

Cracking Disco App – Mac Disc Burning

By AZA

Target: Dicsoapp (www.discoapp.com)

Tools: otx, otool, gdb, 0xed

Platform: Mac OS X Leopard 10.5.7 @x86

Created On: Sunday, May 31, 2009

0x0 - INTRODUCTION

Hmm, beberapa hari yang lalu aku cari-cari software untuk burning di OSX, cari-cari di Apple.com, aku nemuin DiscoApp, dari www.discoapp.com, dari penjelasan di web nya, sepertinya program yang bagus, trus aku coba download. Oops, ternyata shareware, hmm, males ngecrack aku cari-cari crackannya tapi gak nemu, akhirnya aku memutuskan buat ngecrack software ini.

Penting untuk kalian ketahui, aku reversing program ini hanya cuma untuk belajar bukan untuk komersil dan mencari keuntungan buat dijual, jadi untuk saran pembelajaran, akhirnya aku memutuskan untuk share cara me-reverse engineer program ini biar registered.

Awalnya aku mau serial phishing phishing, tapi pas me-reverse dan melihat disassembly code dari validasinya lumayan panjang, dan aku agak malas, jadi aku pake teknik patching aja.

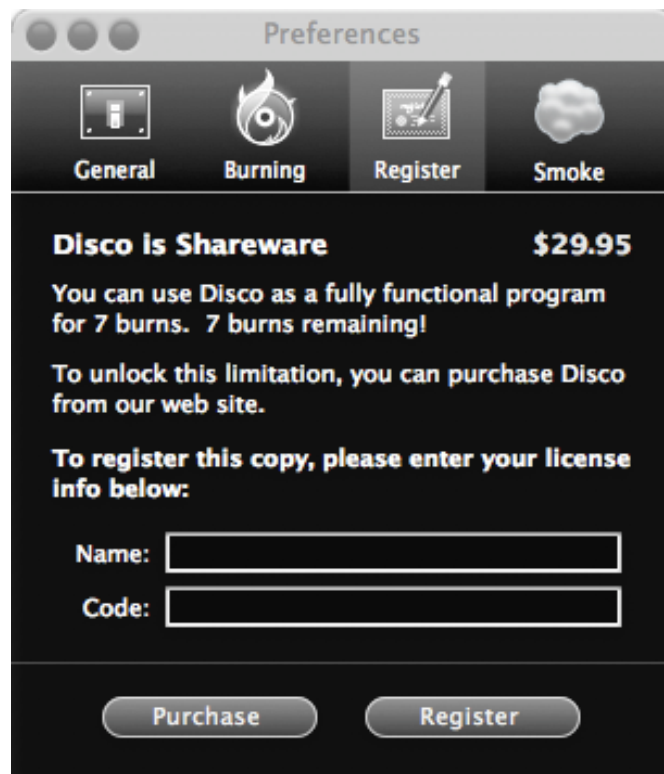
0x1 - Tools

Agar dapat me-reverse engineer program ini, kita memerlukan beberapa tools, antara lain:

- otx (<http://otx.osxninja.com/>)
- otool, bagian dari Xcode, install dari 2nd DVD bawaan Mac, atau bisa download dari (<http://developer.apple.com>)
- gdb, bawaan Mac OS X
- 0xed, Hex editor (lupa urlnya, Google: 0xed)

0x2 - Startup

Download program Disco dari situsnya untuk versi 1.3, kemudian coba kamu install, taruh programnya di /Applications. Setelah itu coba kamu jalankan programnya, maka akan tampil nag screen berikut:



Coba kamu isikan name dan code, kemudian klik Register, maka tidak ada respon apa-apa, tidak akan muncul pesan kesalahan sama sekali. Hal ini akan sedikit sulit untuk mencari dari mana kita akan memulai untuk reverse.

Nah, program ini cukup memberi challenge untuk kita, karena kita tidak menemukan pesan kesalahan. Untuk itu kita perlu memahami Program Control Flow dari Aplikasi Mac.

Window di atas muncul setiap kali kita menjalankan program, maka kita harus mencurigai bahwa program ini akan mengecek apakah registrasi valid pernah di simpan, kalau tidak ada registrasi valid maka tampilkan nag screen di atas.

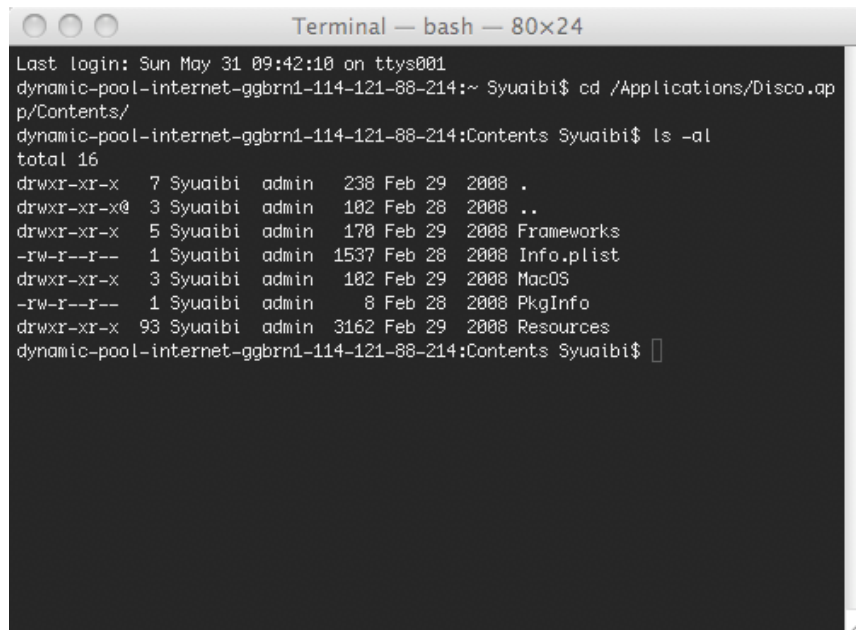
Untuk mengecek setiap kali program di jalankan, maka kita harus tahu bagaimana sebuah program Mac berjalan pertama kali. Kalau di WPF kita biasanya bisa mendelegasikan event OnLoaded. Sementara di Mac, tidak ada OnLoaded, tetapi ada sebuah instance method yang selalu di kirim pesan setiap kali Cocoa berhasil meload Nib (NextStep Interface Builder), method itu adalah **awakeFromNib**, method ini merupakan bagian dari NSObject (untuk mengetahui NSObject, NIB, dll saya mempelajari Cocoa Programming).

0x3 - Disassembly

Sekarang kita sudah punya startup yang bisa kita gunakan untuk memulai cracking, selanjutnya kita disassembly program ini menggunakan otx, untuk itu coba kamu buka /Applications/Utility/Terminal. Kemudian ketikkan:

cd /Applications/Disco.app/Contents/

Yup, program Disco adalah sebuah folder yang bernama Disco.app. Pada MacOSX semua aplikasi adalah sebuah folder yang berextensi .app, tetapi end user mengenal folder ini sebagai file, Mac OS X programmer menyebut folder tipe ini adalah file wrapper. Di dalam folder Contents dari Disco.app ada beberapa folder lainnya, cobalah ketik: **ls -al**



```
Terminal — bash — 80x24
Last login: Sun May 31 09:42:10 on ttys001
dynamic-pool-internet-ggbrn1-114-121-88-214:~ Syuaibi$ cd /Applications/Disco.ap
p/Contents/
dynamic-pool-internet-ggbrn1-114-121-88-214:Contents Syuaibi$ ls -al
total 16
drwxr-xr-x  7 Syuaibi  admin   238 Feb 29  2008 .
drwxr-xr-x@ 3 Syuaibi  admin   102 Feb 28  2008 ..
drwxr-xr-x  5 Syuaibi  admin   170 Feb 29  2008 Frameworks
-rw-r--r--  1 Syuaibi  admin  1537 Feb 28  2008 Info.plist
drwxr-xr-x  3 Syuaibi  admin   102 Feb 29  2008 MacOS
-rw-r--r--  1 Syuaibi  admin    8 Feb 28  2008 PkgInfo
drwxr-xr-x 93 Syuaibi  admin  3162 Feb 29  2008 Resources
dynamic-pool-internet-ggbrn1-114-121-88-214:Contents Syuaibi$
```

Dari gambar di atas ada folder MacOS, di dalam folder ini terdapat binary program Disco, pada folder Resources terdapat semua resource yang diperlukan oleh program baik itu berupa file gambar, video, dll. Sementara frameworks adalah framework dependensi yang di perlukan oleh program, kalau di Windows ini biasanya berupa .dll.

Ok, sekarang kita masuk ke MacOS, dengan cara **cd MacOS**, kemudian ketik **ls-al**.

```
Terminal — bash — 80x24
Last login: Sun May 31 09:42:10 on ttys001
dynamic-pool-internet-ggbrn1-114-121-88-214:~ Syuaibi$ cd /Applications/Disco.ap
p/Contents/
dynamic-pool-internet-ggbrn1-114-121-88-214:Contents Syuaibi$ ls -al
total 16
drwxr-xr-x  7 Syuaibi  admin   238 Feb 29  2008 .
drwxr-xr-x@ 3 Syuaibi  admin   102 Feb 28  2008 ..
drwxr-xr-x  5 Syuaibi  admin   170 Feb 29  2008 Frameworks
-rw-r--r--  1 Syuaibi  admin  1537 Feb 28  2008 Info.plist
drwxr-xr-x  3 Syuaibi  admin   102 Feb 29  2008 MacOS
-rw-r--r--  1 Syuaibi  admin    8 Feb 28  2008 PkgInfo
drwxr-xr-x 93 Syuaibi  admin  3162 Feb 29  2008 Resources
dynamic-pool-internet-ggbrn1-114-121-88-214:Contents Syuaibi$ cd MacOS/
dynamic-pool-internet-ggbrn1-114-121-88-214:MacOS Syuaibi$ ls -al
total 1456
drwxr-xr-x  3 Syuaibi  admin   102 Feb 29  2008 .
drwxr-xr-x  7 Syuaibi  admin   238 Feb 29  2008 ..
-rwxr-xr-x  1 Syuaibi  admin 743672 Feb 28  2008 Disco
dynamic-pool-internet-ggbrn1-114-121-88-214:MacOS Syuaibi$
```

Kita lihat terdapat file Disco, file inilah file yang dieksekusi oleh MacOSX, sekarang ketik **"open ."** untuk membuka finder pada folder ini. Kemudian drag file Disco ke dalam window otx yang sudah kita buka. Kemudian klik Save. Maka kita akan mendapatkan file .txt yang merupakan disassembly dari program ini.

Setelah kita mendapatkan file disassembly (biasanya nama file binary + _x86.txt), coba buka file nya : Disco_x86.txt menggunakan text editor.

Kembali ke Terminal, ketikkan perintah: **grep -i "awakeFromNib" Disco_x86.txt**, perintah ini berguna mencari text "awakeFromNib" dari file disassembly. Maka akan tampil beberapa daftar, coba lihat bagian paling atas yang bertuliskan: **-(void)[DCAppController awakeFromNib]**

Pada pemrograman Mac, kita selalu menggunakan Design Pattern yang disebut dengan MVC (Model-View-Controller), dimana pattern ini memisahkan antara view (tampilan) dengan model (data), tetapi menjembatani keduanya dengan controller. Nah, setiap aplikasi Cocoa pasti mempunyai sebuah application delegate, nah terkadang application delegate itu berupa controller. Dari namanya kita tahu bahwa ini adalah sebuah controller untuk application delegate. Prefix DC merupakap prefix aplikasi Disco (DC) dan ApplicationController merupakan nama dari controller ke application (biasanya berupa delegate). Untuk mengetahui delegate, saya mempelajari Cocoa Programming terlebih dahulu.

0x4 - Analysis

Sekarang kita sudah mendapatkan sebuah string awakeFromNib dari controller yang bernama DCAppController, untuk itu buka kembali file disassembly yang tadi kita generate, dan coba cari "DCAppController awakeFromNib", hit pertama langsung menuju method "DCAppController awakeFromNib". Karena

panjangnya code, maka tidak akan ditampilkan semua disasemblynya. Coba kamu scroll ke bawah untuk melihat-lihat apakah ada sesuatu yang mencurigakan dari method ini.

Kita menemukan `[(%esp, 1) validateName:code:]`. Inilah method yang dikirimkan pesan untuk validasi nama dan code, tetapi jangan dahulu kita bukan method ini. Kita amati lagi ke bawahnya.

Kita mendapatkan “_handleRegistrationNag”, agak ke bawah lagi kita akan menemukan **jmp 0x00002f93** pada offset **00002e87**. Yang harus diingat dalam sebuah analysis disassembly, bahwa jmp biasanya digunakan untuk kondisi percabangan, yang biasanya dalam bahasa C ditulis:

```
if (statement) {
```

```
} else {}
```

Jmp adalah melompat ke alamat tertentu tanpa kondisi apapun. Dan jmp biasanya digunakan bersama `jne`, atau `je` untuk membentuk `if () else`. Untuk membuktikan apakah jmp di sini membentuk blok `if () else`, kita scroll ke atas, untuk mencari apakah kita mendapatkan conditional jump (`jne`, `je`, `jz`, dll).

Ya, pada offset `00002d9d` kita mendapatkan sebuah conditional jump

jel 0x00002e8c (atau dalam syntax Intel: **je 0x00002e8c**)

Nah, ini adalah sebuah conditional jump, coba lihat di atasnya terdapat :

```
calll    0x0004d280    -[(%esp,1) boolValue]
testb    %al,%al
movb     %al,0x5c(%edi)    (BOOL)_distributionVersion
jel      0x00002e8c
```

Sebuah pengiriman pesan (kalau di Windows biasanya disebut pemanggilan method, sementara di Mac disebut dengan mengirimkan pesan) yang mempunyai nilai kembali Boolean (dilihat dari nama `boolValue`, kita bisa menerka ini adalah Boolean: 0/1), kemudian code selanjutnya mentest apakah nilai kembali dari method `[boolValue] = 0`, kode di bawahnya tidak perlu kita hiraukan. Bagi yang pernah belajar Assembly, pastinya hal ini aneh, kenapa instruksi **test** tidak dibarengkan dengan **je**, tetapi malah di gandengkan dengan instruksi **mov**, untuk sekedar info, inilah yang disebut dengan code interleaved, dimana compiler menaruh code di antara `test/cmp` dengan `je`, untuk kegunaan paralelisme.

Jadi kita tahu bahwa jika `boolValue` mengembalikan nilai 0 atau false maka lompat ke `0x2e8c`.

Jika di tulis dalam bahasa Objective C adalah seperti berikut:

```
if (![something objectValue])
{
    // Block code mulai dari 0x2e8c
}
```

Sekarang kita kembali ke alamat 00002e87, kita lihat bahwa 0x2e8c tepat berada di bawah instruksi pada offset 00002e87 (**jmp 0x00002f93**). Kemudian, coba kita lihat ada apa pada offset **0x00002f93**. Wow, ternyata offset tersebut dekat dengan function epilog (akhir dari sebuah fungsi).

Nah, setelah kita tracing dead code (non live debugging) ini, kita bisa memperkuat dugaan kita bahwa jmp, dan jne ini membentuk sebuah if () else.

Kalau kita simpulkan dari investigasi yang kita lakukan sebelumnya, maka kita akan menuliskannya dalam bahasa Objective C sebagai berikut:

```
if ([something objectValue]) {
    // eksekusi code mulai dari 0x2da3 hingga 0x2e87
} else {
    // eksekusi code mulai dari 0x2e8c hingga akhir fungsi
}
```

Jadi jika objectValue bernilai benar, maka eksekusi blok code dari 0x2da3-0x2e87, jika bernilai salah, maka eksekusi blok 0x2e8c hingga akhir fungsi.

Dari hasil analysis ini kita sekarang mulai berfokus pada satu pertanyaan?, apa yang terjadi kalau kita balik, misalnya object value bernilai benar, dia akan mengeksekusi blok kebalikannya. Untuk membuktikannya kita memerlukan **GDB (GNU Debugger)**

0x5 - Debugging

Setelah kita menganalisis dead code dari disassembly file yang kita generate sekarang kita akan melakukan live debugging. Ingat, bahwa awal percabangan if () else kita ada pada offset 00002d93, sekarang kembali ke Terminal dan ketikkan **gdb -q**, kemudian pada prompt (gdb) atau gdb\$, ketikkan :

```
gdb$ exec-file Disco
```

```
Reading symbols for shared libraries ..... done
```

Kemudian kita breakpoint pada offset 00002d93

```
gdb$ b *0x2d93
```

Breakpoint 1 at 0x2d93

// Kemudian ketik **run** untuk menjalankan program Disco

gdb\$ run

```
Reading symbols for shared libraries
..... done
```

Reading symbols for shared libraries . done

Reading symbols for shared libraries . done

Reading symbols for shared libraries . done

Reading symbols for shared libraries . done

Reading symbols for shared libraries . done

Reading symbols for shared libraries . done

Reading symbols for shared libraries . done

Reading symbols for shared libraries . done

Breakpoint 1, 0x00002d93 in -[DCAppController awakeFromNib] ()

-----[regs]

EAX: A05183F8 EBX: 00000000 ECX: 00000000 EDX: 92826128 o d I t s z A p C

ESI: 15908AC0 EDI: 003B3AD0 EBP: BFFFF4A8 ESP: BFFFF470 EIP: 00002D93

CS: 0017 DS: 001F ES: 001F FS: 0000 GS: 0037 SS: 001F

[001F:BFFFF470]-----[stack]

BFFFF4C0 : 10 00 00 00 20 01 34 00 - D3 C1 B1 A1 05 00 00 004.....

BFFFF4B0 : D0 3A 3B 00 C8 58 8D 92 - C8 58 8D 92 10 00 00 00 .;.X..X.....

BFFFF4A0 : 09 00 00 00 03 00 00 00 - 58 F5 FF BF B5 48 E7 94X...H..

BFFFF490 : 09 00 00 00 03 00 00 00 - 58 F5 FF BF 14 48 E7 94X...H..

BFFFF480 : 30 D2 30 00 28 CD 87 92 - F3 71 80 92 14 48 E7 94 0.0.(...q...H..

BFFFF470 : F8 83 51 A0 28 61 82 92 - AC 20 04 00 54 C5 E6 94 ..Q.(a... ..T...

[0017:00002D93]-----[code]

```
0x2d93 <-[DCAppController awakeFromNib]+365>:    call                0x4d280
<dyld_stub_objc_msgSend>
```

```
0x2d98 <-[DCAppController awakeFromNib]+370>:    test    al,al
```

```
0x2d9a <-[DCAppController awakeFromNib]+372>:    mov     BYTE PTR [edi+0x5c],al
```

```

0x2d9d <-[DCAppController awakeFromNib]+375>:    je      0x2e8c <-[DCAppController
awakeFromNib]+614>

0x2da3 <-[DCAppController awakeFromNib]+381>:    mov    eax,ds:0x45040

0x2da8 <-[DCAppController awakeFromNib]+386>:    mov    DWORD PTR [esp+0x4],eax

0x2dac <-[DCAppController awakeFromNib]+390>:    mov    eax,ds:0x45ddc

0x2db1 <-[DCAppController awakeFromNib]+395>:    mov    DWORD PTR [esp],eax

```

```
// Sekarang kita berada pada breakpoint pada alamat 0x00002d93
```

```
// Coba kita stepo beberapa kali hingga kita tiba pada context berikut:
```

```

-----[regs]
EAX: 00000000  EBX: 00000000  ECX: A04E4B80  EDX: 00000015  o d I t s Z a P c
ESI: 15908AC0  EDI: 003B3AD0  EBP: BFFFF4A8  ESP: BFFFF470  EIP: 00002D9D
CS: 0017  DS: 001F  ES: 001F  FS: 0000  GS: 0037  SS: 001F  Jump is taken (Z flag)
[0017:00002D9D]-----[code]

0x2d9d <-[DCAppController awakeFromNib]+375>:    je      0x2e8c <-[DCAppController
awakeFromNib]+614>

0x2da3 <-[DCAppController awakeFromNib]+381>:    mov    eax,ds:0x45040

0x2da8 <-[DCAppController awakeFromNib]+386>:    mov    DWORD PTR [esp+0x4],eax

0x2dac <-[DCAppController awakeFromNib]+390>:    mov    eax,ds:0x45ddc

0x2db1 <-[DCAppController awakeFromNib]+395>:    mov    DWORD PTR [esp],eax

0x2db4 <-[DCAppController awakeFromNib]+398>:    call   0x4d280
<dyled_stub_objc_msgSend>

0x2db9 <-[DCAppController awakeFromNib]+403>:    mov    esi,eax

0x2dbb <-[DCAppController awakeFromNib]+405>:    mov    eax,ds:0x45034

```

```
// Lihat pada bagian regs ada tulisan Jump is taken, dan Zero flag di set (huruf Z
besar di atasnya).
```

```
// Lihat juga pada register EAX yang bernilai 0.
```

```
// Jadi, pengiriman pesan pada [something boolValue] mengembalikan nilai false
= 0.
```

```
// Dari hasil analysis kita mau mencoba menukar blok eksekusi if () else,
caranya adalah membuat agar instruksi je tidak jadi melakukan lompatan (Jump
is not taken)
```

```
// Untuk menggagalkan lompatan kita set Zero flag menjadi 1 dengan
memasukkan perintah berikut
```

```
gdb$ cfz
```

`gdb$ context`

```
-----  
[regs]  
EAX: 00000000 EBX: 00000000 ECX: A04E4B80 EDX: 00000015 o d I t s z a  
P c  
ESI: 1594BB70 EDI: 00340010 EBP: BFFFFFF4A8 ESP: BFFFFFF470 EIP: 00002D9D  
CS: 0017 DS: 001F ES: 001F FS: 0000 GS: 0037 SS: 001F Jump is NOT  
taken (z flag)  
[0017:00002D9D]-----  
[code]  
0x2d9d <-[DCAppController awakeFromNib]+375>: je 0x2e8c <-[  
DCAppController awakeFromNib]+614>  
0x2da3 <-[DCAppController awakeFromNib]+381>: mov eax,ds:0x45040  
0x2da8 <-[DCAppController awakeFromNib]+386>: mov DWORD PTR [esp+0x4],eax  
0x2dac <-[DCAppController awakeFromNib]+390>: mov eax,ds:0x45ddc  
0x2db1 <-[DCAppController awakeFromNib]+395>: mov DWORD PTR [esp],eax  
0x2db4 <-[DCAppController awakeFromNib]+398>: call 0x4d280  
<dyld_stub_objc_msgSend>  
0x2db9 <-[DCAppController awakeFromNib]+403>: mov esi,eax  
0x2dbb <-[DCAppController awakeFromNib]+405>: mov eax,ds:0x45034  
-----
```

// Sekarang kita lihat pada bagian regs bahwa Jump is not taken, bahwa lompatan tidak akan di lakukan

// Kemudian lanjutkan program dengan mengetikkan `c` (continue)

`gdb$ c`

Reading symbols for shared libraries . done

Reading symbols for shared libraries .. done

Reading symbols for shared libraries . done

Coba kamu lihat pada program Disco, sekarang dia akan menampilkan End User License Agreement window, pilih Agree. Nah window inilah yang ditampilkan bila kita dianggap sudah valid registrasi.

Jadi asumsi kita benar bahwa `boolValue` mengembalikan nilai apakah kita sudah ter-registrasi atau belum. Berarti pada offset inilah entry point kita, yaitu pada offset **0x00002d9d**.

Proteksi sudah kita taklukan dengan pembuktian menggunakan **gdb**. Sekarang tahap selanjutnya adalah patching.

0x6 - Patching

Dari hasil live debugging menggunakan GDB kita sudah mendapatkan dimana entry point yang akan kita gunakan untuk patching, yaitu pada offset **0x00002d9d**. Tahap selanjutnya kita menggunakan program `Oxed` untuk patching. Tetapi offset `0x00002d9d` tidak bisa kita gunakan untuk langsung patching pada file, kenapa?

Pada MacOSX binary file bisa bertipe fat binary (Universal Binary) yang mendukung Intel dan PPC processor sekaligus, tetapi ada juga binary yang khusus Intel, tetapi kita tidak bisa berasumsi, kita harus melihat offsetnya secara langsung dari file yang akan kita patch, untuk ini kita bisa menggunakan **otool**.

Pada Terminal ketikkan perintah berikut:

```
$ otool -f Disco
```

```
Fat headers
```

```
fat_magic 0xcafebabe
```

```
nfat_arch 2
```

```
architecture 0
```

```
  cputype 18
```

```
  cpusubtype 0
```

```
  capabilities 0x0
```

```
  offset 4096
```

```
  size 363340
```

```
  align 2^12 (4096)
```

```
architecture 1
```

```
  cputype 7
```

```
  cpusubtype 3
```

```
  capabilities 0x0
```

```
  offset 368640
```

```
  size 375032
```

```
  align 2^12 (4096)
```

Dari info di atas kita melihat bahwa binary ini bertipe fat binary (Universal binary) yang mendukung 2 tipe processor, cputype 18 (PPC) dan cputype 7 (Intel). Pada 2 arsitektur info di atas ada offset. Kita lihat offset untuk cputype 7 adalah 368640 (0x5A000), sementara offset CPU lainnya adalah 4096 (0x1000)

Caranya untuk mendapatkan offset asli dari `0x00002d9d` bila di dahului oleh header dari cpu lainnya adalah dengan cara:

offset_cpu_yang_diinginkan - offset_cpu_lainnya + offset patch

0x5A000 - 0x1000 + 0x2d9d = 0x5BD9D

Sekarang kita sudah mendapatkan offset pada file, yaitu 0x5BD9D.

Kemudian kita buka binary filenya menggunakan 0xED, pada bagian ujung kanan window 0xED kita masukkan offset nya, maka kursor akan mengarah pada bagian yang diinginkan:

0f84e9000000

0f84 adalah kode mesin untuk **jel**, kebalikannya adalah **jnel**, dengan kode mesin **0f85**, gantilah 84 dengan 85 kemudian Save.

Sekarang coba jalankan kembali program Disco.

Nag screen yang menanyakan registrasi tidak akan pernah muncul lagi. Disco sudah di crack.

0x07 - Conclusion

Kita sudah tiba pada akhir tutorial, bagi sebagian reverser mungkin tutorial ini terlihat sangat mudah sekali, tetapi tidak pernah salah bila kita mengulang-ulang hal untuk melatih kemampuan kita dan sebagai saran pembelajaran.

Pembahasan tutorial ini terbagi menjadi 2, yaitu dead code debugging yang murni menggunakan analysis code yang programnya tidak sedang berjalan sehingga kita tidak bisa melihat stack, register, hanya murni analysis dari code disassembly.

Tahap kedua menggunakan live debugging, dimana kita bisa membuktikan analysis kita pada dead code debugging pada program yang sedang berjalan.

Diharapkan tutorial ini bisa memberikan pemahaman kepada teman-teman sekalian.

Have fun!.

Suhendra Ahmad "AZA"