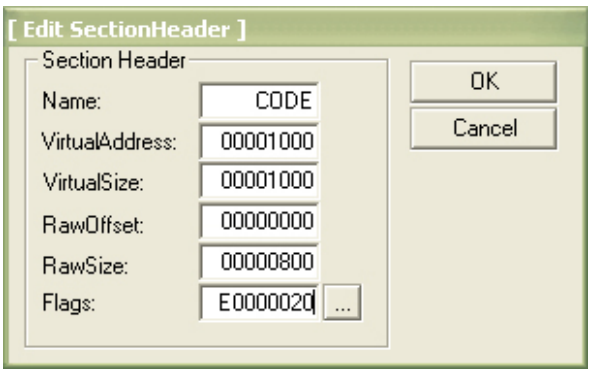
```
00411110 > $ /E9 A6000000   JMP Artemis_.004111BB
...
004111BB  > \8B4424 04      MOV EAX,DWORD PTR SS:[ESP+4]
004111BF  . 2305 21114100   AND EAX,DWORD PTR DS:[411121]
004111C5  . E8 ED040000     CALL Artemis_.004116B7
004111CA  . FE05 BA114100   INC BYTE PTR DS:[4111BA]
004111D0  . FFE0            JMP EAX
004111D2  . 803D BA114100>CMP BYTE PTR DS:[4111BA],0
004111D9  . 75 13           JNZ SHORT Artemis_.004111EE
004111DB  . 90              NOP
004111DC  . 90              NOP
004111DD  . 90              NOP
004111DE  . 90              NOP
004111DF  . 50              PUSH EAX
004111E0  . 2BC0            SUB EAX,EAX
004111E2  . E8 D0040000     CALL Artemis_.004116B7
004111E7  . 58              POP EAX
004111E8  . FE05 BA114100   INC BYTE PTR DS:[4111BA]
004111EE  > C3              RET
```

Da bi ste uspesno odpakovali ovaj paker samo treba da pritiskate F8 polako izvrsavajuci jedan po jedan red sve dok ne izvrsite i JMP EAX komandu. Posle cega cete se naci ovde:

```
00401000   6A 00            PUSH 0
00401002   E8 2A060000      CALL Artemis_.00401631
00401007   A3 36214000      MOV DWORD PTR DS:[402136],EAX
0040100C   C705 7B214000 0>MOV DWORD PTR DS:[40217B],0B
00401016   C705 7F214000 E> MOV DWORD PTR DS:[40217F],Artemis_.00401>
00401020   C705 83214000 0>MOV DWORD PTR DS:[402183],0
```

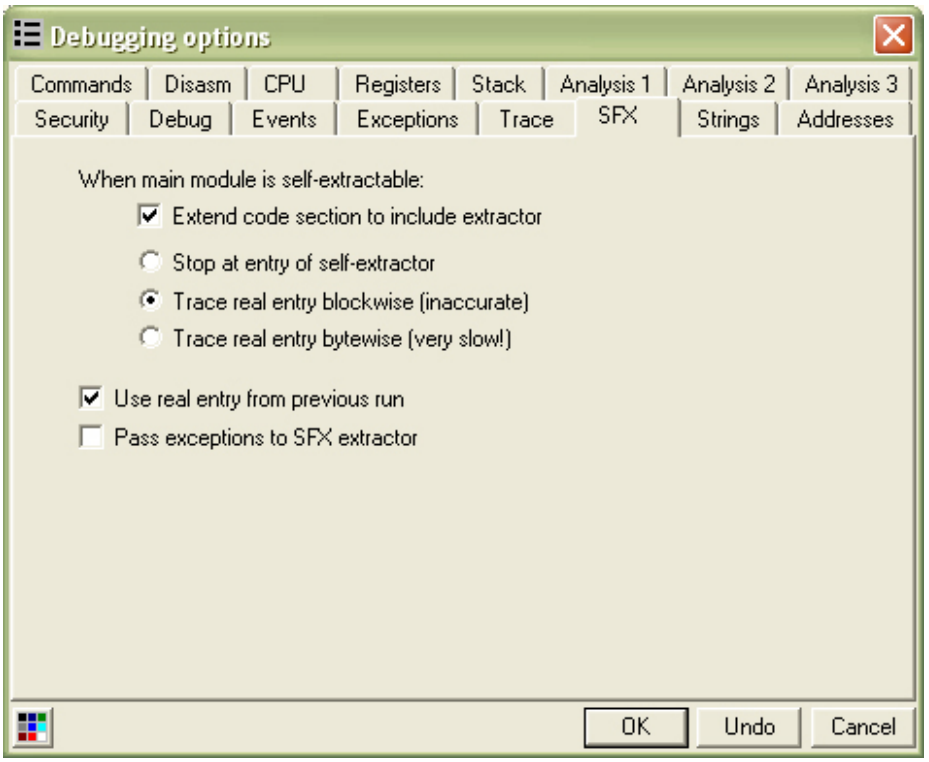
a ovo je ujedno i pravi OEP. Ovde samo treba da uradimo full memory dump i da popravimo importe. Ako necete Neolite 2.0 da odpakujete rucno mozete to da uradite i pomocu PeIDovog generickog unpackera. Lako zar ne ???

NAPOMENA: Ako ovaj packer odpakujete rucno moracete da podesite karakteristike sekcije CODE na E0000020. Ovo se moze uraditi pomocu PE editora koji se nalazi u LordPEu. Kada otvorite fajl u LordPEu onda izaberite dgume Sections pa na CODE, desnim dugmeto m izaberite Edit Section Header i u polje Flags unesiti E0000020. Ovo radimo zato sto sekcija CODE mora da se cita i izvrsava. Mada moze i bez ovoga ali je ovako sigurnije.



PE Lock NT 2.04

Na ovom packeru cu pokazati kako nam neke opcije iz Ollya mogu pomoci da nadjemo OEP za samo jednu sekundu. Meta se nalazi u folderu Cas10 a zove se TSC_crkme10.PeLockNt204.exe, da li ovaj paker radi samo na NT sistemima ne znam, ali znam da ima jednu gresku. Naime kada je kod pakovan sa ovim pakerom crackme jednostavno nece da se startuje. Ovo cemo da resimo jednostavnim odpakivanjem. Pre nego sto pocnemo podesicemo Olly ovako:



Ova opcija nam omogucuje da analiziramo SFX kod tako da se odma nadjemo na samom OEPu. Ovo je jako korisno jer nas kod ovog pakera vodi direktno da sam OEP koji izgleda ovako:

```
00401260  . 55              PUSH EBP                    ; Real entry point of SFX code
00401261  . 8BEC            MOV EBP,ESP
00401263  . 6A FF           PUSH -1
00401265  . 68 00304000     PUSH TSC_crkm.00403000
0040126A  . 68 78224000     PUSH TSC_crkm.00402278      ; SE handler installation
0040126F  . 64:A1 0000000>  MOV EAX,DWORD PTR FS:[0]
```

Posto se vec nalazimo na OEPu mozemo slobodno da uradimo memory dump i da popravimo importe pomocu ImpReca i zavrsili smo brzo i lako i sa ovim pakerom.

Virogen Crypt 0.75

Na PELocku sam vam vec pokazao kako se konfigurise Olly da bi sam pronasao pravi OEP. Ovaj primer ce vam pokazati kako iskoristite ovu opciju kada vas Olly dovede samo na pola puta do OEPa. Meta na kojoj cu vam ovo pokazati se zove crackme #1.VirogenCrypt075.exe, kada je ucitamo u Olly on ce za nas naci "pravi" OEP. On izgleda ovako:

```
0040100E  > \66:B8 0000      MOV AX,0                    ; Real entry point of SFX code
00401012  . EB 00            JMP SHORT crackme_.00401014
00401014  >  40              INC EAX
00401015  . 83F8 64          CMP EAX,64
00401018  .^ 72 FA           JB SHORT crackme_.00401014
0040101A  . BB 64000000      MOV EBX,64
0040101F  . 3BC3             CMP EAX,EBX
00401021  . 8D3D F7104000    LEA EDI,DWORD PTR DS:[4010F7]
00401027  . 2BFB             SUB EDI,EBX
00401029  . FFE7             JMP EDI
```

Naravno ovo sigurno nije pravi OEP. Ako pocnemo da izvrsavamo ovaj dugacak loop:

```
00401018  .^\72 FA           JB SHORT crackme_.00401014
```

to ce potrajati, pa cemo postaviti jedan break-point odmah ispod ovog skoka. Postavicemo break-point na 0040101A i pritisnucemo F9. Kada se ovaj loop zavrsi mi cemo se naci na nasem break-pointu. Sada cemo izvrsiti sve do skoka JMP EDI sa F8 i kada izvrsimo i taj skok nacemo se ovde:

```
00401093  . 6A 00            PUSH 0
00401095  . 8D05 52304000    LEA EAX,DWORD PTR DS:[403052]
0040109B  . 50               PUSH EAX
0040109C  . 8D1D 00304000    LEA EBX,DWORD PTR DS:[403000]
004010A2  . 53               PUSH EBX
```

Ovo lici na sam OEP, a da li je ??? Naravno mozete probati i da izvrsite kod na dole i videcete da je ovo ustvari pravi OEP. Znaci adresa na kojoj mozemo izvrsiti dump je 00401093. Posle ovoga nam ostaje samo da popravimo importe.

NAPOMENA:

Kako znamo da je ovo pravi OEP ???

Pa i ne znamo ustvari mozemo samo da nagadjamo. Ali ako se dugo vremena bavite reversingom videcete da se sami OEPi razlikuju samo ako reversujete exe koji su kompajlovani sa drugim kompajlerima. Tako ce VB uvek imati isti OEP, Delphi isto, i svi ostali programski jezici ne racunajuci TASM / MASM jer kod njih OEP moze da izgleda proizvoljno i zavisi od samog kodera sto nije slucaj sa VB,Delphijem,...

EZIP 1.0

Ovo je jos jedan lak paker za odpakivanje. Otvorite primer zapakovan sa ovim pakerom koji se zove Unpackme12.EZIP1.exe pomocu Ollya. Prva linija pakera ce izgledati ovako:

```
004060BE > $ /E9 19320000          JMP Unpackme.004092DC
```

Izvrsite ovaj skok sa F8 i zavrsicete ovde:

```
004092DC  /> \55                   PUSH EBP
004092DD  |. 8BEC                   MOV EBP,ESP
004092DF  |. 81EC 28040000         SUB ESP,428
004092E5  |. 53                    PUSH EBX
004092E6  |. 56                    PUSH ESI
```

Mozete da traceujete kroz ovaj kod ali nema potrebe :) samo odskrolujte do samog kraja ovog CALLa i videcete sledece:

```
00409688  |. FFE0                  JMP EAX       <- BPX ON
0040968A  |> 5F                    POP EDI
0040968B  |. 5E                    POP ESI
0040968C  |. 5B                    POP EBX
0040968D  |. C9                    LEAVE
0040968E  \. C3                    RET
```

Sada samo treba da postavite jedan obican break-point na JMP EAX komandu, pritisnete F9 da bi ste dosli do te komande i izvrsite je sa F8. Posle ovoga cete se naci na nasem pravom OEPu, to jest ovde:

```
00401000   6A 00                   PUSH 0
00401002   E8 0D010000             CALL Unpackme.00401114
```

Sada mozete da uradite dump pomocu OllyDMP plugina. Default podesavanja su ok. Posle ovoga cete dobiti uspesno odpakovan EZIPovan fajl.

SPEC b3

Kao sto smo videli kod EZIPa, JMP EAX komanda nas kod nekih pakera moze odvesti do samog OEPa. Ovo cemo iskoristiti za odpakivanje SPEC b3a. Primer se nalazi u folderu Cas10, a zove se Notepad.SPEC3.exe. Ucitajte ga u Olly i samo odskrolujte dole dok ne naidjete na:

```
0040D14F   FFE0                    JMP EAX
```

Postavite break-point na JMP EAX, pritisnite F9 da bi ste dosli do tog break-pointa i sa F8 izvrsite skok i zavrsicete na OEPu. Posle ovoga samo uradite dump pomocu OllyDMP plugina i odpakivanje je gotovo :)

CExe 1.0a - 1.0b

Ovo je klasican primer kako ne treba pisati zastitu za vase aplikacije. Naime iako se fajl nalazi enkriptovan unutar samog .exe fajla on se prilikom dekripcije kompletno odpakuje na disk. Primer ove veoma lose zastite se nalazi u folderu Cas10 a zove se unpackme9.CExe.exe. Kako cete odpakovati program “zasticen” ovim kripterom ??? Jednostavno startujte zapakovani fajl i sacekajte da se on pokrene, kada se pokrene primeticete da se u istom direktorijumu nalazi jedan .tmp fajl. Vi samo treba da iskopirate taj tmp fajl gde zelite i preimenujete ga u .exe fajl. Dekriptovanje je gotovo, a nismo cak ni upalili Olly :)

MEW v.1.1 SE

Ovo je jos jedan lak paker za odpakivanje. Otvorite primer zapakovan sa ovim pakerom koji se zove unpackme18.MEW11SE.exe pomocu Ollya. Prva linija pakera ce izgledati ovako:

```
0040732B >- E9 248EFFFF          JMP unpackme.00400154
00407330    0C 60                OR AL,60
```

Izvrsite ovaj skok sa F8 i zavrsicete ovde:

```
00400154   BE 1C604000           MOV ESI,unpackme.0040601C
00400159   8BDE                  MOV EBX,ESI
0040015B   AD                    LODS DWORD PTR DS:[ESI]
0040015C   AD                    LODS DWORD PTR DS:[ESI]
```

Iako ovaj kod izgleda malo komplikovano odpakuje se za samo nekoliko sekundi. Odskrolujte do samog kraja ovog CALL, to jest do prve RET komande. Ona se nalazi ovde:

```
004001F0   FF53 F4               CALL DWORD PTR DS:[EBX-C]
004001F3   AB                    STOS DWORD PTR ES:[EDI]
004001F4   85C0                  TEST EAX,EAX
004001F6  ^ 75 E5                JNZ SHORT unpackme.004001DD
004001F8   C3                    RET
```

Secate se sta smo rekli ??? Do OEPa se moze doci pomocu JMP, CALL, RET komandi. Probacemo ovu teoriju u praksi i postavicemo jedan break-point na adresu 004001F8, to jest na RET komandu. Posle toga cemo izvrsiti sav kod koji je potreban da se stigne do nje sa F9. Kada konacno stanemo kod nje izvrsicemo i nju sa F8 i to ce nas odvesti ovde:

```
00401000    6A                   DB 6A                ; CHAR 'j'
00401001    00                   DB 00
00401002    E8                   DB E8
00401003    0D                   DB 0D
00401004    01                   DB 01
00401005    00                   DB 00
```

Da li je ovo mesto koje smo trazili ??? Da li je ovo OEP ??? Znam da vam ovo izgleda kao neke kuke i kvake ali ako analiziramo ovaj kod pritiskom na CTRL + A videcemo sta se to zaista nalazi na adresi 00401000. Videcemo ovo:

```
00401000  . 6A 00                PUSH 0
00401002  . E8 0D010000          CALL unpackme.00401114
00401007  . A3 00304000          MOV DWORD PTR DS:[403000],EAX
```

E ovo vec izgleda kao pravi OEP pa stoga ovde mozemo uraditi memory dump pomocu alata koji vi izaberete. Posto MEW nema nikakave api redirekcije ili kriptovanje API poziva dump i popravku APIa mozemo da uradimo preko standardnog OllyDMP plugina i sve ce biti ok.

PEBundle 2.0x - 2.4x

Sa pakerom Jeremy Collaka smo se vec sreli kada smo odpakivali PECompact. Sada je na redu jos jedan njegov paker PEBundle. Otvorite primer zapakovan sa ovim pakerom koji se zove unpackme21.PEBundle2x.exe pomocu Ollya. Prva linija pakera ce izgledati ovako:

```
00442000 > 9C                    PUSHFD
00442001   60                    PUSHAD
00442002   E8 02000000           CALL unpackme.00442009
```

Pritisnucemo F7 3x da bi smo usli prvi CALL. To ce nas odvesti ovde:

```
00442009   8BC4                  MOV EAX,ESP
0044200B   83C0 04               ADD EAX,4
```

sada cemo samo odskrolovati do samog kraja ovog CALLa, to jest do prve RET komande. Ovo radimo zato sto smo usli u CALL, a jedini nacin da se vratimo iz CALLa je pomocu RET komande. Prvu RET komandu cemo naci tek ovde:

```
00442466   61                    POPAD
00442467   9D                    POPFD
00442468   68 00B04200           PUSH unpackme.0042B000
0044246D   C3                    RET
```

Postavicemo jedan break-point na adresu 0044246D, to jest na samu RET komandu. Pritisnucemo F9 da bi smo dosli do te RET komande, a sa F8 cemo izvrsiti i tu RET komandu posle cega cemo se naci ovde:

```
0042B000   9C                    PUSHFD
0042B001   60                    PUSHAD
0042B002   E8 02000000           CALL unpackme.0042B009
```

Pritisnucemo F7 3x da bi smo usli i u ovaj CALL i ponovo cemo potraziti najblizu RET komandu. Ona se nalazi ovde:

```
0042B466   61                    POPAD
0042B467   9D                    POPFD
0042B468   68 EC154000           PUSH unpackme.004015EC
0042B46D   C3                    RET
```

Posle ovoga cemo postaviti break-point na RET komandu sa F9 cemo doci do nje i sa F8 cemo je izvrsiti, posle cega cemo zavrsiti ovde:

```
004015EC    68                   DB 68                  ; CHAR 'h'
004015ED    8C264000             DD unpackme.0040268C   ; ASCII "VB5!6&*"
004015F1    E8                   DB E8
004015F2    EE                   DB EE
```

Kao sto vidimo nalazimo se na OEPu koji i bez i sa analizom izgleda ovako kako izgleda jer je u pitanju Visual Basic aplikacija. Ovo je adresa na kojoj mozemo da uradimo dump pomocu LordPEa ili OllyDMPa. Posto PEBundle ne enkriptuje IAT mozemo da uradimo dump pomocu OllyDMPa koji ce za nas popraviti i importe.

ap0x Crypt 0.01 - aC

Posto predpostavljam da ste naucili kako se odpakuje vecina prednodnih pakera, sada cemo preci na odpakivanje mog licnog kriptera. Ova varijacija istog kriptera je samo ogoljena verzija onoga sto vas ceka na nightmare levelu ove knjige. Stoga bi odpakivanje aC bar na ovom nivou trebalo da bude setnja u parku :) Prvo cemo skenirati metu unpackme#1.aC.exe sa PeIDom da vidimo sta on ima da kaze o njoj. Posto mog kriptera nema u PeIDeovoj biblioteci on ce cutati po pitanju identifikacije ovoga kriptera. Nema veze neka nas to ne obeshrabri probacemo sa odpakivanje aC-a pomocu PeID generickog unpackera. Da vidimo, detect OEP - OK :), unpack OK, rebuild IAT ne zato sto IAT nije kriptovan i kada startujemo "raspakovani" fajl, crash. Naime ovo se desilo zato sto PeID nije detektovao tacan OEP. Izgleda da cemo morati ovaj kripter da odpakujemo rucno, otvorite Olly i pogledajte kako izgleda "OEP":

```
00401000 >/$ 60                PUSHAD
00401001  |. E8 E3000000        CALL unpackme.004010E9
00401006  \. C3                RET
```

Nista specijalno, pa cemo pritiskom 2x na F7 uci u prvi CALL da vidimo sta se tu desava. Kada to uradimo nacemo se ovde:

```
004010E9  /$ B8 F5104000       MOV EAX,unpackme.004010F5
004010EE  |. 50                PUSH EAX
004010EF  |. E8 A7FFFFFF       CALL unpackme.0040109B
004010F4  \. C3                RET
```

Ako pogledamo sada prvu liniju videcemo da se u nju smesta adresa 004010F5 ili adresa za koju je PeID mislio da je OEP. Ovo je izgleda ispala moja licna anti-generic zastita. Posto i ovo nije nista specijalno ucicemo u sledeci CALL sa F7. To ce nas dovesti ovde:

```
0040109B  /$ 50                PUSH EAX                 ; unpackme.004010F5
0040109C  |. 8BD8                MOV EBX,EAX
0040109E  |. B9 54010000         MOV ECX,154
004010A3  |> 8033 44             /XOR BYTE PTR DS:[EBX],44
004010A6  |. 83E9 01             |SUB ECX,1
004010A9  |. 43                  |INC EBX
004010AA  |. 83F9 00             |CMP ECX,0
004010AD  |.^ 75 F4              \JNZ SHORT unpackme.004010A3
004010AF  |. 50                  PUSH EAX
004010B0  |. E8 08000000         CALL unpackme.004010BD
004010B5  |. 50                  PUSH EAX
004010B6  |. E8 7EFFFFFF         CALL unpackme.00401039
004010BB  |. 58                  POP EAX
004010BC  \. C3                RET
```

Ovo izgleda malo ozbiljnije, ali opet nista komplikovano. Jednostavno crackme uzima adresu 004010F5 kao pocetnu i xoruje sledecih 154h bajtova sa vrednoscu 44h. Posle ovog loopa se poziva jos jedan CALL sa istim parametrom EAX. Da nebi smo izvrsavali loop 154h puta postavicemo break-point na adresu 00401AF odnosno na PUSH EAX komandu. Kada ovo uradimo pritisnucemo F9 da bi smo dosli do naseg break-pointa. Sada cemo sa F7 uci u sledeci CALL i nacemo se ovde:

```
004010BD  /$ 50                PUSH EAX                 ; unpackme.004010F5
004010BE  |. BB 07104000         MOV EBX,unpackme.00401007
004010C3  |. B9 7F000000         MOV ECX,7F
004010C8  |> 8033 07             /XOR BYTE PTR DS:[EBX],7
004010CB  |. 83E9 01             |SUB ECX,1
004010CE  |. 43                  |INC EBX
```

```
004010CF  |. 83F9 00                |CMP ECX,0
004010D2  |.^ 75 F4                 \JNZ SHORT unpackme.004010C8
0040 10D4  |. 8BD8                  MOV EBX,EAX
004010D6  |. B9 54010000            MOV ECX,154
004010DB  |>  8033 11               XOR BYTE PTR DS:[EBX],11
004010DE  |. 83E9 01                SUB ECX,1
004010E1  |. 43                     INC EBX
004010E2  |. 83F9 00                CMP ECX,0
004010E5  |.^ 75 F4                 JNZ SHORT unpackme.004010DB
004010E7  |. 58                     POP EAX
004010E8  \. C3                     RET
```

Kao sto vidimo u EBX se smesta adresa 00401007 i 7F bajtova od te adrese se xoruje sa 7h, posle cega se ponovo xoruje 154h bajtova od adrese 004010F5 sa 11h. Posto nema potrebe da prolazimo kroz sve ove loopove pritisnucemo CTRL + F9 sto ce izvrsiti ovaj CALL i vratice nas na sledecu adresu posle izvrsenja RET komande:

```
004010B5  |. 50                     PUSH EAX                 ; unpackme.004010F5
004010B6  |. E8 7EFFFFFF            CALL unpackme.00401039
004010BB  |. 58                     POP EAX
004010BC  \. C3                     RET
```

Kao sto vidimo ovo nas je odvelo do druge CALL komande u gornjem CALLu. Sada cemo sa F7 uci i u ovaj drugi CALL:

```
00401007  . /EB 27                  JMP SHORT unpackme.00401030
00401009  . |43 72 43 20 6>         ASCII "CrC of this file"
00401019  . |20 68 61 73 2>         ASCII " has been modifi"
00401029  . |65 64 20 21 2>         ASCII "ed !!!",0
00401030  > \EB 07                  JMP SHORT unpackme.00401039
00401032  .  45 72 72 6F 7>         ASCII "Error:",0
00401039  $  50                     PUSH EAX                 ; unpackme.004010F5
0040103A  .  8BD8                   MOV EBX,EAX
0040103C  .  B9 54010000            MOV ECX,154
0040 1041  .  BA 00000000           MOV EDX,0
00401046  >  0313                   ADD EDX,DWORD PTR DS:[EBX]
00401048  .  83E9 01                SUB ECX,1
0040104B  .  43                     INC EBX
0040104C  .  83F9 00                CMP ECX,0
0040104F  .^ 75 F5                  JNZ SHORT unpackme.00401046
00401051  .  B8 4A124000            MOV EAX,unpackme.0040124A
00401056  .  BE 80124000            MOV ESI,unpackme.00401280
0040105B  .  50                     PUSH EAX
0040105C  .  56                     PUSH ESI
0040105D  .  E8 28000000            CALL unpackme.0040108A
00401062  .  81FA B08DEB31          CMP EDX,31EB8DB0
00401068  .  74 19                  JE SHORT unpackme.00401083
0040106A  .  6A 30                  PUSH 30
0040106C  .  68 32104000            PUSH 00401032  "Error:"
00401071  .  68 09104000            PUSH 00401009  "CrC of this file has been modified !!!"
00401076  .  6A 00                  PUSH 0
00401078  .  E8 E5010000            CALL unpackme.00401262
0040107D  .  50                     PUSH EAX
0040107E  .  E8 F1010000            CALL unpackme.00401274
00401083  >  E9 96010000            JMP unpackme.0040121E
00401088  .  58                     POP EAX
00401089  .  C3                     RET
```

Ovo je malo duzi CALL ali sam ga ja podelio na celine kako bi ste vi lakse shvatili sta se ovde desava. Analizu cemo raditi po obelezenim bojama.

Zeleni deo predstavlja samo poruke koje ce se prikazati ako smo modifikovali ovaj fajl, a posto ovo nismo uradili o ovome necemo brinuti.

Narandzasti deo predstavlja kod za racunanje “CRCa” dela koda koji pocinje na adresi 004010F5 a dugacak je 154h i predstavlja glavni deo koda. Ni ovo nam nije bitno posto nismo modifikovali ovaj deo koda nego se bavimo samo

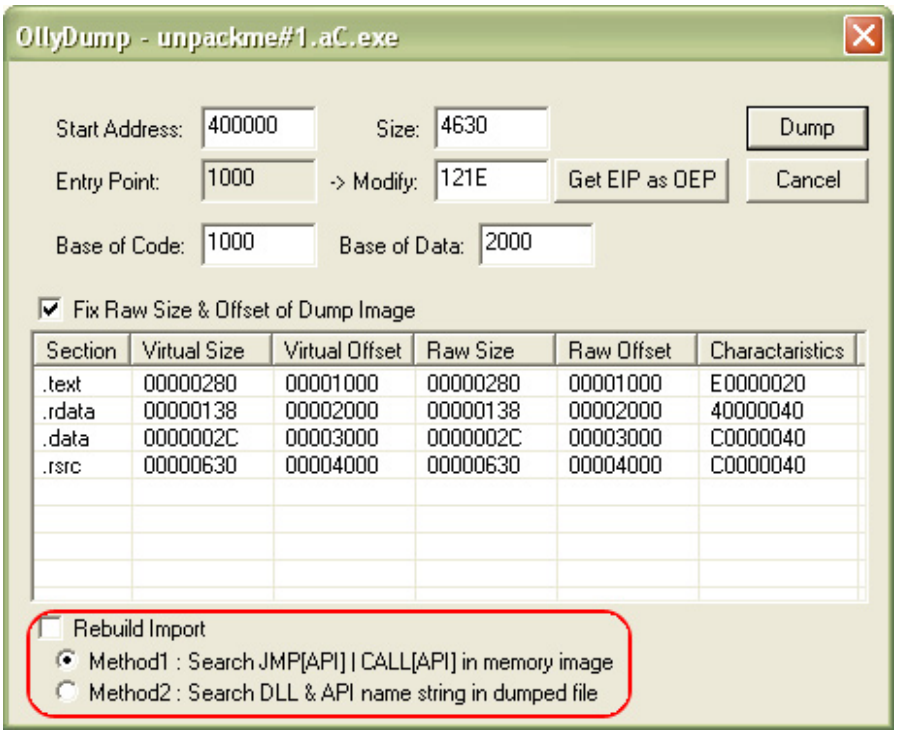
odpakivanjem. Roze deo koda predstavlja novi CALL sa parametrima 0040124A i 00401280 a izgleda ovako:

```
...
00401090  |>  8033 17              /XOR BYTE PTR DS:[EBX],17
00401093  |.  43                   |INC EBX
00401094  |.  3BD9                 |CMP EBX,ECX
00401096  |.^ 75 F8                \JNZ SHORT unpackme.00401090
...
```

Ako analiziramo malo ovaj kod videcemo da se adrese od 0040124A do 00401280 xoruju sa 17h i da taj deo koda predstavljaju redirekcije ka API funkcijama.
Svetlo plavi deo koda poredi EDX sa vrednoscu tacnog CRCa sekcije od 004010F5 i dugacke 154h bajtova. Ako su CRC vrednosti iste onda se skace na tamno plavi deo koda koji vodi ovde:

```
0040121E > $  6A 00                 PUSH 0
...
0040123E   .  E8 0D000000           CALL <JMP.&user32.DialogBoxParamA>
```

Odnosno na pravi OEP. Vidimo da se ovde poziva API funkcija za prikazivanje dialoga - DialogBoxParamA, zbog cega cemo ovde uraditi dump pomocu OllyDMG plugina. Podesite ovaj plugin ovako:



zato sto IAT tabela nije kriptovana nego je kriptovan samo deo fajla. Neophodno je da razumete kako smo dosli do OEPa ovog kriptera kako bi ste bili u stanju da nadjete OEP bilo kojeg kriptera ili pakera, a ovo ce nam trebati kasnije u dvanaestom poglavlju zato predlazem da pazljivo procitate ovaj postupak.

11 Patching it...

Ovo je veoma bitno poglavlje vezano za cracking jer obradjuje vecinu trenutno koriscenih nacina za modifikaciju gotovih exe fajlova. Ovde ce biti objasnjeni nacini funkcionisanja i pravljenja file patchera i memory patchera. Takodje ce biti objasnjeni i principi patchovanja zapakovanih exe fajlova.

+ “Hard” patchers

Fizicki patchevi su patchevi koji se primenjuju direktno na fajlove tako da kod samog patchovanog fajla biva modifikovan fizicki na disku. Ova vrsta patcheva ima svoje podvrste koje se primenjuju u zavisnosti od slucaja koji se reversuje. Najvesce podvrste fizickih patchera su:

- Obicni fajl patcheri
- Patcheri koji podrzavaju vise verzija istog programa
- Seek and Replace patcheri koji traze odredjene paterne u fajlovima koje patchuju i zamenjuju ih novim bajtovima. Ovi paterni su ustvari nizovi bajtova koji mogu sadrzati i jocker karaktere koji zamenjuju bilo koji bajt.

+ Registry patchers

Ovo je vrsta patchera koja vrsi manipulaciju nad sistemski registrijem. Ova vrsta patchera moze biti dosta kompleksnija od obicnih .reg fajlova. Tako na primer ovi patcheri mogu vrsiti backup odpredjenih kljuceva, mogu raditi periodicno brisanje i / ili modifikaciju celih grana kljuceva. Jedina mana ovih patchera je to sto se najcesce prave posebno za svaku aplikaciju. Ova vrsta patchera nema svoje pod vrste.

+ Memory patchers

Posebna vrsta patchera koja se bavi modifikacijom radne RAM memorije. Ovi patchevi su korisni u slucajevim kada nije moguce napraviti obican “hard” patch. Do ovakve situacije dovodi pakovanje “meta” externim pakerima ili enkripcijom sadrzaja fajla. Tada se prave takozvani loaderi koji staruju program i posle odpakivanja programa ili njegove dekripcije u memoriju vrse patchovanje dekriptovanih / odpakovanih memoriskih adresa. Ovi patcheri mogu biti podeljeni u tri pod vrste:

- Game traineri
- Obicni memoriski patcheri (*ustvari isto sto i game traineri*)
- Loaderi koji mogu biti konfigurisani tako da vrse patchovanje posle odredjenog broja sekundi / milisekundi, oni takodje mogu pauzirati procese koje patchuju da bi posle patchovanja nastavili sa radom procesa.

Ova sistematizacija je uradjena iz razloga sto cemo dalje kroz ovo poglavlje biti suoceni sa razlicitim problemima koje cemo morati da resimo. Dalje kroz poglavlje cu vam pokazati kako se patchuju razlicite vrste pakera i kako mozete modifikovati memoriju.

Making an inline patch

Kao sto je vec objasnjeno loaderi se koriste kada zelimo da napravimo patch za pakovane ili kriptovane programe. Njih je veoma lako napraviti ali posto u vecini slucajeva nisu pouzdani koristicemo klasicne patcheve na pakovanim programima. Zvuci li interesantno??? Program specijalno pisan za potrebe ovog poglavlja se nalazi u folderu Cas11 a zove se NAG.exe. Otvoricemo ovaj fajl pomocu Ollya da bi smo nasli mesto koje treba da patchujemo. Da bi smo nasli mesto koje treba da patchujemo, prvo cemo odpakovati program i nacicemo bajt koji zelimo da patchujemo. Sam NAG se nalazi ovde:

```
00407F1F  /74 19          JE SHORT NAG_exe_.00407F3A
00407F21  |6A 40          PUSH 40
00407F23  |68 507F4000    PUSH NAG_exe_.00407F50          ; ASCII "NAG:"
00407F28  |68 587F4000    PUSH NAG_exe_.00407F58          ; ASCII "Ovo je NAG screen
koji treba da ubijete !!!"
00407F2D  |53             PUSH EBX
00407F2E  |E8 B5C6FFFF    CALL <JMP.&USER32.MessageBoxA>
```

Ocigledno je da treba samo da patchujemo skok JNZ u JMP, to jest sa 74 u EB. Potrebni podaci za inline patch su adresa 00407F1F i vrednost sa kojom cemo da patchujemo. Potrebno je samo da nadjemo mesto gde cemo da patchujemo. Ovo moramo da uradimo tik posle sto se program odpakuje. Pre nego sto skoci na sam OEP. Stoga cemo prepraviti skok ispod donje POPAD komande UPX pakera tako da nas preusmeri na kod za patchovanje. Evo kako bi to trebalo da izgleda (*narandzasti su izmenjeni / dodati redovi*)

```
0041291E  > \61           POPAD
0041291F  . EB 14         JMP SHORT NAG2.00412935
00412921    90            NOP
00412922    90            NOP
00412923    90            NOP
00412924    3C294100      DD NAG2.0041293C
00412928    44294100      DD NAG2.00412944
0041292C    08A74000      DD NAG2.0040A708
00412930    00            DB 00
00412931    00            DB 00
00412932    00            DB 00
00412933    00            DB 00
00412934    00            DB 00
00412935    C705 1F7F4000> MOV DWORD PTR:[407F1F],019EB
0041293F  .- E9 B056FFFF  JMP NAG2.00407FF4
```

Vidite sta se desava. Izmenili smo skok tako da on vodi ka delu za patchovanje, posle cega se sa starim UPX skokom vracamo na OEP. Primeticete oblik komande za patchovanje memorije:

MOV DWORD PTR:[407F1F],019EB

Mislim da je jasno sta ovde radimo. Na adresu 00407F1F cemo ubaciti bajtove EB19. Primetite samo da su bajtovi u obrnutom redosledu, prvo ide 19 pa EB, a uvek ce im predhoditi nula ako patchujete pomocu Ollya. Bitno je da zapamtite da ako nema mesta za dodavanje bajtova odmah iza patcher koda, morate osloboditi mesto negde pomocu nekog Hex Editora. Ovaj postupak je vec objasnjen kod dodavanja funkcija tako da nema potrebe da ovo ponovo objasnjavam.

Making a loader

Kao sto je vec objasnjeno loaderi se koriste kada zelimo da napravimo patch za pakovane ili kriptovane, ali mi cemo napraviti loader za obican crackme.

Kao sto vidimo primer koji se nalazi u folderu Cas11 a zove se LOADME.exe prikazuje poruku da nije crackovan. Ovo znaci da se negde u programu nalazi poruka o tacno crackovanom programu. Ta poruka se nalazi ovde:

```
00401052  . 47 6F 6F 64 2>          ASCII "Good Cracker !!!"
00401062  . 00                      ASCII 0
```

A da se ona prikazuje ovde:

```
004010BF  . BB 01000000             MOV EBX,1
004010C4  . 83FB 01                 CMP EBX,1
004010C7  . 75 11                   JNZ SHORT LOADME.004010DA
004010C9  . 68 40104000             PUSH LOADME.00401040   ; /Text = "Bad Cracker !!!"
004010CE  . 6A 64                   PUSH 64
004010D0  . FF75 08                 PUSH DWORD PTR SS:[EBP+8]
004010D3  . E8 96000000             CALL <JMP.& user32.SetDlgItemTextA>
004010D8  . EB 15                   JMP SHORT LOADME.004010EF
004010DA  > 837D 10 02              CMP DWORD PTR SS:[EBP+10],2
004010DE  . 75 0F                   JNZ SHORT LOADME.004010EF
004010E0  . 68 52104000             PUSH LOADME.00401052   ; /Text = "Good Cracker !!!"
004010E5  . 6A 64                   PUSH 64
004010E7  . FF75 08                 PUSH DWORD PTR SS:[EBP+8]
004010EA  . E8 7F000000             CALL <JMP.&user32.SetDlgItemTextA>
```

Patchovacemo skok na adresi 004010C7 JNZ u JMP, to jest sa 75 11 u EB 11 i drugi skok na adresi 004010DE sa 75 0F u 90 90. Ovde necemo uraditi fizicki nego memoriski patch. Za ovo cemo iskoristiti R!SC Process Patcher koji se nalazi zipovan u folderu Cas11 kao fajl rpp.zip. Da bi ste napravili loader morate prvo napraviti jedan .rpp fajl. Njegov sadrzaj ce izgledati ovako:

```
F=LOADME.exe:                ; PROCESS TO PATCH
O=loader.exe:                ; LOADER TO CREATE
P=4010C7/75,11/EB,11:        ; JNZ 2 JMP
P=4010DE/75,0F/90,90:        ; JNZ 2 NOP
$
```

Posle ovoga mozete startovati Rpp.exe i pomocu njega kompajlovati ovaj .rpp skript. Kao rezultat dobicete fajl loader.exe koji ce uspesno patchovati memoriju LOADME.exe fajla. Sama struktura .rpp fajla je jednostavna pa je necu objasnjavati, a ako vam na prvi pogled nije jasno uporedite adrese i bajtove u redovima koji pocinju sa P=.

12 Nightmare

Ovo poglavlje je koncepciski zamisljeno malo drugacije od ostalih. U predhodnim poglavljima sam vam detaljno objasnjavao probleme vezane za razlicite RCE probleme. Sada cu vam ja zadavati probleme koje cete morati sami da resite u skladu sa vasim mogucnostima. Naravno i ovde ce biti objasnjena resenja svih problema samo sto se nadam da cete ih slabije citati jer ste do sada trebali sami da shvatite sustinu RCEa i da se osamostalite tako da mozete sami da resavate probleme. Shvatite ovo poglavlje kao jedan konacan ispit.

Keygening Scarabee #4

Zadatak:

Relativno lak zadatak ...\Cas12\crackme#4.exe treba reversovati tako da se ukljuci Check dugme i da posle pritiska na ovo dugme program prikazuje poruku da je program registrovan. Dozvoljeno je patchovati samo bajtove koji sluze za iskljucivanje / ukljucivanje dugmeta. Pored ovoga treba otkriti nacin na koji se racuna seriski broj i napraviti keygenerator.

Resenje:

Da bi smo ukljucili dugme Check iskoristicemo program DeDe. Ako nemate ovaj program samo patchujete program na objasnjeni nacin.

Ucitajte metu u DeDe i u procedurama izaberiti FormCreate i videcete sledece:

```
0044FBA8  33D2                    xor    edx, edx
* Reference to control TForm1.Button1 : TButton
0044FBAA  8B80F0020000            mov    eax, [eax+$02F0]
0044FBB0  8B08                    mov    ecx, [eax]
* Reference to method TButton.SetEnabled(Boolean)
0044FBB2  FF5164                  call   dword ptr [ecx+$64]
0044FBB5  C3                      ret
```

Kao sto se vidi iz prilozenog kada se kreira forma dugme se iskljuci pozivom na komandu CALL DWORD ptr[ecx+$64], zbog cega cemo NOPovati ovaj CALL. Posle ovoga mozemo otvoriti Olly i nastaviti sa reversingom ove mete. Potrazicemo sve moguce stringove koji bi nam ukazali na to gde se to racuna seriski broj. Videcemo ovu zanimljivu referencu:

```
Text strings referenced in crackme#:CODE, item 1850
 Address=0044FB5D
 Disassembly=MOV EDX,crackme#.0044FB98
 Text string=ASCII "  Registered!"
```

koja ce nas odvesti ovde:

```
0044FB5D  |.  BA 98FB4400   MOV EDX,crackme#.0044FB98          ;  ASCII "  Registered!"
0044FB62  |.  E8 4DF1FDFF   CALL crackme#.0042ECB4
```

Ovo znaci da se CALL koji se koristi za racunanje seriskog broja nalazi izmedju adresa 0044FA64 i 0044FB8C.

Postavicemo break-point na adresu 0044FA64 kako bi smo analizirali deo koda koji sluzi za proveru seriskog broj i pritisnucemo dugme Check u samom programu posle cega cemo zavrsiti na nasem break-pointu. Procicemo kroz kod 1x pritiskajuci F8 da bi smo saznali sto vise o kodu za proveru seriskog broja. Primeticemo sledece:

```
0044FA8D  |.  8B45 F8         MOV EAX,DWORD PTR SS:[EBP-8] ; U EAXu je putanja do fajla
0044FAA3  |.  83F8 15         CMP EAX,15                   ; U EAXu je duzina putanje
0044FAA6  |.  0F85 BB000000   JNZ crackme#.0044FB67        ; EAX se poredi sa 15h
```

Kada se ovaj skok izvrsi zavrsicemo odmah ispod poruke o tacnom seriskom broju zbog cega se ovaj skok nikada ne sme izvrsiti. Zbog ovoga cemo ovaj crackme iskopirati u folder c:\aaaaaaaaaaaaaaaaaa i reversovacemo ga tu. Ovo radimo jer duzina putanje do exe fajla mora da bude 15h ili 21 decimalno a posto se racuna i c:\ u duzinu putanje folder mora biti bas ovaj ili neki drugi sa istom duzinom. Sada cemo pomocu Ollya otvoriti fajl koji se nalazi na ovoj novoj lokaciji. Primeticete da se sada ne izvrsava skok na adresi 0044FAA6 i da cemo malo ispod toga naici na sledece provere:

```
0044FAAC  |.  8B45 F8         MOV EAX,DWORD PTR SS:[EBP-8]
0044FAAF  |.  8078 09 5C      CMP BYTE PTR DS:[EAX+9],5C
```

```
0044FAB3  |. 0F85 AE000000  JNZ crackme#.0044FB67
0044FAB9  |. 8B45 F8        MOV EAX,DWORD PTR SS:[EBP-8]
0044FABC  |. 8078 0F 5C     CMP BYTE PTR DS:[EAX+F],5C
0044FAC0  |. 0F85 A1000000  JNZ crackme#.0044FB67
```

Sledece dve provere na adresama 0044FABC i 0044FAAF su nam jako bitne. Primetite da se odredjena slova iz imena porede sa vrednoscu 5C odnosno sa karakterom '\'. Ovi bajtovi se nalaze na adresama 008A3701 i 008A3707. Ovo ujedno znaci da se 9 i 15 slovo iz putanje do fajla moraju biti '\' a ovo znaci da se fajl nalazi u dodatnim poddirektorijumima zbog cega cemo ovaj exe prebaciti u sledeci folder C:\aaaaaa\bbbba\ccccc i primeticemo da se sada oba JNZ skoka ne izvrsavaju i da cemo nastaviti sa analizom koda od adrese 0044FAC6. Dok polako izvrsavate kod videcete u donjem desnom delu CPU prozora Ollya sledece:

```
0012F3EC  008A2078  ASCII "aaaaaabbbba"
0012F3F0  008A2064  ASCII "bbbba"
0012F3F4  008A2050  ASCII "aaaaaa"
0012F3F8  008A36F8  ASCII "C:\aaaaaa\bbbba\ccccc"
```

Ono sto vidimo je da se putanja rastavlja na poddirektorijume i da se imena prvog i drugog poddirektorijuma slepljuju u jedan string. Docicemo do sledeceg loopa koji se koristi za izaracunavanje seriskog broja.

```
0044FB13  |> /8B45 EC       /MOV EAX,DWORD PTR SS:[EBP-14]
0044FB16  |. |0FB64408 FF   |MOVZX EAX,BYTE PTR DS:[EAX+ECX-1]
0044FB1B  |. |0FAF45 DC     |IMUL EAX,DWORD PTR SS:[EBP-24]
0044FB1F  |. |0145 E0       |ADD DWORD PTR SS:[EBP-20],EAX
0044FB22  |. |41            |INC ECX
0044FB23  |. |4A            |DEC EDX
0044FB24  |.^\75 ED         \JNZ SHORT crackme#.0044FB13
```

Posle izvrsenja ovog loopa i par komandi ispod njega u donjem desnom delu CPU prozora cemo videti:

```
0012F3E8  008A2090  ASCII "71757"
```

tacan seriski broj za ovu putanju. Ispod ovoga sledi poredjenje dve vrednosti

```
0044FB47  |. 8B45 E4        MOV EAX,DWORD PTR SS:[EBP-1C]
0044FB4A  |. 8B55 E8        MOV EDX,DWORD PTR SS:[EBP-18]
0044FB4D  |. E8 2A4BFBFF    CALL crackme#.0040467C
```

preostalog dela putanje "ccccc" i "71757". Ovo znaci da kranji folder koji smo mi nazvali "ccccc" treba da se zove "71757". Zatvoricemo Olly i reimenovacemo ovaj direktorijum u "71757" posle cega cemo videti sledece:

Ovo znaci da smo uspeli, uspesno smo pronasli resenje ovog crackmea. Interesantan je, zar ne? Sada nam samo preostaje da napravimo keygenerator za ovaj crackme.

Keygen:

Vec smo pronasli gde se to racuna seriski broj a sada smo treba da razumemo taj isti algoritam i da ga prepisemo u nekom drugom programskom jeziku. Za ovaj primer ja cu koristiti samo Delphi. Pre nego sto pocnemo da se bavimo pravljenjem keygeneratora moracemo prvo da se pozabavimo samim algoritmom da bi smo saznali sto je vise moguce o

samom algoritmu. Evo kako izgleda deo koda koji sluzi za generisanje tacnog seriskog broja:

```
0044FB13  |> /8B45 EC         /MOV EAX,DWORD PTR SS:[EBP-14]
0044FB16  |. |0FB64408 FF     |MOVZX EAX,BYTE PTR DS:[EAX+ECX-1]
0044FB1B  |. |0FAF45 DC       |IMUL EAX,DWORD PTR SS:[EBP-24]
0044FB1F  |. |0145 E0         |ADD DWORD PTR SS:[EBP-20],EAX
0044FB22  |. |41              |INC ECX
0044FB23  |. |4A              |DEC EDX
0044FB24  |.^\75 ED           \JNZ SHORT crackme#.0044FB13
```

Objasnicemo sta se ovde desava red po red.

- Prvo se na adresi 0044FB13 u EAX smestaju imena prva dva direktorijuma dodata jedno na drugo.
- Na adresi 0044FB16 se u EAX smesta jedno po jedno slovo stringa dobijenog sabiranjem imena foldera. EAX sadrzi hex vrednost ASCIIa tog slova a ne sam ASCII !!!
- Potom se EAX mnozi sa sadrzajem adrese EBP-24. Da bi smo saznali sa cime se to mnozi EAX selektovacemo sledeci red sa ekrana:

  ```
  Stack SS:[0012F3DC]=00000043
  EAX=0000004C
  ```

  i pritisnucemo desno dugme -> Follow in dump... sto ce nas odvesti ovde:

  ```
  0012F3DC  43 00 00 00 00 00 00 00  C.......
  ```

  kao sto vidimo EAX se mnozi sa ASCIIjem slova uredjaja na kojem se nalazi crackme (e.g. C:\, D:\,...)
- Na adresi 0044FB1F se na sadrzaj adrese EBP-20 dodaje vrednost EAXa. Posto je u prvom prolazu EBP-20 jednak nuli zakljucujemo da ce se ovde naci rezultat svih sabiranja EAXa + EBP-20.
- Ostali redovi nam samo govore koliko ce se puta izvrsiti ovaj loop. Taj broj je jednak duzini stringa koji se dobije dodavanjem imena dva foldera jednog na drugo.

Mislim da je sada svima jasno kako treba napraviti keygen. Prvo treba uneti dva bilo koja imena foldera tako da je duzina prvog imena foldera sest a drugog pet. Posle ovoga koristeci formulu iz loopa treba odrediti ime treceg podfoldera koji ce se sastojati samo od brojeva. Ovaj algoritam ce izgledati bas ovako:

```
var
 drv,nrd:string;
 i,tmp,eax:integer;
begin
  Drv := ComboBox1.Text;
  nrd := '';
 for i := 1 to 11 do begin
  nrd := nrd + Chr(65 + Random(25));
 end;
  tmp := 0;
  eax := 1;
 for i := 1 to length(nrd) do begin
  eax := Ord(nrd[i]);
  eax := eax * ord(Drv[1]);
  tmp := tmp + eax;
 end;
  nrd := Drv + nrd[1] + nrd[2] + nrd[3] + nrd[4] + nrd[5] + nrd[6] + '\' + nrd[7] + nrd[8] +
nrd[9] + nrd[10] + nrd[11] + '\' + IntToStr(tmp);
  Edit1.Text := nrd;
```

Ceo source i kompajlovani .exe fajl se nalaze u folderu ..\Cas12\KeyGen\

Patching aC 0.1

Vec sam vam objasnio kako da odpakujete aC 0.1 u poglavlju koje se bavilo odpakivanjem. Sada pred vas stavljam sledeci zadatak:

Zadatak:
Fizicki (*bez loadera*) patchovati NAG u fajlu ...\Cas12\unpackme#1.aC.exe bez odpakivanja na disk i bez modifikacije skoka posle provere CRCa sekcije.

Resenje:
Odpakujmo prvo crackme u memoriju. Posto je ovo vec objasnjeno recicu samo da se trenutno nalazimo na OEPu 0040121E. Ako pogledamo sada adresu:

```
0040122C  . 68 F5104000          PUSH unpackme.004010F5
```

videcemo da se glavna procedura za obradu WM poruka nalazi bas na adresi 004010F5. Sada cemo pogledati sta se tamo nalazi:

```
...
004011B4  . 6A 40                PUSH 40
004011B6  . 68 41114000          PUSH unpackme.00401141
004011BB  . 68 23114000          PUSH unpackme.00401123
004011C0  . FF75 08              PUSH DWORD PTR SS:[EBP+8]
004011C3  . E8 9A000000          CALL <JMP.&user32.MessageBoxA>
004011C8  . 33C0                 XOR EAX,EAX
```

Kao sto vidimo ovde se nalazi nas NAG. Posto program ne smemo odpakivati restartovacemo ga sa CTRL + F2 i oticemo na adresu 004011B4 da bi smo pogledali sta se to nalazi na toj adresi kada je program kriptovan. Vidimo:

```
004011B4    3F                   DB 3F                ; CHAR '?'
004011B5    15                   DB 15
004011B6    3D                   DB 3D                ; CHAR '='
004011B7    14                   DB 14
004011B8    44                   DB 44                ; CHAR 'D'
004011B9    15                   DB 15
004011BA    55                   DB 55                ; CHAR 'U'
004011BB    3D                   DB 3D                ; CHAR '='
004011BC    76                   DB 76                ; CHAR 'v'
004011BD    44                   DB 44                ; CHAR 'D'
004011BE    15                   DB 15
004011BF    55                   DB 55                ; CHAR 'U'
004011C0    AA                   DB AA
004011C1    20                   DB 20                ; CHAR ' '
004011C2    5D                   DB 5D                ; CHAR ']'
004011C3    BD                   DB BD
004011C4    CF                   DB CF
004011C5    55                   DB 55                ; CHAR 'U'
004011C6    55                   DB 55                ; CHAR 'U'
004011C7    55                   DB 55                ; CHAR 'U'
```

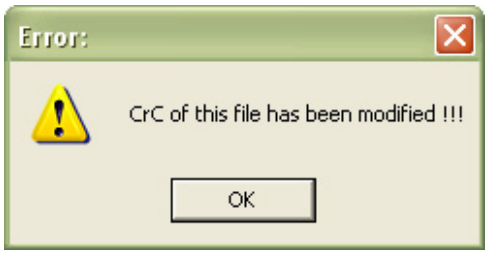
veoma kriptovan kod. Posto sve ovo moramo da promenimo u NOP moramo da znamo kako se enkriptuje 90 da bi smo znali u sta da promenimo ove bajtove. Da bi smo ovo saznali moramo da pogledamo kod kriptera i da vidimo u kako se kriptuju adrese od 004010F5. Za to su zaduzeni sledeci loopovi:

```
004010A3  |>  8033 44            /XOR BYTE PTR DS:[EBX],44
004010A6  |.  83E9 01            |SUB ECX,1
004010A9  |.  43                 |INC EBX
004010AA  |.  83F9 00            |CMP ECX,0
004010AD  |.^ 75 F4              \JNZ SHORT unpackme.004010A3
```

i sledeci loop:

```
004010DB  |> /8033 11          XOR BYTE PTR DS:[EBX],11
004010DE  |. |83E9 01          SUB ECX,1
004010E1  |. |43               INC EBX
004010E2  |. |83F9 00          CMP ECX,0
004010E5  |.^\75 F4            JNZ SHORT unpackme.004010DB
```

Sada imamo dovoljno informacija da bi smo mogli da napravimo patch. Iz ova dva loopa smo saznali da se adrese od 004010F5 xoruju prvo sa 44h a onda sa 11h. Posto posle dekripcije bajtovi od 004011B4 do 004011C7 moraju da budu 90 uradicemo reversni xor **90h xor 11h xor 44h = C5h** a kao rezultat cemo dobiti bajt koji treba da se nalazi na svim adresama izmedju 004011B4 i 004011C7. Ove izmene mozemo da uradimo pomocu Ollya i da ih snimimo. Kada ovo uradimo startovacemo novo-snimljeni fajl i videcemo da



smo zaboravili na cinjenicu da ovaj kripter ima mogucnost provere modifikacije koda ispod adrese 004010F5. Ali ono sto ovaj kripter nema je mogucnost provere CRCa iznad adrese 004010F5 a bas ovde se nalazi kod za proveru modifikacije sekcije koda. Posto bi bilo previse lako samo modifikovati skok JE u JNE ja sam kao uslov zadatka postavio da se ovaj skok ne sme modifikovati. Iako ovo zvuci komplikovano uopste nije. Otvoricemo patchovani program pomocu Ollya i docemo do CMP komande koja se nalazi odmah iznad ovog skoka. Ovde smo:

```
00401062  . 81FA B08DEB31          CMP EDX,31EB8DB0
```

Primeticemo da se sadrzaj EAXa (*F9B35572*) poredi sa 31EB8DB0. Stoga cemo samo logicno promeniti CMP komandu u

```
00401062    81FA 7255B3F9          CMP EDX,F9B35572
```

Pre nego sto ovo uradimo moramo da nadjemo kako se adresa na kojoj se nalazi CMP komanda dekriptuje. Odgovor lezi ovde:

```
004010BE  |. BB 07104000           MOV EBX,unpackme.00401007
004010C3  |. B9 7F000000           MOV ECX,7F
004010C8  |> 8033 07               /XOR BYTE PTR DS:[EBX],7
004010CB  |. 83E9 01               |SUB ECX,1
004010CE  |. 43                    |INC EBX
004010CF  |. 83F9 00               |CMP ECX,0
004010D2  |.^ 75 F4                \JNZ SHORT unpackme.004010C8
```

Odnosno dekriptcija bajtova izgleda ovako:

```
86 xor 7 = 81                      81 xor 7 = 86
FD xor 7 = FA                      FA xor 7 = FD
B7 xor 7 = B0                      72 xor 7 = 75
8A xor 7 = 8D                      55 xor 7 = 52
EC xor 7 = EB                      B3 xor 7 = B4
36 xor 7 = 31                      F9 xor 7 = FE
```

gde je prva kolona orginalna dekripcija a druga patchovana enkripcija. U prevodu treba samo patchovati bajtove na adresi 00401062 u 86FD7552B4FE da bi smo prilikom dekripcije na istoj adresi dobili:

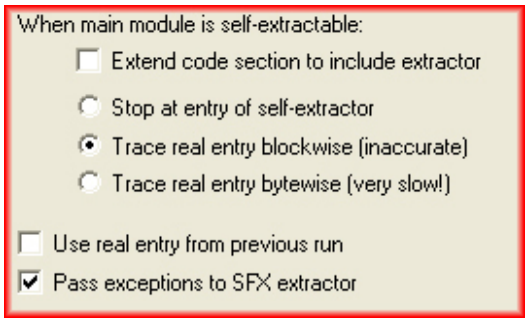
```
00401062    81FA 7255B3F9          CMP EDX,F9B35572
```

Posle ovog patchovanja mozemo pokrenuti patchovani fajl i videti da se na ekranu sada ne pojavljuje NAG. Dakle, uspeli smo !!!

Unpacking Obsidium 1.2

Iako je u knjizi objasnjeno kako se odpakuje dosta velik broj pakera u ovom nightmare delu knjige cete morati da odpakujete paker koji do sada niste imali prilike da sretnete u ovoj knjizi.

Zadatak:
Odpakovati fajl ...\Cas12\crackme.Obsidium12.exe tako da se u fajl PE (PEid i slicnima) identifikatorima prikaze pravo stanje a ne identifikacija pakera. Fajl posle odpakivanja ne sme prikazivati Obsidium NAG poruku i mora raditi bez izvrsavanja koda pakera Obsidium.



Resenje:
Iako sam ovom pakeru dao visu ocenu, tacnije dao sam mu level 4, paker je lak za odpakivanje kada se Olly lepo podesi. Podesite Olly kao na slici. I prvo cete se sresti sa NAGom Obsidiuma a posle toga cete zavrsiti na OEPu. Kao sto se vidi u samom Olly Obsidium je "progutao" par bajtova sa OEPa to jest oni su izvrseni negde drugde. Nas zadatak je da iskoristimo ovaj OEP sto bolje znamo i umemo:

```
004012C0  /. 55              PUSH EBP
004012C1  |. 8BEC            MOV EBP,ESP
004012C3  |. 6A FF           PUSH -1
004012C5  |. 68 F8404000     PUSH crackme_.004040F8
004012CA  |. 68 F41D4000     PUSH crackme_.00401DF4          ; SE handler installation
004012CF  |. 64:A1 0000000>  MOV EAX,DWORD PTR FS:[0]       ; Real entry point of SFX
```

Iako ste sigurno pomislili da je Olly prilikom traceovanja progutao ove bajtove i da se zbog toga nalazimo na 004012CF umesto na 004012C0. E pa Olly nije nista progutao jer Obsidium bas ovako radi kradju bajtova: Negde se izvrsi sve sa adrese 004012C0 do 004012CF a to se ne radi ovde. Nama samo predstavlja problem to sto ako sada uradimo dump imacemo iskvarene adrese jer se deo koda vec izvrsio a dump moramo da radimo kada se ni jedna komanda pravog nepakovanog ExE fajla neizvrsi. Ali mozda mozemo da uradimo dump ovde ? Pogledajmo malo bolje adrese od 004012C0 do 004012CF da vidimo sta se tu desava. Nista specijalno nema nikakvih CALLova ili operacija sa registrima, imamo samo par PUSH komandi, dakle tipican VC++ OEP :) Ono sto sigurno niste znali je da ako se izvrse PUSH komande njihovo stanje se snima u memoriji a to znaci da nema veze da li cemo uraditi dump ovde ili na 004012C0. Dakle uradicemo dump ovde i pokusacemo da popravimo importe sa ImpRecom. Kao sto cete sigurno primetiti ImpRec je nasao OEP i importe ali ne moze da ih identifikuje. Mozete probati i sa obsidiumIAT pluginom ali nista necete postici. Dakle Importe moramo popraviti rucno. Ovo necu ovde objasniti jer zahteva podpuno znanje API referenci i potrajalo bi dosta dugo, stoga cu vam samo dati fajl iat.txt koji sadrzi ceo IAT nepakovanog fajla. Vi samo treba da ga ucitate u ImpRec pomocu opcije Load Tree i uradite Fix dump. E zato je ovo level 4...

13 Tricks of Trade

Ovo je poslednje poglavlje knjige posvecene reversnom inzinjeringu. U njemu ce biti obradje teme vezane za zastitu softwarea od “lakog” reversnog inzenjeringa. U ovom poglavlju cu se potruditi da vam predocim veoma ceste programerske, ali i crackerske zablude...

Coding Tricks

Shvatite ovaj deo poglavlja kao deo za programere koji zele da pisu sigurnije aplikacije, koje ce nama, reverserima, zadata dosta glavobolje kako bi smo ih reversovali.

Mark's famous protector's commandments

- Nikada ne koristite imena fajlova ili imena procedura tako da ona sama po sebi imaju smisla za reversere, npr. IsValidSerialNum. Ako vec ne zelite da se odreknete ove dobre programerske navike onda bar iskodirajte vas program tako da on zavisi od iste funkcije, tako da se njenim ne izvrsavanjem srusi.
- Ne upozoravajte korisnika da je uneti seriski broj netacan odmah nakon provere. Obavestite korisnika posle dan ili dva, crackeri mrze to :)
- Koristite provere checksumove u EXEu i DLLu. Napravite sistem tako da EXE i DLL proveravaju checksumove jedan drugom.
- Napravite pauzu od jednu ili dve sekunde pre provere validnosti seriskog broja. Ovako cete povecati vreme koje potrebno da se pronadje validan seriski broj putem bruteforceinga (veoma lako ali se ne koristi cesto).
- Popravljajte sami, automatski, svoj software.
- Patchujte sami svoj software. Naterajte svoj program da se modifikuje i tako svaki put poziva drugu proceduru za proveru validnosti seriskog broja.
- Cuvajte seriske brojeve na neocekivanim mestima, npr. u propertiju nekog polja baze podataka.
- Cuvajte seriske brojeve na vise razlicitih mesta.
- Ne pouzdajte se u sistemsko vreme. Umesto toga proveravajte vreme kreacije fajlova SYSTEM.DAT i BOOTLOG.TXT i uporedjute ih sa sistemskim vremenom. Zahtevajte da je vreme vece od predhodnog vremena startovanja programa.
- Ne koristite stringove koji ce obavestiti korisnika o isteku probnog vremena softwarea. Umesto ovoga kriptujte stringove ili ih pravite dinamicki.
- Zavarajte crackerima trag pozivajuci veliki broj CALLova, koriscenjem laznih mamaca, enkriptovanih stringova...
- Koristite enkripcije i sisteme koji nisu bazirani na suvom poredjenju validnog i unetog broja.
- Nikada se ne oslanjajte na klasicne pakere / protektore da ce oni zastiti vasu aplikaciju.
- Mi vam nikada necemo otkriti nase najbolje zastite :)

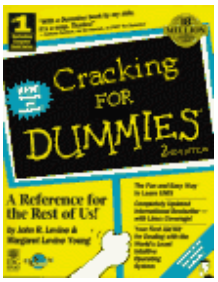
Cracking Tricks

Shvatite ovaj deo poglavlja kao deo za crackere, to jest kao neku vrstu koraka koje treba primenjivati u reversingu.

Ap0x`s deprotector's commandments

- Uvek prvo skenirajte vasu metu sa PeIDom ili nekim drugim dobrim PE identifikatorom
- Pre pristupa reversingu skupite sto je vise moguce inforamcija o samoj meti. Ovo radite startovanjem i unosenjem pogresnih podataka u polja za unos seriskog broja.
- Uvek prilikom prvog startovanja programa nadgledajte kojim fajlovima meta pristupa i koje registry kljuceve otvara, cita, ili modifikuje.
- Ako meta ima anti-debugging trikove prvo ih sve eliminisite pre nego sto pocnete sa reversingom.
- Ako meta koristi NAGove iskoristite ReSHacker ili W32dsam da pronadjete IDove dialoga koji se koriste kao NAG. Kada saznate ovu informaciju bice vam lakse da patchujete kod.
- Nikada ne koristite ResHacker u cilju uklanjanja dialoga. Ovo ce vam samo ukinuti mogucnost pravljenja patcha ili ce ga povecati za veci broj bajtova.
- Ako ste se odlucili da patchujete program uvek pronadjite najjednostavnije resenje koje ce modifikovati najmanji broj bajtova.
- Pri trazenju seriskog broja uvek trazite karakteristicne stringove, tipa "serial", "register", "valid",...
- Prilikom trazenja seriskih brojeva ili drugih provera uvek prvo postavite break-pointe na adekvatnim API pozivima pre nego sto pocnete sa trazenjem seriskog broja.
- Ako je meta zapakovana uvek je rucno raspakujte. Postoji veliki broj pakera koji se moze reversovati i direktno u memoriji ali je uvek lakse da se prilikom restarta programa krene od odpakovanog dela, da ne bi ste uvek trazili OEP ispocetka, plus posle popravke importa mocicete da postavljate break-pointove na njih.
- Uvek traceujte u CALLove koji se nalaze blizu provere seriskog broja ili poruke o pogresnom seriskom broju.
- Detaljno analizirajte kod koji se nalazi u istom CALLu kao i kod za prikazivanje poruke o pogresnom seriskom broj.
- Uvek testirajte da li vasi crackovi rade uvek, na svim kompjuterima, da li rade u neogranicenom vremenskom periodu, da li imaju online proveru tacnosti seriskog broja...
- Nikada javno ne publikujte vase patcheve. Ovo podilazi pod zakon o krsenju intelektualne svojine i autorskih prava, zakon o pirateriji zbog cega se krivicno odgovara. Upozoreni ste....

Only Fools and Horses



Ova strana sadrzi veliki broj zabuna vezanih za reversni inzenjering i njegovo polje delovanja. Vecina ovih gluposti je dosla iz usta samih programera “dobrih” zastita. Jedno je sigurno a to je da sam bio sarkastican sa svakim od ovih citata :)

- Password se nemoze vratiti ako se nalazi iza zvezdica *******
- Seriske brojeve je najbolje cuvati u registriju pod sistemskim granama.
- Nije lose koristiti jedan kljuc ili promenjivu koja ce govoriti programu da li je registrovan ili ne.
- Source kod programa se ne moze vratiti iz kompajlovanog exe fajla.
- Nije moguce otkriti stingove koje sam koristio / la unutar samog exe pa se provera passworda moze raditi ovako: if pwd = “mypwd” then ...
- Nema potrebe traciti vreme na kodiranje zastite.
- Ok je distribuirati software sa iskljucenim opcijama, tako ce korisnici ustedeti vreme za download cele verzije.
- Crackeri plate za originalni seriski broj a onda ga puste na internet
- Crackeri ne mogu da nadju seriski broj ako ga cuvam u nekom .dll fajlu u sistemskom direktorijumu....
- Ko ce da crackuje bas moju aplikaciju...

Glupost je neunistiva....

Cracker`s guide

Ovo je samo mali podsetnik svima kako bi pravi crackeri trebali da se ponasaju.

- ✓ Dobro poznajte “svoje” alate. Svaki cracker je dobar koliko i alati koje koristi
- ✓ Nikada ne ripujte (kradite) tude patcheve
- ✓ Nikada ne koristite tudje keygeneratore da bi ste poturili seriski broj kao vas fishing
- ✓ Nikada ne reversujte program posle koriscenja tudjeg patcha
- ✓ Nikada ne pisite keygeneratore i patcheve na osnovu tudjih tutorijala
- ✓ Uvek prvo pokusajte sami da reversujete aplikaciju. Ne odustajte ni posle vise sati mucenja :) ovo je draz reversinga, verujte mi na rec
- ✓ Uvek se informisite o najnovijim zastitama
- ✓ Svoje znanje produbljujte na crackmeima [http://www.crackemes.de]
- ✓ Nikada ne objavljujte svoje crackove, pravite ih samo za sebe...

F.A.Q.

Ovo su odgovori na cesto postavljana pitanja vezana za reversni inzenjering, u vezi ove knjige ali i odgovori na cesto postavljana pitanja koja zavrsavaju u mom mailboxu.

Q: Zasto kada patchujem program preko HIEWa pisem tacku ispred adrese ?
A: Zato sto se adrese koje se patchuju nalaze na virtualnim lokacijama a ne na stvarnim fizickim. Da bi ste videli stvarne fizicke adrese morate uci u edit mode sa F3.

Q: Kako da pocnem “pecanje” seriskog broja?
A: Prvo sto uvek treba uraditi je potraziti poruke koje se pojavljuju kao poruke o pogresno unetom seriskom broju u string referencama fajla. Ako ovo nema rezultate koje ste ocekivali predjite na postavljanje break-pointa na standardne API pozive.

Q: Kako da odpakujem XX.xx paker?
A: Ako se navedeni paker ne nalazi u knjizi to znaci da nije standardan stoga ga nisam opisao. Postoji mnogo tutorijala na internetu, potrazite ih.

Q: Nisam razumeo / la deo knjige koji se odnosi na xxx?
A: Siguran sam da sam sve detaljno i potanko objasnio. Ako vam nesto na prvi pogled izgleda tesko potrudite se da se udubite u materiju i da iscitate odredjeno poglavlje vise puta, konsultujte Time table.

Q: Pronasao / la sam gresku u pravopisu ili u semantici na strani xx?
A: Super, ja sam se trudio da broj gresaka bude sveden na minimum, ali neke sitne greske su mi sigurno promakle. Posaljite mi email na adresu: ap0x.rce@gmail.com (*0 je nula a ne O*)

Q: Mislim da je knjiga stvarno super i zeleo / la bih da pomognem u nekom buducem izdanju ove i sledecih knjiga?
A: Drago mi je sto tako mislite, ako imate konkretne ideje kako da mi pomognete u pisanju napisite mi email.

Q: Mislim da kniga nije nista posebno i da je nisi ni trebao pisati !!!
A: A jel te to neko pitao za misljenje... Pritisni ALT+F4 odmah....

Q: Da li bi mogao da mi pomognes oko crackovanja programa xx.xx?
A: NE, nemam ni vremena ni zelje da crackujem shareware programe stoga ustedite sebi vreme pisuci mi email ove sadrzine.

Q: Kada ce izaci knjiga?
A: Evo izasla je....

Pogovor

Dosli ste do samog kraja ove knige, cestitam vam na strpljenju i volji da je procitate u celosti. Ja sam se trudio da vas kroz ovu knjigu uvedem u svet reversnog inzenjeringa pocevsi od fundamentalnih pojmova, preko najcesce sretanih problema pa do filozofije reversinga. Priznajem da je ovaj projekat iziskivao dosta veoma pozrtvovanog rada i vremena provedenog za kompjuterom ali kada ovako pogledam do sada napisano delo mogu slobodno da kazem da predstavlja jednu zaokruzenu celinu koja ce vas na mala vrata uvesti u “nas” svet.

Svako poglavlje ove knjige predstavlja jednu celinu koja se samo na prvi pogled cini odvojenom od drugih, ali nije tako. Svako od ovih poglavlja je odabrano tako da se gradaciski nadovezuje na predhodna, uvlaceci vas tako sve dublje i dublje u samu materiju reversinga. Ona najbitnija misao koja se provlaci kroz celu knjigu je kranje nedefinisana i ne izrecena, ali ona predstavlja samu nit reversinga kao nacina razmisljanja. Ovaj stepen svesti i nacina razmisljanja se ne stice lako ali tokom vremena ce vam se podvuci pod kozu i shvaticete da se RCE primenu nalazi svugde oko nas.

Iako ova knjiga predstavlja jednu kompletnu celinu, trenutno nemam osecaj da je ona upodpunosti zavrsena. Mislim,... ne siguran sam, da ce ova knjiga doziveti jos nekoliko dopunjenih izdanja ali ce se to zasnivati na dve jako bitne osnove: interesovanje stalne publike i zivot stalnog pisca.

Nastavice se...