Mario Vuksan, Tomislav Peričin & Vojislav Milunovic,
ReversingLabs Corporation

FAST & FURIOUS REVERSE ENGINEERING WITH TITANENGINE

Agenda

- Obligatory Scare Talk
- Why should you care?
- What is the problem?
- How can TitanEngine change the world?
- Show ME!
- Show ME!
- Show ME!
- How can I help?

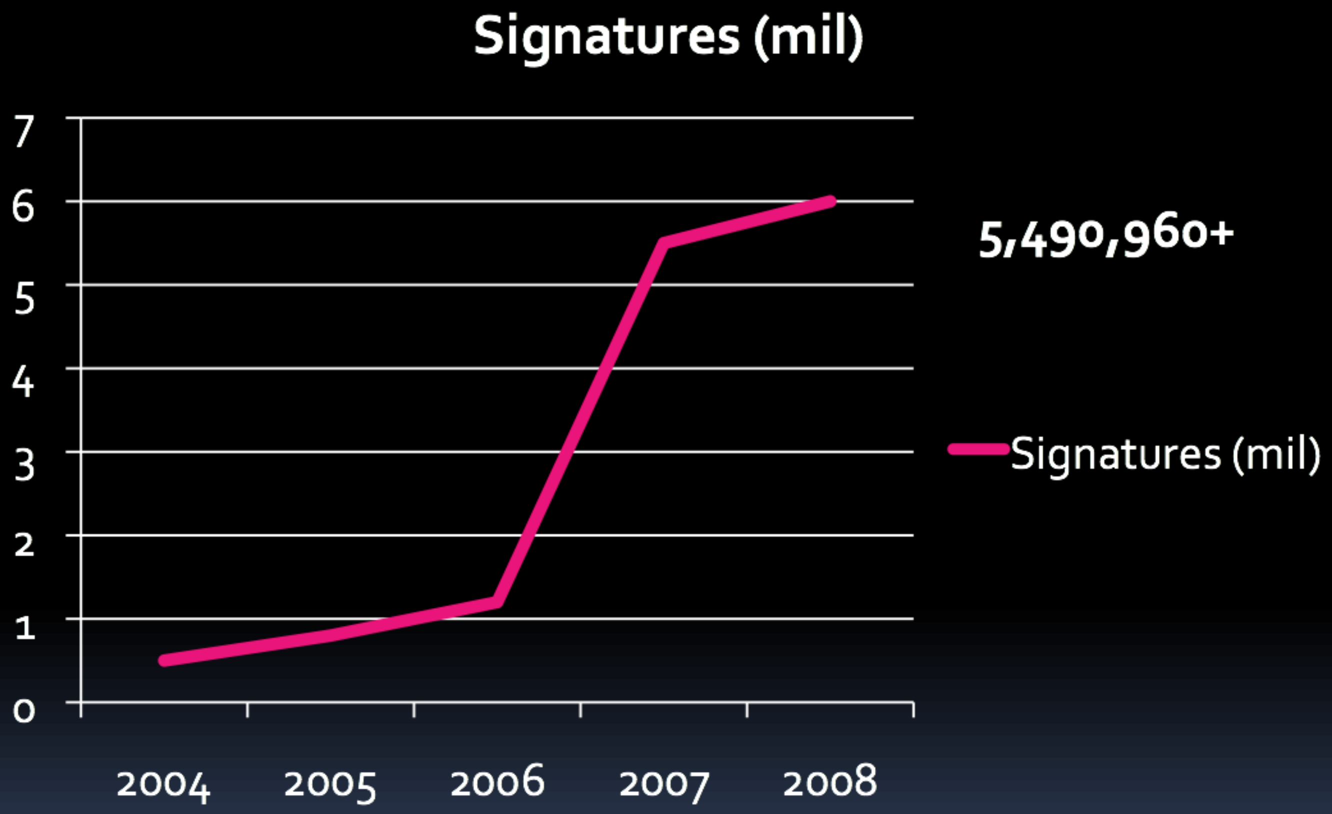
Fighting Malware: Old Problem

Inadequate Infrastructure: New Problem

Exponential Growth in Malware

YIELDS

Exponential **Growth** in Signatures



5,490,960+

DEMANDING

Malware Wars

Army of Threat Researchers

RESULTING IN

Denial of Service on Threat Response Teams

So What?

Security Industry is a For-Profit Entity

We'll Simply Hire More Bodies

But Could We Get Enough Bodies?

Can’t Hire Enough? Combine those we have into one Worldwide Non-profit Entity

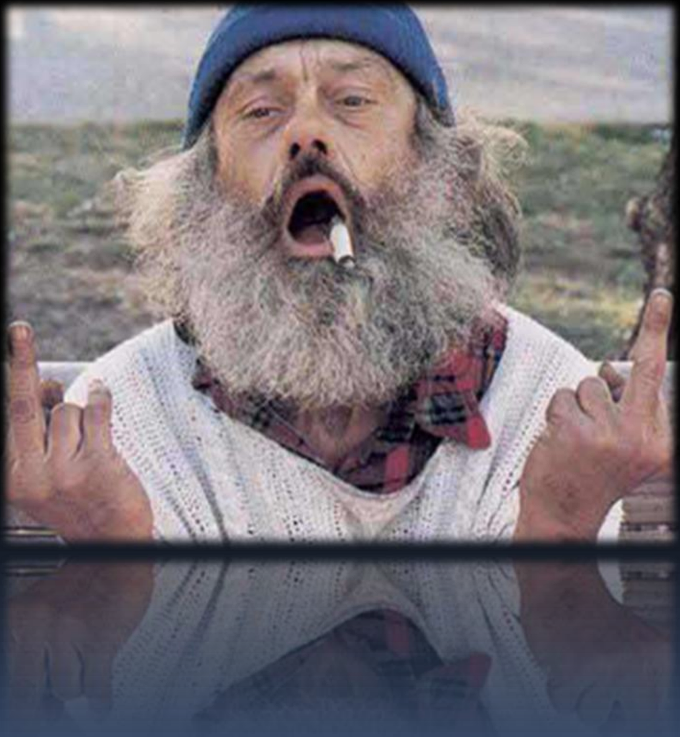
(Bwa-ha-ha!)

OR…

We could

simply *overload* them…

Is an overloaded anti-malware analyst an asset or a liability?



Henry Ford

- Anti-Malware labs are factories
- 100-200+ Analyst teams
- Advanced workflows
- Multiple levels of management
- Modern labor laws apply: No 20+ hour days
- Productivity can be improved
- Work process can be studied
- Improvements ***COULD*** be devised...

So how can Labs do more?

- Charge more, Hire more
- Invest in automation, Invest in heuristics
- Deploy proactive modules, Buy competitors
- All the usual stuff
- … and they could revise their processes

So how can Labs do more?

- 1,000s of OllyDBG and IDAPro scripts can better be reused; could be generalized
- Sample analysis, OEP discovery could benefit all team members
- Reversing should be a team effort

We have to do it better…

Competition is tough

- Bad guys
  - Rise of $$ motivated custom attacks
  - Resourceful crime syndicates

Protection is lacking

- Signatures only “important” for threats
- Need for other types of protection
- Behavioral & HIPS tools ***that work***

Yet manual analysis is still the only certain bet!

Passion for binary protection

- Meatiest task today is dealing with protection techniques
- Task repetition, Error prone, Not reusable
- Large number of file formats can be infected and used for malware

Passion for binary protection

- Executable files == most significant threat
- Executables == the “usual suspect” for malware
  - 85% of malware samples are packed
  - Packing hides malware, hardens its detection
- Packed or protected doesn’t mean bad!
  - 10% of legitimate software is packed

Passion for binary protection

- Legit use for packers & protectors:
  - Compressed binaries decrease bandwidth usage
  - Protect intellectual property
  - Protect from code theft
  - Anti-tampering in multi-player games
  - Safeguard licensing code
- Successfully used by malware authors
  - For all the same reasons

Analyzing Malware

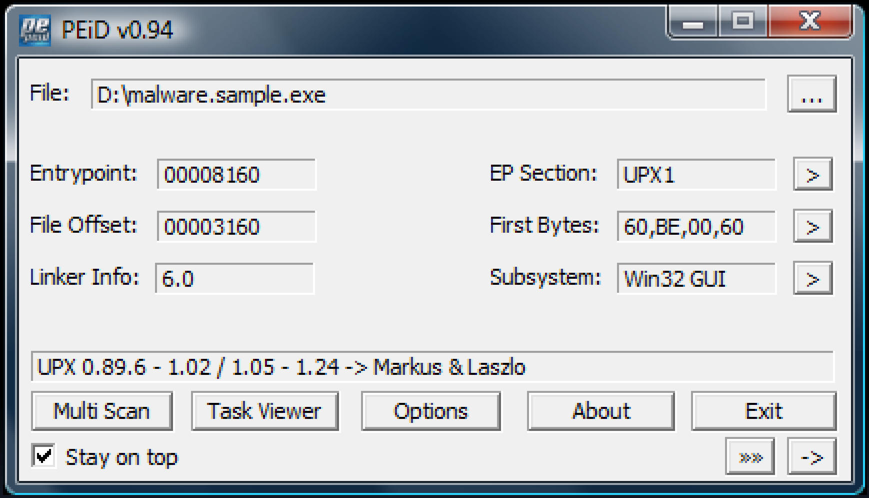
- Malware File Analysis Requires:
  - In-depth knowledge of how PE works
  - In-depth knowledge of how Windows works
  - Various tools to make you reach your goal
- Understanding of Basic Shell Divisions:
  - Packers, Protectors, Crypters, Bundlers & Hybrids
  - Custom malware-specific packers & protectors

What's the Reversing Process Today?

```
/*408160*/  PUSHAD
/*408161*/  MOV ESI,crackme_.00406000
/*408166*/  LEA EDI,DWORD PTR DS:[ESI+FFFFB000]
/*40816C*/  PUSH EDI
/*40816D*/  OR EBP,FFFFFFFF
/*408170*/  JMP SHORT crackme_.00408182
/*408172*/  NOP
/*408173*/  NOP
/*408174*/  NOP
/*408175*/  NOP
/*408176*/  NOP
/*408177*/  NOP
/*408178*/  MOV AL,BYTE PTR DS:[ESI]
/*40817A*/  INC ESI
/*40817B*/  MOV BYTE PTR DS:[EDI],AL
/*40817D*/  INC EDI
/*40817E*/  ADD EBX,EBX
/*408180*/  JNZ SHORT crackme_.00408189
/*408182*/  MOV EBX,DWORD PTR DS:[ESI]
/*408184*/  SUB ESI,-4
/*408187*/  ADC EBX,EBX
```

Reversing in action|Today

- Inspect the Sample
  - Identify the packing shell or compiler



File: D:\malware.sample.exe

Entrypoint: 00008160

File Offset: 00003160

Linker Info: 6.0

EP Section: UPX1

First Bytes: 60,BE,00,60

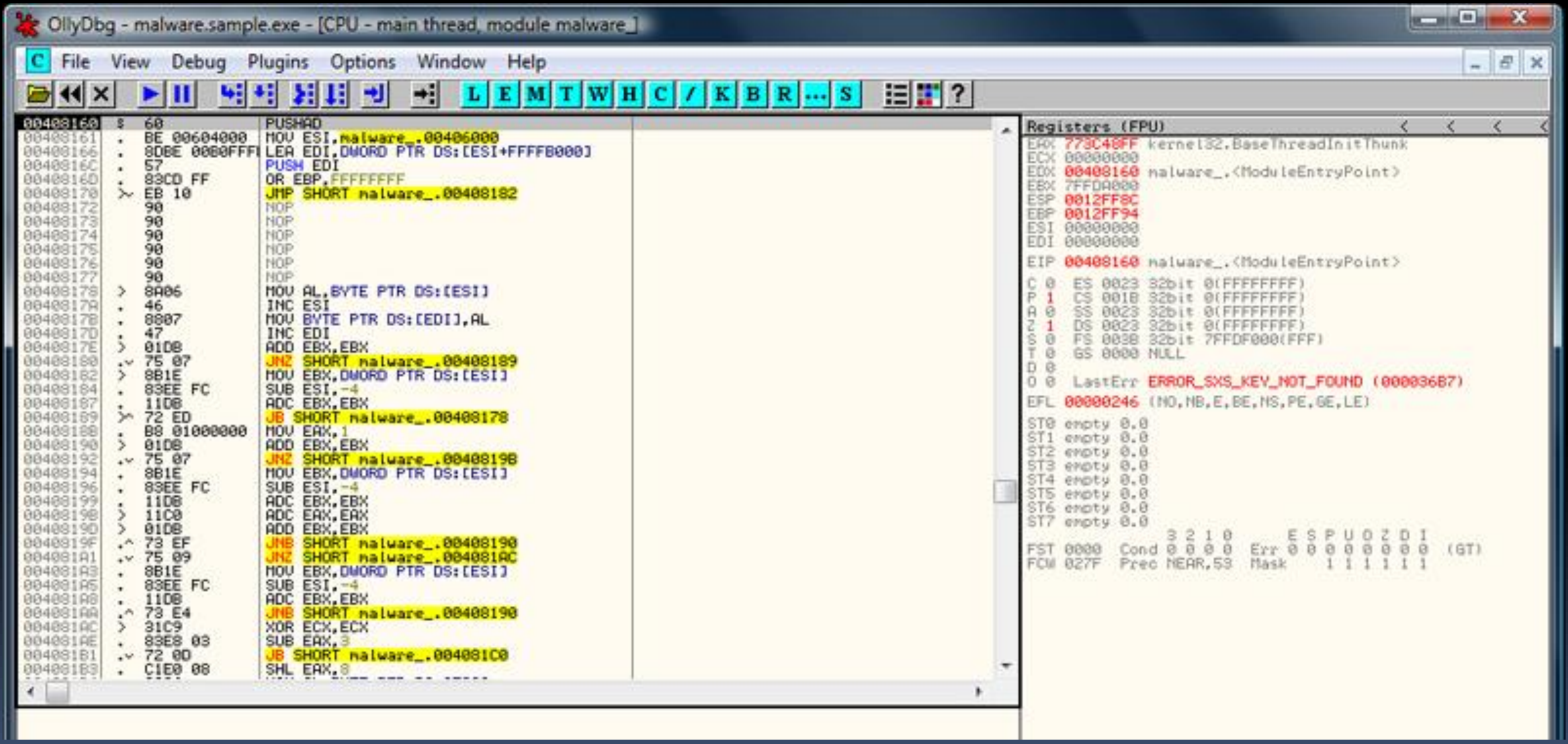
Subsystem: Win32 GUI

UPX 0.89.6 - 1.02 / 1.05 - 1.24 -> Markus & Laszlo

PEiD

Reversing in action|Today

- Unpack the Sample
    - Execute it to the original entry point



OllyDbg - malware.sample.exe - [CPU - main thread, module malware_]
C
File
View
Debug
Plugins
Options
Window
Help
L
E
M
T
W
H
C
/
K
B
R
...
S
?

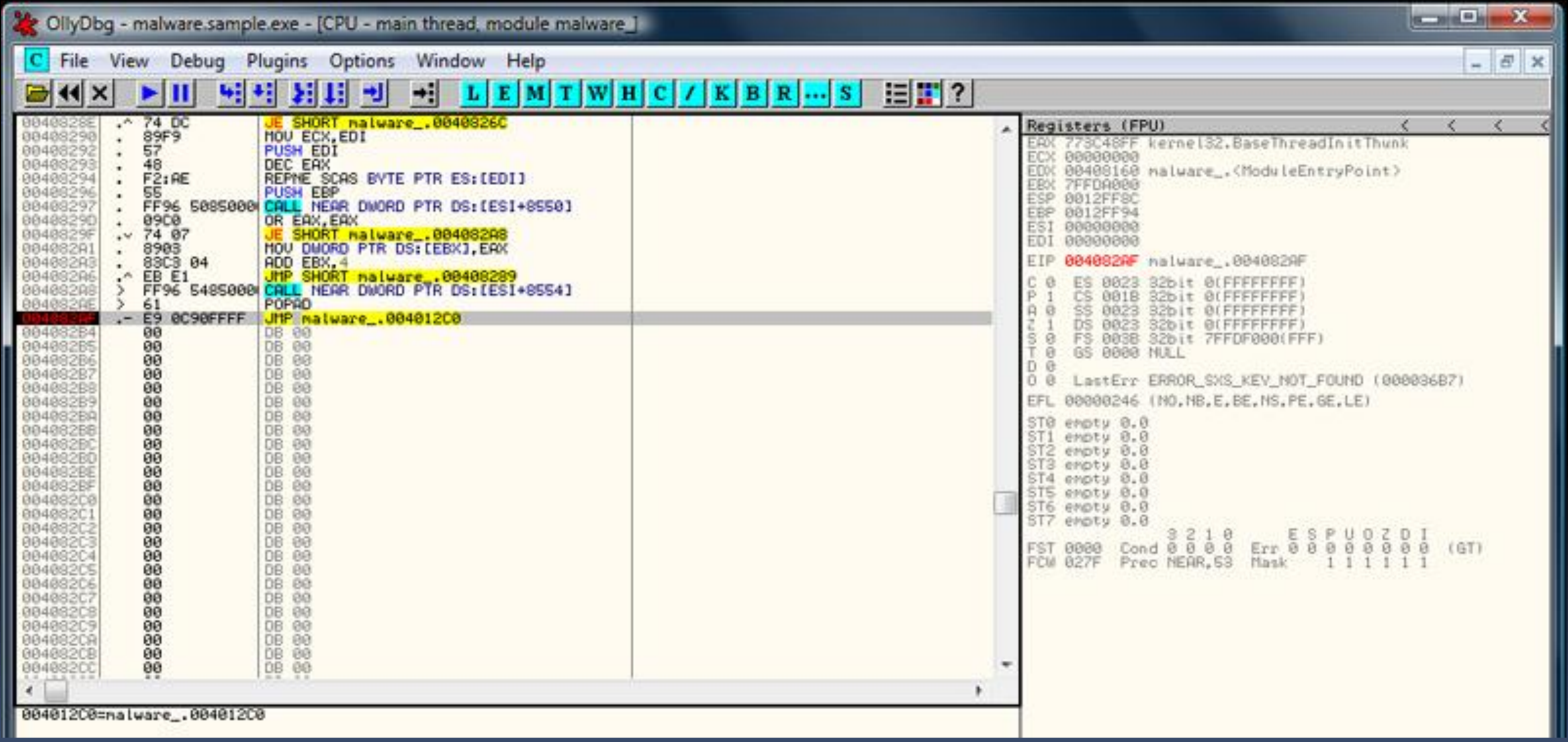
```
00408160  $   60              PUSHAD
00408161  .   BE 00604000     MOV ESI,malware_.00406000
00408166  .   8DBE 0080FFFF   LEA EDI,DWORD PTR DS:[ESI+FFFFB000]
0040816C  .   57              PUSH EDI
0040816D  .   83CD FF         OR EBP,FFFFFFFF
00408170  >-  EB 10           JMP SHORT malware_.00408182
00408172      90              NOP
00408173      90              NOP
00408174      90              NOP
00408175      90              NOP
00408176      90              NOP
00408177      90              NOP
00408178  >   8A06            MOV AL,BYTE PTR DS:[ESI]
0040817A  .   46              INC ESI
0040817B  .   8807            MOV BYTE PTR DS:[EDI],AL
0040817D  .   47              INC EDI
0040817E  >   01DB            ADD EBX,EBX
00408180  .v  75 07           JNZ SHORT malware_.00408189
00408182  >   8B1E            MOV EBX,DWORD PTR DS:[ESI]
00408184  .   83EE FC         SUB ESI,-4
00408187  .   11DB            ADC EBX,EBX
00408189  >^  72 ED           JB SHORT malware_.00408178
0040818B  .   B8 01000000     MOV EAX,1
00408190  >   01DB            ADD EBX,EBX
00408192  .v  75 07           JNZ SHORT malware_.0040819B
00408194  .   8B1E            MOV EBX,DWORD PTR DS:[ESI]
00408196  .   83EE FC         SUB ESI,-4
00408199  .   11DB            ADC EBX,EBX
0040819B  >   11C0            ADC EAX,EAX
0040819D  >   01DB            ADD EBX,EBX
0040819F  .^  73 EF           JNB SHORT malware_.00408190
004081A1  .v  75 09           JNZ SHORT malware_.004081AC
004081A3  .   8B1E            MOV EBX,DWORD PTR DS:[ESI]
004081A5  .   83EE FC         SUB ESI,-4
004081A8  .   11DB            ADC EBX,EBX
004081AA  .^  73 E4           JNB SHORT malware_.00408190
004081AC  >   31C9            XOR ECX,ECX
004081AE  .   83E8 03         SUB EAX,3
004081B1  .v  72 0D           JB SHORT malware_.004081C0
004081B3  .   C1E0 08         SHL EAX,8
```

```
Registers (FPU)
EAX 773C48FF kernel32.BaseThreadInitThunk
ECX 00000000
EDX 00408160 malware_.<ModuleEntryPoint>
EBX 7FFDA000
ESP 0012FF8C
EBP 0012FF94
ESI 00000000
EDI 00000000

EIP 00408160 malware_.<ModuleEntryPoint>

C 0  ES 0023 32bit 0(FFFFFFFF)
P 1  CS 001B 32bit 0(FFFFFFFF)
A 0  SS 0023 32bit 0(FFFFFFFF)
Z 1  DS 0023 32bit 0(FFFFFFFF)
S 0  FS 003B 32bit 7FFDF000(FFF)
T 0  GS 0000 NULL
D 0
O 0  LastErr ERROR_SXS_KEY_NOT_FOUND (000036B7)

EFL 00000246 (NO,NB,E,BE,NS,PE,GE,LE)

ST0 empty 0.0
ST1 empty 0.0
ST2 empty 0.0
ST3 empty 0.0
ST4 empty 0.0
ST5 empty 0.0
ST6 empty 0.0
ST7 empty 0.0
               3 2 1 0      E S P U O Z D I
FST 0000  Cond 0 0 0 0  Err 0 0 0 0 0 0 0 0  (GT)
FCW 027F  Prec NEAR,53  Mask    1 1 1 1 1 1
```

OllyDbg

Reversing in action|Today

- Unpack the Sample
  - Execute it to the original entry point



```
0040828E  .^ 74 DC          JE SHORT malware_.0040826C
00408290  .  89F9           MOV ECX,EDI
00408292  .  57             PUSH EDI
00408293  .  48             DEC EAX
00408294  .  F2:AE          REPNE SCAS BYTE PTR ES:[EDI]
00408296  .  55             PUSH EBP
00408297  .  FF96 5085000   CALL NEAR DWORD PTR DS:[ESI+8550]
0040829D  .  09C0           OR EAX,EAX
0040829F  .v 74 07          JE SHORT malware_.004082A8
004082A1  .  8903           MOV DWORD PTR DS:[EBX],EAX
004082A3  .  83C3 04        ADD EBX,4
004082A6  .^ EB E1          JMP SHORT malware_.00408289
004082A8  >  FF96 5485000   CALL NEAR DWORD PTR DS:[ESI+8554]
004082AE  >  61             POPAD
004082AF  .- E9 0C90FFFF    JMP malware_.004012C0
004082B4     00             DB 00
004082B5     00             DB 00
004082B6     00             DB 00
004082B7     00             DB 00
004082B8     00             DB 00
004082B9     00             DB 00
004082BA     00             DB 00
004082BB     00             DB 00
004082BC     00             DB 00
004082BD     00             DB 00
004082BE     00             DB 00
004082BF     00             DB 00
004082C0     00             DB 00
004082C1     00             DB 00
004082C2     00             DB 00
004082C3     00             DB 00
004082C4     00             DB 00
004082C5     00             DB 00
004082C6     00             DB 00
004082C7     00             DB 00
004082C8     00             DB 00
004082C9     00             DB 00
004082CA     00             DB 00
004082CB     00             DB 00
004082CC     00             DB 00
```

```
EAX 773C48FF kernel32.BaseThreadInitThunk
ECX 00000000
EDX 00408160 malware_.<ModuleEntryPoint>
EBX 7FFDA000
ESP 0012FF8C
EBP 0012FF94
ESI 00000000
EDI 00000000

EIP 004082AF malware_.004082AF

C 0  ES 0023 32bit 0(FFFFFFFF)
P 1  CS 001B 32bit 0(FFFFFFFF)
A 0  SS 0023 32bit 0(FFFFFFFF)
Z 1  DS 0023 32bit 0(FFFFFFFF)
S 0  FS 003B 32bit 7FFDF000(FFF)
T 0  GS 0000 NULL
D 0
O 0  LastErr ERROR_SXS_KEY_NOT_FOUND (000036B7)

EFL 00000246 (NO,NB,E,BE,NS,PE,GE,LE)

ST0 empty 0.0
ST1 empty 0.0
ST2 empty 0.0
ST3 empty 0.0
ST4 empty 0.0
ST5 empty 0.0
ST6 empty 0.0
ST7 empty 0.0
               3 2 1 0      E S P U O Z D I
FST 0000  Cond 0 0 0 0  Err 0 0 0 0 0 0 0 0  (GT)
FCW 027F  Prec NEAR,53  Mask    1 1 1 1 1 1
```

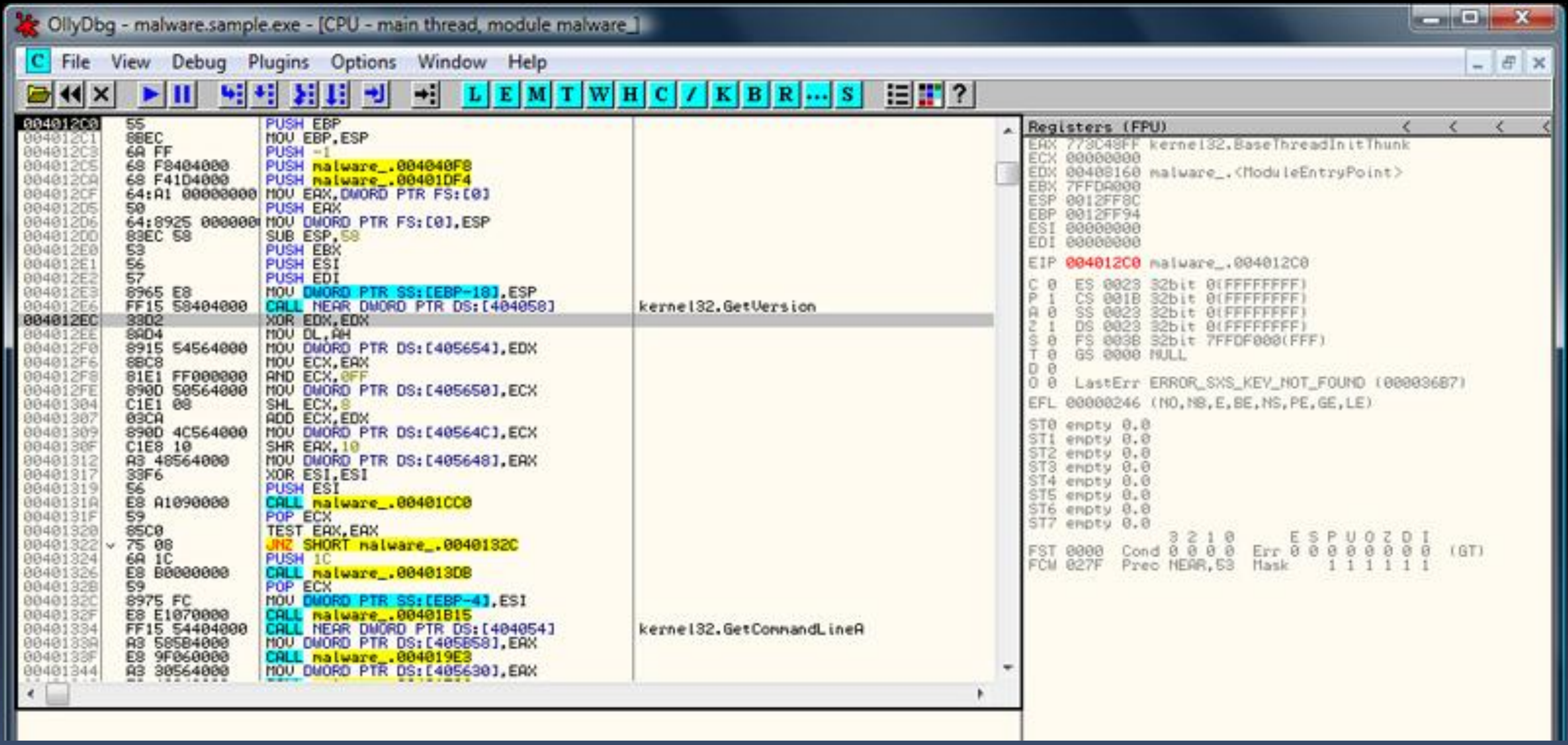
```
004012C0=malware_.004012C0
```

OllyDbg

ReversingLabs Corporation

Reversing in action|Today

- Unpack the Sample
  - Execute it to the original entry point


OllyDbg - malware.sample.exe - [CPU - main thread, module malware_]
C File View Debug Plugins Options Window Help
L E M T W H C / K B R ... S
Registers (FPU)

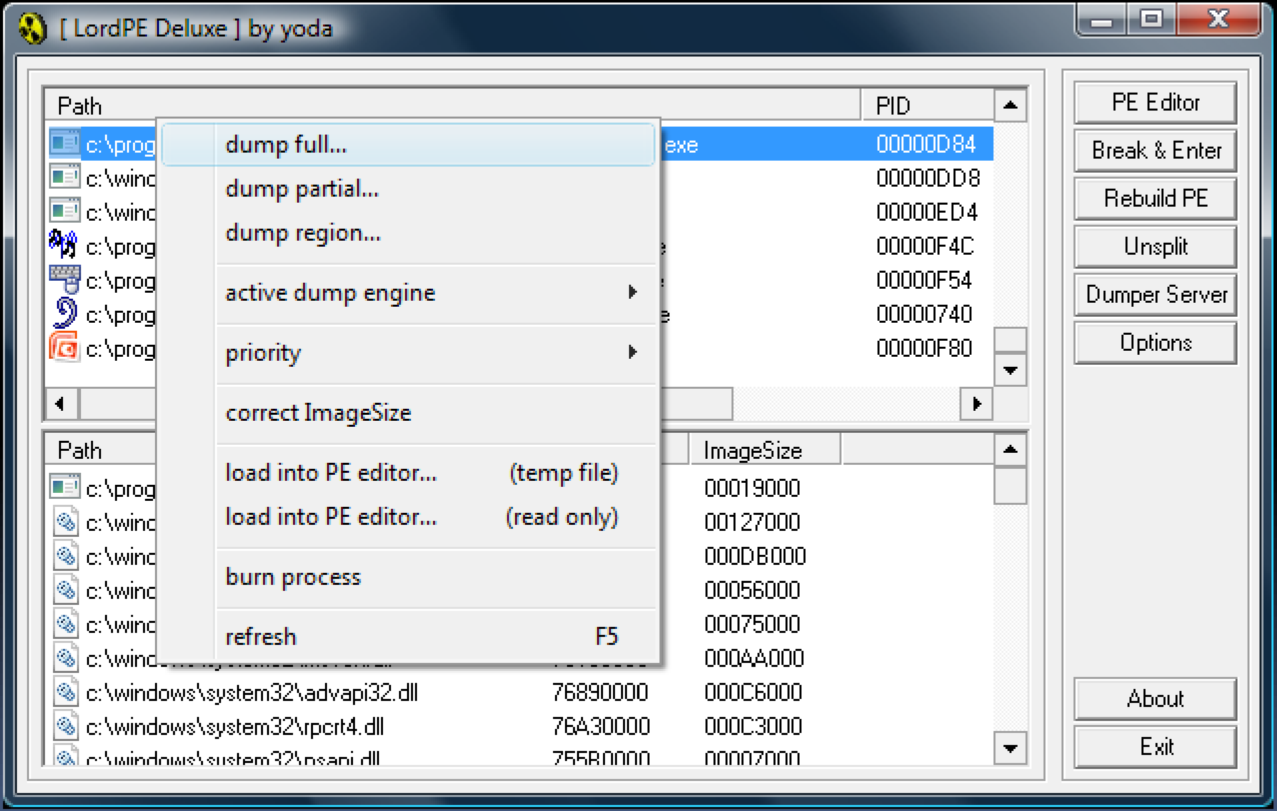
```
004012C0    55              PUSH EBP
004012C1    8BEC            MOV EBP,ESP
004012C3    6A FF           PUSH -1
004012C5    68 F8404000     PUSH malware_.004040F8
004012CA    68 F41D4000     PUSH malware_.00401DF4
004012CF    64:A1 00000000  MOV EAX,DWORD PTR FS:[0]
004012D5    50              PUSH EAX
004012D6    64:8925 000000  MOV DWORD PTR FS:[0],ESP
004012DD    83EC 58         SUB ESP,58
004012E0    53              PUSH EBX
004012E1    56              PUSH ESI
004012E2    57              PUSH EDI
004012E3    8965 E8         MOV DWORD PTR SS:[EBP-18],ESP
004012E6    FF15 58404000   CALL NEAR DWORD PTR DS:[404058]          kernel32.GetVersion
004012EC    33D2            XOR EDX,EDX
004012EE    8AD4            MOV DL,AH
004012F0    8915 54564000   MOV DWORD PTR DS:[405654],EDX
004012F6    8BC8            MOV ECX,EAX
004012F8    81E1 FF000000   AND ECX,0FF
004012FE    890D 50564000   MOV DWORD PTR DS:[405650],ECX
00401304    C1E1 08         SHL ECX,8
00401307    03CA            ADD ECX,EDX
00401309    890D 4C564000   MOV DWORD PTR DS:[40564C],ECX
0040130F    C1E8 10         SHR EAX,10
00401312    A3 48564000     MOV DWORD PTR DS:[405648],EAX
00401317    33F6            XOR ESI,ESI
00401319    56              PUSH ESI
0040131A    E8 A1090000     CALL malware_.00401CC0
0040131F    59              POP ECX
00401320    85C0            TEST EAX,EAX
00401322  v 75 08           JNZ SHORT malware_.0040132C
00401324    6A 1C           PUSH 1C
00401326    E8 B0000000     CALL malware_.004013DB
0040132B    59              POP ECX
0040132C    8975 FC         MOV DWORD PTR SS:[EBP-4],ESI
0040132F    E8 E1070000     CALL malware_.00401B15
00401334    FF15 54404000   CALL NEAR DWORD PTR DS:[404054]          kernel32.GetCommandLineA
0040133A    A3 58584000     MOV DWORD PTR DS:[405858],EAX
0040133F    E8 9F060000     CALL malware_.004019E3
00401344    A3 30564000     MOV DWORD PTR DS:[405630],EAX
```

```
EAX 773C48FF kernel32.BaseThreadInitThunk
ECX 00000000
EDX 00408160 malware_.<ModuleEntryPoint>
EBX 7FFDA000
ESP 0012FF8C
EBP 0012FF94
ESI 00000000
EDI 00000000

EIP 004012C0 malware_.004012C0

C 0  ES 0023 32bit 0(FFFFFFFF)
P 1  CS 001B 32bit 0(FFFFFFFF)
A 0  SS 0023 32bit 0(FFFFFFFF)
Z 1  DS 0023 32bit 0(FFFFFFFF)
S 0  FS 003B 32bit 7FFDF000(FFF)
T 0  GS 0000 NULL
D 0
O 0  LastErr ERROR_SXS_KEY_NOT_FOUND (00003687)

EFL 00000246 (NO,NB,E,BE,NS,PE,GE,LE)

ST0 empty 0.0
ST1 empty 0.0
ST2 empty 0.0
ST3 empty 0.0
ST4 empty 0.0
ST5 empty 0.0
ST6 empty 0.0
ST7 empty 0.0
               3 2 1 0      E S P U O Z D I
FST 0000  Cond 0 0 0 0  Err 0 0 0 0 0 0 0 0  (GT)
FCW 027F  Prec NEAR,53  Mask    1 1 1 1 1 1
```

OllyDbg

Reversing in action|Today

- Unpack the Sample
  - Dump the process memory

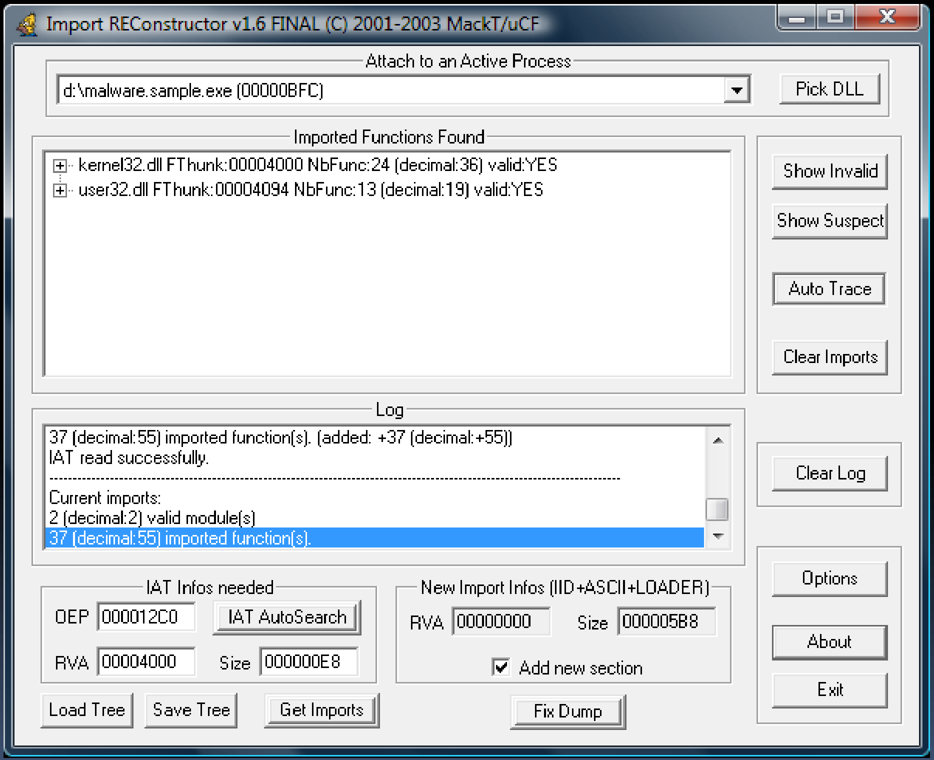


```
Path                                        PID
c:\prog                       exe           00000D84
c:\winc                                     00000DD8
c:\winc                                     00000ED4
c:\prog                                     00000F4C
c:\prog                                     00000F54
c:\prog                                     00000740
c:\prog                                     00000F80

Path                                        ImageSize
c:\prog                                     00019000
c:\winc                                     00127000
c:\winc                                     000DB000
c:\winc                                     00056000
c:\winc                                     00075000
c:\winc                                     000AA000
c:\windows\system32\advapi32.dll  76890000  000C6000
c:\windows\system32\rpcrt4.dll    76A30000  000C3000
c:\windows\system32\psapi.dll     755B0000  00007000
```

LordPE

Reversing in action|Today

- Unpack the Sample
  - Fix the import table



Import REConstructor v1.6 FINAL (C) 2001-2003 MackT/uCF

Attach to an Active Process: d:\malware.sample.exe (00000BFC)

Imported Functions Found

```
kernel32.dll FThunk:00004000 NbFunc:24 (decimal:36) valid:YES
user32.dll FThunk:00004094 NbFunc:13 (decimal:19) valid:YES
```

Log

```
37 (decimal:55) imported function(s). (added: +37 (decimal:+55))
IAT read successfully.
------------------------------------------------------------------------------------------
Current imports:
2 (decimal:2) valid module(s)
37 (decimal:55) imported function(s).
```

IAT Infos needed
OEP 000012C0
RVA 00004000 Size 000000E8

New Import Infos (IID+ASCII+LOADER)
RVA 00000000 Size 000005B8
☑ Add new section

ImpRec

ReversingLabs Corporation

Problems with File analysis

- File analysis takes time
  - Identifying requires keeping up with shells
  - Shells evolve & have different forms
- Analysts get more samples then they can handle
- File unpacking takes even more time
  - Protection “tricks” continue to evolve
  - Yet, this process can be **automated!**

TitanEngine

ReversingLabs Corporation

Fast Reversing|Tomorrow

- TitanEngine key features:
  - Framework designed to work with PE files
  - SDK has 250 documented functions
  - Easy automation of all reversing tools
  - Supports both x86 and x64
  - Can create:
    - Static, Dynamic & Generic unpackers
    - New file analysis tools
  - Tested on over 150 unpackers
  - Its **free** and **open source – LGPL3**!

Furious Reversing|Tomorrow

- Engine simulates reverse engineer's presence
  - Unpacking process has the same steps:
    - Debugs until entry point
    - Dumps memory to disk
    - Collects data for import fixing
    - Collects data for relocation fixing
    - Custom fixes (Code splices, Entry point, …)

TitanEngine|Content

- SDK Contains:
  - Integrated x86/x64 debugger
  - Integrated x86/x64 disassembler
  - Integrated memory dumper
  - Integrated import tracer & fixer
  - Integrated relocation fixer
  - Integrated file realigner
  - TLS, Resources, Exports…

TitanEngine|Debugger

- Integrated x86/x64 Debugger
  - Attach / Detach
  - Trace, including single stepping
  - Set several types of breakpoints:
    - Software (INT3)
    - Hardware
    - Memory
    - Flexible
    - API
  - Access debugged file's context

TitanEngine|Debugger

- Integrated x86/x64 Debugger
  - Disassembly instructions
    - Disassemble a length
    - Full disassemble
  - Memory manipulation
    - Find, Replace, Patch, Fill…
  - Get call/jump destination
  - Check if the jump will execute or not
  - Thread module for thread manipulation
  - Librarian module for module manipulation

TitanEngine|Dumper

- Integrated Memory Dumper
  - Dump memory
    - Process, regions or modules
  - Paste PE header from disk to memory
  - Manipulate file sections
    - Extract, resort, add, delete & resize
  - Manipulate file overlay
    - Find, extract, add, copy & remove

TitanEngine|Dumper

- Integrated Memory Dumper
  - Convert addresses
    - From relative to physical, and vice-versa
    - Get section number from address
  - PE header data
    - Get and set PE header values

TitanEngine|Importer

- Integrated Import Fixer
  - Build new import tables on the fly
  - Get API information
    - API address in both your & debugged process
    - DLL to hold API from API address
    - Remote & local DLL loaded base
    - API name from address
    - API Forwarders

TitanEngine|Importer

- Integrated Import Fixer
  - Automatic import table functions:
    - Locate import table in the memory
    - Fix the import table automatically
    - Fix import eliminations, automatically
  - Enumerate and handle import table data
  - Move import table from one file to another
  - Load import table from any PE file

TitanEngine|Tracer

- Integrated Import Tracer
  - Identify import redirections and eliminations
    - Fix known import protections
  - Use integrated tracers to resolve imports
    - Static disassembly tracer
    - Static hasher disassembly tracer
  - Use ImpRec modules to fix redirections

TitanEngine|Relocater

- Integrated Relocation Fixer
  - Build new relocation table on the fly
  - Resolve relocation table
    - Grab relocation table directly from the process
    - Make & compare memory snapshots
  - Remove relocation table from the file
  - Relocate file to new image base

TitanEngine|Realigner

- Integrated File Realigner
  - Validate PE files
  - Fix broken PE files
  - Realign files: reduce size & validate
  - Fix header checksum
  - Wipe sections

TitanEngine|The Rest…

- TLS
  - Remove callbacks
  - Break at callbacks
- Exporter
  - Build export tables on the fly
- Handler
  - Close remote handles
  - Get file lock handles
  - Find open mutexes

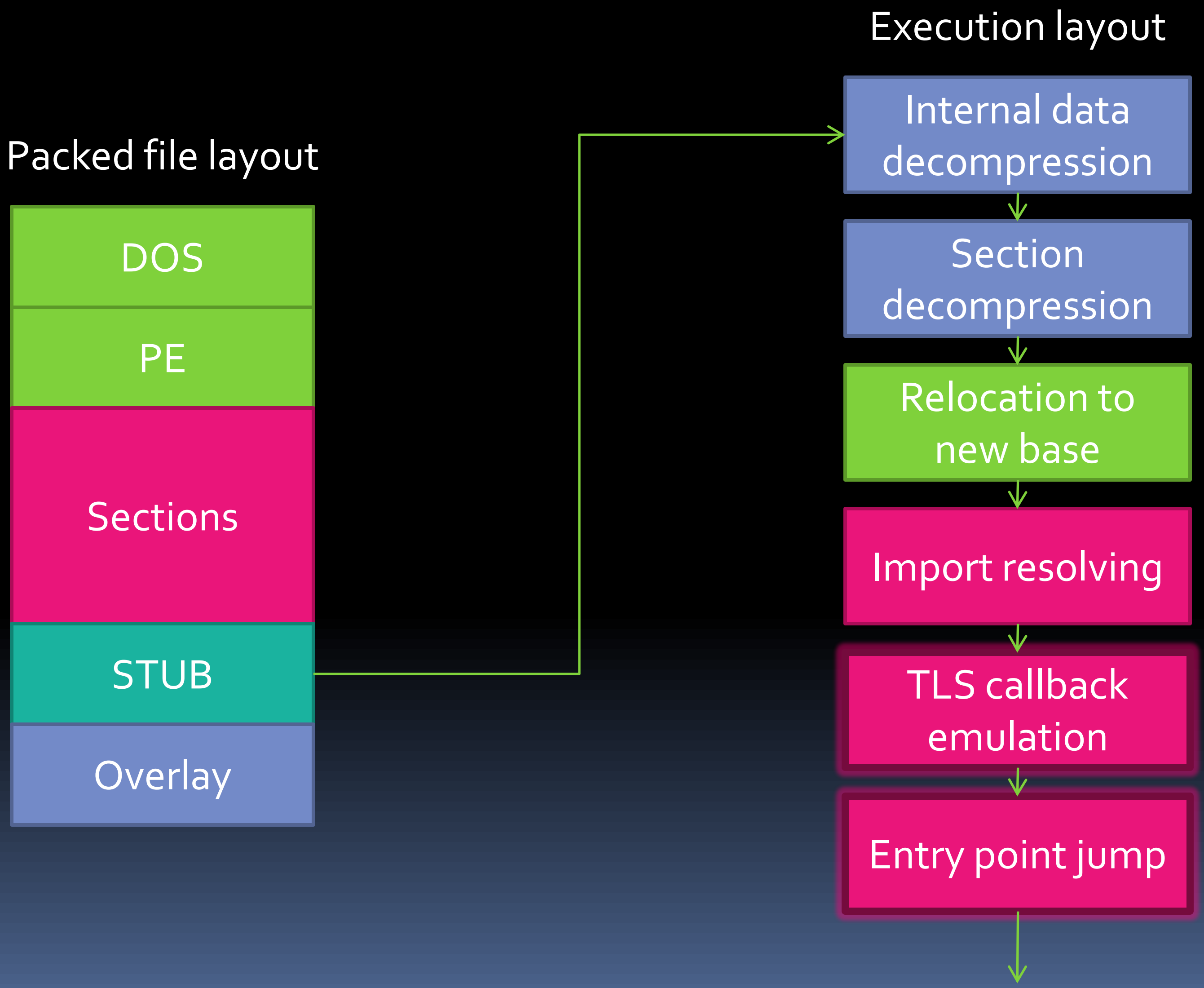
TitanEngine|The Rest…

- Resource
  - Extract resource
- Remote
  - Load & Free libraries into running process
- OEP Finder
  - Get OEP location generically
- Static
  - Unpack files statically

Back to Basics:
Shell Modifier Types

- Shell Division
  - Crypters
  - Packers
  - Protectors
  - Bundlers
    - Data bundlers
    - Overlay/Resource bundlers
  - Hybrids

Packed File Layout



Unpacker Types…

- Basic Unpacker Division
  - Static unpackers:
    - **Pro:** simple, fast & supported by TitanEngine
    - **Con:** don't work if internal shell mechanisms change
  - Dynamic unpackers:
    - **Pro:** "simple", fast & supported by TitanEngine
    - **Con:** carry a certain risk of file execution!
  - Generic unpackers:
    - **Pro:** Can support large number of similar shells
    - **Con:** Can be highly inaccurate!

Writing an Unpacker…

- Analyze the Packing Shell
    - Step 1
        - Determine protection types
            - Design ways to avoid them
            - Determine method to resolve custom protections
            - Determine method to skip entry point layer protection
            - Determine if we can automate file identification

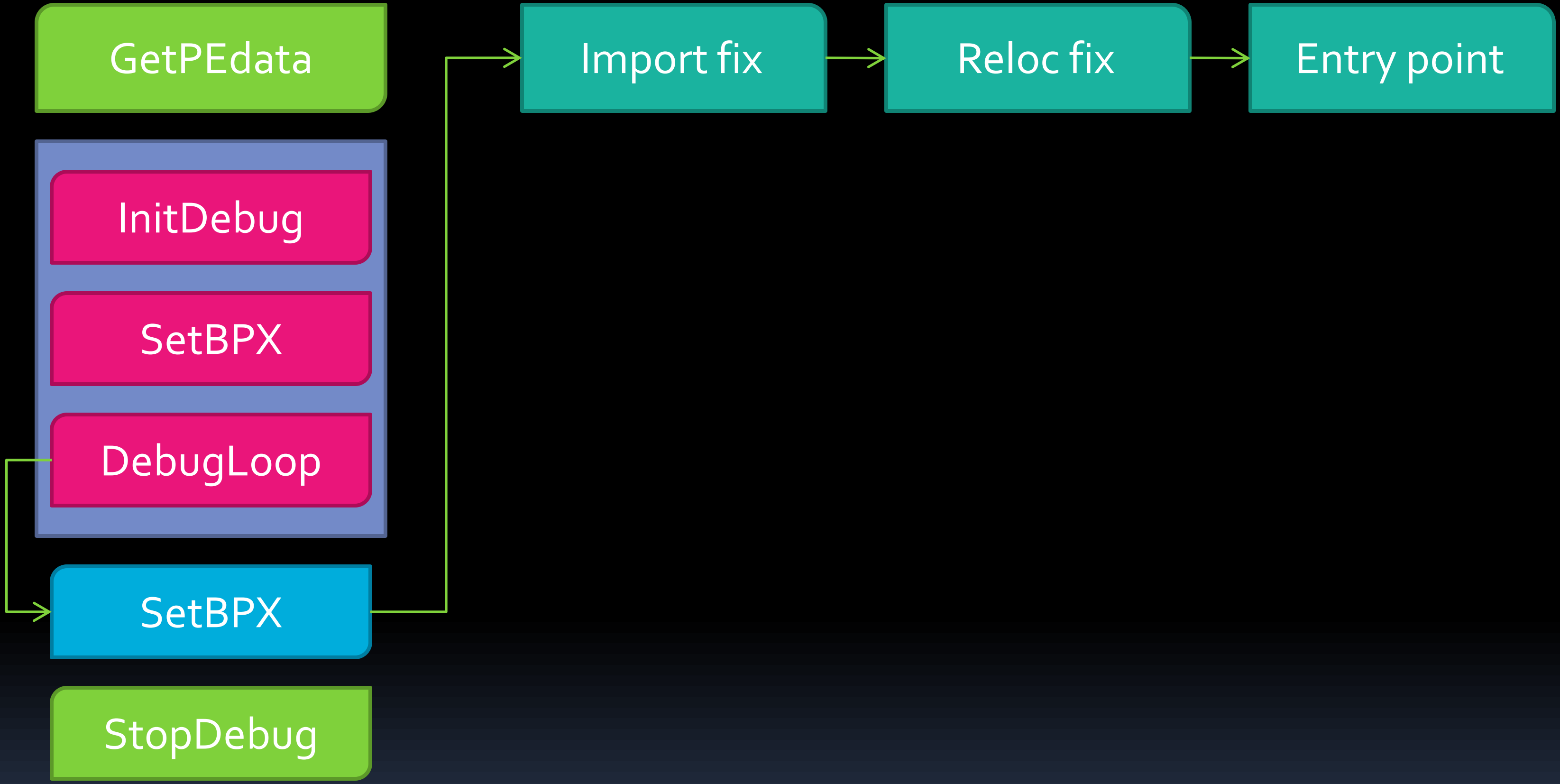
Writing an Unpacker…

- Analyze the Packing Shell
  - Step 2
    - Locate packing shell's important parts
      - Where does it fill import table?
      - Where does it relocate the file?
      - How does it jump to OEP?
    - Identify byte patterns, using ***lots*** of samples!
      - Proper patterns contain wild cards
      - Proper patterns work on all samples
      - Proper patterns are based on ***multiple*** compiler cases!

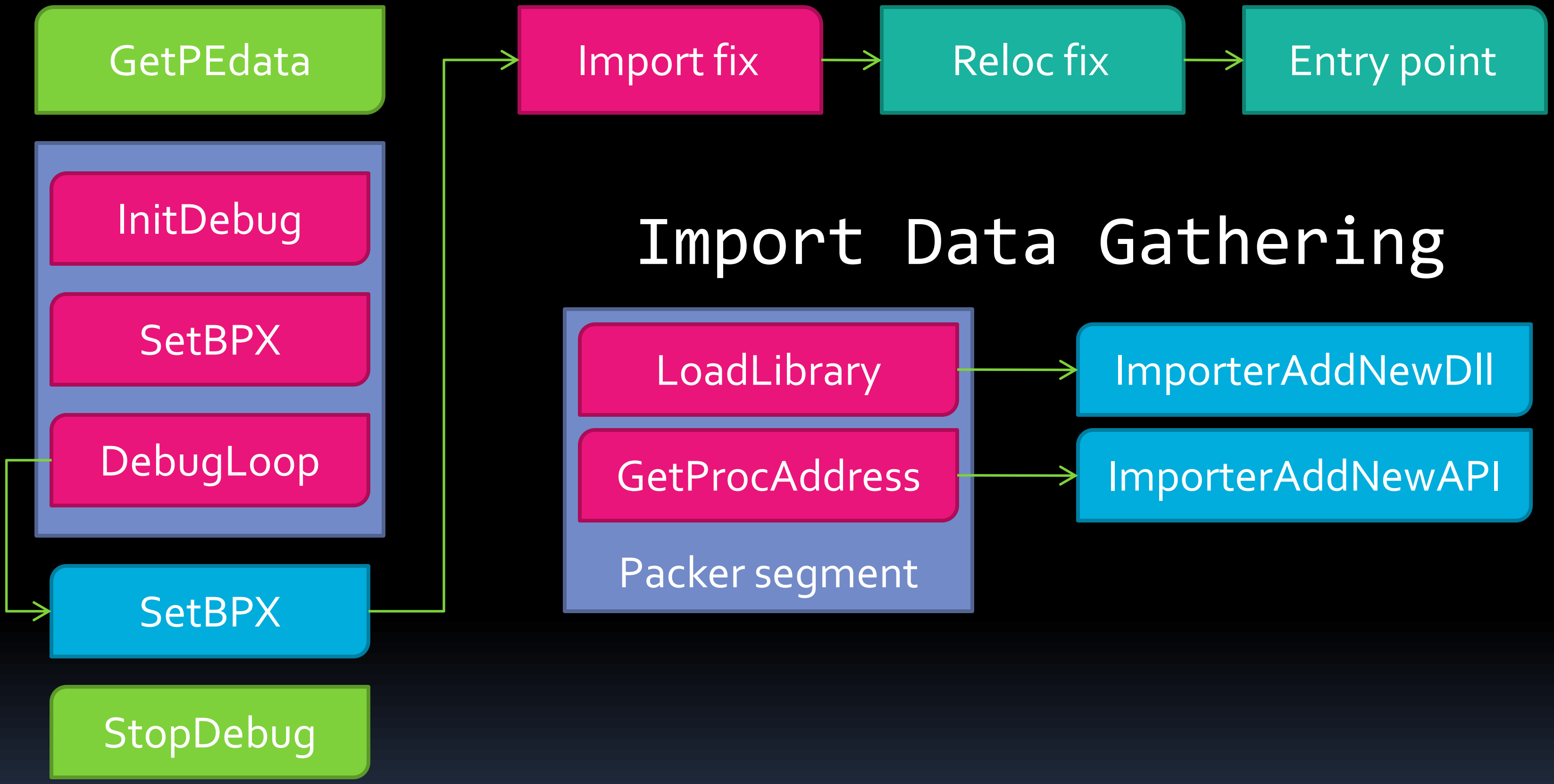
Writing an Unpacker…

- Writing the Unpacking Code
  - Step 3
    - Select the best platform for unpacker creation
      - Select framework
        - Write a custom one, or select existing
      - Select programming language
  - Step 4
    - Write and test it
      - Test on as many samples as you can get your hands on!

Dynamic Unpacker Layout



Dynamic Unpacker Layout



GetPEdata

InitDebug

SetBPX

DebugLoop

SetBPX

StopDebug

Import fix

Reloc fix

Entry point

Import Data Gathering

LoadLibrary

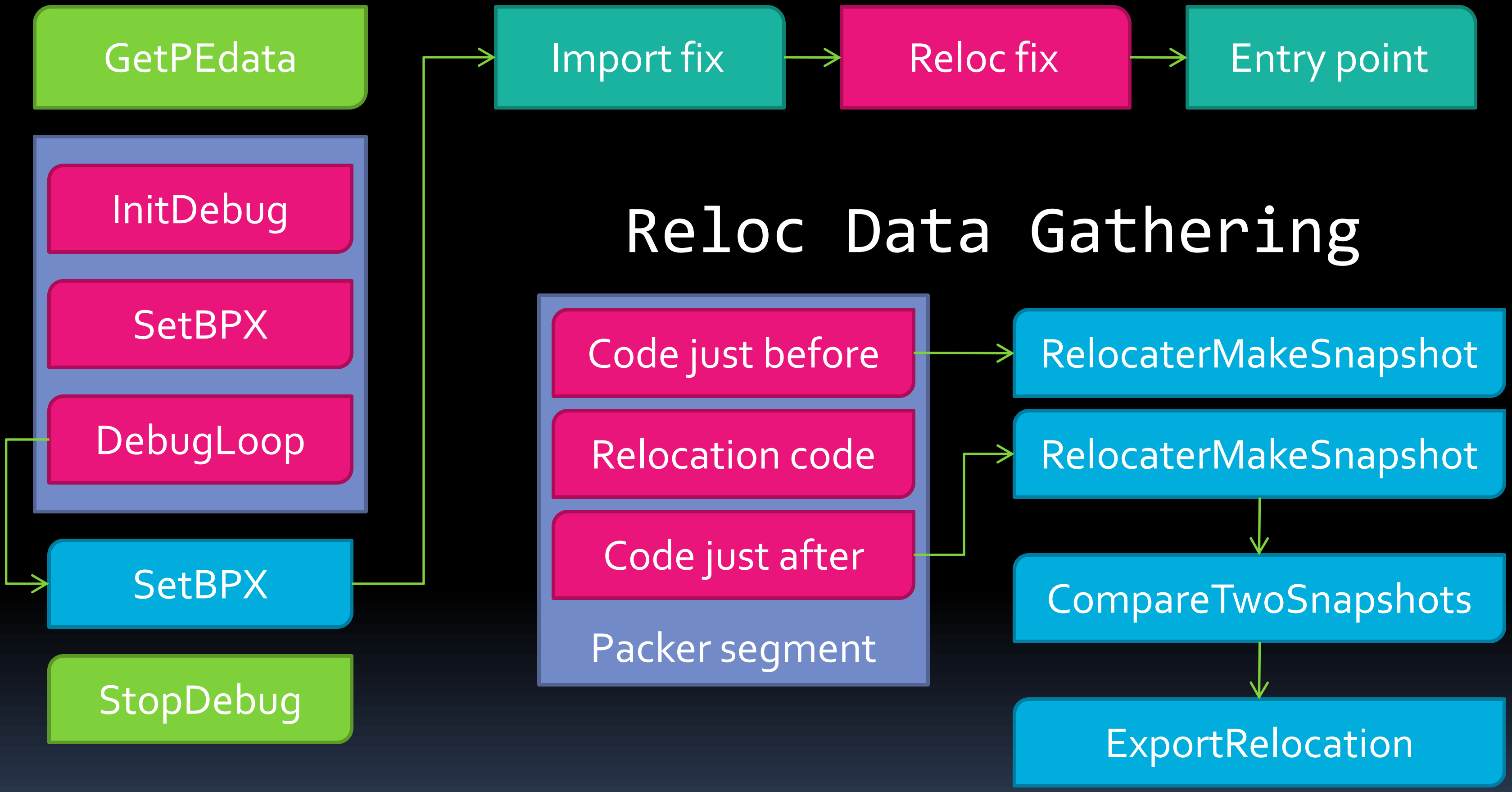
GetProcAddress

Packer segment

ImporterAddNewDll

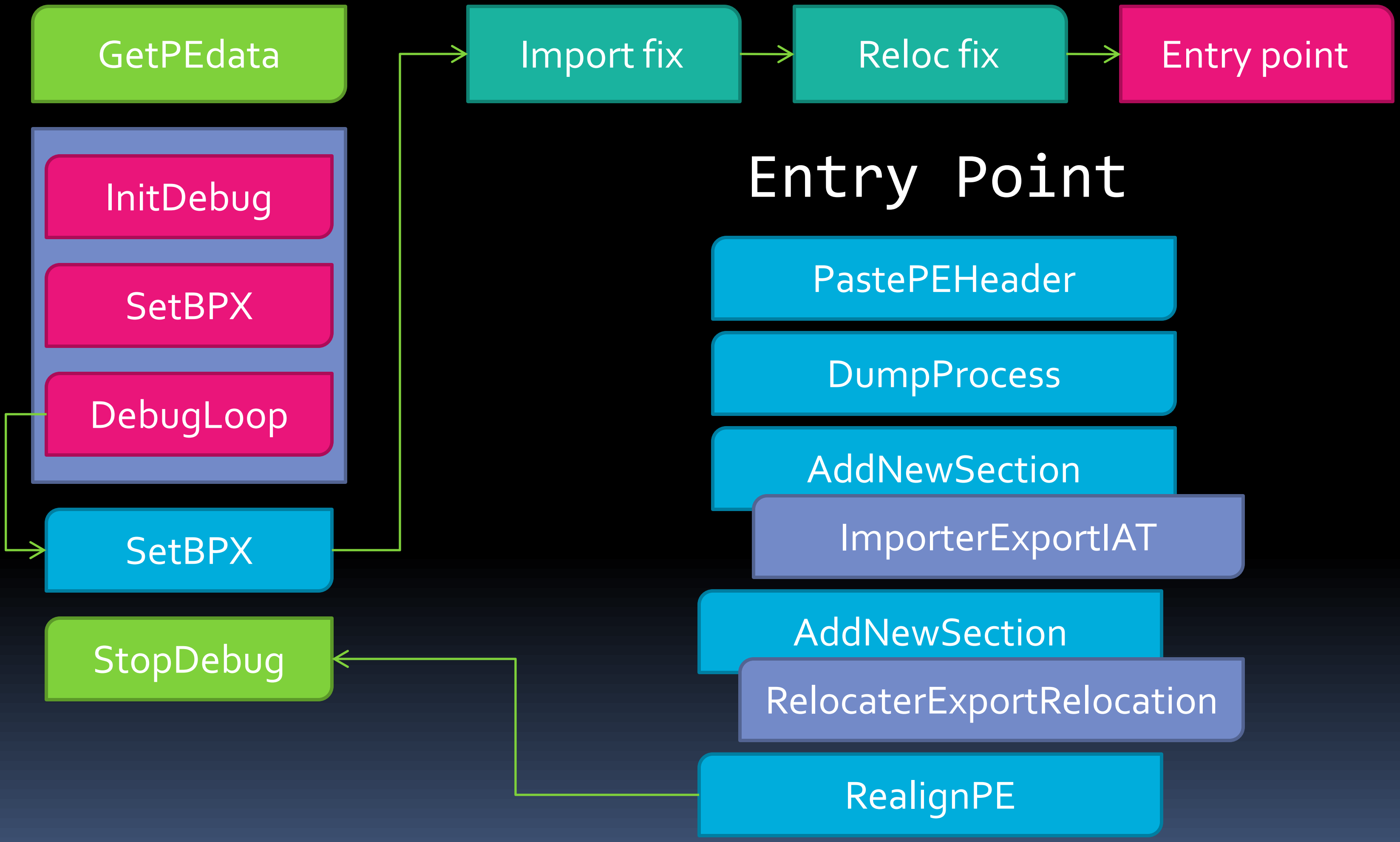
ImporterAddNewAPI

Dynamic Unpacker Layout



Reloc Data Gathering

Dynamic Unpacker Layout



File -> New Unpacker…

- Creating a Dynamic Unpacker for UPX:
  - Gathering info on the packer
    - Free & open source
    - Can pack DLL & EXE files
    - Multiple platforms supported
    - DEP supported but no x64 support
  - Multiple unpackers exist
    - UPX can decompress itself!
  - Multiple signatures available

UPX | Analysis

- Packer Code Points of Interest
  - Point of interest #1:
    - Import table filling (string case)

```
/*40826C*/  MOV EAX,DWORD PTR DS:[EDI]
/*40826E*/  OR EAX,EAX
/*408270*/  JE SHORT crackme_.004082AE
/*408272*/  MOV EBX,DWORD PTR DS:[EDI+4]
/*408275*/  LEA EAX,DWORD PTR DS:[EAX+ESI+8510]
/*40827C*/  ADD EBX,ESI
/*40827E*/  PUSH EAX
/*40827F*/  ADD EDI,8
/*408282*/  CALL NEAR DWORD PTR DS:[ESI+854C]
/*408288*/  XCHG EAX,EBP
```

Bytes: 50 83 C7 08 FF 96 4C 85 00 00

↑

BPX

UPX | Analysis

- Packer Code Points of Interest
  - Point of interest #1:
    - Import table filling (ordinal case)

```
/*40C304*/  MOVZX EAX,WORD PTR DS:[EDI]
/*40C307*/  INC EDI
/*40C308*/  PUSH EAX
/*40C309*/  INC EDI
/*40C30A*/  DB B9
/*40C30B*/  PUSH EDI
/*40C30C*/  DEC EAX
/*40C30D*/  REPNE SCAS BYTE PTR ES:[EDI]
/*40C30F*/  PUSH EBP
/*40C310*/  CALL NEAR DWORD PTR DS:[ESI+CBF8]
/*40C316*/  OR EAX,EAX
```

Bytes: 50 47 ?? 57 48 F2 AE **(BPX @ start)**

Bytes: 57 48 F2 AE ??FF96 F8 CB 00 00

↑

BPX

UPX | Analysis

- Packer Code Points of Interest
  - Point of interest #2:
    - Relocating file to loaded base

```
/*3D2C4A*/  ADD EDI,4
/*3D2C4D*/  LEA EBX,DWORD PTR DS:[ESI-4]
/*3D2C50*/  XOR EAX,EAX
/*3D2C52*/  MOV AL,BYTE PTR DS:[EDI]
/*3D2C54*/  INC EDI
/*3D2C55*/  OR EAX,EAX
/*3D2C57*/  JE SHORT iPackage.003D2C7B
/*3D2C59*/  CMP AL,0EF
/*3D2C5B*/  JA SHORT iPackage.003D2C6E
/*3D2C5D*/  ADD EBX,EAX
/*3D2C5F*/  MOV EAX,DWORD PTR DS:[EBX]
/*3D2C61*/  XCHG AH,AL
/*3D2C63*/  ROL EAX,10
/*3D2C66*/  XCHG AH,AL
/*3D2C68*/  ADD EAX,ESI
/*3D2C6A*/  MOV DWORD PTR DS:[EBX],EAX
/*3D2C6C*/  JMP SHORT iPackage.003D2C50
/*3D2C6E*/  AND AL,0F
/*3D2C70*/  SHL EAX,10
/*3D2C73*/  MOV AX,WORD PTR DS:[EDI]
/*3D2C76*/  ADD EDI,2
/*3D2C79*/  JMP SHORT iPackage.003D2C5D
```

Snapshot

UPX | Analysis

- Packer Code Points of Interest
    - Point of interest #3:
        - Entry point jump (old method)

```
/*4082A1*/  MOV DWORD PTR DS:[EBX],EAX
/*4082A3*/  ADD EBX,4
/*4082A6*/  JMP SHORT crackme_.00408289
/*4082A8*/  CALL NEAR DWORD PTR DS:[ESI+8554]
/*4082AE*/  POPAD
/*4082AF*/  JMP crackme_.004012C0
```

Bytes: 61 E9 0C 90 FF FF

BPX

UPX | Analysis

- Packer Code Points of Interest
  - Point of interest #3:
    - Entry point jump (new method)

```
/*45F5F5*/  POPAD
/*45F5F6*/  LEA EAX,DWORD PTR SS:[ESP-80]
/*45F5FA*/  PUSH 0
/*45F5FC*/  CMP ESP,EAX
/*45F5FE*/  JNZ SHORT dELPHI_u.0045F5FA
/*45F600*/  SUB ESP,-80
/*45F603*/  JMP dELPHI_u.0044CF38
```

Bytes: 83 EC ?? E9 30 D9 FE FF

BPX

UPX | Unpacker

- Starting the "Engine"
    - Read interesting file data
        - ImageBase, AddressOfEntryPoint, …
    - Initialize the debugger
        - InitDebugEx for executables
        - InitDLLDebug for libraries
    - Set initial breakpoint at packer EP
    - DebugLoop();

UPX | Unpacker EP Callback

- Finding Our Points of Interest
  - Find import filling code
    - Set breakpoints pointing to import handle code
      - There are one or two breakpoints here
  - Find “relocate to new base” code
    - Set breakpoints pointing to snapshot code
      - There is one breakpoint here (optional)
  - Find entry point jump
    - Set breakpoints pointing to unpack finalization
      - There is one breakpoint here (but two patterns!)

UPX | Unpacker Breakpoints

- Assign Callbacks to Our Breakpoints
  - Import fixing callback
    - Breakpoint #1; Loading *new* library
      - In this callback call ImporterAddNewDLL
      - Data: EAX holds the pointer to string in ***remote*** **process**

UPX | Unpacker Breakpoints

- Assign Callbacks to Our Breakpoints
  - Import fixing callback
    - Breakpoint #2: Getting API address (string case)
      - In this callback call ImporterAddNewAPI
      - Data: EAX holds the pointer to string in ***remote* process**
      - Data: EBX holds the data write address
    - Breakpoint #3: Getting API address (ordinal case)
      - In this callback call ImporterAddNewAPI
      - Data: EDI holds the ordinal number
      - Data: EBX holds the data write address

UPX | Unpacker Breakpoints

- Assign Callbacks to Our Breakpoints
  - Relocation fixing callback
    - Breakpoint #4; Snapshot #1
      - This is optional breakpoint, present only if file is DLL
      - In this callback we create a snapshot file
      - Function RelocaterMakeSnapshoot
        - Memory which will be snapshot is first PE section

UPX | Unpacker Breakpoints

- Assign Callbacks to Our Breakpoints
  - Original entry point callback
    - Breakpoint #5
      - Fix (possibly broken) PE header with PastePEHeader
      - Dump the process with DumpProcess function
      - Make second relocation snapshot & compare them
      - Add new section for IAT and export IAT to it
        - ImporterExportIAT
      - Add new section for relocations and export them
        - RelocaterExportRelocation / RelocaterChangeFileBase
      - Realign the file with RealignPE
      - Move overlay from original to unpacked file
      - StopDebug();

UPX | DEMO

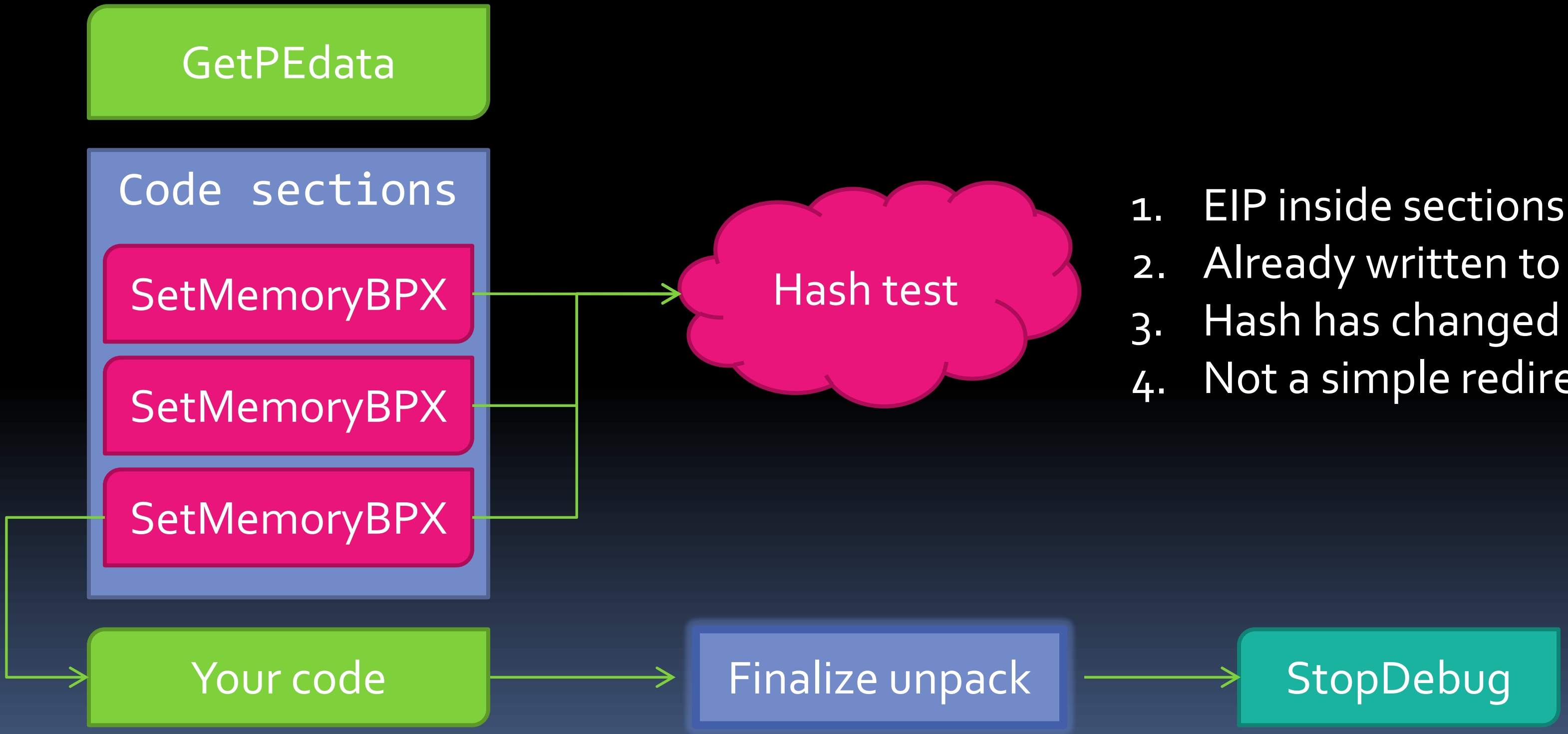
- DEMO - UPX Unpacker
  - But does it actually work?

File -> New Unpacker…

- Create a Generic *Executable* Unpacker
  - No signatures, no patterns, no problem…
    - Generically determine OEP location
    - EP can not be fixed without getting into specifics
    - Automatically fix imports
      - Fix redirections & import eliminations
    - No hassle with relocations
      - But generic DLL unpacker is possible!

Generic OEP finder blueprint

- Creating a generic entry point finder



1. EIP inside sections
2. Already written to
3. Hash has changed
4. Not a simple redirection

Generic Unpacker | DEMO

- RL!dePacker 1.5
  - But does it actually work?

AlexProtector | DEMO

- ImportStudio 2.0
  - Tool similar to ImpRec used to fix imports
  - Demo: fixing import eliminations

tELock | DEMO

- ImportStudio 2.0
  - Tool similar to ImpRec used to fix imports
  - Demo: using ImpRec plugins

TitanEngine | What's Next?

- Extend Framework
  - File function analysis
  - Plugins, modules and scripts
  - Integrated file identification
  - Extend SDK to Delphi and MASM
  - Extend SDK to python and ruby
- More Samples of Usage
  - One unpacker per week project
- More Analysis Tools Built Around It
  - UnpackStudio, MFK…

TitanEngine – How to Help?

- http://titan.reversinglabs.com
- Open Source Project
- Contribute Solutions
- Help others with tutorials
- Contribute Code
- Forums

Questions?

Questions?

(What Would You Like to Know)