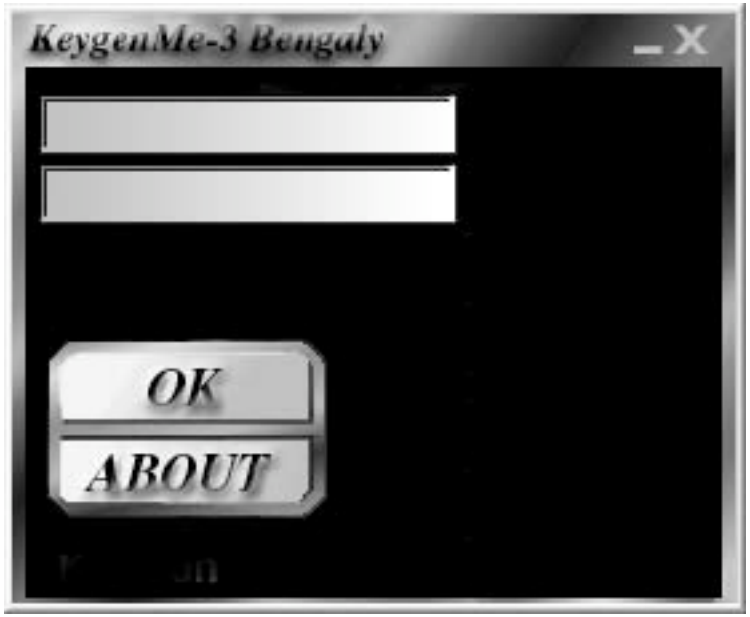
Breaking Protection

Overview

- Here, we discuss cracking examples
- Examples are not from real software
  - "Crackme" --- program designed for studying cracking/protection techniques
- Why learn cracking?
  - So that you can better protect software
  - "…protection technologies developed by people who have never attempted cracking are *never* effective!"

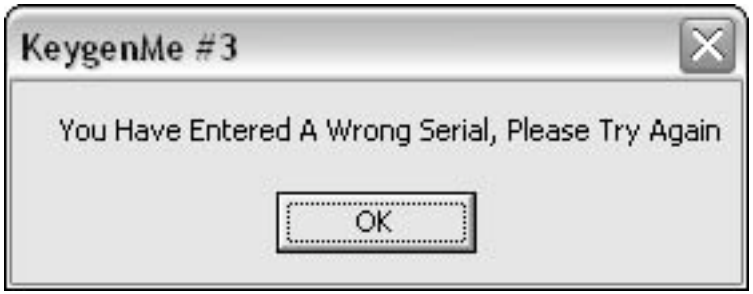
Patching

- Consider the following application
  - KeygenMe-3 by Bengaly
- No useful info here
- What to do?
- Enter some data and see what happens



Patching

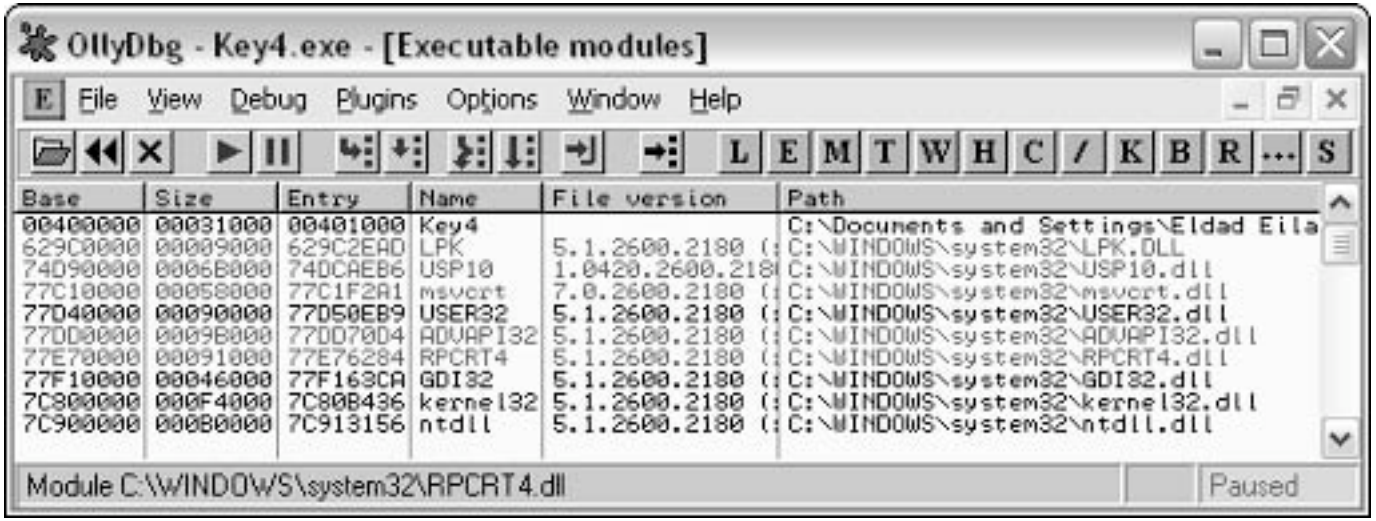
- Get invalid serial number message:



- Now what?
- OllyDbg, of course…

Patching

- Looking for message box



Patching

- What about lpk.dll?

Patching

- Imports/exports

OllyDbg - Key4.exe - [Names in Key4]

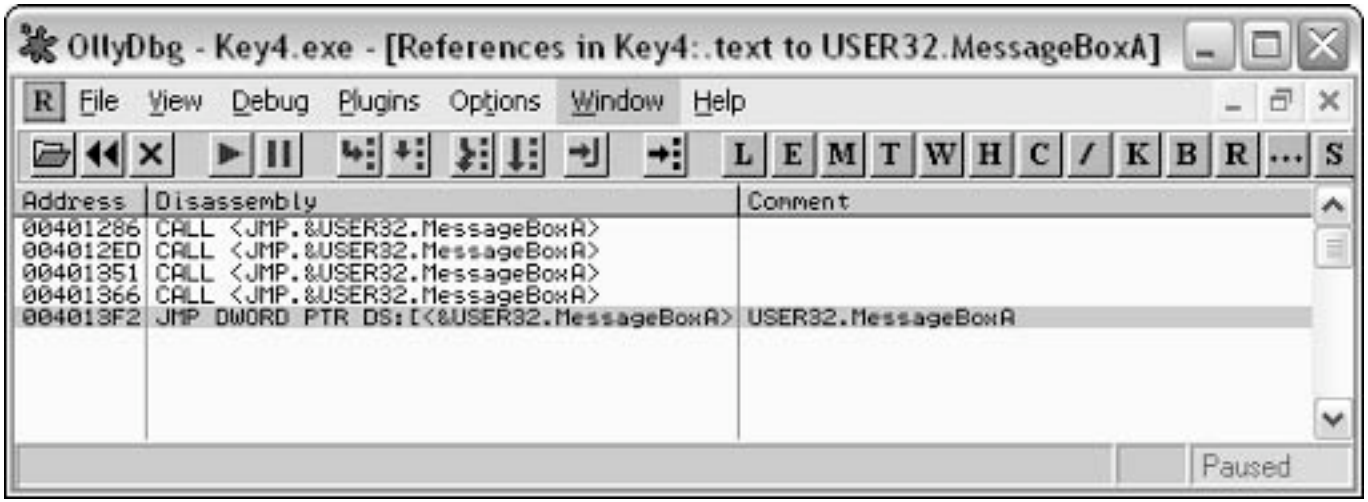
File View Debug Plugins Options Window Help

Address	Section	Type	Name	Comment
00402060	.rdata	Import	USER32.CreateDialogParamA	
00402054	.rdata	Import	USER32.DefWindowProcA	
00402000	.rdata	Import	GDI32.DeleteObject	
00402058	.rdata	Import	USER32.DispatchMessageA	
0040200C	.rdata	Import	KERNEL32.ExitProcess	
00402050	.rdata	Import	USER32.GetDlgItem	
0040204C	.rdata	Import	USER32.GetDlgItemTextA	
00402048	.rdata	Import	USER32.GetMessageA	
00402008	.rdata	Import	KERNEL32.GetModuleHandleA	
00402044	.rdata	Import	USER32.IsDialogMessageA	
0040202C	.rdata	Import	USER32.LoadBitmapA	
00402018	.rdata	Import	USER32.LoadCursorA	
0040201C	.rdata	Import	USER32.LoadIconA	
00402010	.rdata	Import	KERNEL32.lstrlenA	
00402020	.rdata	Import	USER32.MessageBoxA	
00401000	.text	Export	<ModuleEntryPoint>	
00402024	.rdata	Import	USER32.PostQuitMessage	
00402028	.rdata	Import	USER32.RegisterClassExA	
0040205C	.rdata	Import	USER32.ReleaseCapture	
00402030	.rdata	Import	USER32.SendMessageA	
00402034	.rdata	Import	USER32.SetWindowTextA	
00402038	.rdata	Import	USER32.ShowWindow	
0040203C	.rdata	Import	USER32.TranslateMessage	
00402040	.rdata	Import	USER32.UpdateWindow	

Module C:\WINDOWS\system32\RPCRT4.dll Paused

Patching

- References to MessageBoxA



- OK, now what?

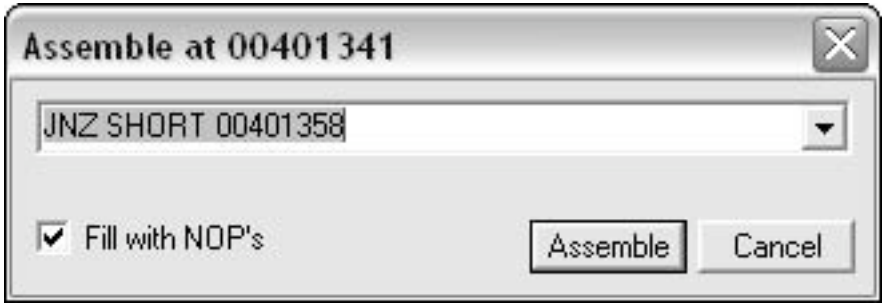
Patching

- Third MsgBoxA reference

```
0040133F    CMP EAX,ESI
00401341    JNZ SHORT Key4.00401358
00401343    PUSH 0
00401345    PUSH Key4.0040348C                ; ASCII "KeygenMe #3"
0040134A    PUSH Key4.004034DD                ; Text = " Great, You are
                                                ranked as Level-3 at
                                                Keygening now"
0040134F    PUSH 0                            ; hOwner = NULL
00401351    CALL <JMP.&USER32.MessageBoxA>    ; MessageBoxA
00401356    JMP SHORT Key4.0040136B
00401358    PUSH 0                            ; Style =
                                                MB_OK|MB_APPLMODAL
0040135A    PUSH Key4.0040348C                ; Title = "KeygenMe #3"
0040135F    PUSH Key4.004034AA                ; Text = " You Have
                                                Entered A Wrong Serial,
                                                Please Try Again"
00401364    PUSH 0                            ; hOwner = NULL
00401366    CALL <JMP.&USER32.MessageBoxA>    ; MessageBoxA
0040136B    JMP SHORT Key4.00401382
```

Patching

- Now patch it in OllyDbg…



- …success

Keygenning

- Spse program asks for ID & serial number
- Such a program may have keygen algorithm
  - Generate a "key" or serial number based on ID
- Attacker might want access to keygen algorithm
- Why?
  - To generate many valid ID/serial number pairs
  - Why isn't 1 such pair sufficient?

Ripping Keygen Algorithm

- Goal is to create working copy of keygen algorithm
- Just for creating valid ID/serial number pairs
- This code can be "ripped" from the application
- Following example is from…
  - KeygenMe-3 by Bengaly

Ripping Keygen Algorithm

- Code Part 1

```
004012B1    PUSH 40                                  ; Count = 40 (64.)
004012B3    PUSH Key4.0040303F                       ; Buffer = Key4.0040303F
004012B8    PUSH 6A                                  ; ControlID = 6A (106.)
004012BA    PUSH DWORD PTR [EBP+8]                   ; hWnd
004012BD    CALL <JMP.&USER32.GetDlgItemTextA>       ; GetDlgItemTextA
004012C2    CMP EAX,0
004012C5    JE SHORT Key4.004012DF
004012C7    PUSH 40                                  ; Count = 40 (64.)
004012C9    PUSH Key4.0040313F                       ; Buffer = Key4.0040313F
004012CE    PUSH 6B                                  ; ControlID = 6B (107.)
004012D0    PUSH DWORD PTR [EBP+8]                   ; hWnd
```

Ripping Keygen Algorithm

Code Part 2

```
004012D3    CALL <JMP.&USER32.GetDlgItemTextA>  ; GetDlgItemTextA
004012D8    CMP EAX,0
004012DB    JE SHORT Key4.004012DF
004012DD    JMP SHORT Key4.004012F6
004012DF    PUSH 0                              ; Style =
                                                  MB_OK|MB_APPLMODAL
004012E1    PUSH Key4.0040348C                  ; Title = "KeygenMe #3"
004012E6    PUSH Key4.00403000                  ; Text = "    Please
                                                  Fill In 1 Char to
                                                  Continue!!"
004012EB    PUSH 0                              ; hOwner = NULL
004012ED    CALL <JMP.&USER32.MessageBoxA>      ; MessageBoxA
004012F2    LEAVE
004012F3    RET 10
004012F6    PUSH Key4.0040303F                  ; String = "Eldad Eilam"
004012FB    CALL <JMP.&KERNEL32.lstrlenA>       ; lstrlenA
00401300    XOR ESI,ESI
00401302    XOR EBX,EBX
00401304    MOV ECX,EAX
00401306    MOV EAX,1
0040130B    MOV EBX,DWORD PTR [40303F]
00401311    MOVSX EDX,BYTE PTR [EAX+40351F]
00401318    SUB EBX,EDX
0040131A    IMUL EBX,EDX
0040131D    MOV ESI,EBX
```

Ripping Keygen Algorithm

- Code Part 3

```
0040131F    SUB EBX,EAX
00401321    ADD EBX,4353543
00401327    ADD ESI,EBX
00401329    XOR ESI,EDX
0040132B    MOV EAX,4
00401330    DEC ECX
00401331    JNZ SHORT Key4.0040130B
00401333    PUSH ESI
00401334    PUSH Key4.0040313F                  ; ASCII "12345"
00401339    CALL Key4.00401388
0040133E    POP ESI
0040133F    CMP EAX,ESI
```

Ripping Keygen Algorithm

- Take a look at Key4.00401388

```
00401388    PUSH EBP
00401389    MOV EBP,ESP
0040138B    PUSH DWORD PTR [EBP+8]                  ; String
0040138E    CALL <JMP.&KERNEL32.lstrlenA>           ; lstrlenA
00401393    PUSH EBX
00401394    XOR EBX,EBX
00401396    MOV ECX,EAX
00401398    MOV ESI,DWORD PTR [EBP+8]
0040139B    PUSH ECX
0040139C    XOR EAX,EAX
0040139E    LODS BYTE PTR [ESI]
0040139F    SUB EAX,30
004013A2    DEC ECX
004013A3    JE SHORT Key4.004013AA
004013A5    IMUL EAX,EAX,0A
004013A8    LOOPD SHORT Key4.004013A5
004013AA    ADD EBX,EAX
004013AC    POP ECX
004013AD    LOOPD SHORT Key4.0040139B
004013AF    MOV EAX,EBX
004013B1    POP EBX
004013B2    LEAVE
004013B3    RET 4
```

Ripping Keygen Algorithm

- Code for keygen algorithm…
- Uppercase asm is ripped from app
- Note: there is no need to understand the details!

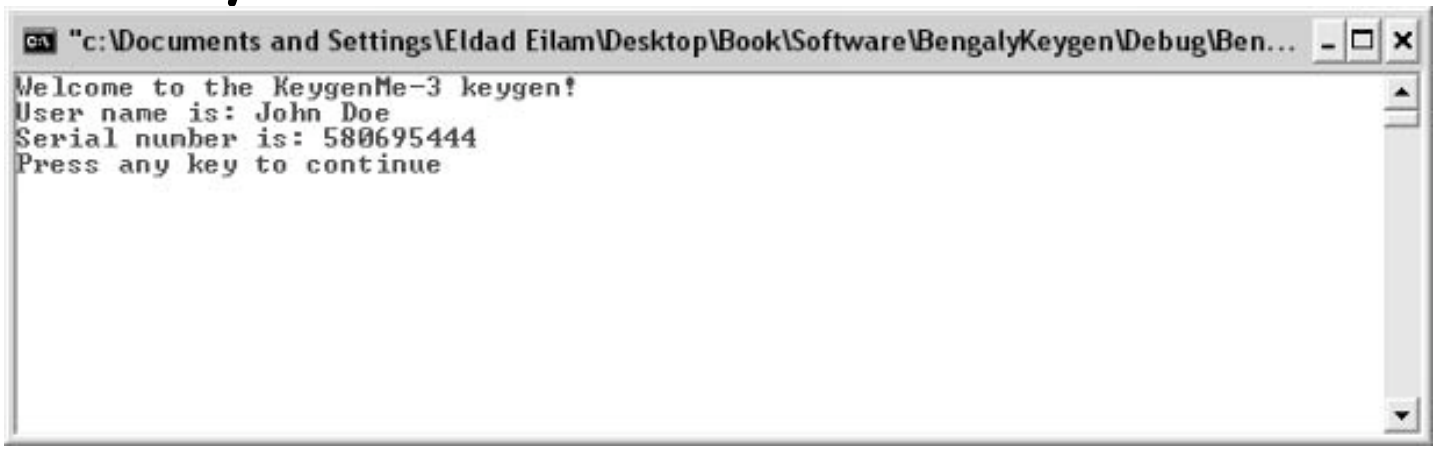
```
ULONG ComputeSerial(LPSTR pszString)
{
  DWORD dwLen = lstrlen(pszString);
  _asm
  {
    mov ecx, [dwLen]
    mov edx, 0x25
    mov eax, 1
LoopStart:
    MOV EBX, DWORD PTR [pszString]
    mov ebx, dword ptr [ebx]
    //MOVSX EDX, BYTE PTR DS:[EAX+40351F]
    SUB EBX, EDX
    IMUL EBX, EDX
    MOV ESI, EBX
    SUB EBX, EAX
    ADD EBX, 0x4353543
    ADD ESI, EBX
    XOR ESI, EDX
    MOV EAX, 4
    mov edx, 0x65
    DEC ECX
    JNZ LoopStart
    mov eax, ESI
  }
}
```

Ripping Keygen Algorithm

- Insert previous code into console app

```
int _tmain(int argc, _TCHAR* argv[])
{
 printf ("Welcome to the KeygenMe-3 keygen!\n");
 printf ("User name is: %s\n", argv[1]);
 printf ("Serial number is: %u\n", ComputeSerial(argv[1]));
 return 0;
}
```

- And try it out…



Advanced Cracking: Defender

- Application developed to demonstrate protection techniques
  - "…similar to what you would find in real-world commercial protection…"
- Difficult, but not impossible
  - "…all it takes is a lot of knowledge and a lot of patience"

Defender Interface

- Launch without command-line options

```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\Eldad Eilam>defender
Defender Version 1.0 - Written by Eldad Eilam
Bad parameters!
Usage: Defender <Full Name> <16-digit hexadecimal number>

C:\Documents and Settings\Eldad Eilam>_
```

Defender Interface

- Launched with “random” username/serial number

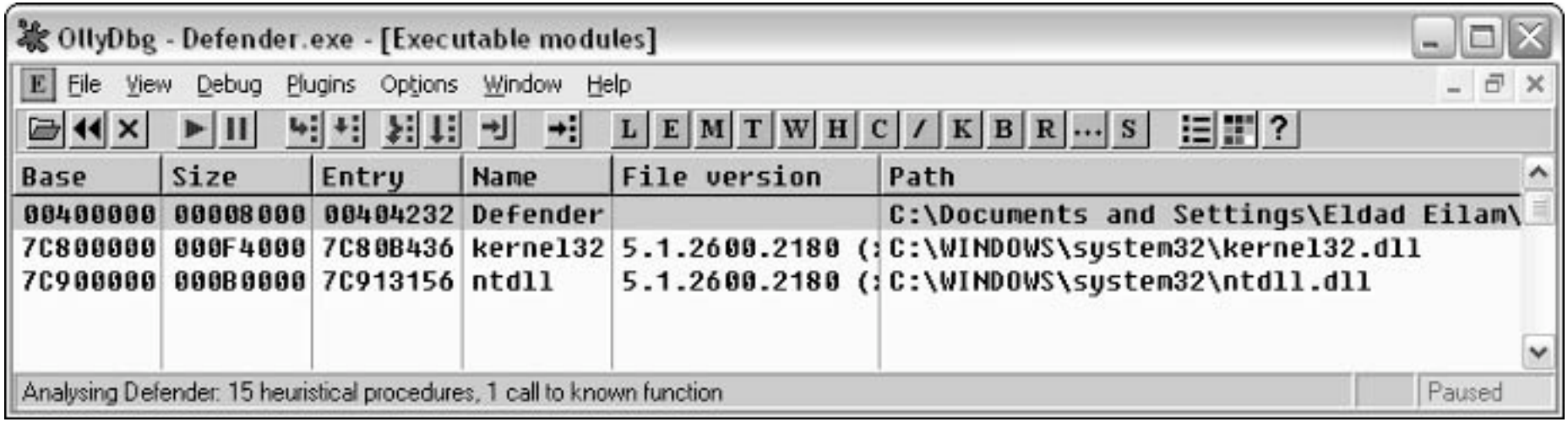
```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\Eldad Eilam>defender "John Doe" 1234567890abcdef
Defender Version 1.0 - Written by Eldad Eilam
Sorry... Bad key, try again.

C:\Documents and Settings\Eldad Eilam>
```

Defender: Linked Modules

- Load into OllyDbg and look at Executable Modules window
  - Gives exe modules that are statically linked



- Just standard stuff here

Defender: Imports/Exports

- Imports/exports

- Only API called is IsDebuggerPresent?
  - This is very strange

Defender: DUMPBIN

- Anything?
- Still just one API?
- What about summary?

```
Microsoft (R) COFF/PE Dumper Version 7.10.3077
Copyright (C) Microsoft Corporation. All rights reserved.


Dump of file defender.exe
File Type: EXECUTABLE IMAGE

  Section contains the following imports:

    KERNEL32.dll
                 405000 Import Address Table
                 405030 Import Name Table
                      0 time date stamp
                      0 Index of first forwarder reference

                    22F IsDebuggerPresent

  Summary

        1000 .data
        4000 .h3mf85n
        1000 .h477w81
        1000 .rdata
```

DUMPBIN /HEADERS

- Try long listing --- find the following

```
OPTIONAL HEADER VALUES
             10B magic # (PE32)
            7.10 linker version
            3400 size of code
             600 size of initialized data
               0 size of uninitialized data
            4232 entry point (00404232)
            1000 base of code
            5000 base of data
          400000 image base (00400000 to 00407FFF)
            1000 section alignment
             200 file alignment
            4.00 operating system version
            0.00 image version
```

- ……

DUMPBIN /HEADERS

- And…

```
SECTION HEADER #1
.h3mf85n name
    3300 virtual size
    1000 virtual address (00401000 to 004042FF)
    3400 size of raw data
     400 file pointer to raw data (00000400 to 000037FF)
       0 file pointer to relocation table
       0 file pointer to line numbers
       0 number of relocations
       0 number of line numbers
E0000020 flags
         Code
         Execute Read Write
```

- ……

DUMPBIN /HEADERS

- And…

```
SECTION HEADER #2
  .rdata name
       95 virtual size
     5000 virtual address (00405000 to 00405094)
      200 size of raw data
     3800 file pointer to raw data (00003800 to 000039FF)
        0 file pointer to relocation table
        0 file pointer to line numbers
        0 number of relocations
        0 number of line numbers
40000040 flags
         Initialized Data
         Read Only
```

- ……

DUMPBIN /HEADERS

- And…

```
SECTION HEADER #4
.h477w81 name
      8C virtual size
    7000 virtual address (00407000 to 0040708B)
     200 size of raw data
    3A00 file pointer to raw data (00003A00 to 00003BFF)
       0 file pointer to relocation table
       0 file pointer to line numbers
       0 number of relocations
       0 number of line numbers
C0000040 flags
         Initialized Data
         Read Write
```

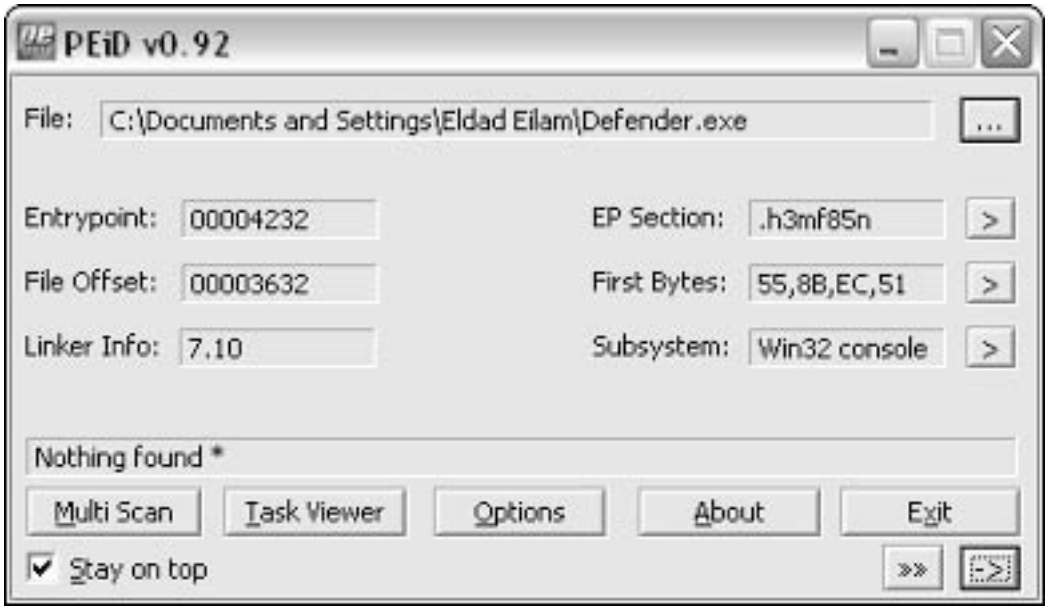
- ……

Strange Section Names

- May be indication that program is packed
- What to do?
- Try unpacking
- Will only work if it is standard packer

Defender: PEiD

- Try PEiD for common packers
- Nothing interesting…



Defender: Initialization

- Want to figure out where "Bad key, try again" msg comes from
  - But, Defender does not call any API???
  - So, no obvious place to set break point
- What to do?
- Look at initialization routine…

Initialization Disassembly I

```
.h3mf85n:00404232 start          proc near
.h3mf85n:00404232
.h3mf85n:00404232 var_8          = dword ptr -8
.h3mf85n:00404232 var_4          = dword ptr -4
.h3mf85n:00404232
.h3mf85n:00404232                push    ebp
.h3mf85n:00404233                mov     ebp, esp
.h3mf85n:00404235                push    ecx
.h3mf85n:00404236                push    ecx
.h3mf85n:00404237                push    esi
.h3mf85n:00404238                push    edi
.h3mf85n:00404239                call    sub_402EA8
.h3mf85n:0040423E                push    eax
.h3mf85n:0040423F                call    loc_4033D1
.h3mf85n:00404244                mov     eax, dword_406000
.h3mf85n:00404249                pop     ecx
.h3mf85n:0040424A                mov     ecx, eax
.h3mf85n:0040424C                mov     eax, [eax]
.h3mf85n:0040424E                mov     edi, 6DEF20h
.h3mf85n:00404253                xor     esi, esi
.h3mf85n:00404255                jmp     short loc_404260
.h3mf85n:00404257 ; ---------------------------------------------------
```

Initialization Disassembly II

```
.h3mf85n:00404257
.h3mf85n:00404257 loc_404257:                    ; CODE XREF: start+30_j
.h3mf85n:00404257                 cmp     eax, edi
.h3mf85n:00404259                 jz      short loc_404283
.h3mf85n:0040425B                 add     ecx, 8
.h3mf85n:0040425E                 mov     eax, [ecx]
.h3mf85n:00404260
.h3mf85n:00404260 loc_404260:                    ; CODE XREF: start+23_j
.h3mf85n:00404260                 cmp     eax, esi
.h3mf85n:00404262                 jnz     short loc_404257
.h3mf85n:00404264                 xor     eax, eax
.h3mf85n:00404266
.h3mf85n:00404266 loc_404266:                    ; CODE XREF: start+5A_j
.h3mf85n:00404266                 lea     ecx, [ebp+var_8]
.h3mf85n:00404269                 push    ecx
.h3mf85n:0040426A                 push    esi
.h3mf85n:0040426B                 mov     [ebp+var_8], esi
.h3mf85n:0040426E                 mov     [ebp+var_4], esi
.h3mf85n:00404271                 call    eax
.h3mf85n:00404273                 call    loc_404202
.h3mf85n:00404278                 mov     eax, dword_406000
.h3mf85n:0040427D                 mov     ecx, eax
.h3mf85n:0040427F                 mov     eax, [eax]
.h3mf85n:00404281                 jmp     short loc_404297
```

Initialization Disassembly III

```
.h3mf85n:00404283 ; ---------------------------------------------------
.h3mf85n:00404283
.h3mf85n:00404283 loc_404283:                    ; CODE XREF: start+27_j
.h3mf85n:00404283                 mov     eax, [ecx+4]
.h3mf85n:00404286                 add     eax, dword_40601C
.h3mf85n:0040428C                 jmp     short loc_404266
.h3mf85n:0040428E ; ---------------------------------------------------
.h3mf85n:0040428E
.h3mf85n:0040428E loc_40428E:                    ; CODE XREF: start+67_j
.h3mf85n:0040428E                 cmp     eax, edi
.h3mf85n:00404290                 jz      short loc_4042BA
.h3mf85n:00404292                 add     ecx, 8
.h3mf85n:00404295                 mov     eax, [ecx]
.h3mf85n:00404297
.h3mf85n:00404297 loc_404297:                    ; CODE XREF: start+4F_j
.h3mf85n:00404297                 cmp     eax, esi
.h3mf85n:00404299                 jnz     short loc_40428E
.h3mf85n:0040429B                 xor     eax, eax
.h3mf85n:0040429D
.h3mf85n:0040429D loc_40429D:                    ; CODE XREF: start+91_j
.h3mf85n:0040429D                 lea     ecx, [ebp+var_8]
.h3mf85n:004042A0                 push    ecx
.h3mf85n:004042A1                 push    esi
.h3mf85n:004042A2                 mov     [ebp+var_8], esi
```

Initialization Disassembly IV

```
.h3mf85n:004042A5                 mov     [ebp+var_4], esi
.h3mf85n:004042A8                 call    eax
.h3mf85n:004042AA                 call    loc_401746
.h3mf85n:004042AF                 mov     eax, dword_406000
.h3mf85n:004042B4                 mov     ecx, eax
.h3mf85n:004042B6                 mov     eax, [eax]
.h3mf85n:004042B8                 jmp     short loc_4042CE
.h3mf85n:004042BA ; ---------------------------------------------------
.h3mf85n:004042BA
.h3mf85n:004042BA loc_4042BA:                     ; CODE XREF: start+5E_j
.h3mf85n:004042BA                 mov     eax, [ecx+4]
.h3mf85n:004042BD                 add     eax, dword_40601C
.h3mf85n:004042C3                 jmp     short loc_40429D
.h3mf85n:004042C5 ; ---------------------------------------------------
.h3mf85n:004042C5
.h3mf85n:004042C5 loc_4042C5:                     ; CODE XREF: start+9E_j
.h3mf85n:004042C5                 cmp     eax, edi
.h3mf85n:004042C7                 jz      short loc_4042F5
.h3mf85n:004042C9                 add     ecx, 8
.h3mf85n:004042CC                 mov     eax, [ecx]
.h3mf85n:004042CE
```

Initialization Disassembly V

```
.h3mf85n:004042CE loc_4042CE:                    ; CODE XREF: start+86_j
.h3mf85n:004042CE                 cmp     eax, esi
.h3mf85n:004042D0                 jnz     short loc_4042C5
.h3mf85n:004042D2                 xor     ecx, ecx
.h3mf85n:004042D4
.h3mf85n:004042D4 loc_4042D4:                    ; CODE XREF: start+CC_j
.h3mf85n:004042D4                 lea     eax, [ebp+var_8]
.h3mf85n:004042D7                 push    eax
.h3mf85n:004042D8                 push    esi
.h3mf85n:004042D9                 mov     [ebp+var_8], esi
.h3mf85n:004042DC                 mov     [ebp+var_4], esi
.h3mf85n:004042DF                 call    ecx
.h3mf85n:004042E1                 call    loc_402082
.h3mf85n:004042E6                 call    ds:IsDebuggerPresent
.h3mf85n:004042EC                 xor     eax, eax
.h3mf85n:004042EE                 pop     edi
.h3mf85n:004042EF                 inc     eax
.h3mf85n:004042F0                 pop     esi
.h3mf85n:004042F1                 leave
.h3mf85n:004042F2                 retn    8
.h3mf85n:004042F5 ; ---------------------------------------------------
.h3mf85n:004042F5
.h3mf85n:004042F5 loc_4042F5:                    ; CODE XREF: start+95_j
.h3mf85n:004042F5                 mov     ecx, [ecx+4]
.h3mf85n:004042F8                 add     ecx, dword_40601C
.h3mf85n:004042FE                 jmp     short loc_4042D4
.h3mf85n:004042FE start           endp
```

Initialization

- Consider this code

```
mf85n:00402EA8 sub_402EA8      proc near
.h3mf85n:00402EA8
.h3mf85n:00402EA8 var_4           = dword ptr -4
.h3mf85n:00402EA8
.h3mf85n:00402EA8                 push    ecx
.h3mf85n:00402EA9                 mov     eax, large fs:30h
.h3mf85n:00402EAF                 mov     [esp+4+var_4], eax
.h3mf85n:00402EB2                 mov     eax, [esp+4+var_4]
.h3mf85n:00402EB5                 mov     eax, [eax+0Ch]
.h3mf85n:00402EB8                 mov     eax, [eax+0Ch]
.h3mf85n:00402EBB                 mov     eax, [eax]
.h3mf85n:00402EBD                 mov     eax, [eax+18h]
.h3mf85n:00402EC0                 pop     ecx
.h3mf85n:00402EC1                 retn
.h3mf85n:00402EC1 sub_402EA8      endp
```

- fs register for thread-related info
  - What's at offset "+30"?

Initialization

- For any thread fs:0 is "Thread Environment Block" (TEB)
- What to do?
- Look up the TEB data stucture…

TEB

```
+0x000 NtTib              : _NT_TIB
+0x01c EnvironmentPointer : Ptr32 Void
+0x020 ClientId           : _CLIENT_ID
+0x028 ActiveRpcHandle    : Ptr32 Void
+0x02c ThreadLocalStoragePointer : Ptr32 Void
+0x030 ProcessEnvironmentBlock : Ptr32 _PEB
.
.
```

- At +30 we have PEB
  - Process Environment Block
- Just like TEB, but for a process
  - Program access +c in PEB
- So, program accesses PEB via TEB

PEB

```
+0x000 InheritedAddressSpace : UChar
+0x001 ReadImageFileExecOptions : UChar
+0x002 BeingDebugged    : UChar
+0x003 SpareBool        : UChar
+0x004 Mutant           : Ptr32 Void
+0x008 ImageBaseAddress : Ptr32 Void
+0x00c Ldr              : Ptr32 _PEB_LDR_DATA
.
.
```

- What is at +c in PEB?
  - _PEB_LDR_DATA
- Go look at that data structure…

_PEB_LDR_DATA

- Program get +c here too

```
+0x000 Length              : Uint4B
+0x004 Initialized         : UChar
+0x008 SsHandle            : Ptr32 Void
+0x00c InLoadOrderModuleList : _LIST_ENTRY
+0x014 InMemoryOrderModuleList : _LIST_ENTRY
+0x01c InInitializationOrderModuleList : _LIST_ENTRY
+0x024 EntryInProgress     : Ptr32 Void
```

- LIST_ENTRY
- Look at data structure (next slide)

LIST_ENTRY

- Goes to offset +0 here
  - That is, LIST_ENTRY again

```
+0x000 InLoadOrderLinks : _LIST_ENTRY
+0x008 InMemoryOrderLinks : _LIST_ENTRY
+0x010 InInitializationOrderLinks : _LIST_ENTRY
+0x018 DllBase          : Ptr32 Void
+0x01c EntryPoint       : Ptr32 Void
+0x020 SizeOfImage      : Uint4B
+0x024 FullDllName      : _UNICODE_STRING
+0x02c BaseDllName      : _UNICODE_STRING
+0x034 Flags            : Uint4B
+0x038 LoadCount        : Uint2B
+0x03a TlsIndex         : Uint2B
+0x03c HashLinks        : _LIST_ENTRY
+0x03c SectionPointer   : Ptr32 Void
+0x040 CheckSum         : Uint4B
+0x044 TimeDateStamp    : Uint4B
+0x044 LoadedImports    : Ptr32 Void
+0x048 EntryPointActivationContext : Ptr32 _ACTIVATION_CONTEXT
+0x04c PatchInformation : Ptr32 Void
```

LIST_ENTRY

- Goes to offset +18 here
  - That is, DllBase

```
+0x000 InLoadOrderLinks : _LIST_ENTRY
+0x008 InMemoryOrderLinks : _LIST_ENTRY
+0x010 InInitializationOrderLinks : _LIST_ENTRY
+0x018 DllBase           : Ptr32 Void
+0x01c EntryPoint        : Ptr32 Void
+0x020 SizeOfImage       : Uint4B
+0x024 FullDllName       : _UNICODE_STRING
+0x02c BaseDllName       : _UNICODE_STRING
+0x034 Flags             : Uint4B
+0x038 LoadCount         : Uint2B
+0x03a TlsIndex          : Uint2B
+0x03c HashLinks         : _LIST_ENTRY
+0x03c SectionPointer    : Ptr32 Void
+0x040 CheckSum          : Uint4B
+0x044 TimeDateStamp     : Uint4B
+0x044 LoadedImports     : Ptr32 Void
+0x048 EntryPointActivationContext : Ptr32 _ACTIVATION_CONTEXT
+0x04c PatchInformation : Ptr32 Void
```

What Does it all Mean?

- After all of that, program has found base of some DLL
- Dump loader data structures
  - InLoadOrderModuleList from PEB_LDR_DATA
  - Next slide…

Initialization

```
0:000> !dlls -l

0x00241ee0: C:\Documents and Settings\Eldad Eilam\Defender.exe
      Base   0x00400000  EntryPoint  0x00404232  Size        0x00008000
      Flags  0x00005000  LoadCount   0x0000ffff  TlsIndex    0x00000000
             LDRP_LOAD_IN_PROGRESS
             LDRP_ENTRY_PROCESSED

0x00241f48: C:\WINDOWS\system32\ntdll.dll
      Base   0x7c900000  EntryPoint  0x7c913156  Size        0x000b0000
      Flags  0x00085004  LoadCount   0x0000ffff  TlsIndex    0x00000000
             LDRP_IMAGE_DLL
             LDRP_LOAD_IN_PROGRESS
             LDRP_ENTRY_PROCESSED
             LDRP_PROCESS_ATTACH_CALLED

0x00242010: C:\WINDOWS\system32\kernel32.dll
      Base   0x7c800000  EntryPoint  0x7c80b436  Size        0x000f4000
      Flags  0x00085004  LoadCount   0x0000ffff  TlsIndex    0x00000000
             LDRP_IMAGE_DLL
             LDRP_LOAD_IN_PROGRESS
             LDRP_ENTRY_PROCESSED
             LDRP_PROCESS_ATTACH_CALLED
```

Initialization

- Bottom line?
- The function at 00402EA8 obtains in-memory address of NTDLL.DLL
- Program must communicate with OS
  - And this is a highly obfuscated way to (begin to) do so!

Initialization

- Then what?
- Next, goes to function at 004033D1
- Listing starts on next slide…

Function at 004033D1

```
loc_4033D1:
.h3mf85n:004033D1                push    ebp
.h3mf85n:004033D2                mov     ebp, esp
.h3mf85n:004033D4                sub     esp, 22Ch
.h3mf85n:004033DA                push    ebx
.h3mf85n:004033DB                push    esi
.h3mf85n:004033DC                push    edi
.h3mf85n:004033DD                push    offset dword_4034DD
.h3mf85n:004033E2                pop     eax
.h3mf85n:004033E3                mov     [ebp-20h], eax
.h3mf85n:004033E6                push    offset loc_4041FD
.h3mf85n:004033EB                pop     eax
.h3mf85n:004033EC                mov     [ebp-18h], eax
.h3mf85n:004033EF                mov     eax, offset dword_4034E5
.h3mf85n:004033F4                mov     ds:dword_4034D6, eax
.h3mf85n:004033FA                mov     dword ptr [ebp-8], 1
.h3mf85n:00403401                cmp     dword ptr [ebp-8], 0
.h3mf85n:00403405                jz      short loc_40346D
.h3mf85n:00403407                mov     eax, [ebp-18h]
.h3mf85n:0040340A                sub     eax, [ebp-20h]
.h3mf85n:0040340D                mov     [ebp-30h], eax
```

Function at 004033D1

```
.h3mf85n:00403410                    mov     eax, [ebp-20h]
.h3mf85n:00403413                    mov     [ebp-34h], eax
.h3mf85n:00403416                    and     dword ptr [ebp-24h], 0
.h3mf85n:0040341A                    and     dword ptr [ebp-28h], 0
.h3mf85n:0040341E loc_40341E:        ; CODE XREF: .h3mf85n:00403469_j
.h3mf85n:0040341E                    cmp     dword ptr [ebp-30h], 3
.h3mf85n:00403422                    jbe     short loc_40346B
.h3mf85n:00403424                    mov     eax, [ebp-34h]
.h3mf85n:00403427                    mov     eax, [eax]
.h3mf85n:00403429                    mov     [ebp-2Ch], eax
.h3mf85n:0040342C                    mov     eax, [ebp-34h]
.h3mf85n:0040342F                    mov     eax, [eax]
.h3mf85n:00403431                    xor     eax, 2BCA6179h
.h3mf85n:00403436                    mov     ecx, [ebp-34h]
.h3mf85n:00403439                    mov     [ecx], eax
.h3mf85n:0040343B                    mov     eax, [ebp-34h]
.h3mf85n:0040343E                    mov     eax, [eax]
.h3mf85n:00403440                    xor     eax, [ebp-28h]
.h3mf85n:00403443                    mov     ecx, [ebp-34h]
.h3mf85n:00403446                    mov     [ecx], eax
.h3mf85n:00403448                    mov     eax, [ebp-2Ch]
.h3mf85n:0040344B                    mov     [ebp-28h], eax
.h3mf85n:0040344E                    mov     eax, [ebp-24h]
.h3mf85n:00403451                    xor     eax, [ebp-2Ch]
```

Function at 004033D1

```
.h3mf85n:00403454                 mov     [ebp-24h], eax
.h3mf85n:00403457                 mov     eax, [ebp-34h]
.h3mf85n:0040345A                 add     eax, 4
.h3mf85n:0040345D                 mov     [ebp-34h], eax
.h3mf85n:00403460                 mov     eax, [ebp-30h]
.h3mf85n:00403463                 sub     eax, 4
.h3mf85n:00403466                 mov     [ebp-30h], eax
.h3mf85n:00403469                 jmp     short loc_40341E
.h3mf85n:0040346B ; ---------------------------------------------------
.h3mf85n:0040346B
.h3mf85n:0040346B loc_40346B:     ; CODE XREF: .h3mf85n:00403422_j
.h3mf85n:0040346B                 jmp     short near ptr unk_4034D5
.h3mf85n:0040346D ; ---------------------------------------------------
.h3mf85n:0040346D
.h3mf85n:0040346D loc_40346D:     ; CODE XREF: .h3mf85n:00403405_j
.h3mf85n:0040346D                 mov     eax, [ebp-18h]
.h3mf85n:00403470                 sub     eax, [ebp-20h]
.h3mf85n:00403473                 mov     [ebp-40h], eax
.h3mf85n:00403476                 mov     eax, [ebp-20h]
.h3mf85n:00403479                 mov     [ebp-44h], eax
.h3mf85n:0040347C                 and     dword ptr [ebp-38h], 0
.h3mf85n:00403480                 and     dword ptr [ebp-3Ch], 0
.h3mf85n:00403484
.h3mf85n:00403484 loc_403484:     ; CODE XREF: .h3mf85n:004034CB_j
.h3mf85n:00403484                 cmp     dword ptr [ebp-40h], 3
```

Function at 004033D1

```
.h3mf85n:00403488                jbe     short loc_4034CD
.h3mf85n:0040348A                mov     eax, [ebp-44h]
.h3mf85n:0040348D                mov     eax, [eax]
.h3mf85n:0040348F                xor     eax, [ebp-3Ch]
.h3mf85n:00403492                mov     ecx, [ebp-44h]
.h3mf85n:00403495                mov     [ecx], eax
.h3mf85n:00403497                mov     eax, [ebp-44h]
.h3mf85n:0040349A                mov     eax, [eax]
.h3mf85n:0040349C                xor     eax, 2BCA6179h
.h3mf85n:004034A1                mov     ecx, [ebp-44h]
.h3mf85n:004034A4                mov     [ecx], eax
.h3mf85n:004034A6                mov     eax, [ebp-44h]
.h3mf85n:004034A9                mov     eax, [eax]
.h3mf85n:004034AB                mov     [ebp-3Ch], eax
.h3mf85n:004034AE                mov     eax, [ebp-44h]
.h3mf85n:004034B1                mov     ecx, [ebp-38h]
.h3mf85n:004034B4                xor     ecx, [eax]
.h3mf85n:004034B6                mov     [ebp-38h], ecx
.h3mf85n:004034B9                mov     eax, [ebp-44h]
.h3mf85n:004034BC                add     eax, 4
.h3mf85n:004034BF                mov     [ebp-44h], eax
.h3mf85n:004034C2                mov     eax, [ebp-40h]
.h3mf85n:004034C5                sub     eax, 4
.h3mf85n:004034C8                mov     [ebp-40h], eax
.h3mf85n:004034CB                jmp     short loc_403484
.h3mf85n:004034CD ; ---------------------------------------------------
```

Function at 004033D1

- Boxed part represents **12 pages** of "data"
- Why all of this data embedded in code???

```
.h3mf85n:004034CD
.h3mf85n:004034CD loc_4034CD:     ; CODE XREF: .h3mf85n:00403488_j
.h3mf85n:004034CD                 mov     eax, [ebp-38h]
.h3mf85n:004034D0                 mov     dword_406008, eax
.h3mf85n:004034D0 ; ---------------------------------------------------
.h3mf85n:004034D5 db  68h         ; CODE XREF: .h3mf85n:loc_40346B_j
.h3mf85n:004034D6 dd 4034E5h      ; DATA XREF: .h3mf85n:004033F4_w
.h3mf85n:004034DA ; ---------------------------------------------------
.h3mf85n:004034DA                 pop     ebx
.h3mf85n:004034DB                 jmp     ebx
.h3mf85n:004034DB ; ---------------------------------------------------
.h3mf85n:004034DD dword_4034DD    dd 0DDF8286Bh, 2A7B348Ch
.h3mf85n:004034E5 dword_4034E5    dd 88B9107Eh, 0E6F8C142h, 7D7F2B8Bh,
                                 0DF8902F1h, 0B1C8CBC5h
.
.
.
.h3mf85n:00403CE5                 dd 157CB335h
.h3mf85n:004041FD ; ---------------------------------------------------
.h3mf85n:004041FD
.h3mf85n:004041FD loc_4041FD:     ; DATA XREF: .h3mf85n:004033E6_o
.h3mf85n:004041FD                 pop     edi
.h3mf85n:004041FE                 pop     esi
.h3mf85n:004041FF                 pop     ebx
.h3mf85n:00404200                 leave
.h3mf85n:00404201                 retn
```

Function at 004033D1

- "Data" is probably encrypted code
  - Goes from 4034DD to 403CE5
- What about unencrypted parts?
- Looks like a big if-then-else
  - But one clause looks like it's "dead"
- So look at the "live" branch…

Function at 004033D1

- Note XOR at 403431
  - Appear to be XORing within a loop
  - Note that XORing a constant value
- Beginning at 4033DD we see 4034DD put into [ebp-20h], via the stack
  - What's special about address 4034DD??
- At 403410, use [ebp-20h] to get initial address for XORing
- Aha --- the decryption loop!

Decrypted Code

- Use OllyDbg and breakpt at end of decryption loop (40346B)
- Then OllyDbg shows the following

```
004034DD    12          DB 12
004034DE    49          DB 49
004034DF    32          DB 32
004034E0    F6          DB F6
004034E1    9E          DB 9E
004034E2    7D          DB 7D
```

- Tell OllyDbg to re-analyze code
  - Reveals many pages of decrypted code

Decrypted Code

- Code digs thru NTDLL's PE header
  - Gets export directory
- For each export, "performs an interesting … bit of arithmetic on each function name string"
- Code is on next slide…

Unusual Calculation

- Debugger: [ebp-68] is len. of current string
  - [ebp-64] has its address
- Then for each char in string, shifts left by its index, modulo 24
- What the… ?
- It's a "checksum"

```
004035A4    MOV EAX,DWORD PTR [EBP-68]
004035A7    MOV ECX,DWORD PTR [EBP-68]
004035AA    DEC ECX
004035AB    MOV DWORD PTR [EBP-68],ECX
004035AE    TEST EAX,EAX
004035B0    JE SHORT Defender.004035D0
004035B2    MOV EAX,DWORD PTR [EBP-64]
004035B5    ADD EAX,DWORD PTR [EBP-68]
004035B8    MOVSX ESI,BYTE PTR [EAX]
004035BB    MOV EAX,DWORD PTR [EBP-68]
004035BE    CDQ
004035C2    IDIV ECX
004035C4    MOV ECX,EDX
004035C6    SHL ESI,CL
004035C8    ADD ESI,DWORD PTR [EBP-6C]
004035CB    MOV DWORD PTR [EBP-6C],ESI
004035CE    JMP SHORT Defender.004035A4
```

NTDLL

- After all chars have been processed…

```
004035D0    CMP DWORD PTR [EBP-6C],39DBA17A
004035D7    JNZ SHORT Defender.004035F1
```

- What's going on here?
- Looking for an export entry (NTDLL) that has "checksum" 39DBA17A
- Put a breakpoint on line after JNZ…
  - …and [ebp-64] shows you what was found

Allocate Memory

- It turns out that it calls
  - NtAllocateVirtualMemory
- Which is (undocumented) native API equivalent of document API
  - VirtualAlloc
- It's for allocating memory pages

Read Time-stamp Counter

- Code to call NtAllocateVirtualMemory
- What is RDTSC?
  - "Read time-stamp counter"
  - A 64-bit counter, incremented at each tick

```
0040365F    RDTSC
00403661    AND EAX,7FFF0000
00403666    MOV DWORD PTR [EBP-C],EAX
00403669    PUSH 4
0040366B    PUSH 3000
00403670    LEA EAX,DWORD PTR [EBP-4]
00403673    PUSH EAX
00403674    PUSH 0
00403676    LEA EAX,DWORD PTR [EBP-C]
00403679    PUSH EAX
0040367A    PUSH -1
0040367C    CALL DWORD PTR [EBP-10]
```

Parameters

- Timestamp bits ANDed with constant
- 2nd parameter to memory alloc. function
- Look at function prototype
  - Undocumented

```
0040365F    RDTSC
00403661    AND EAX,7FFF0000
00403666    MOV DWORD PTR [EBP-C],EAX
00403669    PUSH 4
0040366B    PUSH 3000
00403670    LEA EAX,DWORD PTR [EBP-4]
00403673    PUSH EAX
00403674    PUSH 0
00403676    LEA EAX,DWORD PTR [EBP-C]
00403679    PUSH EAX
0040367A    PUSH -1
0040367C    CALL DWORD PTR [EBP-10]
```

Base Address

- 2nd param points to "base address"
- This is where memory will be allocated

```
NTSYSAPI
NTSTATUS
NTAPI
NtAllocateVirtualMemory(
  IN HANDLE              ProcessHandle,
  IN OUT PVOID           *BaseAddress,
  IN ULONG               ZeroBits,
  IN OUT PULONG          RegionSize,
  IN ULONG               AllocationType,
  IN ULONG               Protect );
```

Allocate Memory

- What just happened?
- Generated a "random" number using timer
- Use this random number as location (base address) for allocated memory
- Interesting idea!

Parameters

- Consider also 4th parameter
  - This gives the allocated block size
- Loaded from [ebp-4]
- Code on next slide involved with find block size…

Parameters

- Consider 4th parameter
- Recall [ebp+8] is NTDLL base addr
- Accesses PE hdr
- Ptr to PE hdr stored in [ebp-74]
- Get offset +1c

```
004035FE    MOV EAX,DWORD PTR [EBP+8]
00403601    MOV DWORD PTR [EBP-70],EAX
00403604    MOV EAX,DWORD PTR [EBP-70]
00403607    MOV ECX,DWORD PTR [EBP-70]
0040360A    ADD ECX,DWORD PTR [EAX+3C]
0040360D    MOV DWORD PTR [EBP-74],ECX
00403610    MOV EAX,DWORD PTR [EBP-74]
00403613    MOV EAX,DWORD PTR [EAX+1C]
00403616    MOV DWORD PTR [EBP-78],EAX
```

Parameters

- PE header ==>
- What's at +1c?
  - That is, at +4 in OptionalHeader
- SizeOfCode

```
0:000> dt _IMAGE_NT_HEADERS -b
   +0x000 Signature        : Uint4B
   +0x004 FileHeader       :
      +0x000 Machine           : Uint2B
      +0x002 NumberOfSections : Uint2B
      +0x004 TimeDateStamp     : Uint4B
      +0x008 PointerToSymbolTable : Uint4B
      +0x00c NumberOfSymbols   : Uint4B
      +0x010 SizeOfOptionalHeader : Uint2B
      +0x012 Characteristics   : Uint2B
   +0x018 OptionalHeader    :
      +0x000 Magic             : Uint2B
      +0x002 MajorLinkerVersion : UChar
      +0x003 MinorLinkerVersion : UChar
      +0x004 SizeOfCode        : Uint4B
      +0x008 SizeOfInitializedData : Uint4B
      +0x00c SizeOfUninitializedData : Uint4B
      +0x010 AddressOfEntryPoint : Uint4B
      +0x014 BaseOfCode        : Uint4B
      +0x018 BaseOfData        : Uint4B
             .
             .
```

Size Calculation

- Code below related to size calculation
- Value read from [ebp-7c] points into NTDLL header
  - Beginning of NTDLL's export directory

```
0040363D    MOV EAX,DWORD PTR [EBP-7C]
00403640    MOV EAX,DWORD PTR [EAX+18]
00403643    MOV DWORD PTR [EBP-88],EAX
```

- Q: What's at offset +18?
- A: NumberOfFunctions

Block Size

- Final preparation of block size

```
00403649    MOV EAX,DWORD PTR [EBP-88]
0040364F    MOV ECX,DWORD PTR [EBP-78]
00403652    LEA EAX,DWORD PTR [ECX+EAX*8+8]
```

- So computed block size is…
  - NTDLLcodesize + NumExports * 8 + 8
- Why?
- Not clear at this point…

Checksum

- Another strange checksum
  - This time, NTDLL's export list
- Includes following 2 lines:

```
0040380F    MOV DWORD PTR DS:[ECX+EAX*8],EDX

00403840    MOV DWORD PTR DS:[EDX+ECX*8+4],EAX
```

- First, is function's checksum
- Second is function's RVA

Interesting Code

- More "interesting" code

```
004038FD    MOV EAX,DWORD PTR [EBP-C8]
00403903    MOV ESI,DWORD PTR [EBP+8]
00403906    ADD ESI,DWORD PTR [EAX+2C]
00403909    MOV EAX,DWORD PTR [EBP-D8]
0040390F    MOV EDX,DWORD PTR [EBP-C]
00403912    LEA EDI,DWORD PTR [EDX+EAX*8+8]
00403916    MOV EAX,ECX
00403918    SHR ECX,2
0040391B    REP MOVS DWORD PTR ES:[EDI],DWORD PTR [ESI]
0040391D    MOV ECX,EAX
0040391F    AND ECX,3
00403922    REP MOVS BYTE PTR ES:[EDI],BYTE PTR [ESI]
```

Memory Copy

- Code on previous slide is a common "sentence" in assembly code
- A memory copy
  - REP MOV repeatedly copies DWORDS from address at ESI to address at EDI until ECX is 0
- So, what is being copied?

Memory Copy

- ESI is loaded with [ebp+8]
- Why is that familiar?
- NTDLL's base address
- Then increment by value at [eax+2c]
  - BaseOfCode
- EDI gets addr of new memory block

What Just Happened?

- To recap…
- Memory allocated at random location
- In this memory, write a table of
  - Checksums of NTDLL exported functions
  - Corresponding RVAs
- Finally, write a copy of entire NTDLL code section

Data Structure

- Representation of description on previous slide

Function Name Checksum	Function's RVA
Function Name Checksum	Function's RVA
⋮	
Function Name Checksum	Function's RVA
Copy of NTDLL Code Section	

What's Next?

- After this, next function starts with…

```
00403108    CMP ESI,190BC2
0040310E    JE SHORT Defender.0040311E
00403110    ADD ECX,8
00403113    MOV ESI,DWORD PTR [ECX]
00403115    CMP ESI,EBX
00403117    JNZ SHORT Defender.00403108
```

- Followed by…

```
0040311E    MOV ECX,DWORD PTR [ECX+4]
00403121    ADD ECX,EDI
00403123    MOV DWORD PTR [EBP-C],ECX
```

Searching For…

- What does this do?

```
00403108    CMP ESI,190BC2
0040310E    JE SHORT Defender.0040311E
00403110    ADD ECX,8
00403113    MOV ESI,DWORD PTR [ECX]
00403115    CMP ESI,EBX
00403117    JNZ SHORT Defender.00403108
```

- Goes thru export table…
- …looking for checksum 190BC2
- That is, looking for a specific API

Found It —But What Is It?

- This is what happens when entry found

```
0040311E    MOV ECX,DWORD PTR [ECX+4]
00403121    ADD ECX,EDI
00403123    MOV DWORD PTR [EBP-C],ECX
```

- Where have we (just) seen offset +4?
- Apparently, that's the RVA
  - Gets added to "base address" of NTDLL

Leaving User Mode

- Later, we have this…

```
7D03F0F2    MOV EAX,35
7D03F0F7    MOV EDX,7FFE0300
7D03F0FC    CALL DWORD PTR [EDX]
7D03F0FE    RET 20
```

- …which (eventually) calls this

```
7C90EB8B    MOV EDX,ESP
7C90EB8D    SYSENTER
```

- SYSENTER is "kernel-mode switch"
  - So cannot follow with OllyDbg

What Now?

- How to determine which system call?
- Three choices…
  - Switch to kernel mode debugger (SoftICE)
  - Find RVA from checksum table (it's probably the same as actual RVA in NTDLL)
  - Find system call based on order in checksum list (and hope order wasn't changed)
- Author chooses first option — SoftICE

System Call

- First, it goes into KiSystemService
  - All system calls go thru this function
  - Look for CALL EBX, which transfers to actual system call
  - In this case, it's NtAllocateVirtualMemory
  - Again???
- Then back to user mode…
- …and program calls NtCreateThread

Thread and Then…

- After creating thread, calls "function" 006DEF20
- Find that this is NtDelayExecution
  - Equivalent to SleepEx
- This should "cause new thread to execute immediately"
- Then calls "function" 403A41

Function 403A41

- Function call just skips ahead 30 bytes
- Those 30 bytes consist of…

```
004039FA  K.E.R.N.E.L.3.2...D.L.L.
```

- Function's only purpose is to avoid "executing" this string!
- Then searches for 2 more "functions"
  - 6DEF20 and 1974C

SoftICE Disappears

- Before getting to function 1974C, SoftICE disappears
  - Defender has quit
- Apparently, secondary thread has killed primary thread
  - Secondary thread that was just created

Reversing Secondary Thread

- This code is encrypted, like before
- Set breakpoint after it's decrypted
- Obtain code on next few slides…

Function at 00402FFE (I)

- More dead code at line 4030C7?
- Note RDTSC at line 403007

```
00402FFE   XOR EAX,EAX
00403000   INC EAX
00403001   JE Defender.004030C7
00403007   RDTSC
00403009   MOV DWORD PTR SS:[EBP-8],EAX
0040300C   MOV DWORD PTR SS:[EBP-4],EDX
0040300F   MOV EAX,DWORD PTR DS:[406000]
00403014   MOV DWORD PTR SS:[EBP-50],EAX
00403017   MOV EAX,DWORD PTR SS:[EBP-50]
0040301A   CMP DWORD PTR DS:[EAX],0
0040301D   JE SHORT Defender.00403046
0040301F   MOV EAX,DWORD PTR SS:[EBP-50]
00403022   CMP DWORD PTR DS:[EAX],6DEF20
00403028   JNZ SHORT Defender.0040303B
0040302A   MOV EAX,DWORD PTR SS:[EBP-50]
0040302D   MOV ECX,DWORD PTR DS:[40601C]
00403033   ADD ECX,DWORD PTR DS:[EAX+4]
00403036   MOV DWORD PTR SS:[EBP-44],ECX
00403039   JMP SHORT Defender.0040304A
0040303B   MOV EAX,DWORD PTR SS:[EBP-50]
```

Function at 00402FFE (II)

- Note second RDTSC
- Subtracted from first RDTSC ???

```
0040303E   ADD EAX,8
00403041   MOV DWORD PTR SS:[EBP-50],EAX
00403044   JMP SHORT Defender.00403017
00403046   AND DWORD PTR SS:[EBP-44],0
0040304A   AND DWORD PTR SS:[EBP-4C],0
0040304E   AND DWORD PTR SS:[EBP-48],0
00403052   LEA EAX,DWORD PTR SS:[EBP-4C]
00403055   PUSH EAX
00403056   PUSH 0
00403058   CALL DWORD PTR SS:[EBP-44]
0040305B   RDTSC
0040305D   MOV DWORD PTR SS:[EBP-18],EAX
00403060   MOV DWORD PTR SS:[EBP-14],EDX
00403063   MOV EAX,DWORD PTR SS:[EBP-18]
00403066   SUB EAX,DWORD PTR SS:[EBP-8]
00403069   MOV ECX,DWORD PTR SS:[EBP-14]
0040306C   SBB ECX,DWORD PTR SS:[EBP-4]
```

Function at 00402FFE (III)

- Infinite loop at line 4030C2?
- Comparison with constant at line 403077…
- What "function" is 1BF08AE?

```
0040306F   MOV DWORD PTR SS:[EBP-60],EAX
00403072   MOV DWORD PTR SS:[EBP-5C],ECX
00403075   JNZ SHORT Defender.00403080
00403077   CMP DWORD PTR SS:[EBP-60],77359400
0040307E   JBE SHORT Defender.004030C2
00403080   MOV EAX,DWORD PTR DS:[406000]
00403085   MOV DWORD PTR SS:[EBP-58],EAX
00403088   MOV EAX,DWORD PTR SS:[EBP-58]
0040308B   CMP DWORD PTR DS:[EAX],0
0040308E   JE SHORT Defender.004030B7
00403090   MOV EAX,DWORD PTR SS:[EBP-58]
00403093   CMP DWORD PTR DS:[EAX],1BF08AE
00403099   JNZ SHORT Defender.004030AC
0040309B   MOV EAX,DWORD PTR SS:[EBP-58]
0040309E   MOV ECX,DWORD PTR DS:[40601C]
004030A4   ADD ECX,DWORD PTR DS:[EAX+4]
004030A7   MOV DWORD PTR SS:[EBP-54],ECX
004030AA   JMP SHORT Defender.004030BB
004030AC   MOV EAX,DWORD PTR SS:[EBP-58]
004030AF   ADD EAX,8
004030B2   MOV DWORD PTR SS:[EBP-58],EAX
004030B5   JMP SHORT Defender.00403088
004030B7   AND DWORD PTR SS:[EBP-54],0
004030BB   PUSH 0
004030BD   PUSH -1
004030BF   CALL DWORD PTR SS:[EBP-54]
004030C2   JMP Defender.00402FFE
```

"Function" at 1BF08AE

- Stepping into this, the compare (almost) always fails
- This code is checking a to see if process is paused
  - Recall the 2 calls to RTDSC
- If paused, process is terminated
- What's the purpose?

Defeating "Killer" Thread

- Patch code to avoid check…

```
00403075   NOP
00403076   NOP
00403077   CMP DWORD PTR SS:[EBP-60],77359400
0040307E   JMP SHORT Defender.004030C2
```

- However, you cannot save this change
  - So, must do this in each debug session
- Why can't you save this change?
  - Not clear at this point… we'll see later

"Function" 1974C

- This one is not a call into kernel
- Instead, code contained in NTDLL
- How to determine what API?
  - Use RVA or its order in table
  - Author uses order in export table
- Finds result on next slide…

Loading KERNEL32.DLL

```
ordinal hint RVA      name
.
.
70      3E   000161CA LdrLoadDll
```

- What is LdrLoadDll?
- Native API version of LoadLibrary
- What DLL is it loading?
- We saw a name earlier: KERNEL32.DLL

Loading KERNEL32.DLL

- As with NTDLL, Defender generates checksum/RVA table
- Then inserts code section of KERNEL32.DLL

After Loading KERNEL3.DLL

- Another "function" skips 30 bytes or so
- What are those bytes?

```
00404138  44 65 66 65 6E 64 65 72  Defender
00404140  20 56 65 72 73 69 6F 6E   Version
00404148  20 31 2E 30 20 2D 20 57   1.0 - W
00404150  72 69 74 74 65 6E 20 62  ritten b
00404158  79 20 45 6C 64 61 64 20  y Eldad
00404160  45 69 6C 61 6D           Eilam
```

- Defender's welcome message
  - Ready to be printed out!

KERNEL32.DLL

- Next, obfuscated call to something in KERNEL32.DLL
- What could this be?

```
00404167    PUSH DWORD PTR SS:[ESP]
0040416A    CALL Defender.004012DF
```

- No need to work too hard…
- …this must be printing welcome msg

Re-Encrypting

- At end of this function, we have

```
004041E2   MOV EAX,Defender.004041FD
004041E7   MOV DWORD PTR DS:[4034D6],EAX
004041ED   MOV DWORD PTR SS:[EBP-8],0
004041F4   JMP Defender.00403401
004041F9   LODS DWORD PTR DS:[ESI]
004041FA   DEC EDI
004041FB   ADC AL,0F2
004041FD   POP EDI
004041FE   POP ESI
004041FF   POP EBX
00404200   LEAVE
00404201   RETN
```

- JMP is far away, but we've been there…

Re-Encrypting

```
loc_4033D1:
.h3mf85n:004033D1                push    ebp
.h3mf85n:004033D2                mov     ebp, esp
.h3mf85n:004033D4                sub     esp, 22Ch
.h3mf85n:004033DA                push    ebx
.h3mf85n:004033DB                push    esi
.h3mf85n:004033DC                push    edi
.h3mf85n:004033DD                push    offset dword_4034DD
.h3mf85n:004033E2                pop     eax
.h3mf85n:004033E3                mov     [ebp-20h], eax
.h3mf85n:004033E6                push    offset loc_4041FD
.h3mf85n:004033EB                pop     eax
.h3mf85n:004033EC                mov     [ebp-18h], eax
.h3mf85n:004033EF                mov     eax, offset dword_4034E5
.h3mf85n:004033F4                mov     ds:dword_4034D6, eax
.h3mf85n:004033FA                mov     dword ptr [ebp-8], 1
.h3mf85n:00403401                cmp     dword ptr [ebp-8], 0
.h3mf85n:00403405                jz      short loc_40346D
.h3mf85n:00403407                mov     eax, [ebp-18h]
.h3mf85n:0040340A                sub     eax, [ebp-20h]
.h3mf85n:0040340D                mov     [ebp-30h], eax
```

Re-Encrypting

- Dead code … NOT!
- This code very similar to decryption
  - Convincing "dead code"?
- But actually encryption code
  - Computes checksum of encrypted code
  - Jumps to end of encrypted code
- Why re-encrypt???

Back at the Entry Point

- Blah

```
00404202   MOV EAX,DWORD PTR DS:[406004]
00404207   MOV ECX,EAX
00404209   MOV EAX,DWORD PTR DS:[EAX]
0040420B   JMP SHORT Defender.00404219
0040420D   CMP EAX,66B8EBBB
00404212   JE SHORT Defender.00404227
00404214   ADD ECX,8
00404217   MOV EAX,DWORD PTR DS:[ECX]
00404219   TEST EAX,EAX
0040421B   JNZ SHORT Defender.0040420D
0040421D   XOR ECX,ECX
0040421F   PUSH Defender.0040322E
00404224   CALL ECX
00404226   RETN
00404227   MOV ECX,DWORD PTR DS:[ECX+4]
0040422A   ADD ECX,DWORD PTR DS:[406014]
00404230   JMP SHORT Defender.0040421F
```

Back at the Entry Point

- Blah

```
00401785   MOV EAX,DWORD PTR DS:[406008]
0040178A   MOV DWORD PTR SS:[EBP-9C0],EAX
```

Parsing Parameters

- Blah

Parsing Parameters

- Blah

```
00402994    TEST EAX,EAX
00402996    JNZ Defender.00402AC4
0040299C    CALL Defender.004029EC
```

Parsing Parameters

- Blah

```
004029A1  42 61 64 20 70 61 72 61  Bad para
004029A9  6D 65 74 65 72 73 21 0A  meters!.
004029B1  55 73 61 67 65 3A 20 44  Usage: D
004029B9  65 66 65 6E 64 65 72 20  efender
004029C1  3C 46 75 6C 6C 20 4E 61  <Full Na
004029C9  6D 65 3E 20 3C 31 36 2D  me> <16-
004029D1  64 69 67 69 74 20 68 65  digit he
004029D9  78 61 64 65 63 69 6D 61  xadecima
004029E1  6C 20 6E 75 6D 62 65 72  l number
004029E9  3E 0A 00                 >..
```

Processing Username

- Blah

```
00401681    CMP CL,40
00401684    JNB SHORT Defender.0040169B
00401686    CMP CL,20
00401689    JNB SHORT Defender.00401691
0040168B    SHLD EDX,EAX,CL
0040168E    SHL EAX,CL
00401690    RETN
00401691    MOV EDX,EAX
00401693    XOR EAX,EAX
00401695    AND CL,1F
00401698    SHL EDX,CL
0040169A    RETN
0040169B    XOR EAX,EAX
0040169D    XOR EDX,EDX
0040169F    RETN
```

Processing Username

- Blah

```
00402B1C    ADD EAX,DWORD PTR SS:[EBP-190]
00402B22    MOV ECX,DWORD PTR SS:[EBP-18C]
00402B28    ADC ECX,EDX
00402B2A    MOV DWORD PTR SS:[EBP-190],EAX
00402B30    MOV DWORD PTR SS:[EBP-18C],ECX
```

User Info

- Formula used to validate user input

$$Sum = \sum_{n=0}^{len} C_n \times 2^{C_n \bmod 48}$$

User Info

- Blah

```
00401D1F    MOV EAX,DWORD PTR SS:[EBP+8]
00401D22    IMUL EAX,DWORD PTR DS:[406020]
00401D29    MOV DWORD PTR SS:[EBP-10],EAX
```

User Info

- Blah

```
00401D7B    MOV EAX,DWORD PTR SS:[EBP+10]
00401D7E    MOV ECX,DWORD PTR SS:[EBP-10]
00401D81    SUB ECX,EAX
00401D83    MOV EAX,DWORD PTR SS:[EBP-28]
00401D86    XOR ECX,DWORD PTR DS:[EAX]
```

User Info

- Blah

```
00401E32    PUSHFD
00401E33    AAS
00401E34    ADD BYTE PTR DS:[EDI],-22
00401E37    AND DH,BYTE PTR DS:[EAX+B84CCD0]
00401E3D    LODS BYTE PTR DS:[ESI]
00401E3E    INS DWORD PTR ES:[EDI],DX
```

Unlocking Code

Brute-Forcing

Brute-Forcing

- Blah

```
00401D49    MOV DWORD PTR SS:[EBP-4],1
00401D50    CMP DWORD PTR SS:[EBP-4],0
00401D54    JE SHORT Defender.00401DBF
```

Brute-Forcing

- Blah

```
00401D49    C745 FC 01000000          MOV DWORD PTR SS:[EBP-4],1
```

Brute-Forcing

- Blah

```
C7 45 FC 00 00 00 00
```

Brute-Forcing

- Blah

```
00401D2C    PUSH Defender.00401E32
00401D31    POP EAX
00401D32    MOV DWORD PTR SS:[EBP-14],EAX
00401D35    PUSH Defender.00401EB6
00401D3A    POP EAX
00401D3B    MOV DWORD PTR SS:[EBP-C],EAX
```

Brute-Forcing

- Blah

```
for (DWORD dwCurrentBlock = 0;
dwCurrentBlock <= dwBlockCount;
dwCurrentBlock++)
{
dwDecryptedData[dwCurrentBlock] = dwEncryptedData[dwCurrentBlock] ^
 dwCurrentKey;
 dwDecryptedData[dwCurrentBlock] ^= dwPrevBlock;
 dwPrevBlock = dwEncryptedData[dwCurrentBlock];
}
```

Brute-Forcing

- Blah

```
PBYTE pbCurrent = (PBYTE) memchr(dwDecryptedData, Sequence[0],
                                 sizeof(dwEncryptedData));
while (pbCurrent)
{
  if (memcmp(pbCurrent, Sequence, sizeof(Sequence)) == 0)
  {
     printf ("Found our sequence! Key is 0x%08x.\n", dwCurrentKey);
    _exit(1);
  }
  pbCurrent++;
  pbCurrent = (PBYTE) memchr(pbCurrent, Sequence[0],
   sizeof(dwEncryptedData) - (pbCurrent - (PBYTE) dwDecryptedData));
}
```

Brute-Forcing

- Blah

```
DWORD dwEncryptedData[] = {
0x5AA37BEB,    0xD7321D42,    0x2618DDF9,    0x2F1794E3,
0x1DE51172,    0x8BDBD150,    0xBB2954C1,    0x678CB4E3,
0x5DD701F9,    0xE11679A6,    0x501CD9A0,    0x685251B9,
0xD6F355EE,    0xE401D07F,    0x10C218A5,    0x22593307,
0x10133778,    0x22594B07,    0x1E134B78,    0xC5093727,
0xB016083D,    0x8A4C8DAC,    0x1BB759E3,    0x550A5611,
0x140D1DF4,    0xE8CE15C5,    0x47326D27,    0xF3F1AD7D,
0x42FB734C,    0xF34DF691,    0xAB07368B,    0xE5B2080F,
0xCDC6C492,    0x5BF8458B,    0x8B55C3C9 };

unsigned char Sequence[] = {0xC7, 0x45, 0xFC, 0x00, 0x00, 0x00, 0x00 };
```

Brute-Forcing

- Blah

```
Found our sequence! Key is 0xb14ac01a.
```

Brute-Forcing

- Blah

```
__int64  NameToInt64(LPWSTR pwszName)
{
  __int64 Result = 0;
  int iPosition = 0;
  while (*pwszName)
  {
    Result += (__int64) *pwszName << (__int64) (*pwszName % 48);
    pwszName++;
    iPosition++;
  }

  return Result;
}
```

Brute-Forcing

- Blah

```
 char name[256];
 char fsname[256];
 DWORD complength;
 DWORD VolumeSerialNumber;
 GetVolumeInformation("C:\\", name, sizeof(name), &VolumeSerialNumb
&complength, 0, fsname, sizeof(fsname));
 printf ("Volume serial number is: 0x%08x\n", VolumeSerialNumber);
 printf ("Computing serial for name: %s\n", argv[1]);
 WCHAR wszName[256];
 mbstowcs(wszName, argv[1], 256);
 unsigned __int64 Name = NameToInt64(wszName);
 ULONG FirstNum = (ULONG) Name * VolumeSerialNumber;
 unsigned __int64 Result = FirstNum - (ULONG) 0xb14ac01a;

 printf ("Name number is: %08x%08x\n",
(ULONG) (Name >> 32), (ULONG) Name);
 printf ("Name * VolumeSerialNumber is: %08x\n", FirstNum);
 printf ("Serial number is: %08x%08x\n",
(ULONG) (Result >> 32), (ULONG) Result);
```

Brute-Forcing

- Blah

```
Volume serial number is: 0x6c69e863
Computing serial for name: John Doe
Name number is: 000000212ccaf4a0
Name * VolumeSerialNumber is: 15cd99e0
Serial number is: 000000006482d9c6
```

Brute-Forcing

- Blah

```
unsigned char Sequence[] = {0xC7, 0x45, 0xFC, 0x00, 0x00, 0x00, 0x00 };
```

Brute-Forcing

- Blah

```
Found our sequence! Key is 0x8ed105c2.
```

Brute-Forcing

- Blah

```
unsigned __int64 Name = NameToInt64(wszName);
ULONG FirstNum = (ULONG) Name * VolumeSerialNumber;
unsigned __int64 Result = FirstNum - (ULONG) 0xb14ac01a;
Result |= (unsigned __int64) (FirstNum - 0x8ed105c2) << 32;

printf ("Name number is: %08x%08x\n",
      (ULONG) (Name >> 32), (ULONG) Name);
printf ("Name * VolumeSerialNumber is: %08x\n", FirstNum);
printf ("Serial number is: %08x%08x\n",
      (ULONG) (Result >> 32), (ULONG) Result);
```

Brute-Forcing

- Blah

```
Volume serial number is: 0x6c69e863
Computing serial for name: John Doe
Name number is: 000000212ccaf4a0
Name * VolumeSerialNumber is: 15cd99e0
Serial number is: 86fc941e6482d9c6
```

Brute-Forcing

- Blah

```
Defender Version 1.0 - Written by Eldad Eilam
That is correct! Way to go!
```

Brute-Forcing

Cracking Defender: Summary

Protections in Defender

Localized Encryption

Obfuscation

Time-Stamp Thread

Decryption Keys

Inlining

Conclusions

Assignment

- Rip keygen code from "keygen.exe"
  - http://www.cs.sjsu.edu/~stamp/CS286/progs/keygen.exe.zip
- Make a separate app that generates valid serial number for given ID/username
- Test on each of following ID/usernames
  - aaaaa
  - qwert
  - qwerty