BlackHat USA 2010, Las Vegas

Mario Vuksan & Tomislav Pericin

TITANMIST: YOUR FIRST STEP TO REVERSING NIRVANA

Agenda

- **Introduction**
  - 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

Human Aspect of Security

- Boils down to an individual
  - Malware Analysis
  - Reverse Engineering
  - Penetration Testing
- Do we have what it takes?
  - Does we have tools to be successful?
- Tools generally fall into too categories:
  - Either very expensive
  - Or poorly supported
- Black hat down to exceptions
  - OllyDBG
  - Metasploit

Why TitanMist?

- Better Reversing Tools are Needed
  - PeID
  - OllyScripts
  - TrID
- Unified tools are needed
  - Format identification, analysis, unpacking
- Necessary alternative Options To
  - Using AV Products to Unpack
  - Using Sandboxes (Norman, CWSandbox, Anubis, ThreatExpert, etc.)

Unified Solution

- Goals
  - Faster analysis
    - Malware
    - Cracked Software
    - Vulnerable Applications
  - Removal of obfuscation
  - Better data for heuristic systems
  - Accessibility

Bottom Line

- Malware analysis is no longer for AV Labs only
- General public does not have money for high end specialized toolsets
- General public needs well supported projects
- Community needs to coalesce around
  - A unified tool (not one author, but rather one distributions)
  - Information repository (one website)

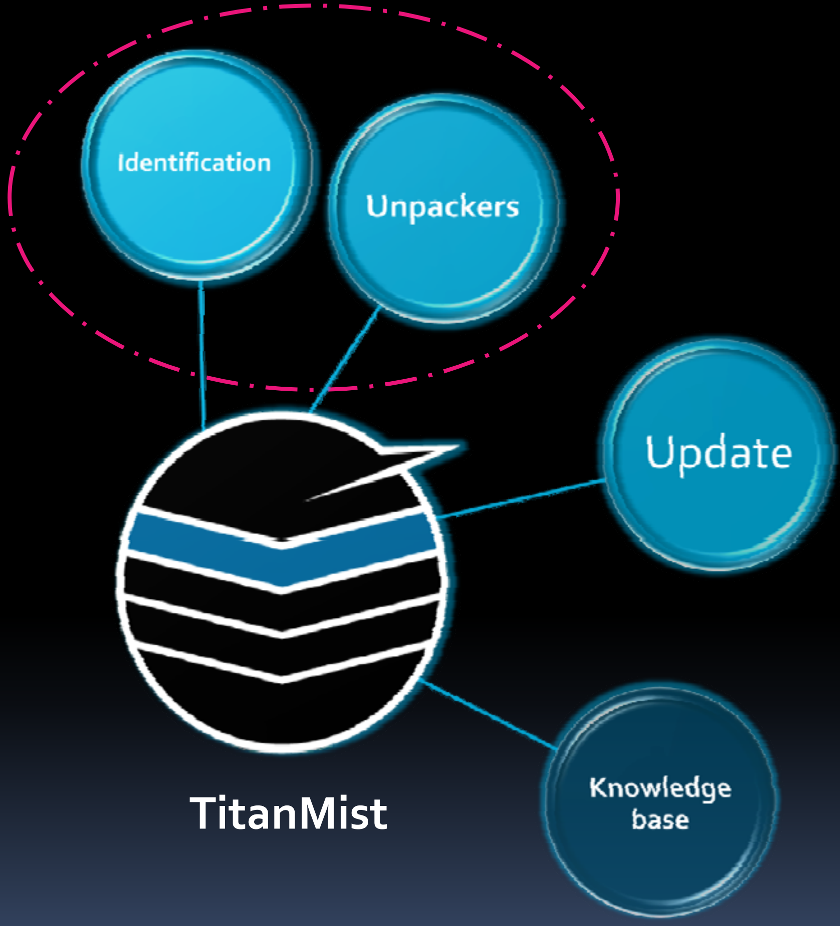
TitanMist

REVERSING LABS | Reverse Engineering & Software Protection

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

- 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 ?? ?? ?? ?? 57 83 CD FF EB 10
90 90 90 90 90 90 8A 06 46 88 07 47 01 DB 75 07
```

REVERSING | Reverse Engineering & Software Protection LABS

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**

REVERSING LABS | Reverse Engineering & Software Protection

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|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|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|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 ??
```

Identification|Complex patterns

- Complex TitanMist identification patterns
  - Enable pattern matching by following rule:
    - "!(+/-" 2/4/5 ")" – 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 *UPX* pattern:

(80 7C 24 08 01 0F 85 !(+6))
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|Future plans

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

TitanMist vs PEiD patterns

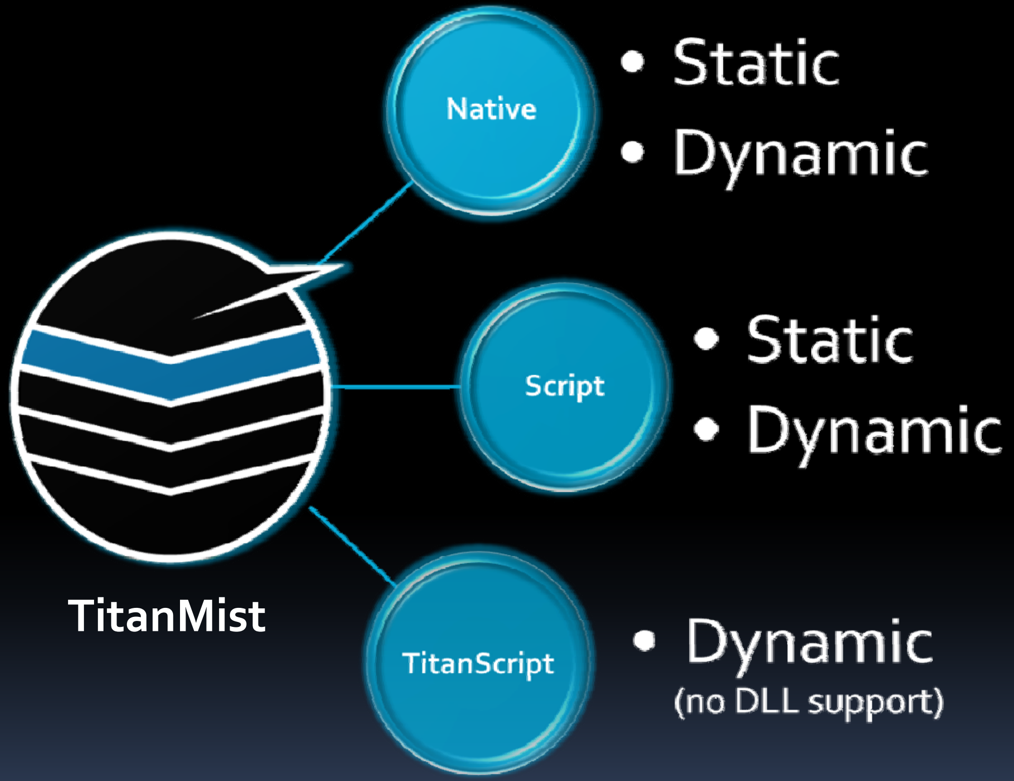
*comparison refers only to user made signatures

TitanMist	PEiD
Complex patterns	Simple patterns only
Any direction patterns	Single direction patterns
Multiple start points	Single start point
Match or seek patterns	
Variable byte patterns	
Skip byte patterns	
Optional patterns	
Code flow following	
Signature priority	

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)

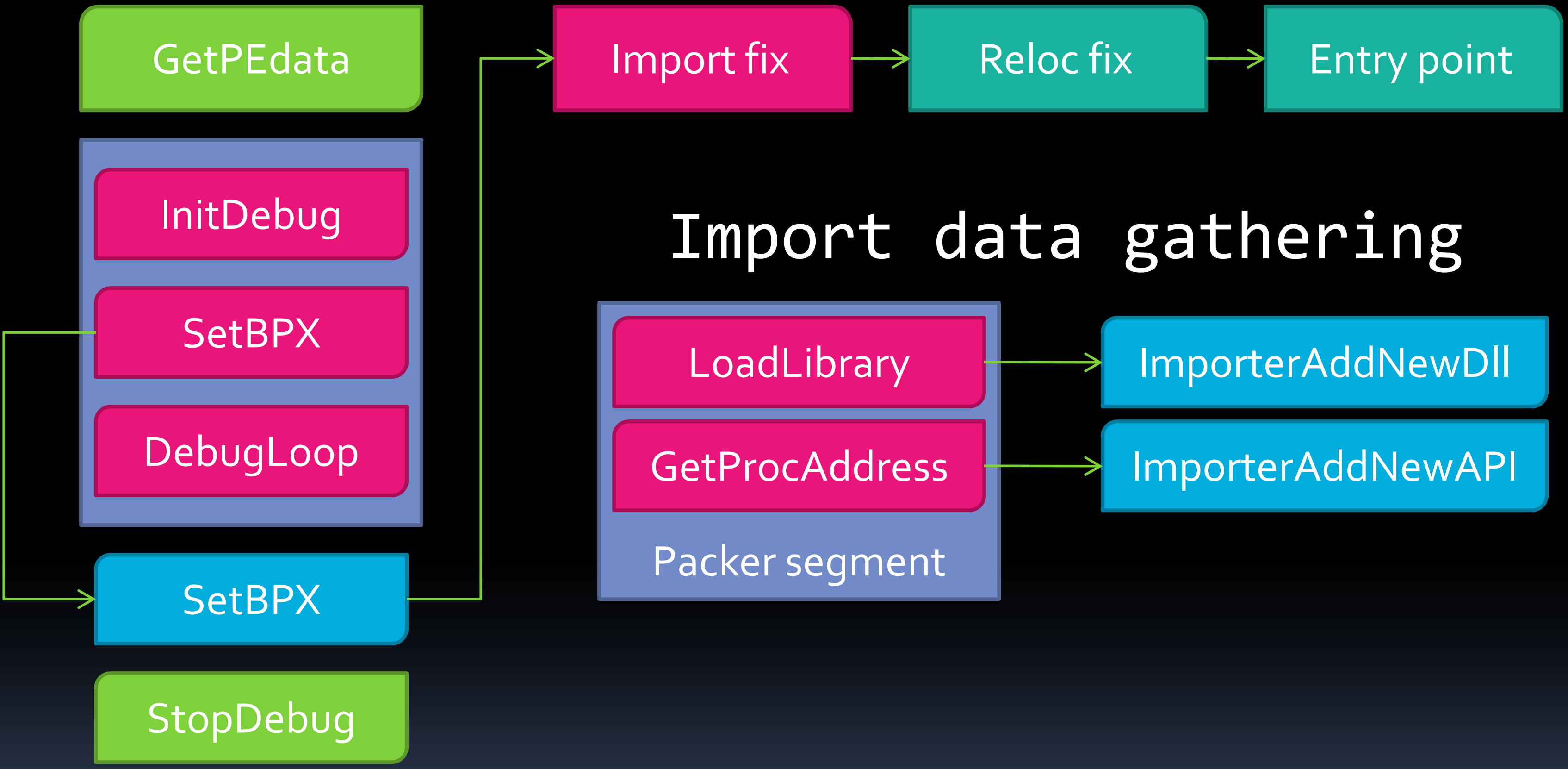
TitanMist|Unpackers



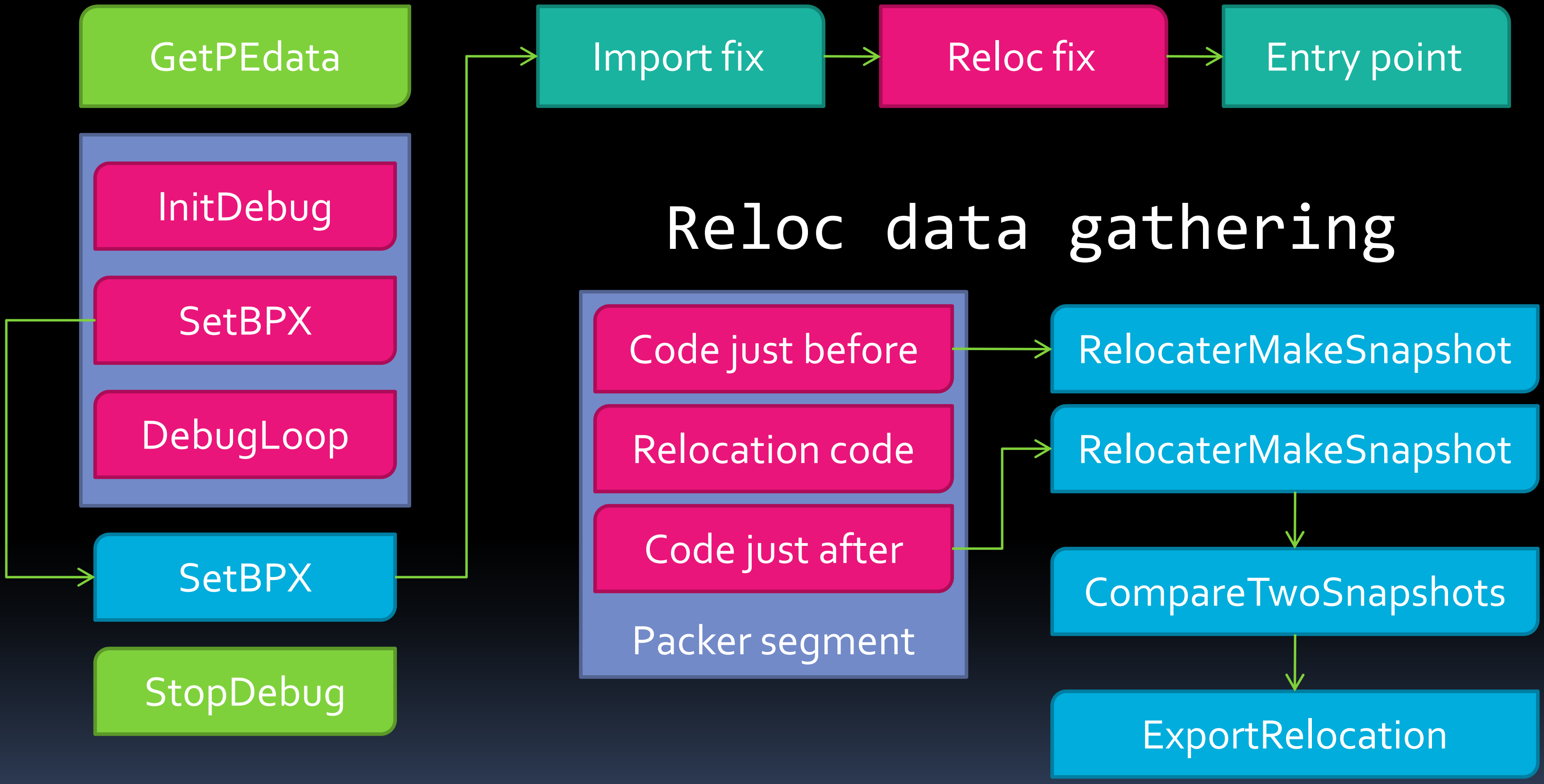
TitanMist|Unpacker coding

- 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

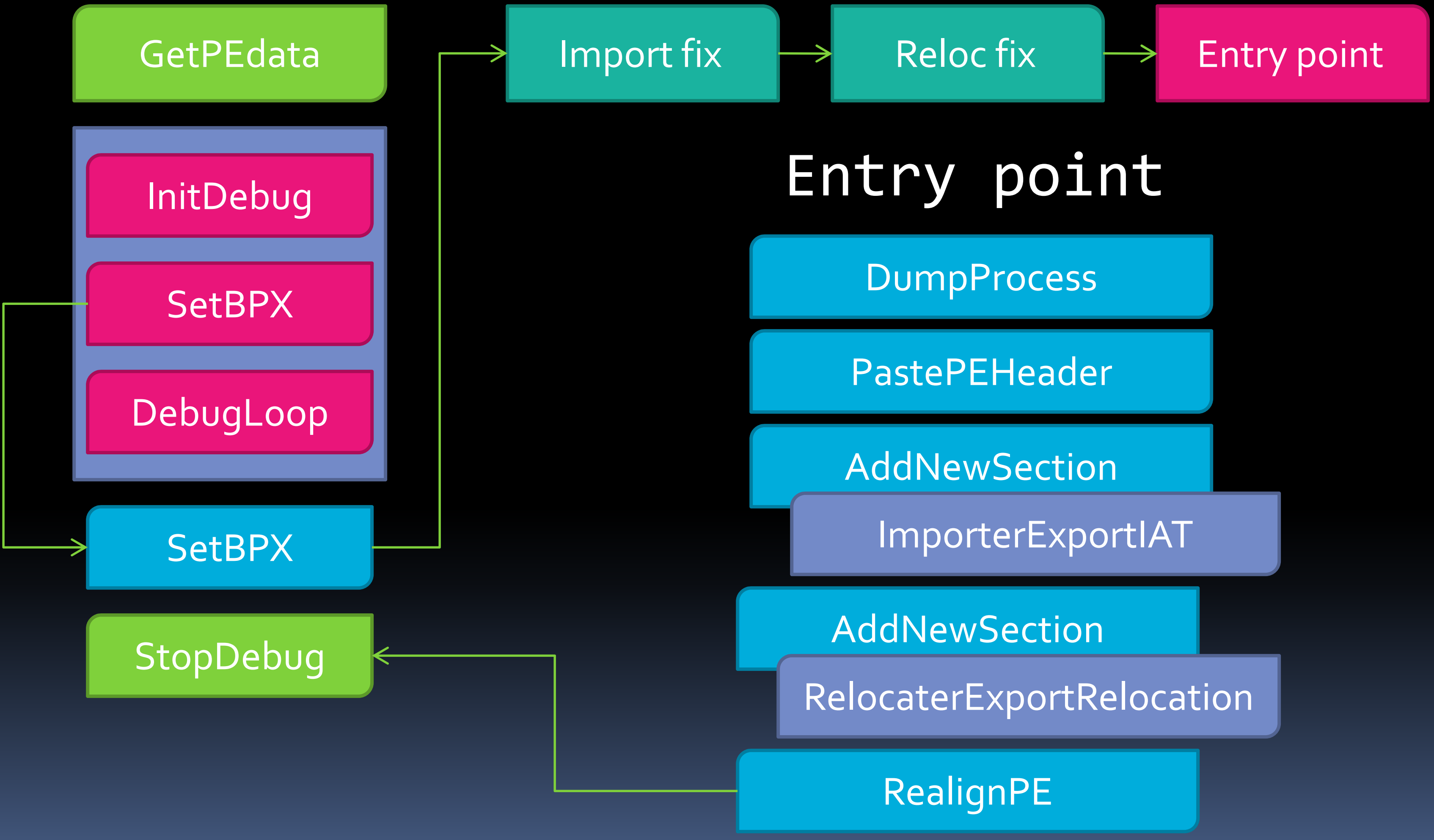
TitanMist|Unpacker coding



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
```

REVERSING LABS | Reverse Engineering & Software Protection

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
- AlexProtector
- LameCrypt
- DEF

Script unpackers

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

REVERSING LABS | Reverse Engineering & Software Protection

Questions?

Questions?

(What Would You Like to Know)