BlackHat USA 2010, Las Vegas

Mario Vuksan & Tomislav Pericin

TITANMIST:
YOUR FIRST STEP TO REVERSING NIRVANA

Agenda

- **Introduction**
  - Why TitanMist? Human aspect of the security industry
- **Introduction and review of known formats**
  - Introduction to dynamic analysis and unpacking
  - Solving dynamic analysis problems
- **Introduction to TitanMist**
  - Defining the needed infrastructure
  - Extending the code base & collaboration
  - Building a unique knowledge base about formats

Why TitanMist? Human Aspect of Security

- Security still boils down to an individual
  - Malware Analysis
  - Reverse Engineering
  - Penetration Testing
- Do we have necessary skills?
  - Do we have tools to be successful?
- Tools generally fall into two categories:
  - Either very expensive
  - Or are free/open source and poorly supported
- Fortunately there are some notable exceptions
  - OllyDBG
  - Metasploit

Why TitanMist? Working Together

- Anti-Malware Research Collaboration
  - For Researchers, Investigators and Companies
  - Number of parties is grown rapidly
  - Information data sets are growing
  - Samples collections are expanding rapidly
- Collaboration Problems
  - How to compare collections or data sets?
  - What is a malware family? Naming & behavior conventions
  - What packing/protection formats are used?
  - Are samples original, unpacked or replicated?
  - What identification standard is used?
  - What unpacking standard is used?

Why TitanMist? Unified Unpacking Solution

- Better Reversing Tools are Needed
  - Tools need to be integrated
  - E.g., PeID, OllyScripts, TrID
- Integrated Functionality
  - Format identification, analysis, unpacking
- Alternatives to Commercial Solutions
  - Using AV Products to Unpack
  - Using Sandboxes (Norman, CWSandbox, etc.)
- Open, free and vendor independent solutions
- IEEE Malware Workgroup
  - Peter Ferrie, Microsoft
    - Format Identification Library for Vendor Collaboration
  - Will be integrated into TitanMist

Why TitanMist? Bottom Line

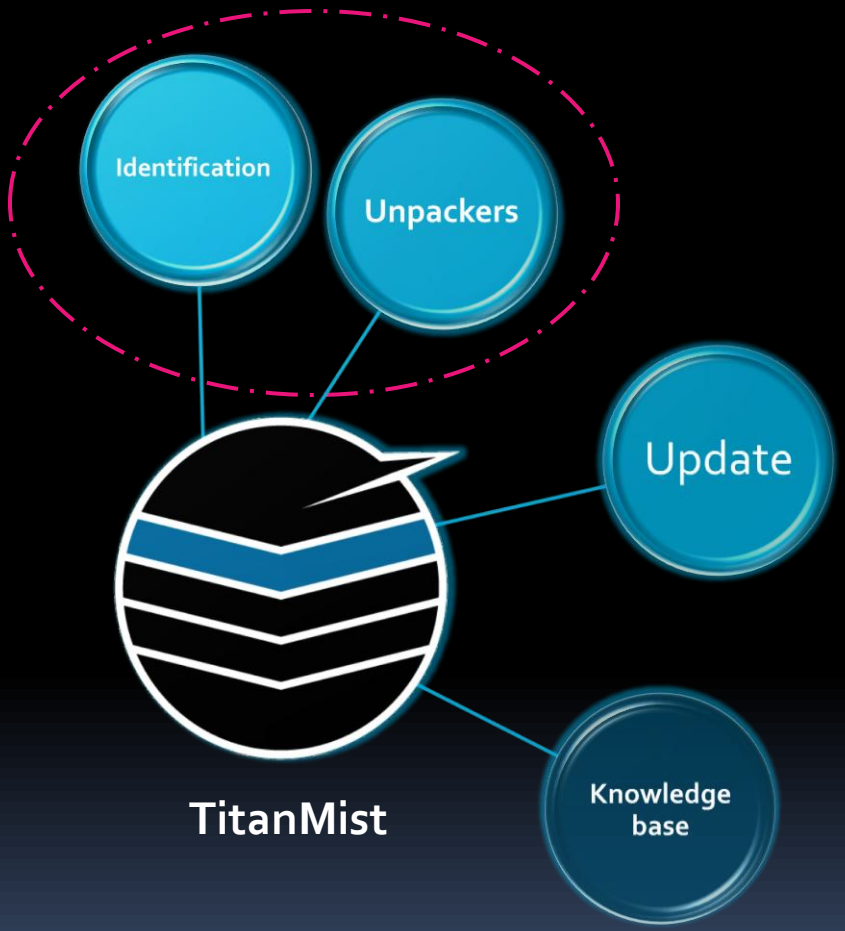
- TitanMist Reversing Goals
  - Faster analysis for different use cases
    - Malware, Cracked Software, Vulnerable Applications
  - Removal of obfuscation
  - Better data for heuristic systems
  - Accessibility: open and free
- TitanMist Community Goals
  - Malware analysis is no longer for AV Labs only
  - While there is a space for specialized and expensive toolsets
  - General public needs open and free alternatives
  - General public needs well supported projects
- Community will grow around
  - A unified tool (multiple author, but rather one distribution)
  - Information repository (multiple authors, one website)

TitanMist

TitanMist|Introduction

- TitanMist's key features:
  - Tool for format identification
  - Tool for format specific unpacking
  - Format info stored in a public knowledge base
  - Easily extendable & community supported
  - Always up to date

TitanMist|Infrastructure



TitanMist|Database

- TitanMist Database
  - Links signatures with format specific unpackers

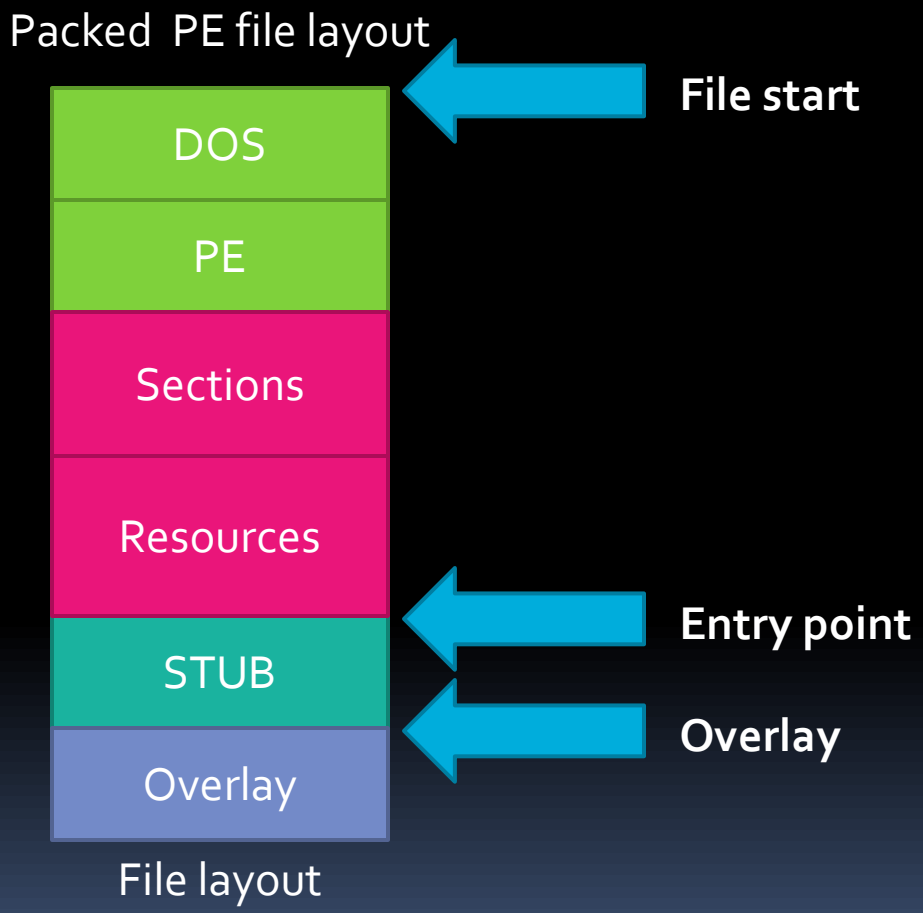
```
<mistdb version="0.1">
        <entry
                name="…"
                url="…"
                version="…"
                description="…"
                priority="1"
                author="…">
                <unpacker type="…" >filename.ext</unpacker>
                <signature start="ep" version="1.x – 3.x" unpacker="…">
                        PATTERN
                </signature>
        </entry>
</mistdb>
```

REVERSING LABS | Reverse Engineering & Software Protection

TitanMist|Identification

- TitanMist identification
  - Signatures can be simple or complex
  - Signatures are stored into XML database
  - Signatures are grouped by formats into entries
  - Detection is defined by the entry or the signature
  - Entries can be linked with multiple unpackers
  - Entries are linked to online knowledge base

Identification|Pattern start



Identification|Pattern start

- TitanMist identification signatures start:
  - **ep** – Match the pattern from the PE entry point
  - **overlay** - Match the pattern from the PE overlay
  - **begin** – Match the pattern from the file start
  - **all** – Scan the entire file for the pattern
- Seek or match can be defined for any search

Identification|Simple patterns

- Simple TitanMist identification patterns
  - Simple patterns are equal to PEiD patterns
  - Enable pattern matching by following rules:
    - ?? – Wild card byte (any byte matches it)
    - ?x – Bit masking for the high bits
    - x? – Bit masking for the low bits
  - Example *UPX* pattern:

    60 BE ?? ?? ?? ?? 8D BE ?? ?? ?? ?? 5? 83 CD FF EB 10
    90 90 90 90 90 90 8A 06 46 88 07 47 01 DB 75 07

Identification|Problem #1

- Arbitrary number of bytes of the same type

```
/*408160*/  PUSHAD
/*408161*/  MOV ESI,00406000
/*408166*/  LEA EDI,DWORD PTR DS:[ESI+FFFFB000]
/*40816C*/  PUSH EDI
/*40816D*/  OR EBP,FFFFFFFF
/*408170*/  JMP SHORT 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
```

Identification|Complex patterns

- Complex TitanMist identification patterns
  - Enable pattern matching by following rule:
    - "*("byte")" – Match the selected byte multiple times
  - Solution to the variable bytes problem
    - Solves variable byte number problem
    - Solves long signatures due to repetition
  - Example *UPX* pattern:

    60 BE ?? ?? ?? ?? 8D BE ?? ?? ?? ?? 57 83 CD FF EB ??
    ***(90) 8A 06 46 88 07 47 01 DB 75 07**

Identification|Problem #2

- Jumps that increase or decrease

```
/*408160*/  PUSHAD
/*408161*/  MOV ESI,00406000
/*408166*/  LEA EDI,DWORD PTR DS:[ESI+FFFFB000]
/*40816C*/  PUSH EDI
/*40816D*/  OR EBP,FFFFFFFF
/*408170*/  JMP SHORT 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
```

Identification|Complex patterns

- Complex TitanMist identification patterns
  - Enable pattern matching by following rule:
    - "[" byte "-" byte "]" – Detect if the byte is in range
  - Solution to the variable bytes problem
    - Solves register permutation problem
    - Solves jump direction problem
  - Example *UPX* pattern:

    60 BE ?? ?? ?? ?? 8D BE ?? ?? ?? ?? 57 83 CD FF EB [00–7F]
    90 90 90 90 90 90 8A 06 46 88 07 47 01 DB 75 07

Identification|Problem #3

- Code that is only in certain cases there

```
/*1222AE0*/ CMP BYTE PTR SS:[ESP+8],1
/*1222AE5*/ JNZ 01222C7C
/*1222AEB*/ PUSHAD
/*1222AEC*/ MOV ESI, 011E6000
/*1222AF1*/ LEA EDI,DWORD PTR DS:[ESI+FFF8B000]
/*1222AF7*/ PUSH EDI
/*1222AF8*/ OR EBP,FFFFFFFF
/*1222AFB*/ JMP SHORT 01222B0A
/*1222AFD*/ NOP
/*1222AFE*/ NOP
/*1222AFF*/ NOP
/*1222B00*/ MOV AL,BYTE PTR DS:[ESI]
/*1222B02*/ INC ESI
/*1222B03*/ MOV BYTE PTR DS:[EDI],AL
/*1222B05*/ INC EDI
```

Identification|Complex patterns

- Complex TitanMist identification patterns
  - Enable pattern matching by following rule:
    - "(" byte pattern ")" – Optional byte pattern
  - Solution to the variable bytes problem
    - Solves optional instructions problem
    - Solves the multiple signatures problem
  - Example *UPX* pattern:

    (80 7C 24 08 01 0F 85 ?? ?? ?? ??)
    60 BE ?? ?? ?? ?? 8D BE ?? ?? ?? ?? 57 83 CD FF EB [00–7F]
    90 90 90 90 90 90 8A 06 46 88 07 47 01 DB 75 07

Identification|Problem #4

- Large unknown blocks of code

```
/*409678*/  JMP 00400154
...
/*400154*/  MOV ESI, 0040701C
/*400159*/  MOV EBX,ESI
/*40015B*/  LODS DWORD PTR DS:[ESI]
/*40015C*/  LODS DWORD PTR DS:[ESI]
/*40015D*/  PUSH EAX
/*40015E*/  LODS DWORD PTR DS:[ESI]
/*40015F*/  XCHG EAX,EDI
/*400160*/  MOV DL,80
/*400162*/  MOVS BYTE PTR ES:[EDI],BYTE PTR DS:[ESI]
/*400163*/  MOV DH,80
/*400165*/  CALL NEAR DWORD PTR DS:[EBX]
/*400167*/  JNB SHORT 00400162
/*400169*/  XOR ECX,ECX
```

Identification|Complex patterns

- Complex TitanMist identification patterns
  - Enable pattern matching by following rule:
    - "+/-(" hex offset ")" – Skip or rewind number of bytes
  - Solution to the unknown bytes problem
    - Solves the problem of increasing bytes patterns
    - Solves the problem of byte patterns being linear
  - Example *MEW* pattern:

    4D 5A +(152) BE ?? ?? ?? ?? 8B DE AD AD 50 AD 97 B2 80 A4 B6 80 FF 13 73 F9 33 C9 FF 13 73 16 …

Identification|Problem #5

- Multi layer packer code

```
/*4012C0*/  MOV EAX, 00407D34
/*4012C5*/  PUSH EAX
/*4012C6*/  PUSH DWORD PTR FS:[0]
/*4012CD*/  MOV DWORD PTR FS:[0],ESP
/*4012D4*/  XOR EAX,EAX
/*4012D6*/  MOV DWORD PTR DS:[EAX],ECX
...
MOV EAX,F0406AB9
LEA ECX,DWORD PTR DS:[EAX+1000129E]
MOV DWORD PTR DS:[ECX+1],EAX
MOV EDX,DWORD PTR SS:[ESP+4]
MOV EDX,DWORD PTR DS:[EDX+C]
MOV BYTE PTR DS:[EDX],0E9
ADD EDX,5
SUB ECX,EDX
```

REVERSING LABS | Reverse Engineering & Software Protection

PeCompact

Identification|Complex patterns

- Complex TitanMist identification patterns
  - Enable pattern matching by following rule:
    - "+(?)" – Follow DWORD virtual address
  - Solution to the multi layer pattern problem
    - Solves the problem of byte patterns not being linear
  - Example *PECompact* pattern:

B8 ?? ?? ?? ?? 50 64 FF 35 00 00 00 00 64 89 25 00 00 00 00
33 C0 89 08 50 45 43 6F 6D 70 61 63 74 -(21) B8 +(?) B8 ?? //cut

Identification|Problem #6

- Multi layer packer code

```
/*407000*/ CALL  00407083
...
/*407083*/ POP EAX
/*407084*/ PUSHAD
/*407085*/ MOV EBP,EAX
/*407087*/ PUSH EBP
/*407088*/ XOR ESI,ESI
/*40708A*/ PUSH 148
/*40708F*/ CALL 004071DD
...
/*4071DD*/ CALL  00407210
...
/*407210*/ POP EDX
/*407211*/ INC EDX
/*407212*/ PUSH EDX
```

REVERSING LABS | Reverse Engineering & Software Protection

ShrinkWrap

Identification|Complex patterns

- Complex TitanMist identification patterns
  - Enable pattern matching by following rule:
    - "!(+/-" 2/5/6 ")" – Follow relative jumps and calls
  - Solution to the multi layer pattern problem
    - Solves the problem of byte patterns not being linear
    - Solves the problem of increasing bytes patterns
  - Example *ShrinkWrap* pattern:

E8 !(+5) 58 60 8B E8 55 33 F6 68 ?? ?? ?? ?? E8 !(+5) E8 !(+5)
5A 42 52 B9 ?? ?? ?? ?? 33 C0 8B 02 83 F0 01 89 02 42 E2 F6
5A 8B C4 64 8B ... // cut

Identification|Problem #7

- Sliding pieces of code

```
/*4072DA*/ TEST EBX,EBX
/*4072DC*/  JE 00407384
/*4072E2*/ TEST EBX,80000000
/*4072E8*/  JNZ SHORT 004072EE
/*4072EA*/  ADD EBX,EDX
/*4072EC*/  INC EBX
/*4072ED*/  INC EBX
/*4072EE*/  PUSH EBX
/*4072EF*/  AND EBX,7FFFFFFF
/*4072F5*/  PUSH EBX
/*4072F6*/  PUSH DWORD PTR SS:[EBP+545]
/*4072FC*/  CALL NEAR DWORD PTR SS:[EBP+F49]
/*407302*/ TEST EAX,EAX
/*407304*/  POP EBX
/*407305*/  JNZ SHORT 00407376
```

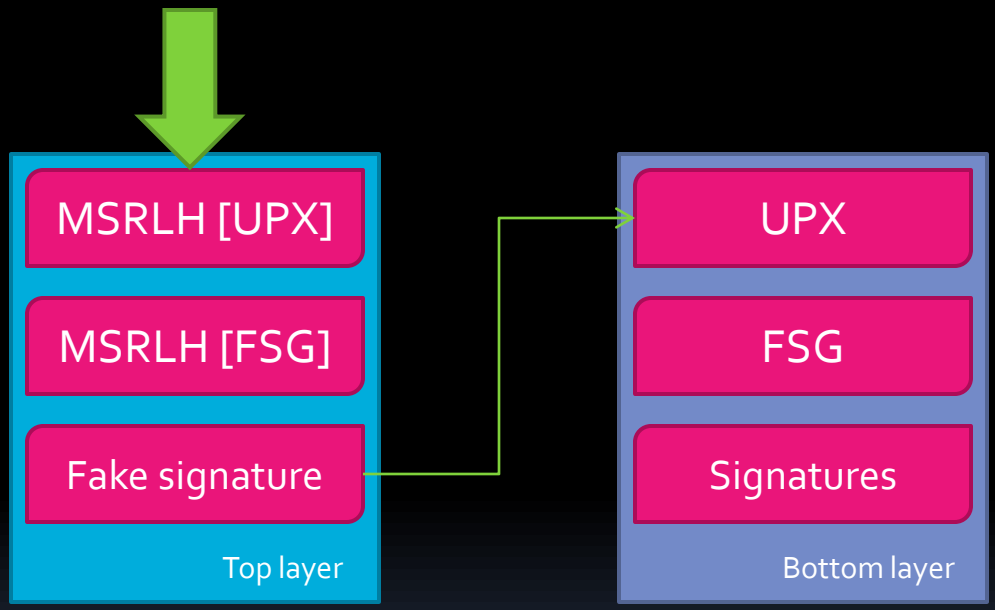
Identification|Complex patterns

- Complex TitanMist identification patterns
  - Enable pattern matching by following rule:
    - "+/-[" range "]{" pattern "}"– Seek the pattern in range
  - Solution to the multiple signatures
    - Solves the problem of byte patterns not matching
    - Enables embedding unpacker patterns in identification
  - Example *ASPack* pattern:

```
60 E8 03 00 00 00 E9 EB 04 5D 45 55 C3 E8 01 00 00 00  ... // cut
+[300]{8B D8 50 FF 95 ?? ?? ?? ?? 85 C0 75 07 53 FF 95}
+[100]{53 81 E3 FF FF FF 7F 53 FF B5 ?? ?? ?? ?? FF 95
?? ?? ?? ?? 85C0 5B} +[100]{61 75 08 B8 01 00 00 00 C2
0C 00 68 ?? ?? ?? ?? C3}
```

Identification|Priority

- TitanMist signature patterns are layered



Identification|Future plans

- Future complex TitanMist signature patterns
  - Making signatures PE format aware
    - Disable signatures for DLL, x64 and .net files
    - Filter files by import table, etc.
  - Combining patterns with logic responses
    - Multiple patterns making a single signature

TitanMist vs PEiD patterns

*comparison refers only to user made signatures

TitanMist

- Complex patterns
- Any direction patterns
- Multiple start points
- Match or seek patterns
- Variable byte patterns
- Skip byte patterns
- Optional patterns
- Code flow following
- Signature priority

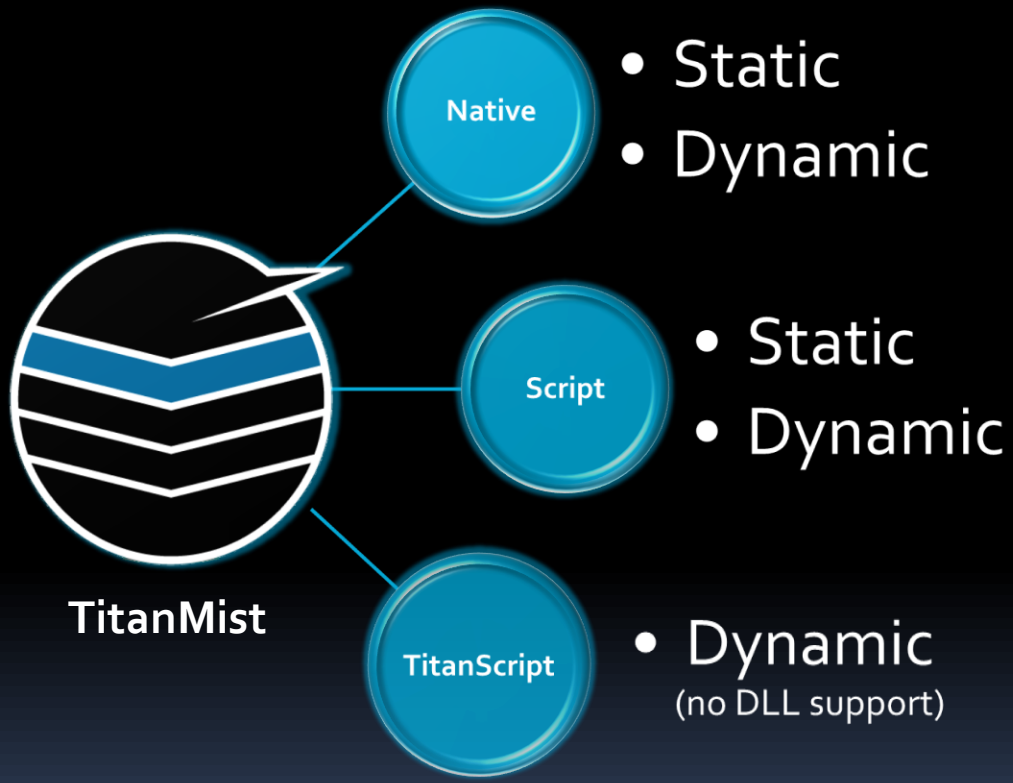
PEiD

- Simple patterns only
- Single direction patterns
- Single start point

TitanMist|Unpackers

- TitanMist unpacking
  - TitanMist uses automated unpackers
  - Unpackers can be written in many languages
    - C, C++, MASM, Delphi, LUA, Python and *TitanScript*
      - *TitanScript is based on **ODbgScript** by SHaG & Epsylon3*
  - *Script unpackers* are based on the TitanEngine
  - *Native unpackers* can be based on the TitanEngine or on any other framework or custom code (DLL)

TitanMist|Unpackers

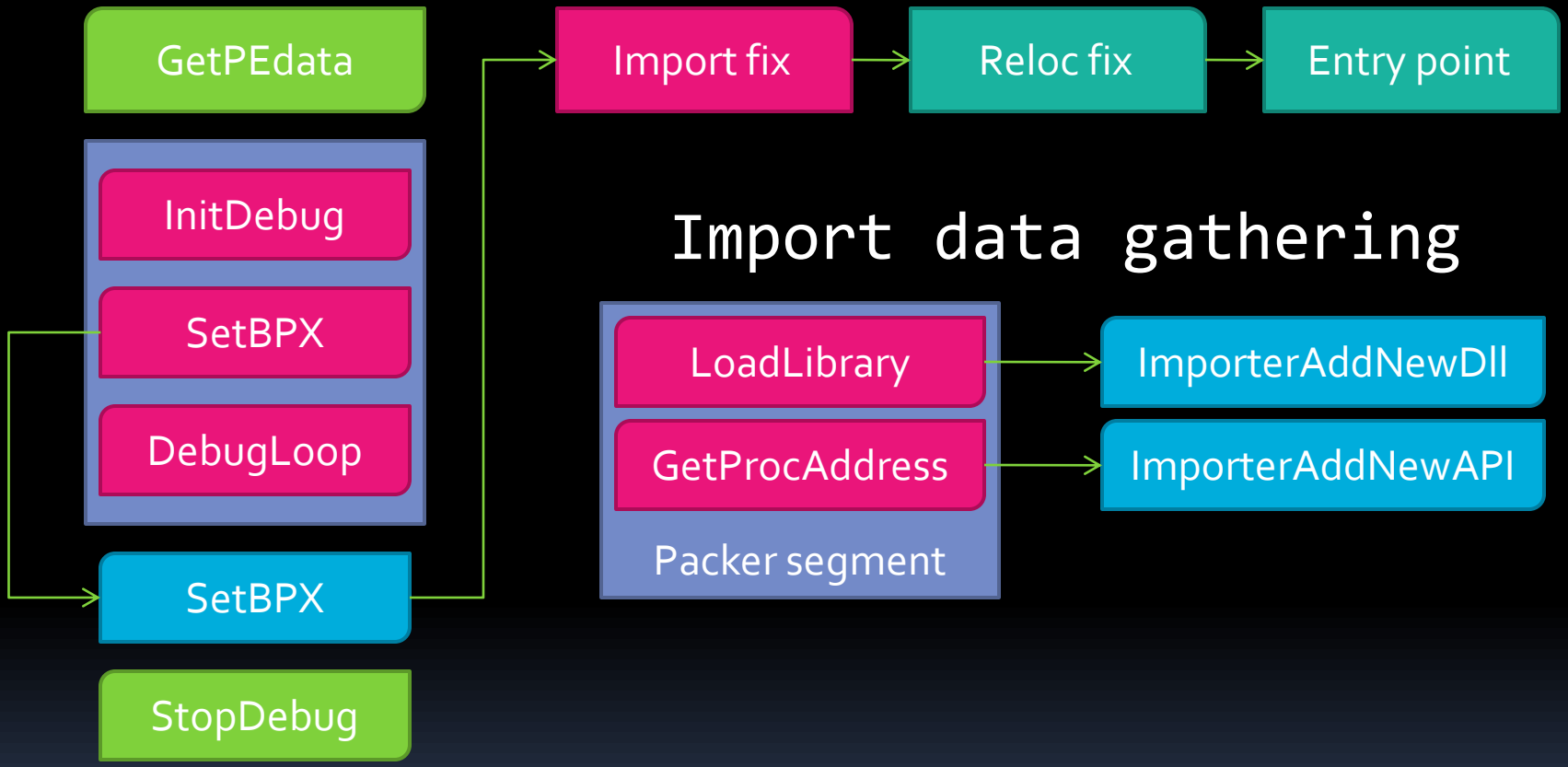


REVERSING LABS | Reverse Engineering & Software Protection

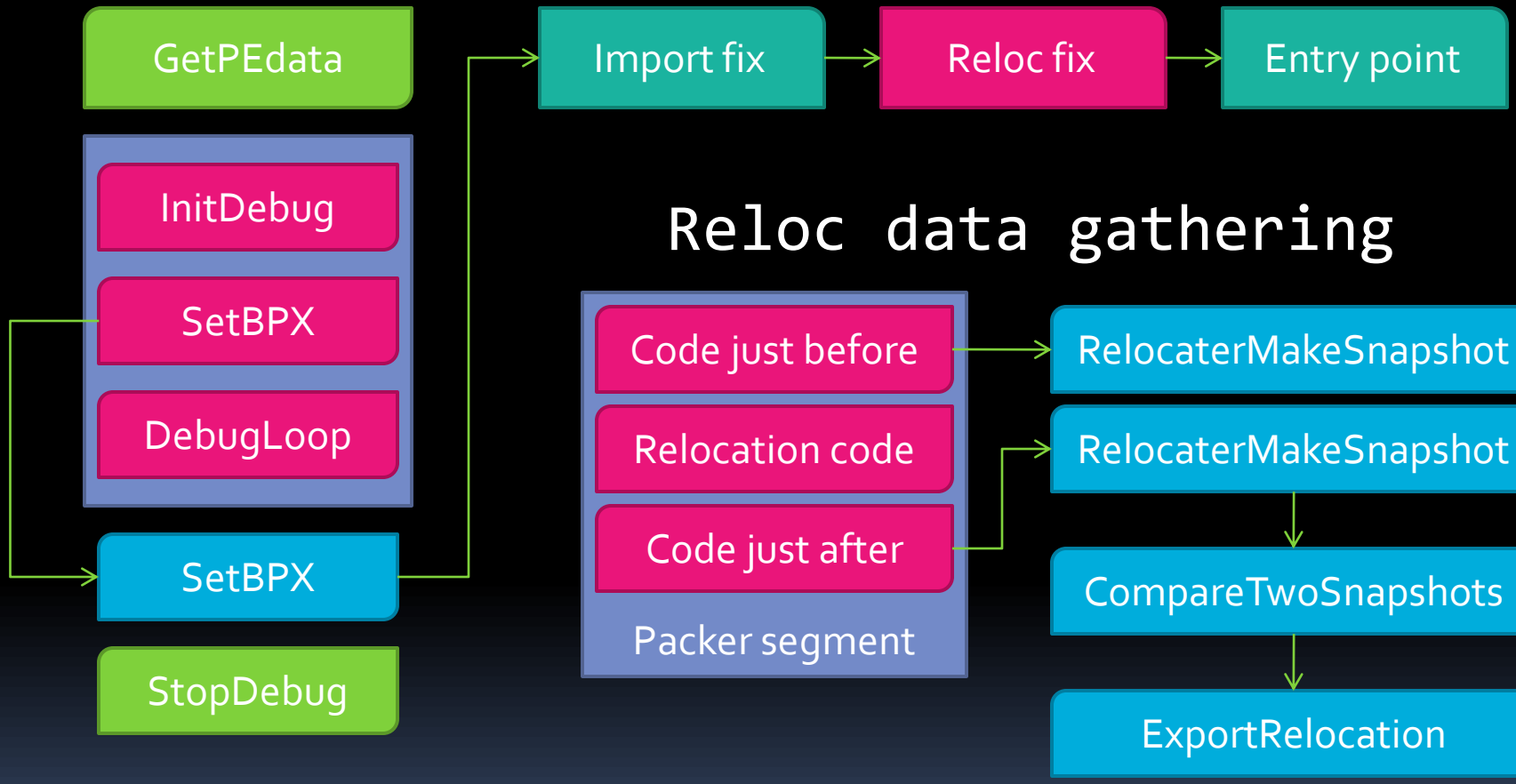
TitanMist|Unpacker coding

- TitanMist unpacker coding
  - TitanEngine simulates reverse engineers presence
    - Dynamic unpacking process has the same steps
      - Debugging until entry point
      - Dumping memory to disk
      - Collection of data for import fixing
      - Collection of data for relocation fixing
      - Custom fixes (Code splices, Entry point, …)
    - Static unpacking process has standard steps
      - Decryption and/or decompression
      - Import table and original entry point correction

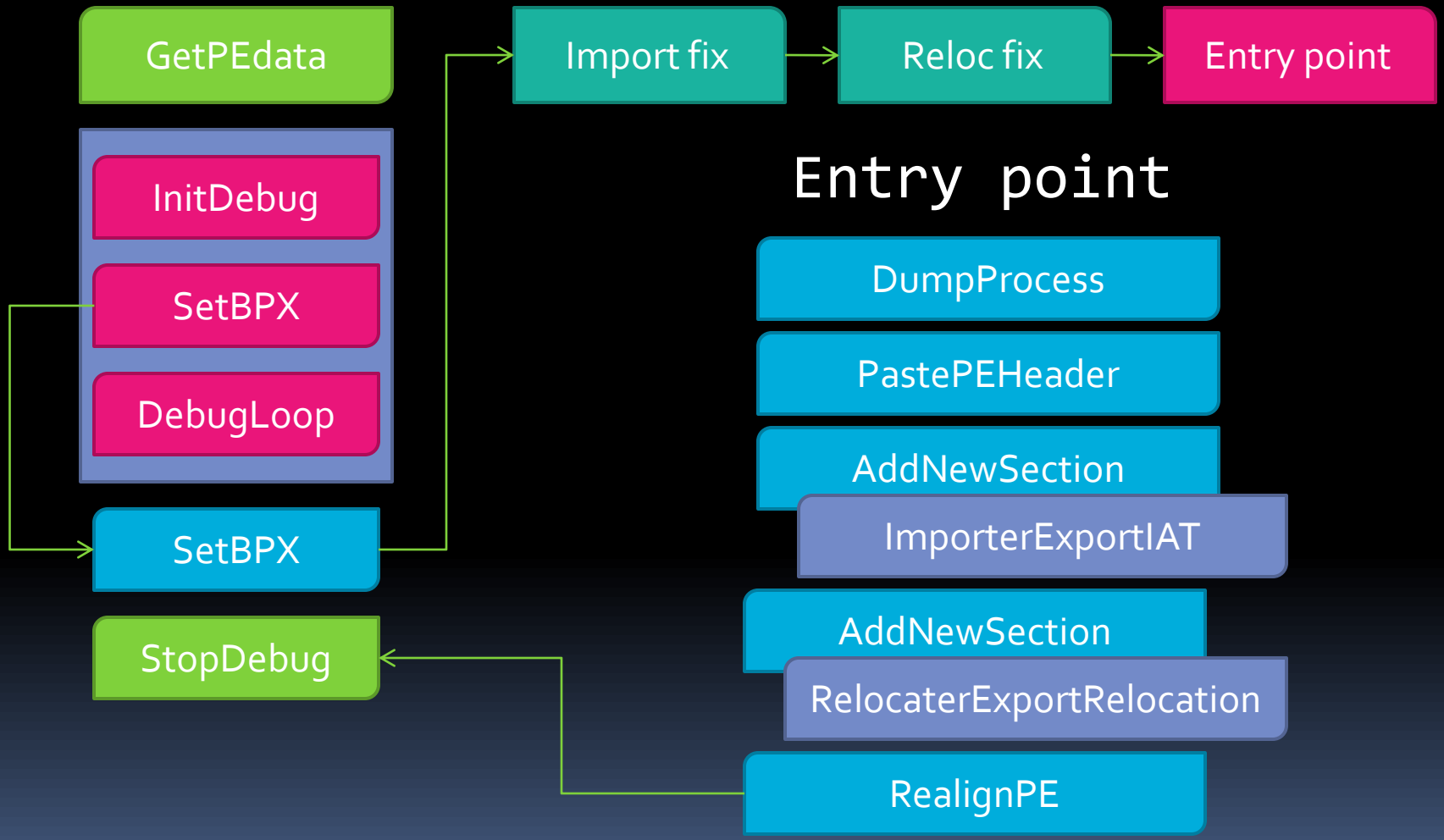
TitanMist|Unpacker coding



TitanMist|Unpacker coding



TitanMist|Unpacker coding



TitanScript|Unpacker coding

- TitanScript unpacker coding
  - TitanScript uses ODbgScript syntax
  - TitanScript enables use of TitanEngine functions
  - TitanScript is *compatible* with existing scripts
  - OllyScripts can easily be upgraded to TitanScripts
    - Partial script recoding
    - Instruction addition

TitanScript|Unpacker coding

- OllyScript to TitanScript conversion
  - Problem: OllyScripts are **not** full blown unpackers!
  - Solution(s):
    1. Recoding to match TitanEngine layout
    2. Instruction adding:
       - DNF – dump and fix
       - ERROR – set unpacking error

TitanScript|UPX Example

- **OllyScript**

```
eob Break
findop eip, #61#
log $RESULT
bphws $RESULT, "x"
Run
Break:
 eob // clear
 bphwc eip
 bp eip + 14.
 run
 sti
 ret
```

- **TitanScript**

```
eob Break
findop eip, #61#
log $RESULT
bphws $RESULT, "x"
Run
Break:
 eob // clear
 bphwc eip
 bp eip + 14.
 run
 sti
 dnf
 ret
```

Dynamic unpacking problems

- Dynamic unpacking yields following problems
  - Damaged or broken files can't be unpacked
  - Files with missing dependencies can't be unpacked
  - DEP non compatible files can't be unpacked
- Good news!
  - There is a solution for each of these problems
  - We ***can:***
    - Repair the damaged files
    - We can simulate presence of needed dependencies
    - We can work around DEP or disable it
  - TitanEngine Nexus plugin performs this automatically!

Nexus|Fixing broken files

- File validation should be done before any unpacking, especially dynamic, is performed
- Validation gives detailed file information
  - Wheatear or not the file is valid
  - Wheatear or not broken file can be fixed
- Validation & repair is done automatically

Nexus|Missing dependencies

- If observed standalone, files can be missing crucial dependencies
- Dependencies are crucial only for packed file not the packer itself, but:
  - Files must be present on disk if the packer imports them statically - *done automatically*
  - Packed must be fooled that actual functions exist in these fake files - *done automatically*

TitanMist | DEMO

TitanMist|Knowledge base

- TitanMist knowledge base
  - Online Wikipedia file format knowledge base
    - File format descriptions
      - Basic file format information
      - Extensive file format analysis
      - Protection options descriptions
    - TitanMist unpackers
    - Sample files

TitanMist|Release

Native unpackers

- AHPack
- CryptoCrack PE Protector
- AlexProtector
- Yoda Crypter
- LameCrypt
- ExeFog
- tELock
- nPack
- MEW5
- DEF

Script unpackers

- ASPack
- RLPack
- BeroExePacker
- PeCompact
- ShrinkWrap
- PackMan
- FSG
- MEW
- PEX
- UPX

Questions?

(What Would You Like to Know)