Relocation Bonus

Attacking the Windows Loader Makes Analysts Switch Careers

Introduction

Introduction

Nick Cano

Introduction

Nick Cano

- 25 years old

Introduction

Nick Cano

- 25 years old
- Senior Security Architect at Cylance

Introduction

Nick Cano

- 25 years old
- Senior Security Architect at Cylance
- Author of *Game Hacking: Developing Autonomous Bots for Online Games*

Introduction

Nick Cano

- 25 years old
- Senior Security Architect at Cylance
- Author of *Game Hacking: Developing Autonomous Bots for Online Games*
- Pluralsight Instructor, *Modern C++ Secure Coding Practices: Const Correctness*

Introduction

Nick Cano

- 25 years old
- Senior Security Architect at Cylance
- Author of *Game Hacking: Developing Autonomous Bots for Online Games*
- Pluralsight Instructor, *Modern C++ Secure Coding Practices: Const Correctness*

Relocation Bonus

Introduction

Nick Cano

- 25 years old
- Senior Security Architect at Cylance
- Author of *Game Hacking: Developing Autonomous Bots for Online Games*
- Pluralsight Instructor, *Modern C++ Secure Coding Practices: Const Correctness*

Relocation Bonus

- A look into the Windows Portable Executable (PE) header and how it can be weaponized to *destroy* parsers, disassemblers, and other tools

Introduction

Nick Cano

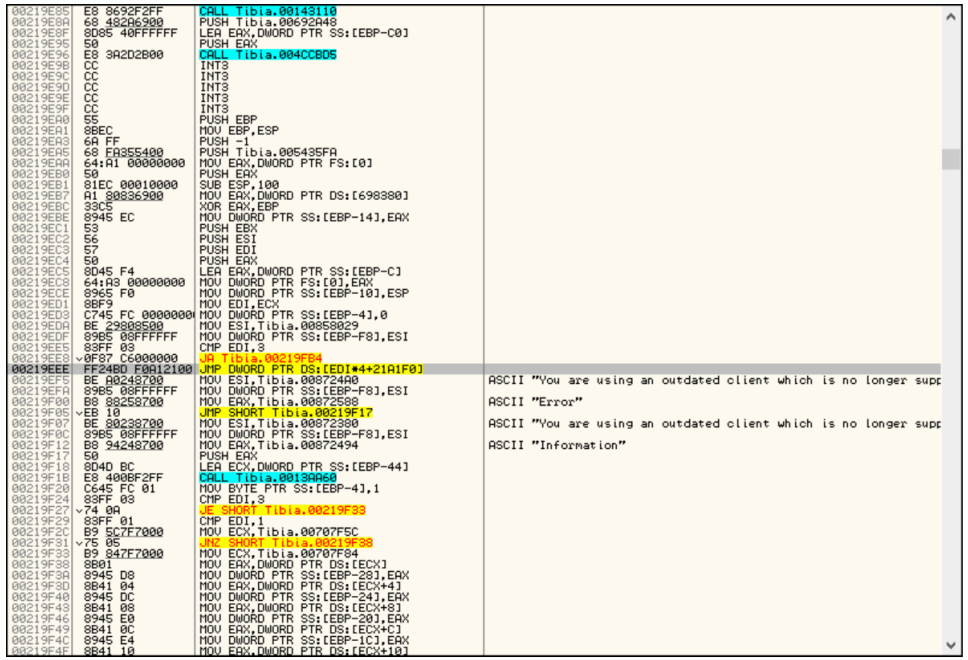
- 25 years old
- Senior Security Architect at Cylance
- Author of *Game Hacking: Developing Autonomous Bots for Online Games*
- Pluralsight Instructor, *Modern C++ Secure Coding Practices: Const Correctness*

Relocation Bonus

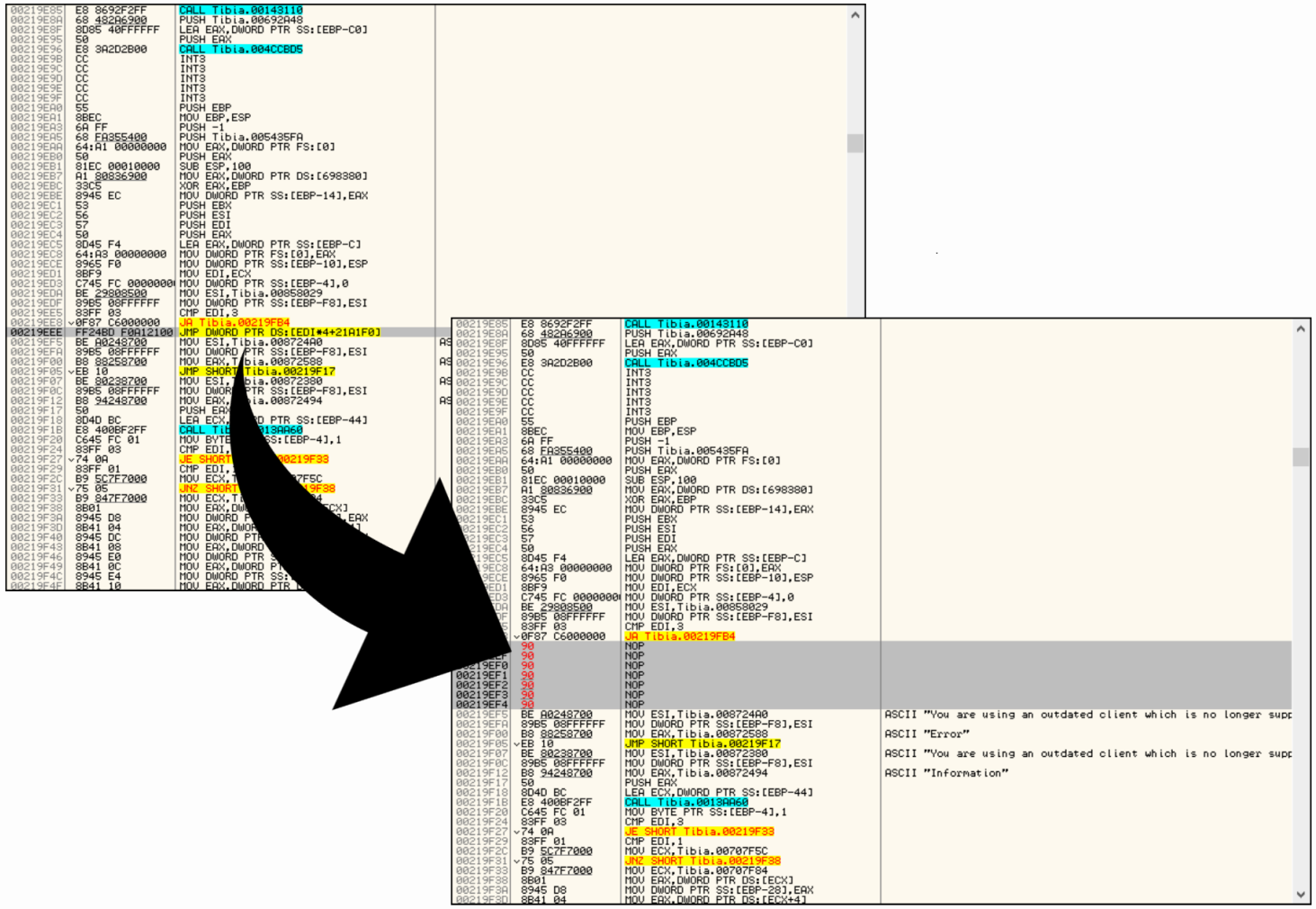
- A look into the Windows Portable Executable (PE) header and how it can be weaponized to *destroy* parsers, disassemblers, and other tools
- A PE rebuilder that takes any 32bit PE then obfuscates and rebuilds it using the attack

Why Attack Relocations?

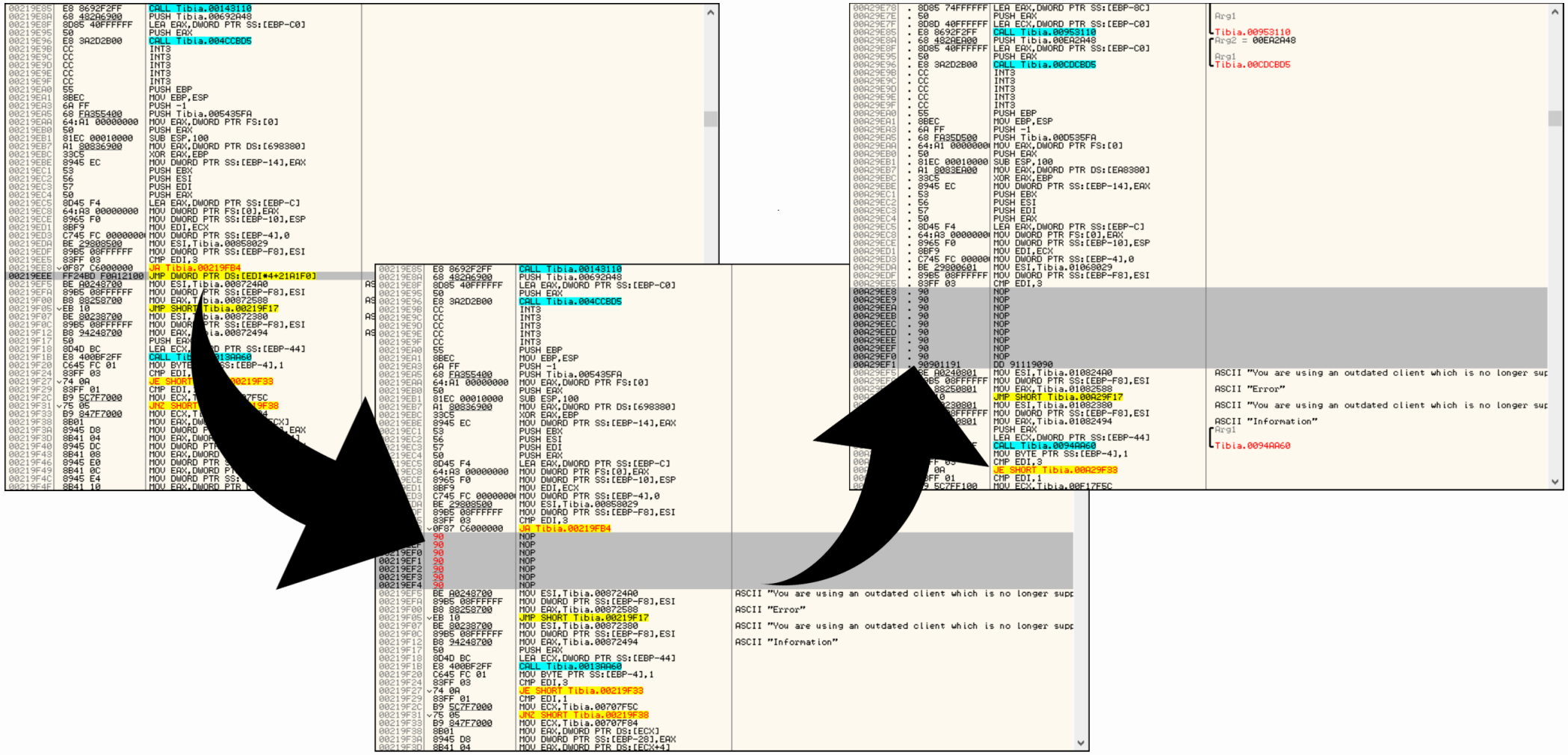
Why Attack Relocations?



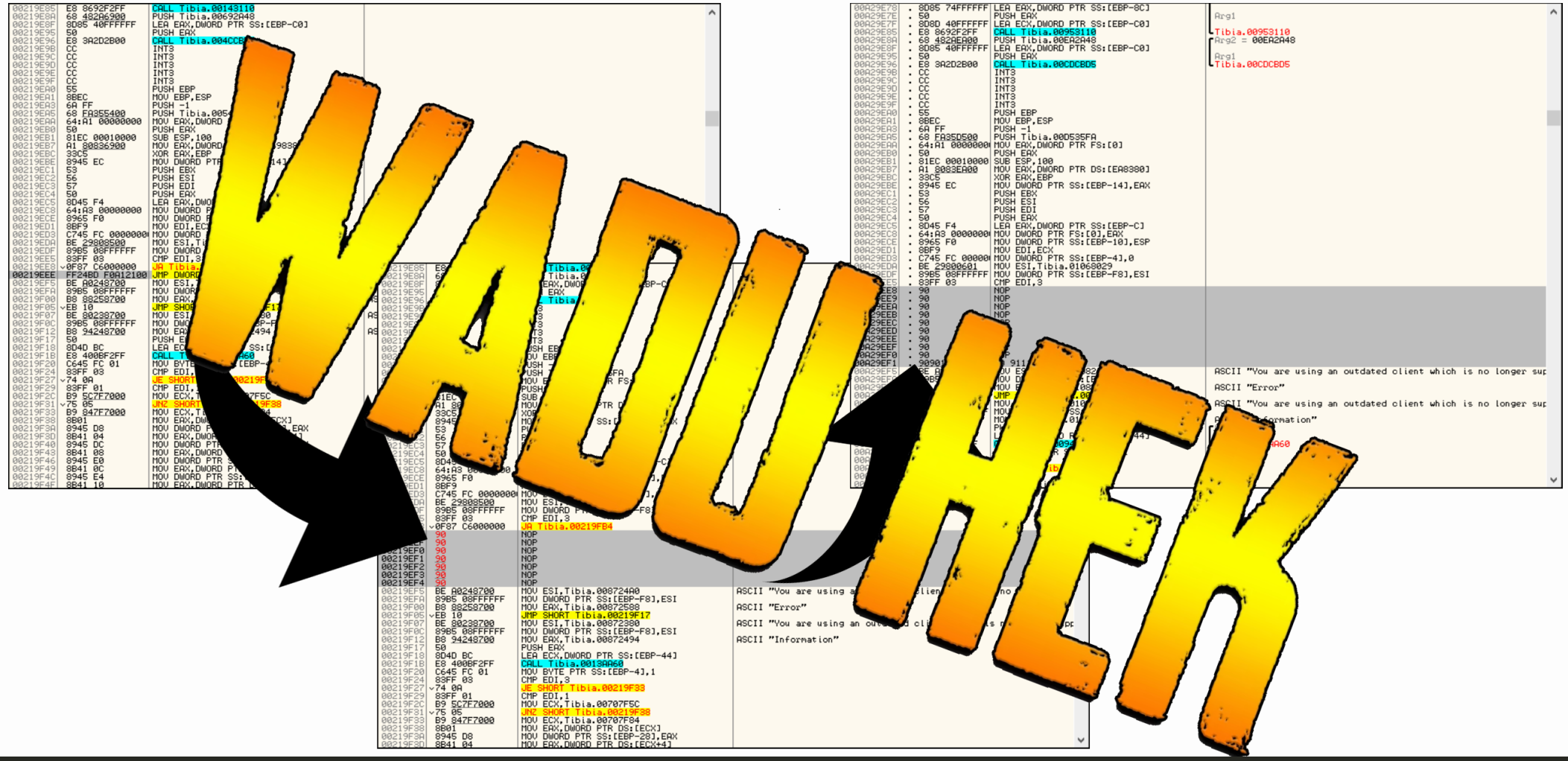
Why Attack Relocations?



Why Attack Relocations?



Why Attack Relocations?



Why Attack Relocations?

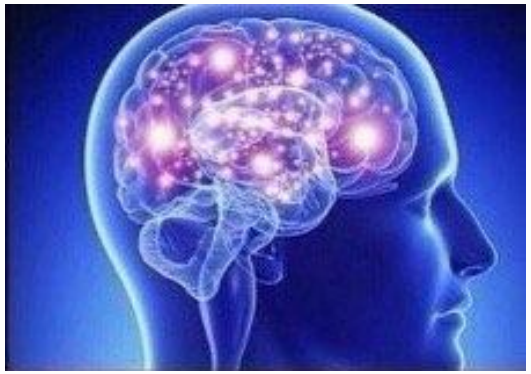
Why Attack Relocations?

its broken for no reason

Why Attack Relocations?

its broken for no reason

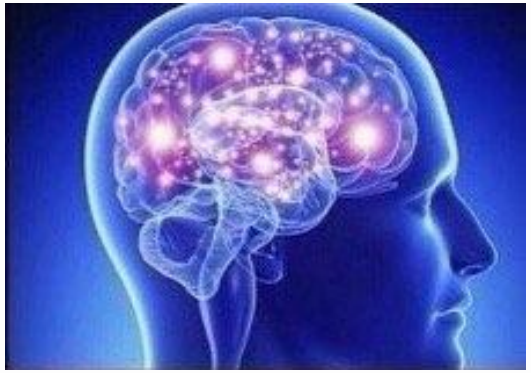
relocations corrupted the patched code



Why Attack Relocations?

its broken for no reason

relocations corrupted the patched code

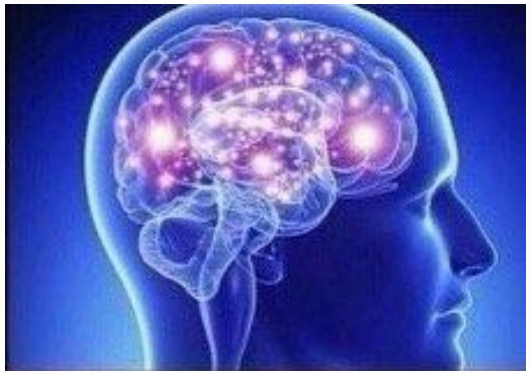


don't patch code that is relocated

Why Attack Relocations?

its broken for no reason

relocations corrupted the patched code



don't patch code that is relocated

relocations can be weaponized to hide my arsenal in the bowels of the machine

Mission Statement

Mission Statement

Me: *slaps roof of exe*
this bad boy can fit so many reloc entries in it

What Are Relocations?

What Are Relocations?

Relocations exist to enable dynamic mapping, specifically ASLR

What Are Relocations?

Relocations exist to enable dynamic mapping, specifically ASLR

0x012E0000 Base

```
0133A00C  . AF9F3301    DD Example.01339FAF
0133A010  . D29F3301    DD Example.01339FD2
0133A014  . 00A03301    DD Example.0133A000
0133A018  . 00A03301    DD Example.0133A000
0133A01C  . 00A03301    DD Example.0133A000
0133A020  . 00A03301    DD Example.0133A000
0133A024  . 00A03301    DD Example.0133A000
0133A028  . 00A03301    DD Example.0133A000
0133A02C  . 00A03301    DD Example.0133A000
0133A030  . 00A03301    DD Example.0133A000
0133A034  . 00A03301    DD Example.0133A000
0133A038  . 00A03301    DD Example.0133A000
0133A03C  . 00A03301    DD Example.0133A000
0133A040  . 00A03301    DD Example.0133A000
0133A044  . 00A03301    DD Example.0133A000
0133A048  . 00A03301    DD Example.0133A000
0133A04C  . 00A03301    DD Example.0133A000
0133A050  . 00A03301    DD Example.0133A000
0133A054  . 00A03301    DD Example.0133A000
0133A058  . 00A03301    DD Example.0133A000
```

0x01120000 Base

```
0117A00C  . AF9F1701    DD Example.01179FAF
0117A010  . D29F1701    DD Example.01179FD2
0117A014  . 00A01701    DD Example.0117A000
0117A018  . 00A01701    DD Example.0117A000
0117A01C  . 00A01701    DD Example.0117A000
0117A020  . 00A01701    DD Example.0117A000
0117A024  . 00A01701    DD Example.0117A000
0117A028  . 00A01701    DD Example.0117A000
0117A02C  . 00A01701    DD Example.0117A000
0117A030  . 00A01701    DD Example.0117A000
0117A034  . 00A01701    DD Example.0117A000
0117A038  . 00A01701    DD Example.0117A000
0117A03C  . 00A01701    DD Example.0117A000
0117A040  . 00A01701    DD Example.0117A000
0117A044  . 00A01701    DD Example.0117A000
0117A048  . 00A01701    DD Example.0117A000
0117A04C  . 00A01701    DD Example.0117A000
0117A050  . 00A01701    DD Example.0117A000
0117A054  . 00A01701    DD Example.0117A000
0117A058  . 00A01701    DD Example.0117A000
```

What Are Relocations?

Relocations exist to enable dynamic mapping, specifically ASLR

0x012E0000 Base

```
0133A00C | . AF9F3301 | DD Example.01339FAF
0133A010 | . D29F3301 | DD Example.01339FD2
0133A014 | . 00A03301 | DD Example.0133A000
0133A018 | . 00A03301 | DD Example.0133A000
0133A01C | . 00A03301 | DD Example.0133A000
0133A020 | . 00A03301 | DD Example.0133A000
0133A024 | . 00A03301 | DD Example.0133A000
0133A028 | . 00A03301 | DD Example.0133A000
0133A02C | . 00A03301 | DD Example.0133A000
0133A030 | . 00A03301 | DD Example.0133A000
0133A034 | . 00A03301 | DD Example.0133A000
0133A038 | . 00A03301 | DD Example.0133A000
0133A03C | . 00A03301 | DD Example.0133A000
0133A040 | . 00A03301 | DD Example.0133A000
0133A044 | . 00A03301 | DD Example.0133A000
0133A048 | . 00A03301 | DD Example.0133A000
0133A04C | . 00A03301 | DD Example.0133A000
0133A050 | . 00A03301 | DD Example.0133A000
0133A054 | . 00A03301 | DD Example.0133A000
0133A058 | . 00A03301 | DD Example.0133A000
```

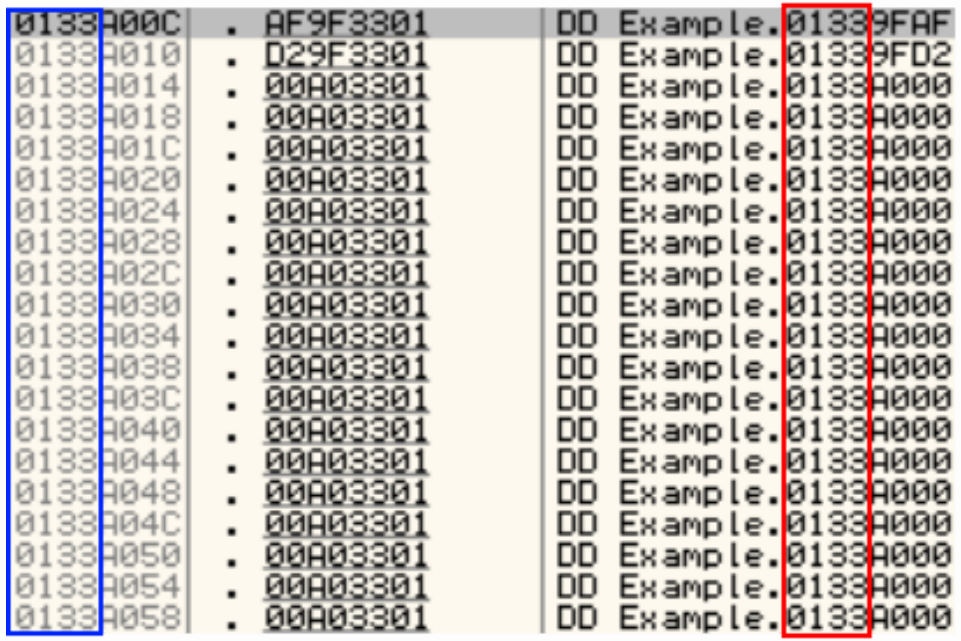
0x01120000 Base

```
0117A00C | . AF9F1701 | DD Example.01179FAF
0117A010 | . D29F1701 | DD Example.01179FD2
0117A014 | . 00A01701 | DD Example.0117A000
0117A018 | . 00A01701 | DD Example.0117A000
0117A01C | . 00A01701 | DD Example.0117A000
0117A020 | . 00A01701 | DD Example.0117A000
0117A024 | . 00A01701 | DD Example.0117A000
0117A028 | . 00A01701 | DD Example.0117A000
0117A02C | . 00A01701 | DD Example.0117A000
0117A030 | . 00A01701 | DD Example.0117A000
0117A034 | . 00A01701 | DD Example.0117A000
0117A038 | . 00A01701 | DD Example.0117A000
0117A03C | . 00A01701 | DD Example.0117A000
0117A040 | . 00A01701 | DD Example.0117A000
0117A044 | . 00A01701 | DD Example.0117A000
0117A048 | . 00A01701 | DD Example.0117A000
0117A04C | . 00A01701 | DD Example.0117A000
0117A050 | . 00A01701 | DD Example.0117A000
0117A054 | . 00A01701 | DD Example.0117A000
0117A058 | . 00A01701 | DD Example.0117A000
```

What Are Relocations?

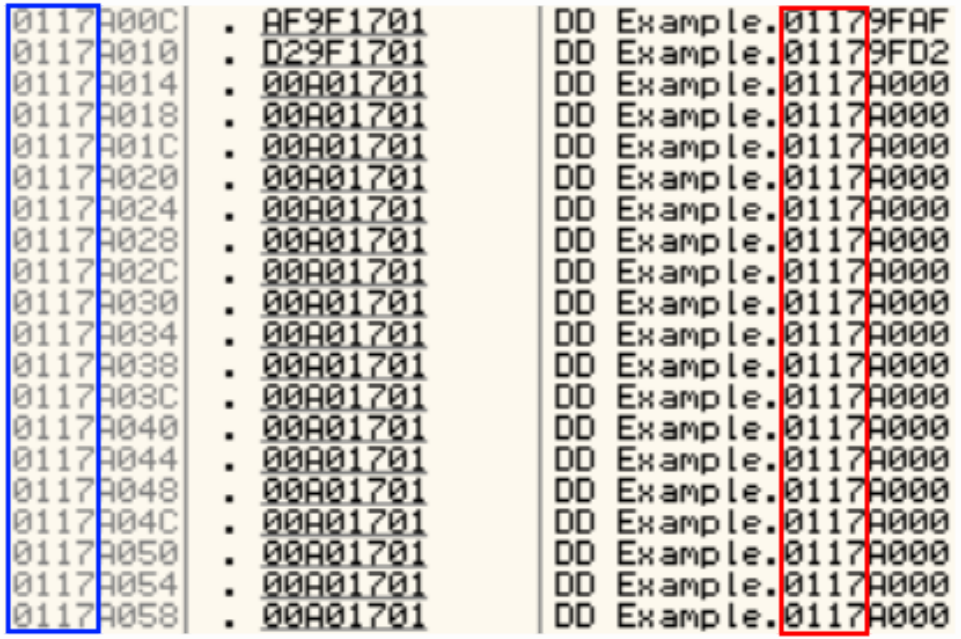
Relocations exist to enable dynamic mapping, specifically ASLR

0x012E0000 Base



```
0133A00C  .  AF9F3301    DD Example.01339FAF
0133A010  .  D29F3301    DD Example.01339FD2
0133A014  .  00A03301    DD Example.0133A000
0133A018  .  00A03301    DD Example.0133A000
0133A01C  .  00A03301    DD Example.0133A000
0133A020  .  00A03301    DD Example.0133A000
0133A024  .  00A03301    DD Example.0133A000
0133A028  .  00A03301    DD Example.0133A000
0133A02C  .  00A03301    DD Example.0133A000
0133A030  .  00A03301    DD Example.0133A000
0133A034  .  00A03301    DD Example.0133A000
0133A038  .  00A03301    DD Example.0133A000
0133A03C  .  00A03301    DD Example.0133A000
0133A040  .  00A03301    DD Example.0133A000
0133A044  .  00A03301    DD Example.0133A000
0133A048  .  00A03301    DD Example.0133A000
0133A04C  .  00A03301    DD Example.0133A000
0133A050  .  00A03301    DD Example.0133A000
0133A054  .  00A03301    DD Example.0133A000
0133A058  .  00A03301    DD Example.0133A000
```

0x01120000 Base

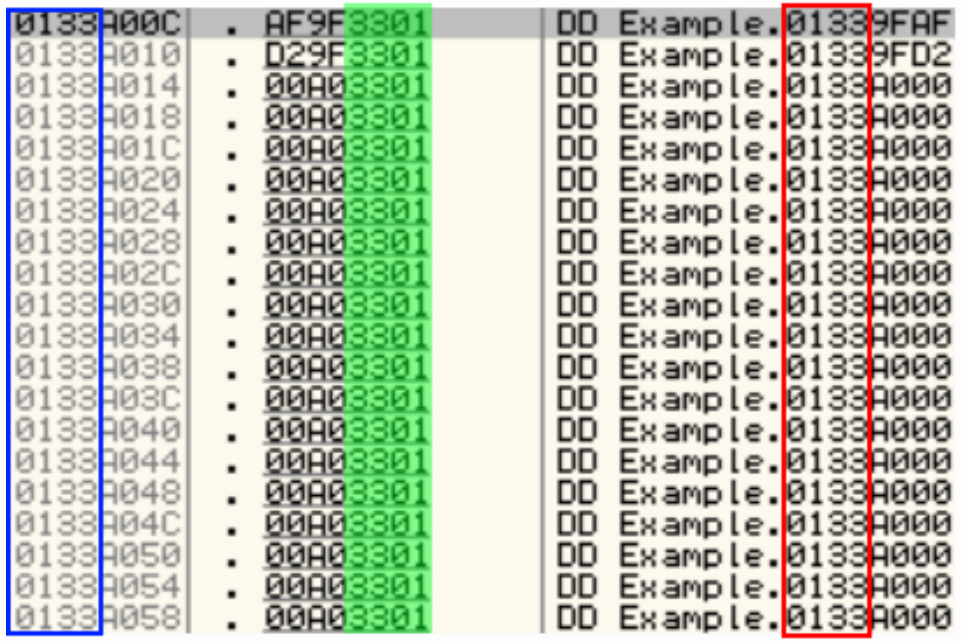


```
0117A00C  .  AF9F1701    DD Example.01179FAF
0117A010  .  D29F1701    DD Example.01179FD2
0117A014  .  00A01701    DD Example.0117A000
0117A018  .  00A01701    DD Example.0117A000
0117A01C  .  00A01701    DD Example.0117A000
0117A020  .  00A01701    DD Example.0117A000
0117A024  .  00A01701    DD Example.0117A000
0117A028  .  00A01701    DD Example.0117A000
0117A02C  .  00A01701    DD Example.0117A000
0117A030  .  00A01701    DD Example.0117A000
0117A034  .  00A01701    DD Example.0117A000
0117A038  .  00A01701    DD Example.0117A000
0117A03C  .  00A01701    DD Example.0117A000
0117A040  .  00A01701    DD Example.0117A000
0117A044  .  00A01701    DD Example.0117A000
0117A048  .  00A01701    DD Example.0117A000
0117A04C  .  00A01701    DD Example.0117A000
0117A050  .  00A01701    DD Example.0117A000
0117A054  .  00A01701    DD Example.0117A000
0117A058  .  00A01701    DD Example.0117A000
```

What Are Relocations?

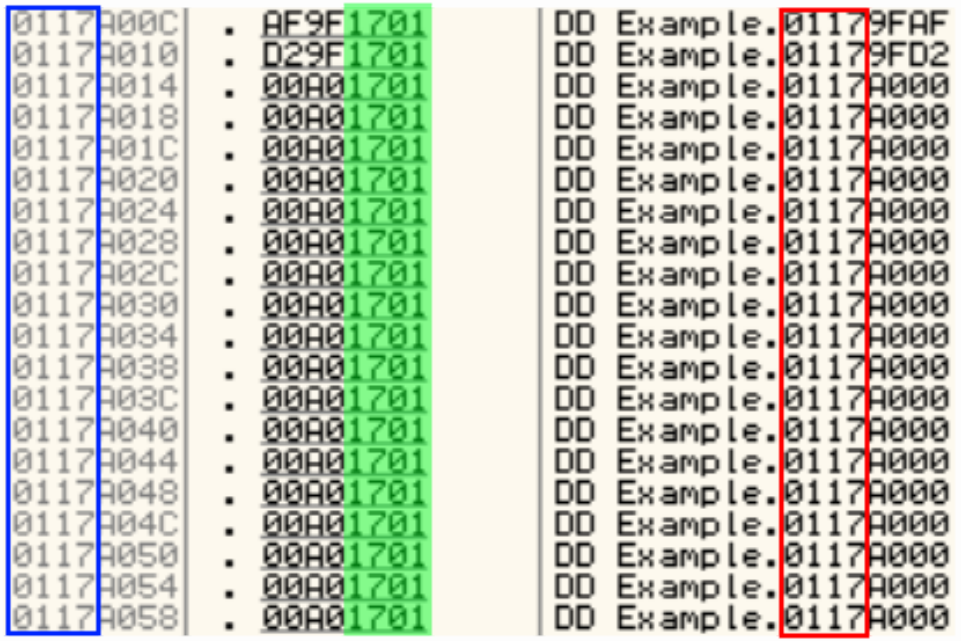
Relocations exist to enable dynamic mapping, specifically ASLR

0x012E0000 Base



```
0133A00C . AF9F3301        DD Example.01339FAF
0133A010 . D29F3301        DD Example.01339FD2
0133A014 . 00A03301        DD Example.0133A000
0133A018 . 00A03301        DD Example.0133A000
0133A01C . 00A03301        DD Example.0133A000
0133A020 . 00A03301        DD Example.0133A000
0133A024 . 00A03301        DD Example.0133A000
0133A028 . 00A03301        DD Example.0133A000
0133A02C . 00A03301        DD Example.0133A000
0133A030 . 00A03301        DD Example.0133A000
0133A034 . 00A03301        DD Example.0133A000
0133A038 . 00A03301        DD Example.0133A000
0133A03C . 00A03301        DD Example.0133A000
0133A040 . 00A03301        DD Example.0133A000
0133A044 . 00A03301        DD Example.0133A000
0133A048 . 00A03301        DD Example.0133A000
0133A04C . 00A03301        DD Example.0133A000
0133A050 . 00A03301        DD Example.0133A000
0133A054 . 00A03301        DD Example.0133A000
0133A058 . 00A03301        DD Example.0133A000
```

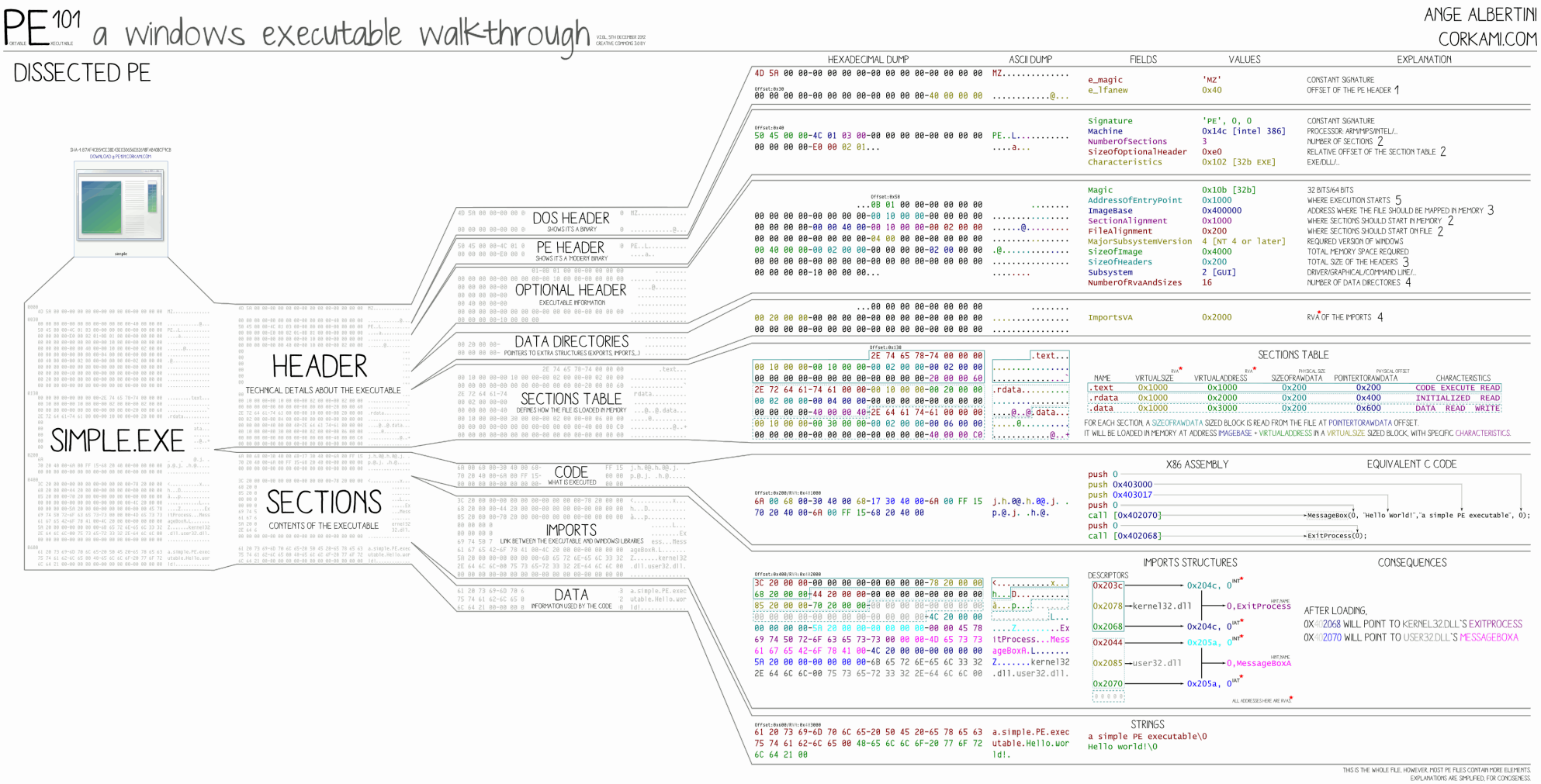
0x01120000 Base



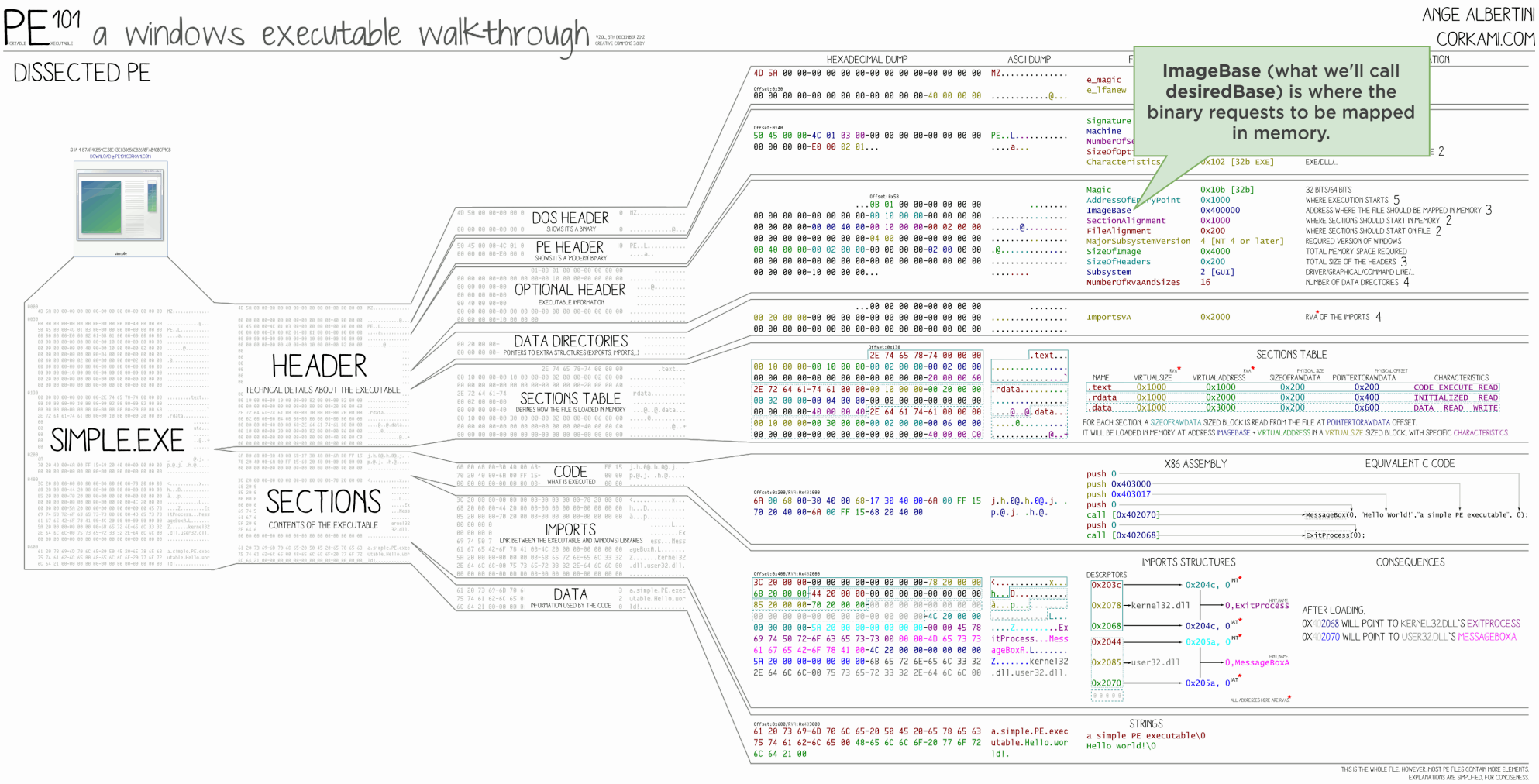
```
0117A00C . AF9F1701        DD Example.01179FAF
0117A010 . D29F1701        DD Example.01179FD2
0117A014 . 00A01701        DD Example.0117A000
0117A018 . 00A01701        DD Example.0117A000
0117A01C . 00A01701        DD Example.0117A000
0117A020 . 00A01701        DD Example.0117A000
0117A024 . 00A01701        DD Example.0117A000
0117A028 . 00A01701        DD Example.0117A000
0117A02C . 00A01701        DD Example.0117A000
0117A030 . 00A01701        DD Example.0117A000
0117A034 . 00A01701        DD Example.0117A000
0117A038 . 00A01701        DD Example.0117A000
0117A03C . 00A01701        DD Example.0117A000
0117A040 . 00A01701        DD Example.0117A000
0117A044 . 00A01701        DD Example.0117A000
0117A048 . 00A01701        DD Example.0117A000
0117A04C . 00A01701        DD Example.0117A000
0117A050 . 00A01701        DD Example.0117A000
0117A054 . 00A01701        DD Example.0117A000
0117A058 . 00A01701        DD Example.0117A000
```

PE Header Sidebar

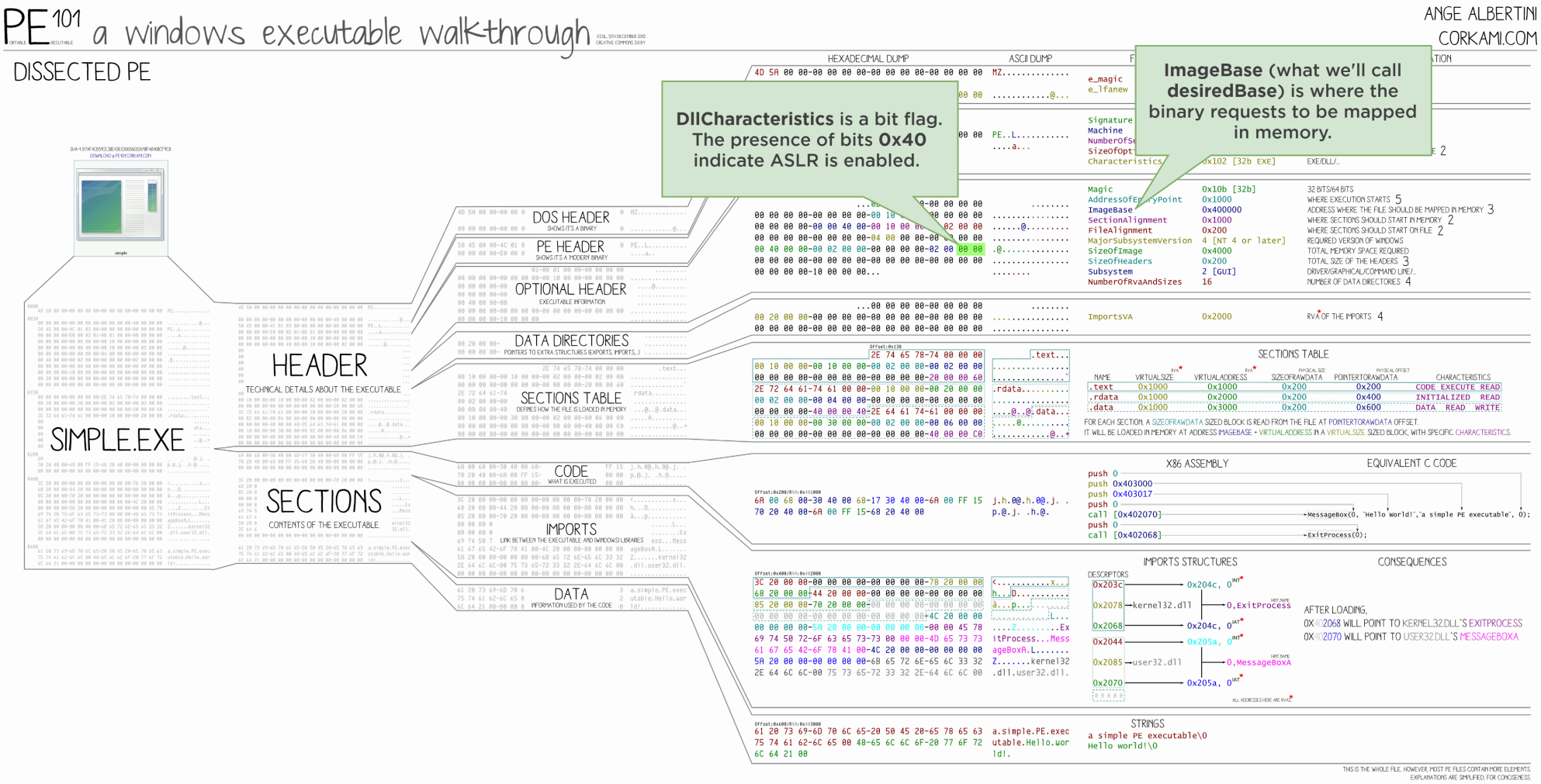
PE Header Sidebar



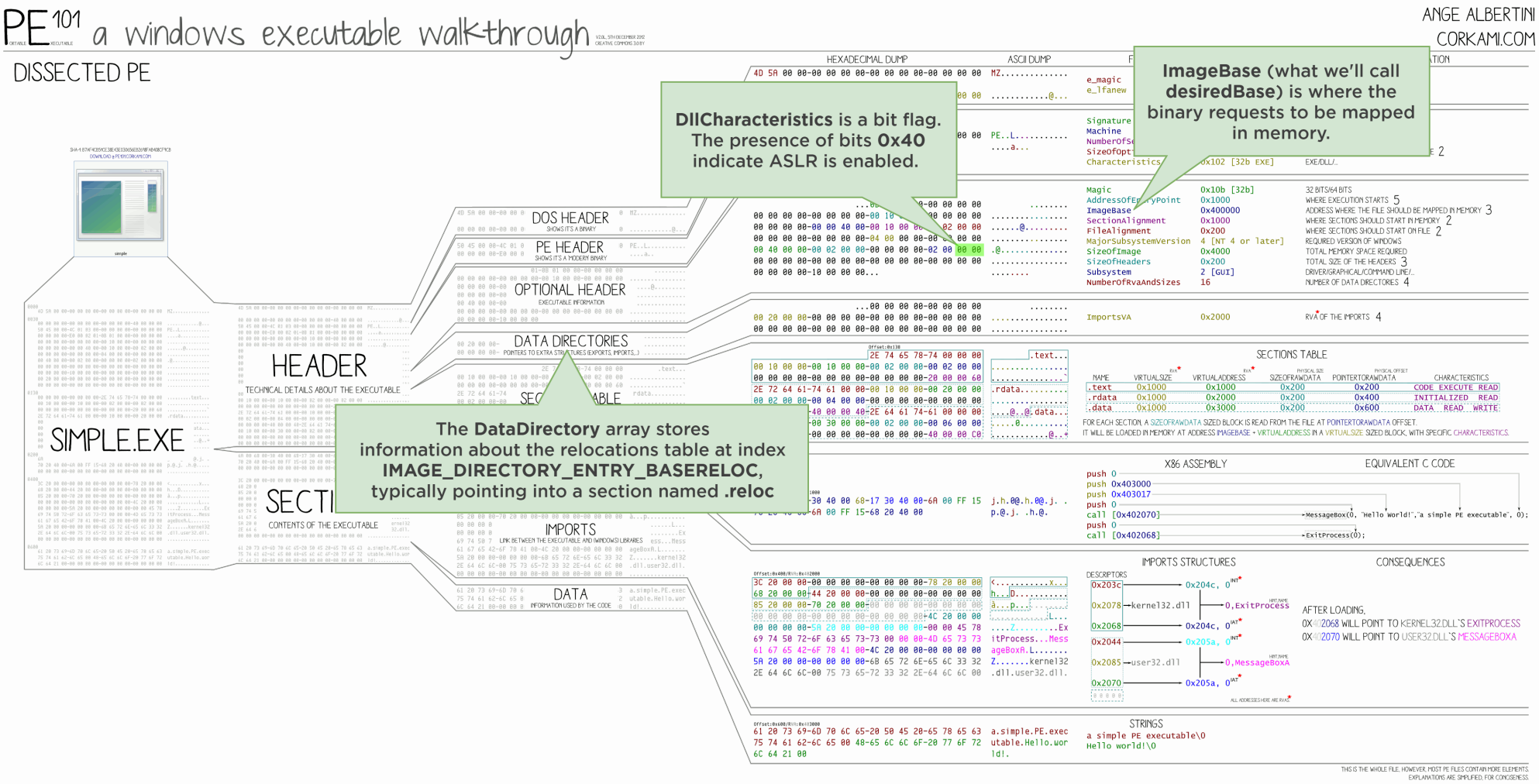
PE Header Sidebar



PE Header Sidebar



PE Header Sidebar



How Do Relocations Work?

How Do Relocations Work?

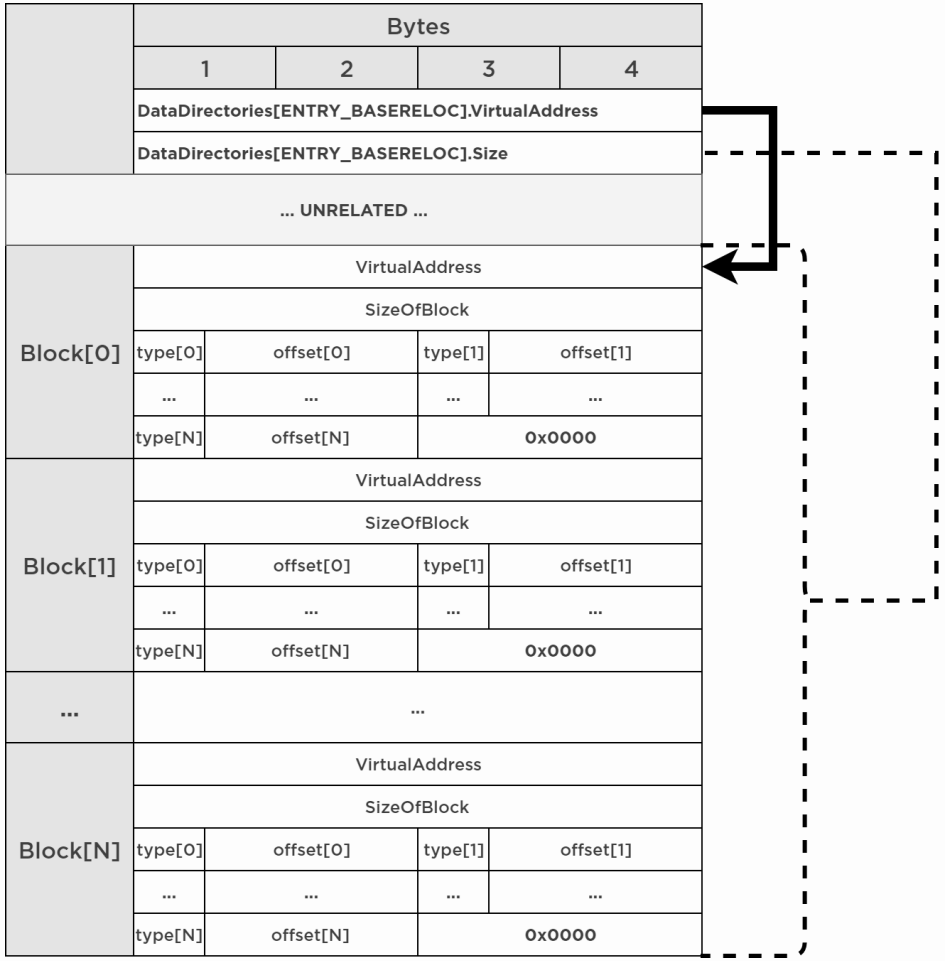
	Bytes			
	1	2	3	4
	DataDirectories[ENTRY_BASERELOC].VirtualAddress			
	DataDirectories[ENTRY_BASERELOC].Size			
... UNRELATED ...				
Block[0]	VirtualAddress			
	SizeOfBlock			
	type[0]	offset[0]	type[1]	offset[1]
	...	...	...	...
	type[N]	offset[N]	0x0000	
Block[1]	VirtualAddress			
	SizeOfBlock			
	type[0]	offset[0]	type[1]	offset[1]
	...	...	...	...
	type[N]	offset[N]	0x0000	
...	...			
Block[N]	VirtualAddress			
	SizeOfBlock			
	type[0]	offset[0]	type[1]	offset[1]
	...	...	...	...
	type[N]	offset[N]	0x0000	

How Do Relocations Work?

	Bytes			
	1	2	3	4
	DataDirectories[ENTRY_BASERELOC].VirtualAddress			
	DataDirectories[ENTRY_BASERELOC].Size			
... UNRELATED ...				
Block[0]	VirtualAddress			
	SizeOfBlock			
	type[0]	offset[0]	type[1]	offset[1]
	...	...	...	...
	type[N]	offset[N]	0x0000	
Block[1]	VirtualAddress			
	SizeOfBlock			
	type[0]	offset[0]	type[1]	offset[1]
	...	...	...	...
	type[N]	offset[N]	0x0000	
...	...			
Block[N]	VirtualAddress			
	SizeOfBlock			
	type[0]	offset[0]	type[1]	offset[1]
	...	...	...	...
	type[N]	offset[N]	0x0000	

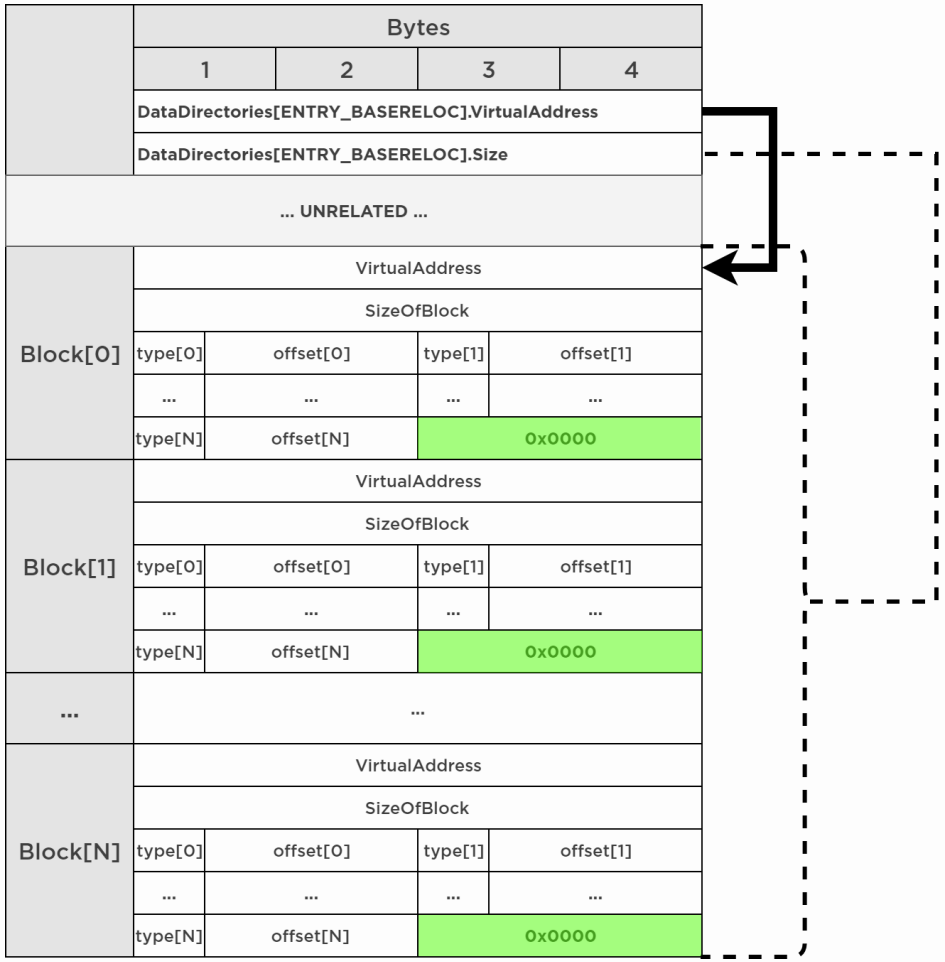
- `VirtualAddress` points to first reloc block

How Do Relocations Work?



- `VirtualAddress` points to first reloc block
- `Size` is the size, in bytes, of all blocks

How Do Relocations Work?



- `VirtualAddress` points to first reloc block
- `Size` is the size, in bytes, of all blocks
- `(uint16_t)0x0000` marks the end of each block

How Do Relocations Work?

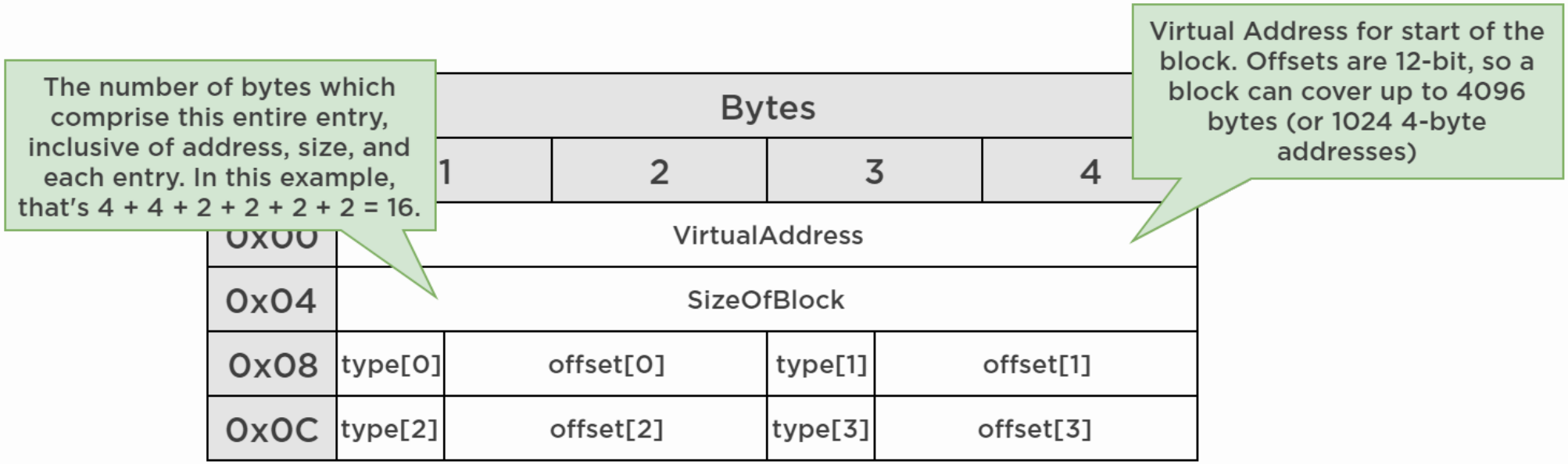
	Bytes			
	1	2	3	4
0x00	VirtualAddress			
0x04	SizeOfBlock			
0x08	type[0]	offset[0]	type[1]	offset[1]
0x0C	type[2]	offset[2]	type[3]	offset[3]

How Do Relocations Work?

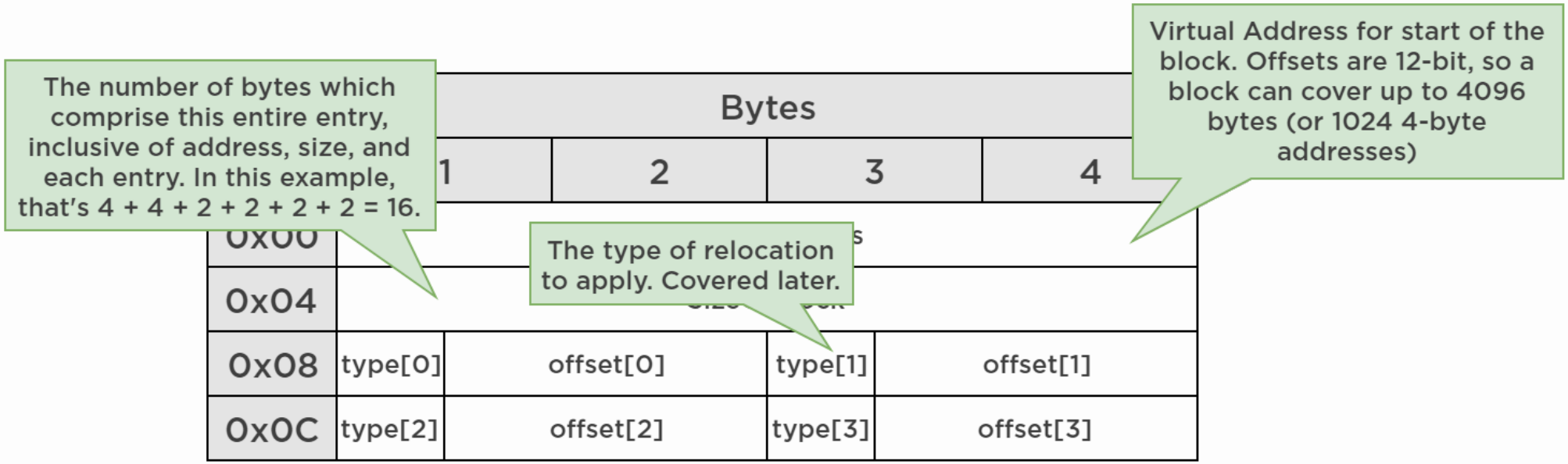
	Bytes			
	1	2	3	4
0x00	VirtualAddress			
0x04	SizeOfBlock			
0x08	type[0]	offset[0]	type[1]	offset[1]
0x0C	type[2]	offset[2]	type[3]	offset[3]

Virtual Address for start of the block. Offsets are 12-bit, so a block can cover up to 4096 bytes (or 1024 4-byte addresses)

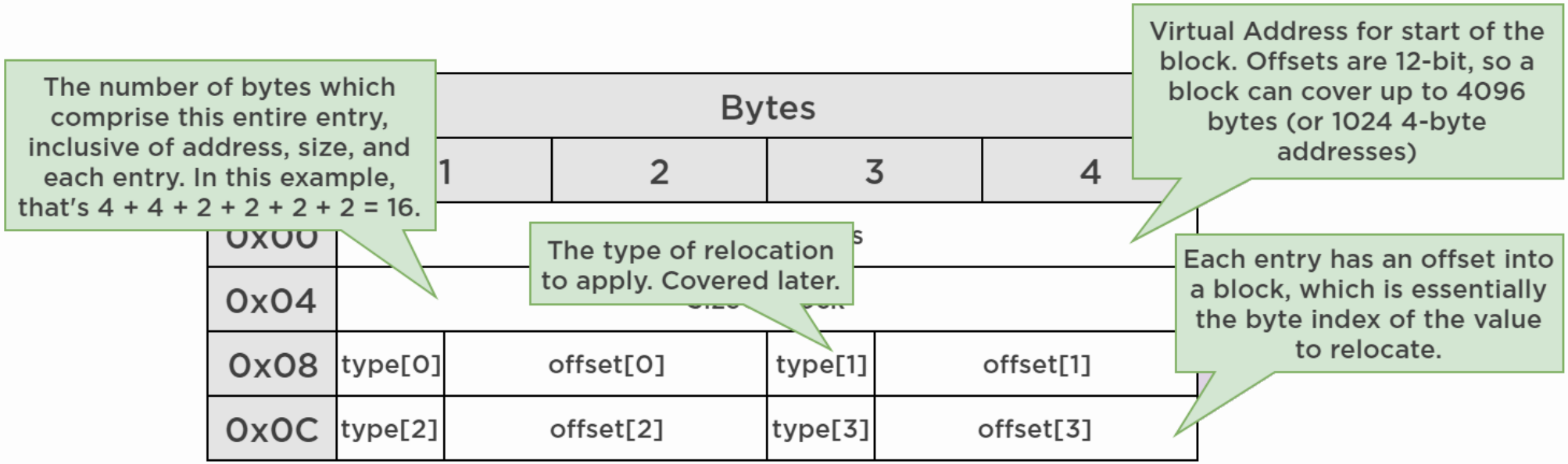
How Do Relocations Work?



How Do Relocations Work?



How Do Relocations Work?



How Do Relocations Work?

How Do Relocations Work?

Something like this

```
auto delta = base - desiredBase;
for (auto reloc : relocs) {
    auto block = (base + reloc.VirtualAddress);
    for (auto entry : reloc.entries) {
        auto adr = block + entry.offset;
        if (entry.type == IMAGE_REL_BASED_HIGHLOW) // <- this
            *((uint32_t*)adr) += delta;
        else if (entry.type == IMAGE_REL_BASED_DIR64)
            *((uint64_t*)adr) += delta;
        else if (entry.type == IMAGE_REL_BASED_HIGH)
            *((uint16_t*)adr) += (uint16_t)((delta >> 16) & 0xFFFF);
        else if (entry.type == IMAGE_REL_BASED_LOW)
            *((uint16_t*)adr) += (uint16_t)delta;
    }
}
```

Controlling Relocations

Controlling Relocations

- Relocations simply use a `+=` operation on data at a specified address

Controlling Relocations

- Relocations simply use a `+=` operation on data at a specified address
- The right-hand side of this operation will be `delta`

Controlling Relocations

- Relocations simply use a `+=` operation on data at a specified address
- The right-hand side of this operation will be `delta`
- `delta` is defined as `base - desiredBase`

Controlling Relocations

- Relocations simply use a `+=` operation on data at a specified address
- The right-hand side of this operation will be `delta`
- `delta` is defined as `base - desiredBase`

Conclusion: to abuse relocations, `base` must be preselected, giving a predictable `delta`. This means ASLR must be tricked.

ASLR Preselection

ASLR Preselection

- `desiredBase` is the only means of controlling ASLR

ASLR Preselection

- `desiredBase` is the only means of controlling ASLR
- `delta` is dependant on `desiredBase`

ASLR Preselection

- `desiredBase` is the only means of controlling ASLR
- `delta` is dependant on `desiredBase`

Conclusion: because of how `delta` is derived, `desiredBase` must be made to force ASLR mapping at a predetermined address which isn't `desiredBase` itself.

ASLR Preselection

- `desiredBase` is the only means of controlling ASLR
- `delta` is dependant on `desiredBase`

Conclusion: because of how `delta` is derived, `desiredBase` must be made to force ASLR mapping at a predetermined address which isn't `desiredBase` itself.

Knowing this, I tried `desiredBase` as:

ASLR Preselection

- `desiredBase` is the only means of controlling ASLR
- `delta` is dependant on `desiredBase`

Conclusion: because of how `delta` is derived, `desiredBase` must be made to force ASLR mapping at a predetermined address which isn't `desiredBase` itself.

Knowing this, I tried `desiredBase` as:

- `0xFFFFFFFF`

ASLR Preselection

- `desiredBase` is the only means of controlling ASLR
- `delta` is dependant on `desiredBase`

Conclusion: because of how `delta` is derived, `desiredBase` must be made to force ASLR mapping at a predetermined address which isn't `desiredBase` itself.

Knowing this, I tried `desiredBase` as:

- `0xFFFFFFFF`: PE fails to load; invalid header

ASLR Preselection

- `desiredBase` is the only means of controlling ASLR
- `delta` is dependant on `desiredBase`

Conclusion: because of how `delta` is derived, `desiredBase` must be made to force ASLR mapping at a predetermined address which isn't `desiredBase` itself.

Knowing this, I tried `desiredBase` as:

- `0xFFFFFFFF`: PE fails to load; invalid header
- `0x00000000`

ASLR Preselection

- `desiredBase` is the only means of controlling ASLR
- `delta` is dependant on `desiredBase`

Conclusion: because of how `delta` is derived, `desiredBase` must be made to force ASLR mapping at a predetermined address which isn't `desiredBase` itself.

Knowing this, I tried `desiredBase` as:

- `0xFFFFFFFF`: PE fails to load; invalid header
- `0x00000000`: PE fails to load; invalid header

ASLR Preselection

- `desiredBase` is the only means of controlling ASLR
- `delta` is dependant on `desiredBase`

Conclusion: because of how `delta` is derived, `desiredBase` must be made to force ASLR mapping at a predetermined address which isn't `desiredBase` itself.

Knowing this, I tried `desiredBase` as:

- `0xFFFFFFFF`: PE fails to load; invalid header
- `0x00000000`: PE fails to load; invalid header
- `0xFFFF0000`

ASLR Preselection

- `desiredBase` is the only means of controlling ASLR
- `delta` is dependant on `desiredBase`

Conclusion: because of how `delta` is derived, `desiredBase` must be made to force ASLR mapping at a predetermined address which isn't `desiredBase` itself.

Knowing this, I tried `desiredBase` as:

- `0xFFFFFFFF`: PE fails to load; invalid header
- `0x00000000`: PE fails to load; invalid header
- `0xFFFF0000`: PE loads at base `0x00010000`

ASLR Preselection

- `desiredBase` is the only means of controlling ASLR
- `delta` is dependant on `desiredBase`

Conclusion: because of how `delta` is derived, `desiredBase` must be made to force ASLR mapping at a predetermined address which isn't `desiredBase` itself.

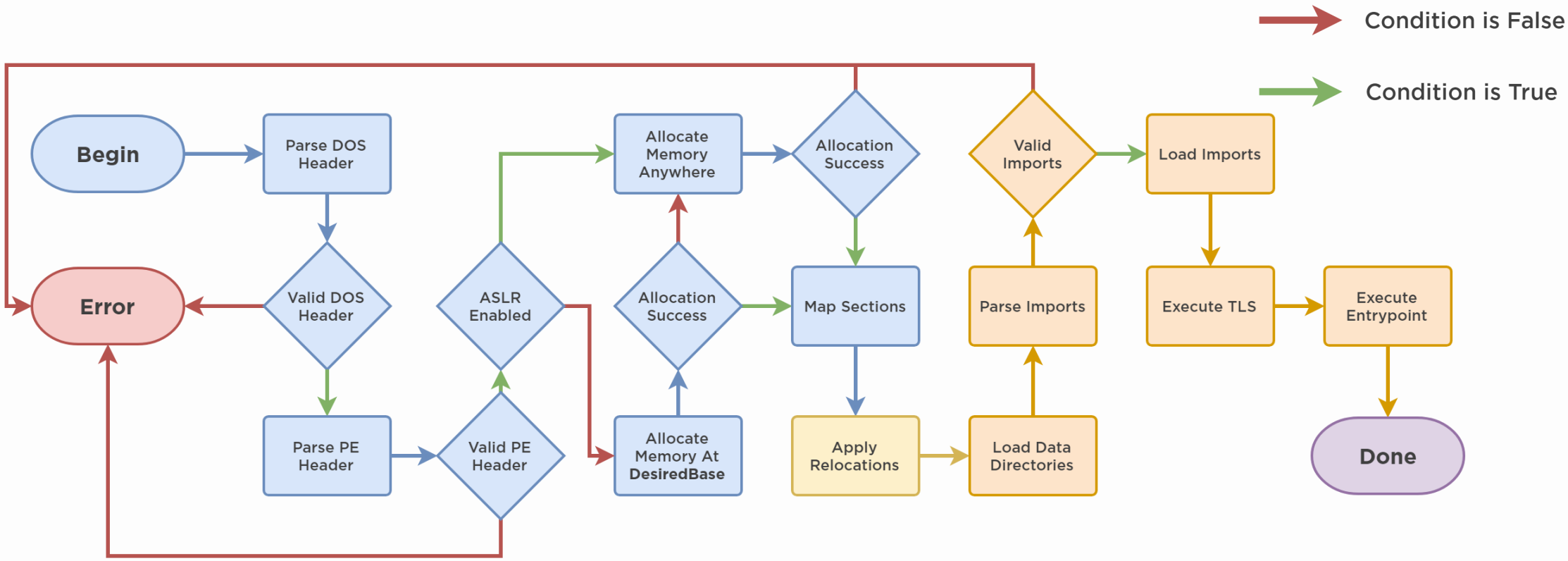
Knowing this, I tried `desiredBase` as:

- `0xFFFFFFFF`: PE fails to load; invalid header
- `0x00000000`: PE fails to load; invalid header
- `0xFFFF0000`: PE loads at base `0x00010000`

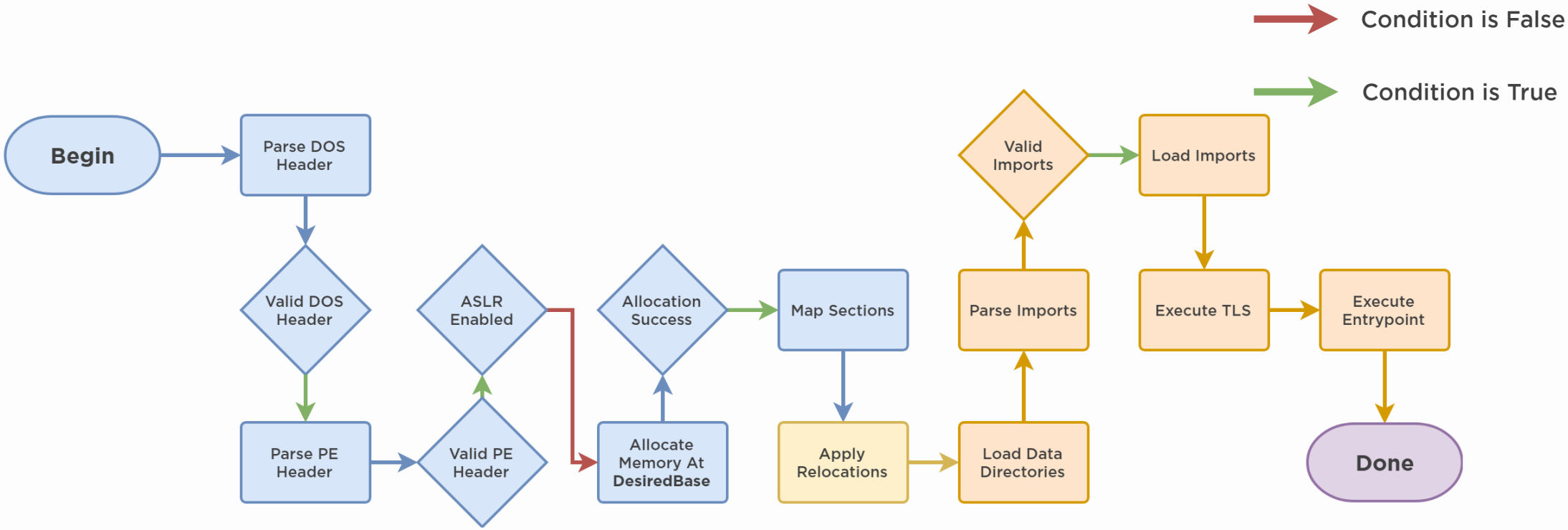
As I later learned, Corkami had already figured all of this out: https://github.com/corkami/pocs/

PE Loader Breakdown

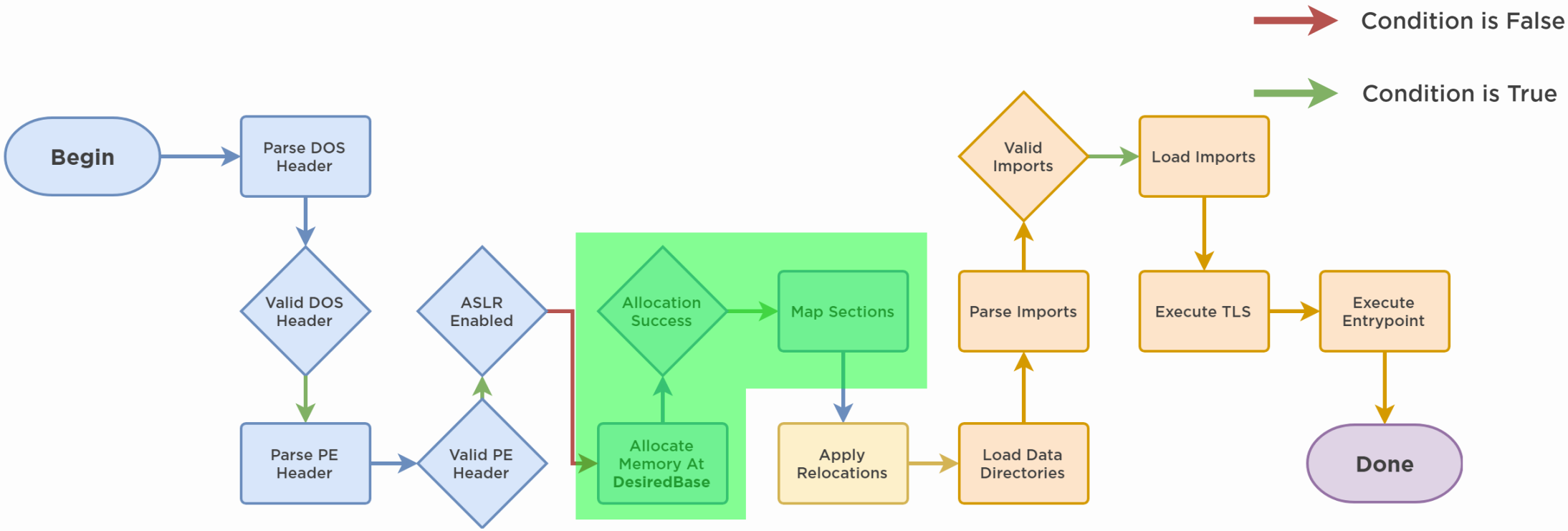
PE Loader Breakdown



PE Loader Breakdown



PE Loader Breakdown



Targets For Relocation Obfuscation

Targets For Relocation Obfuscation

- Import Table is loaded post-reloc

Targets For Relocation Obfuscation

- Import Table is loaded post-reloc
- Many sections are mapped pre-reloc, but not used until execution which is post-reloc

Targets For Relocation Obfuscation

- Import Table is loaded post-reloc
- Many sections are mapped pre-reloc, but not used until execution which is post-reloc
- `entryPoint` isn't used until execution, post-reloc; the memory will be read-only, however, unless DEP is off

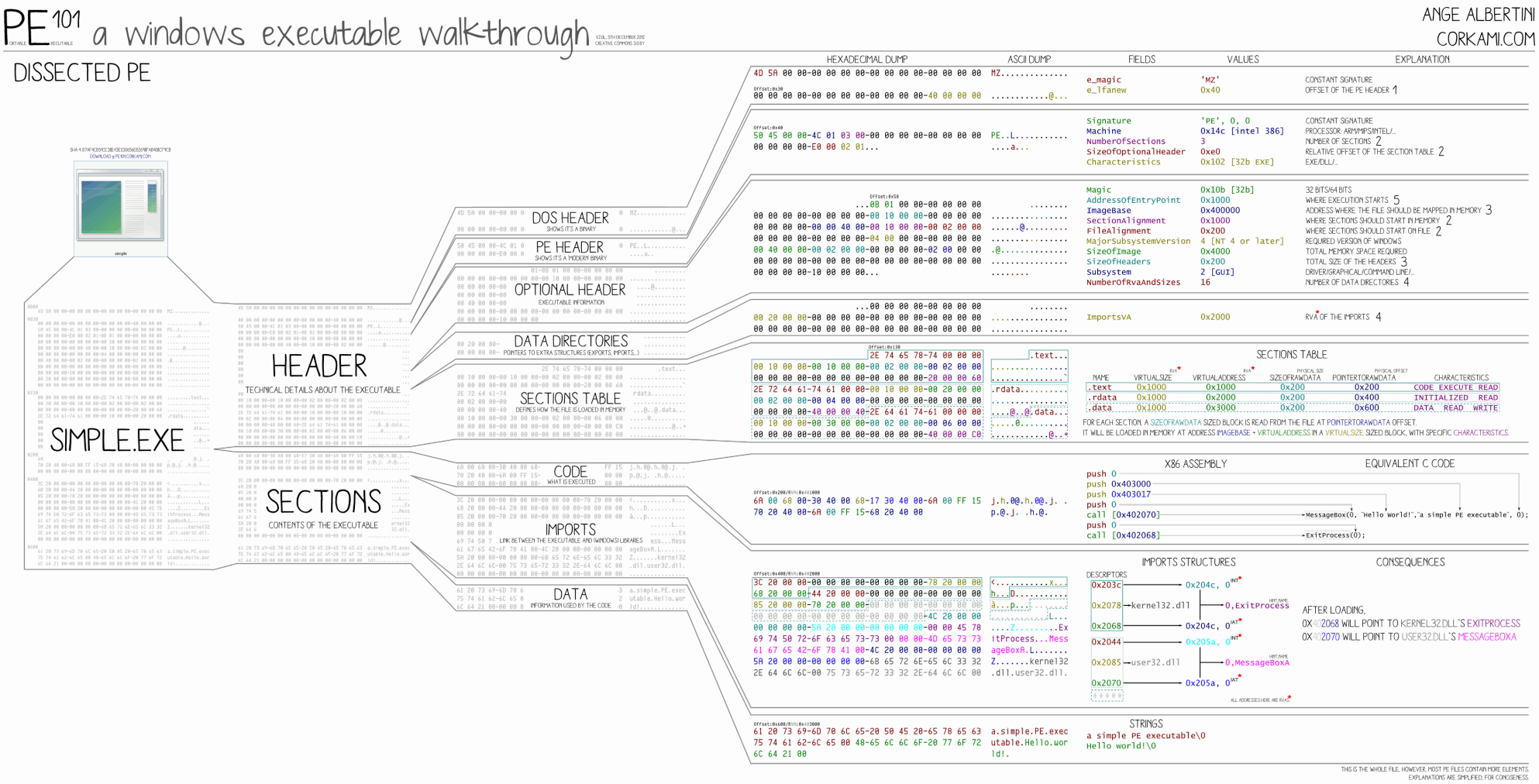
Targets For Relocation Obfuscation

- Import Table is loaded post-reloc
- Many sections are mapped pre-reloc, but not used until execution which is post-reloc
- `entryPoint` isn't used until execution, post-reloc; the memory will be read-only, however, unless DEP is off

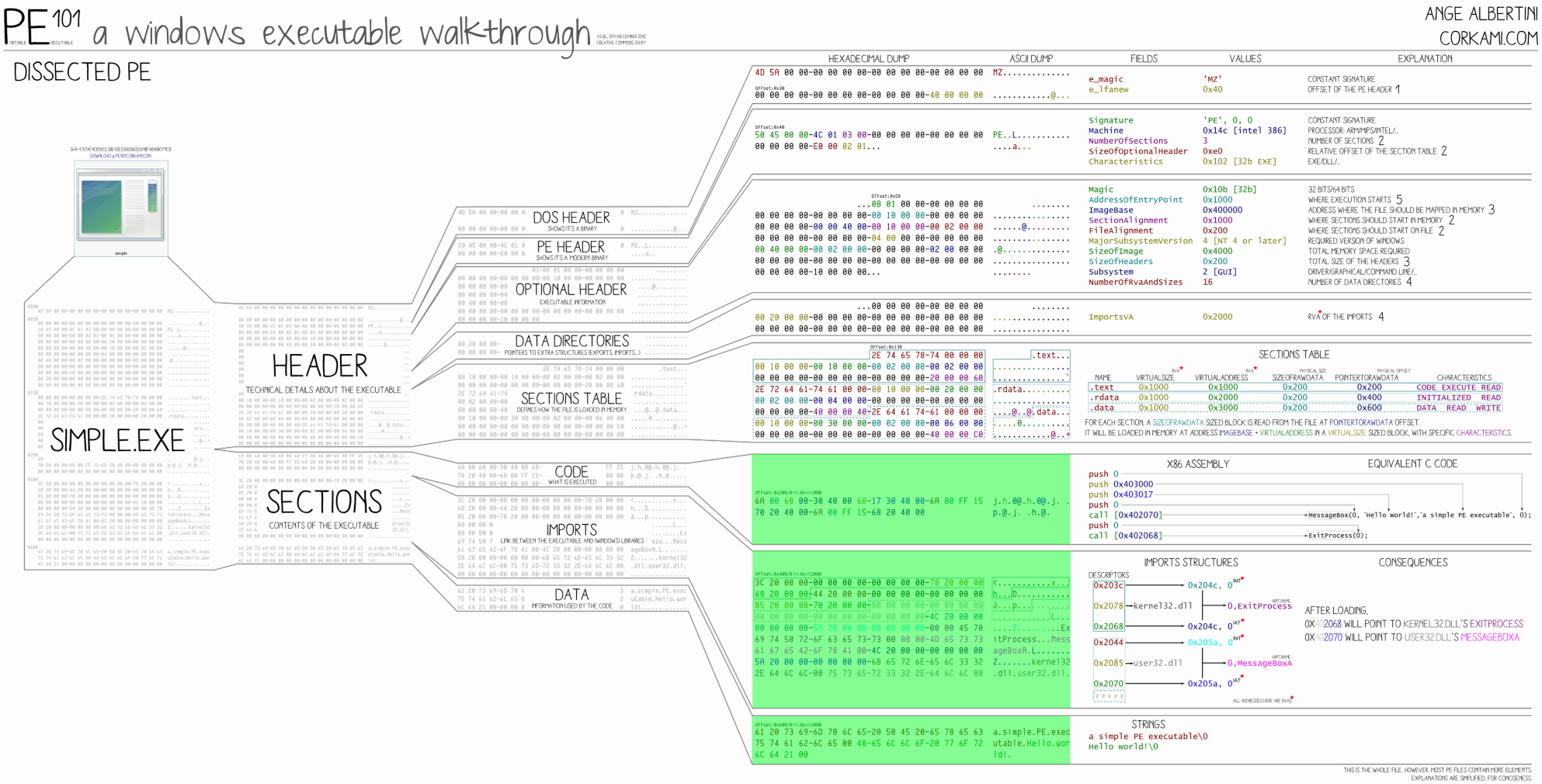
Conclusion: can mangle imports, code and resource sections, and optionally the `entryPoint` if DEP is off on the target machine.

Targets For Relocation Obfuscation

Targets For Relocation Obfuscation



Targets For Relocation Obfuscation



The Final Attack

The Final Attack

Due to the nature of the attack, it works best as a tool which rebuilds regular PE files.

The Final Attack

Due to the nature of the attack, it works best as a tool which rebuilds regular PE files.

- Load target PE file

The Final Attack

Due to the nature of the attack, it works best as a tool which rebuilds regular PE files.

- Load target PE file
- Apply original relocations for `base` of `0x00010000`

The Final Attack

Due to the nature of the attack, it works best as a tool which rebuilds regular PE files.

- Load target PE file
- Apply original relocations for `base` of `0x00010000`
- Turn ASLR off by flipping a bit in the PE Header

The Final Attack

Due to the nature of the attack, it works best as a tool which rebuilds regular PE files.

- Load target PE file
- Apply original relocations for `base` of `0x00010000`
- Turn ASLR off by flipping a bit in the PE Header
- Set `desiredBase` to `0xFFFF0000`

The Final Attack

Due to the nature of the attack, it works best as a tool which rebuilds regular PE files.

- Load target PE file
- Apply original relocations for `base` of `0x00010000`
- Turn ASLR off by flipping a bit in the PE Header
- Set `desiredBase` to `0xFFFF0000`
- Loop over data to obfuscate in `uint32_t`-sized chunks, decrementing each by `0x00010000 - 0xFFFF0000` (expected value of `delta`)

The Final Attack

Due to the nature of the attack, it works best as a tool which rebuilds regular PE files.

- Load target PE file
- Apply original relocations for `base` of `0x00010000`
- Turn ASLR off by flipping a bit in the PE Header
- Set `desiredBase` to `0xFFFF0000`
- Loop over data to obfuscate in `uint32_t`-sized chunks, decrementing each by `0x00010000 - 0xFFFF0000` (expected value of `delta`)
- Discard original relocations table

The Final Attack

Due to the nature of the attack, it works best as a tool which rebuilds regular PE files.

- Load target PE file
- Apply original relocations for `base` of `0x00010000`
- Turn ASLR off by flipping a bit in the PE Header
- Set `desiredBase` to `0xFFFF0000`
- Loop over data to obfuscate in `uint32_t`-sized chunks, decrementing each by `0x00010000 - 0xFFFF0000` (expected value of `delta`)
- Discard original relocations table
- Generate new relocations table containg the location of each decrement done inside of the loop (using `IMAGE_REL_BASED_HIGHLOW`)

The Final Attack

Due to the nature of the attack, it works best as a tool which rebuilds regular PE files.

- Load target PE file
- Apply original relocations for `base` of `0x00010000`
- Turn ASLR off by flipping a bit in the PE Header
- Set `desiredBase` to `0xFFFF0000`
- Loop over data to obfuscate in `uint32_t`-sized chunks, decrementing each by `0x00010000 - 0xFFFF0000` (expected value of `delta`)
- Discard original relocations table
- Generate new relocations table containg the location of each decrement done inside of the loop (using `IMAGE_REL_BASED_HIGHLOW`)
- Save new PE file to disk

The Final Attack

Due to the nature of the attack, it works best as a tool which rebuilds regular PE files.

- Load target PE file
- Apply original relocations for `base` of `0x00010000`
- Turn ASLR off by flipping a bit in the PE Header
- Set `desiredBase` to `0xFFFF0000`
- Loop over data to obfuscate in `uint32_t`-sized chunks, decrementing each by `0x00010000 - 0xFFFF0000` (expected value of `delta`)
- Discard original relocations table
- Generate new relocations table containg the location of each decrement done inside of the loop (using `IMAGE_REL_BASED_HIGHLOW`)
- Save new PE file to disk
- ??? profit

ROUND ONE

FIGHT

Testing The Attack

Testing The Attack

- **Windows 7**

Testing The Attack

- **Windows 7** ... works !

Testing The Attack

- **Windows 7** ... works !
- **Windows 8**

Testing The Attack

- **Windows 7** ... works !
- **Windows 8** ... nobody uses this shit

Testing The Attack

- **Windows 7** ... works !
- **Windows 8** ... nobody uses this shit
- **Windows 10**

Testing The Attack

- **Windows 7** ... works !
- **Windows 8** ... nobody uses this shit
- **Windows 10**

Testing The Attack

- **Windows 7** works
- **Windows 8** nobody uses this shit
- **Windows 10**

THE END

Exploring New Terrain

Exploring New Terrain

- Embed PE copies for all possible base addresses

Exploring New Terrain

- Embed PE copies for all possible base addresses ... way too big

Exploring New Terrain

- Embed PE copies for all possible base addresses ... way too big 🍆
- Tweaking ASLR Configuration

Exploring New Terrain

- Embed PE copies for all possible base addresses ... way too big 🍆
- Tweaking ASLR Configuration ... works!

Exploring New Terrain

- Embed PE copies for all possible base addresses ... way too big 🍆
- Tweaking ASLR Configuration ... works!
  - Set `Mandatory ASLR` to `On`

Exploring New Terrain

- Embed PE copies for all possible base addresses ... way too big 🍆
- Tweaking ASLR Configuration ... works!
  - Set `Mandatory ASLR` to `On`
  - Set `Bottom-Up ASLR` to `Off`

Exploring New Terrain

- Embed PE copies for all possible base addresses ... way too big 🍆
- Tweaking ASLR Configuration ... works!
  - Set `Mandatory ASLR` to `On`
  - Set `Bottom-Up ASLR` to `Off`

```
[HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\
Image File Execution Options\NAME_OF_EXE]
"MitigationAuditOptions"=hex:00,00,00,00,00,00,00,00,\
                            00,00,00,00,00,00,00,00
"MitigationOptions"=hex:00,01,22,00,00,00,00,00,\
                        00,00,00,00,00,00,00,00
```

Exploring New Terrain

- Embed PE copies for all possible base addresses ... way too big 🍆
- Tweaking ASLR Configuration ... works!
  - Set `Mandatory ASLR` to `On`
  - Set `Bottom-Up ASLR` to `Off`

```
[HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\
Image File Execution Options\NAME_OF_EXE]
"MitigationAuditOptions"=hex:00,00,00,00,00,00,00,00,\
                            00,00,00,00,00,00,00,00
"MitigationOptions"=hex:00,01,22,00,00,00,00,00,\
                        00,00,00,00,00,00,00,00
```

Conclusion: it can work on Windows 10

Exploring New Terrain

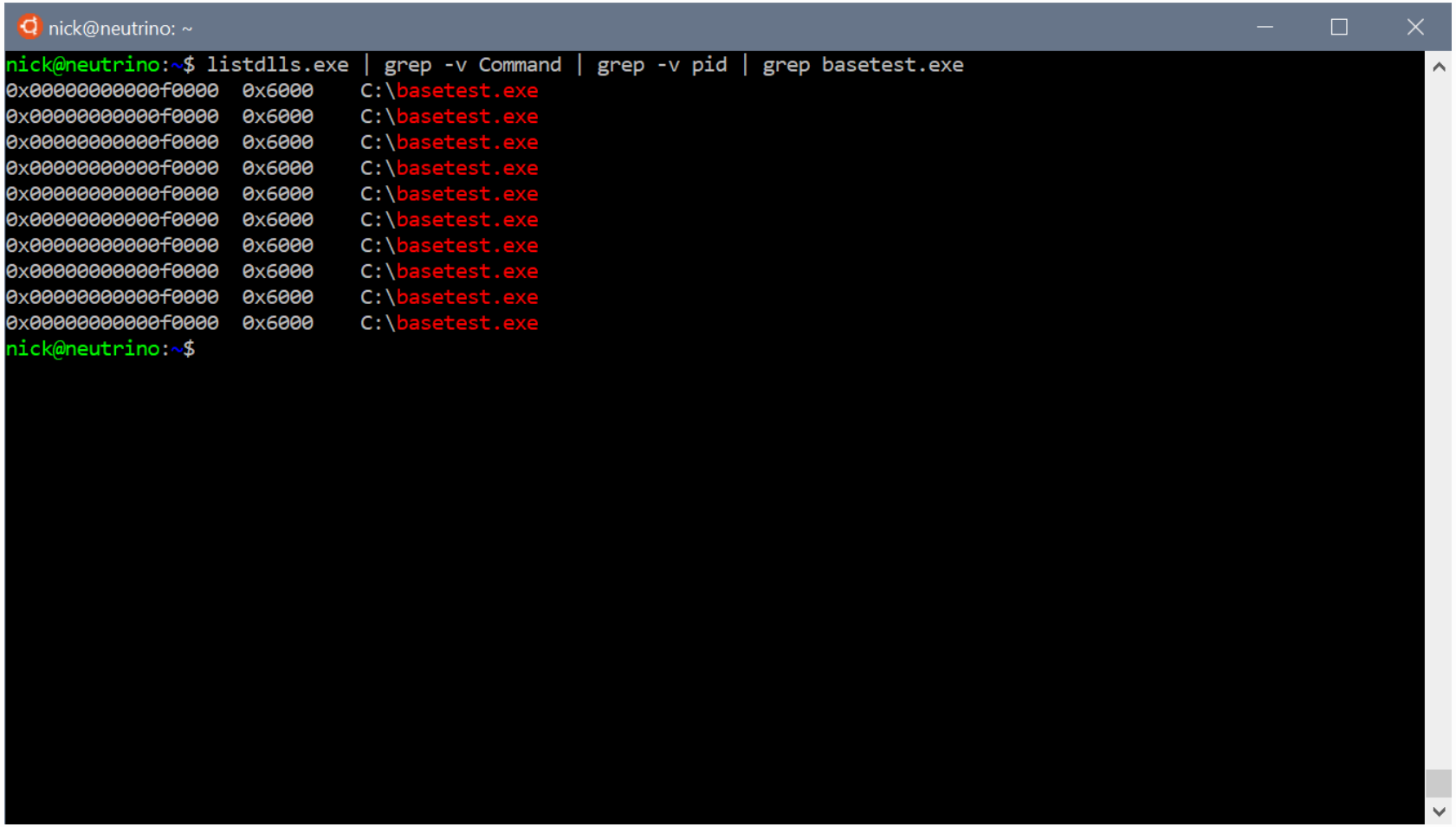
- Embed PE copies for all possible base addresses ... way too big 🍆
- Tweaking ASLR Configuration ... works!
  - Set `Mandatory ASLR` to `On`
  - Set `Bottom-Up ASLR` to `Off`

```
[HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\
Image File Execution Options\NAME_OF_EXE]
"MitigationAuditOptions"=hex:00,00,00,00,00,00,00,00,\
                            00,00,00,00,00,00,00,00
"MitigationOptions"=hex:00,01,22,00,00,00,00,00,\
                        00,00,00,00,00,00,00,00
```

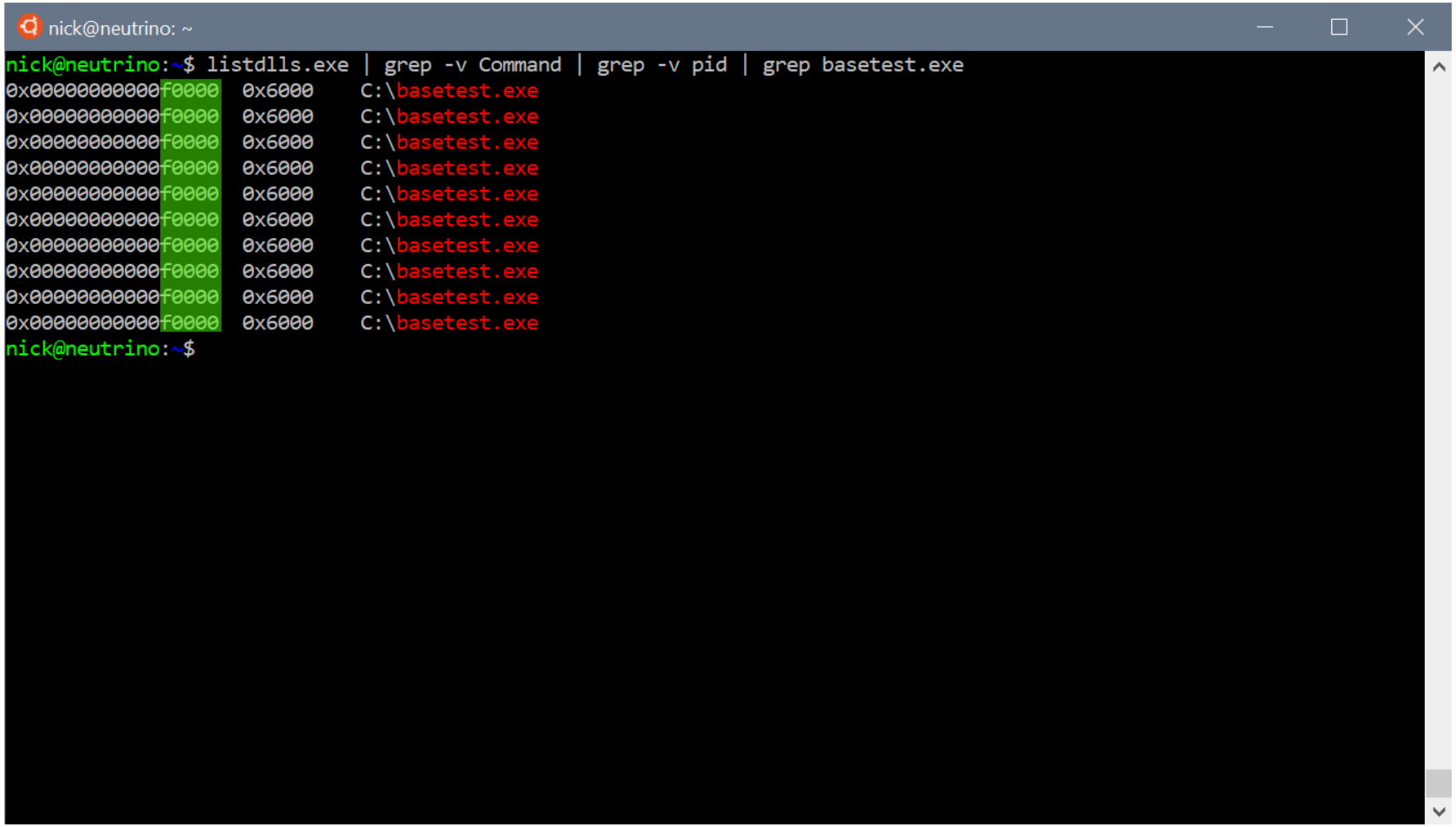
Conclusion: it can work on Windows 10, but I don't like it

A New Hope

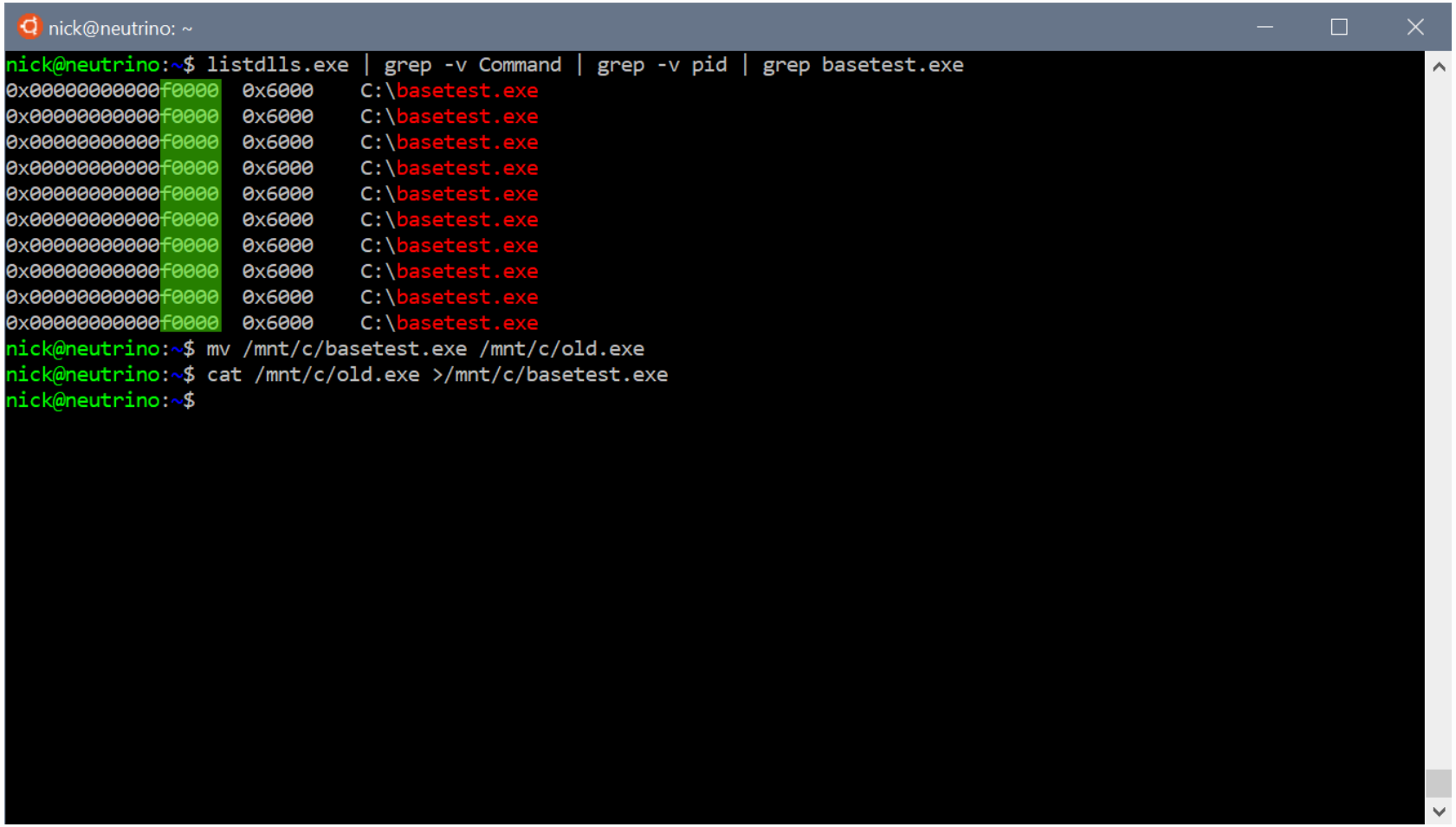
A New Hope


nick@neutrino: ~
nick@neutrino:~$ listdlls.exe | grep -v Command | grep -v pid | grep basetest.exe
0x00000000000f0000 0x6000 C:\basetest.exe
0x00000000000f0000 0x6000 C:\basetest.exe
0x00000000000f0000 0x6000 C:\basetest.exe
0x00000000000f0000 0x6000 C:\basetest.exe
0x00000000000f0000 0x6000 C:\basetest.exe
0x00000000000f0000 0x6000 C:\basetest.exe
0x00000000000f0000 0x6000 C:\basetest.exe
0x00000000000f0000 0x6000 C:\basetest.exe
0x00000000000f0000 0x6000 C:\basetest.exe
0x00000000000f0000 0x6000 C:\basetest.exe
nick@neutrino:~$

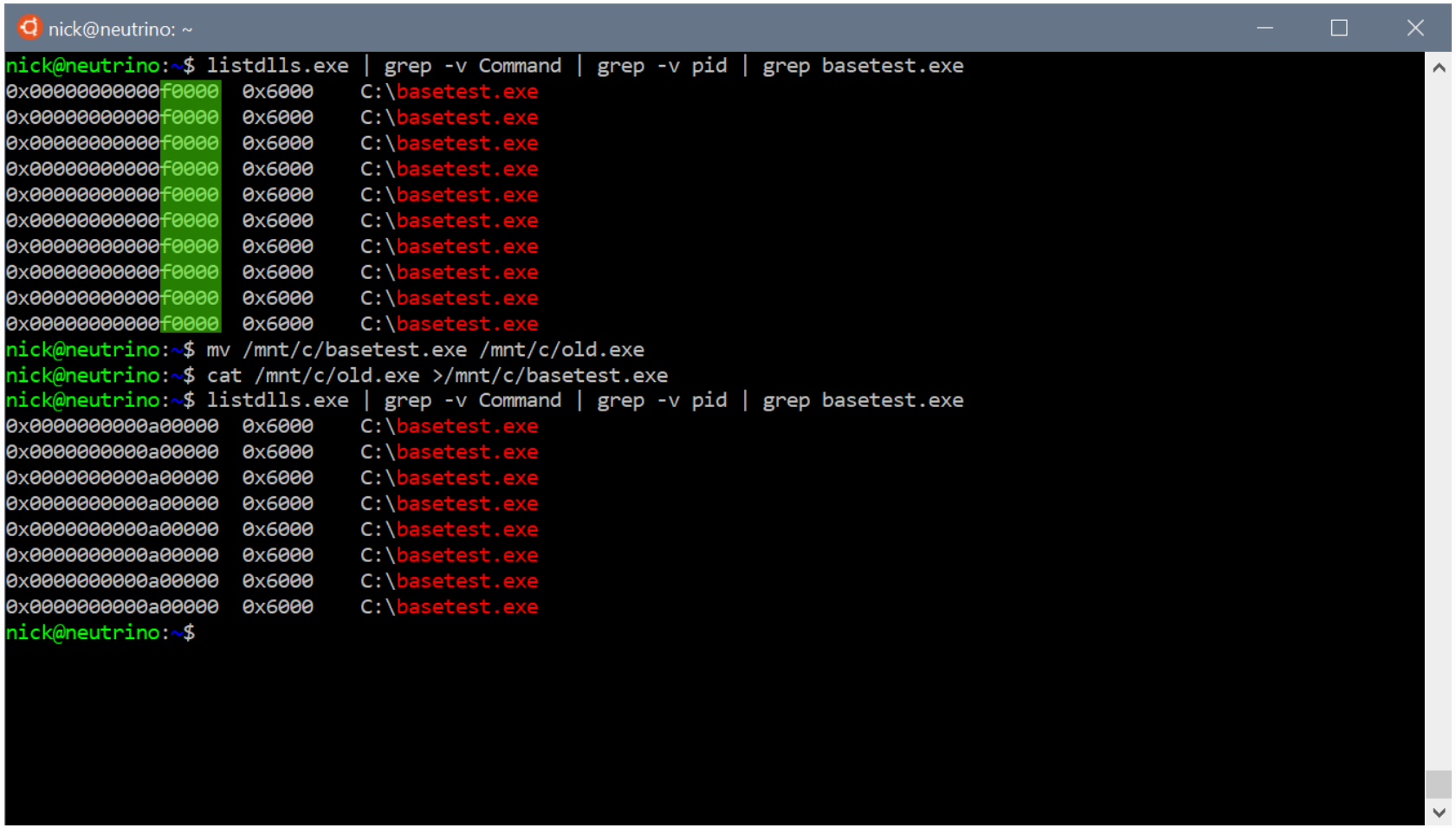
A New Hope


nick@neutrino: ~
nick@neutrino:~$ listdlls.exe | grep -v Command | grep -v pid | grep basetest.exe
0x00000000000f0000 0x6000 C:\basetest.exe
0x00000000000f0000 0x6000 C:\basetest.exe
0x00000000000f0000 0x6000 C:\basetest.exe
0x00000000000f0000 0x6000 C:\basetest.exe
0x00000000000f0000 0x6000 C:\basetest.exe
0x00000000000f0000 0x6000 C:\basetest.exe
0x00000000000f0000 0x6000 C:\basetest.exe
0x00000000000f0000 0x6000 C:\basetest.exe
0x00000000000f0000 0x6000 C:\basetest.exe
0x00000000000f0000 0x6000 C:\basetest.exe
nick@neutrino:~$

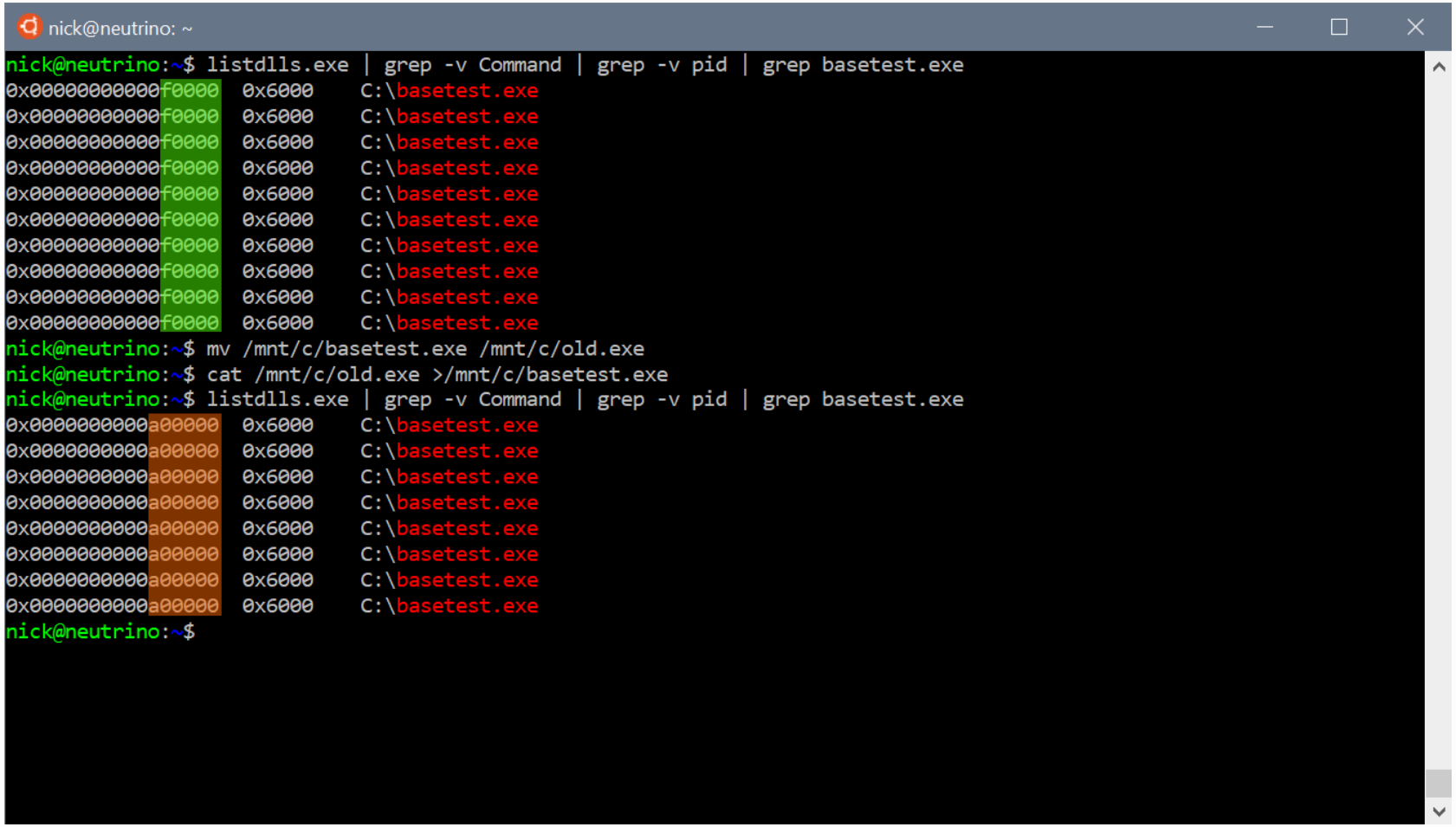
A New Hope


nick@neutrino: ~
nick@neutrino:~$ listdlls.exe | grep -v Command | grep -v pid | grep basetest.exe
0x00000000000f0000 0x6000 C:\basetest.exe
0x00000000000f0000 0x6000 C:\basetest.exe
0x00000000000f0000 0x6000 C:\basetest.exe
0x00000000000f0000 0x6000 C:\basetest.exe
0x00000000000f0000 0x6000 C:\basetest.exe
0x00000000000f0000 0x6000 C:\basetest.exe
0x00000000000f0000 0x6000 C:\basetest.exe
0x00000000000f0000 0x6000 C:\basetest.exe
0x00000000000f0000 0x6000 C:\basetest.exe
0x00000000000f0000 0x6000 C:\basetest.exe
nick@neutrino:~$ mv /mnt/c/basetest.exe /mnt/c/old.exe
nick@neutrino:~$ cat /mnt/c/old.exe >/mnt/c/basetest.exe
nick@neutrino:~$

A New Hope

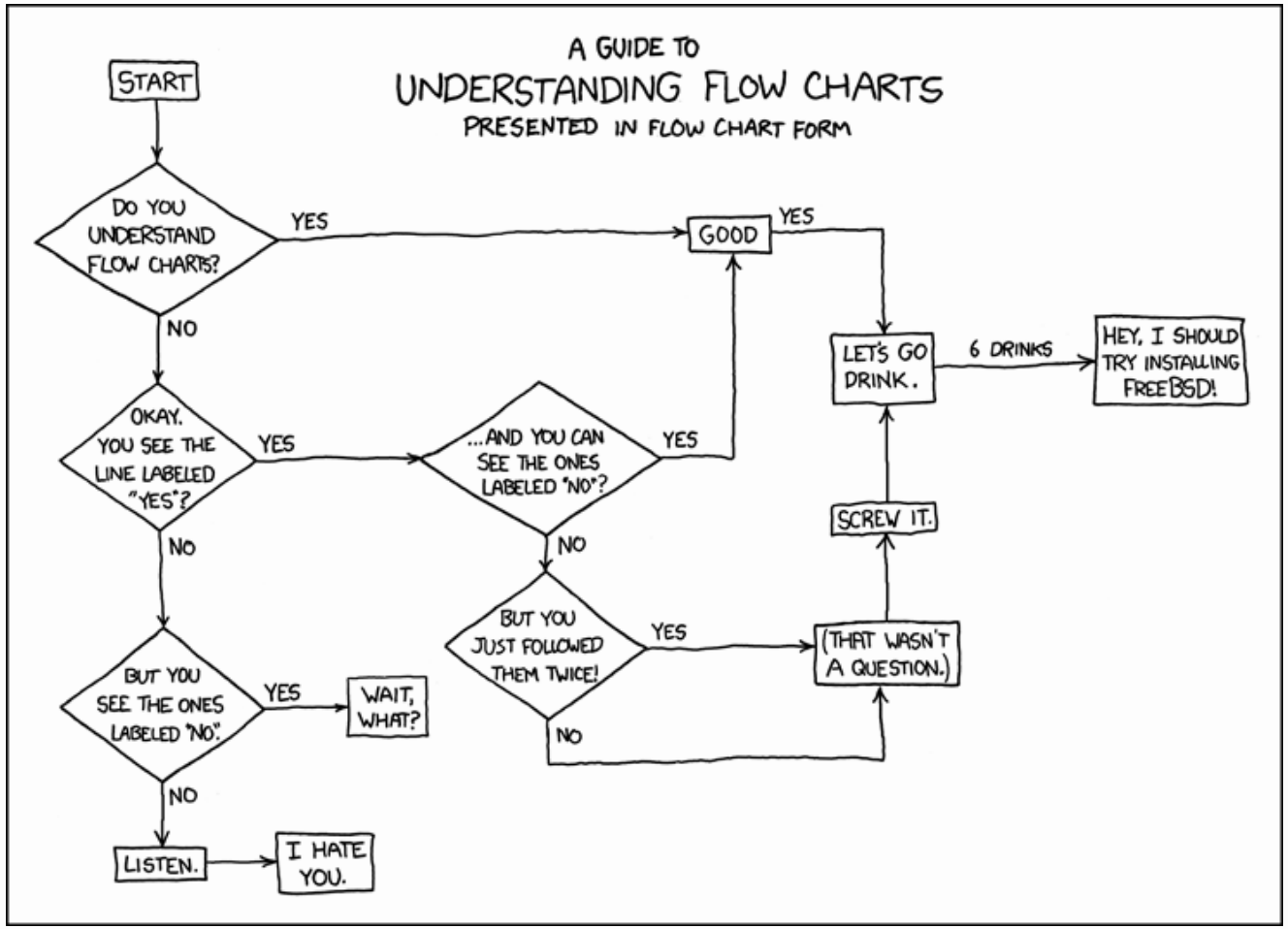

nick@neutrino: ~
nick@neutrino:~$ listdlls.exe | grep -v Command | grep -v pid | grep basetest.exe
0x00000000000f0000 0x6000 C:\basetest.exe
0x00000000000f0000 0x6000 C:\basetest.exe
0x00000000000f0000 0x6000 C:\basetest.exe
0x00000000000f0000 0x6000 C:\basetest.exe
0x00000000000f0000 0x6000 C:\basetest.exe
0x00000000000f0000 0x6000 C:\basetest.exe
0x00000000000f0000 0x6000 C:\basetest.exe
0x00000000000f0000 0x6000 C:\basetest.exe
0x00000000000f0000 0x6000 C:\basetest.exe
0x00000000000f0000 0x6000 C:\basetest.exe
nick@neutrino:~$ mv /mnt/c/basetest.exe /mnt/c/old.exe
nick@neutrino:~$ cat /mnt/c/old.exe >/mnt/c/basetest.exe
nick@neutrino:~$ listdlls.exe | grep -v Command | grep -v pid | grep basetest.exe
0x00000000000a00000 0x6000 C:\basetest.exe
0x00000000000a00000 0x6000 C:\basetest.exe
0x00000000000a00000 0x6000 C:\basetest.exe
0x00000000000a00000 0x6000 C:\basetest.exe
0x00000000000a00000 0x6000 C:\basetest.exe
0x00000000000a00000 0x6000 C:\basetest.exe
0x00000000000a00000 0x6000 C:\basetest.exe
0x00000000000a00000 0x6000 C:\basetest.exe
nick@neutrino:~$

A New Hope

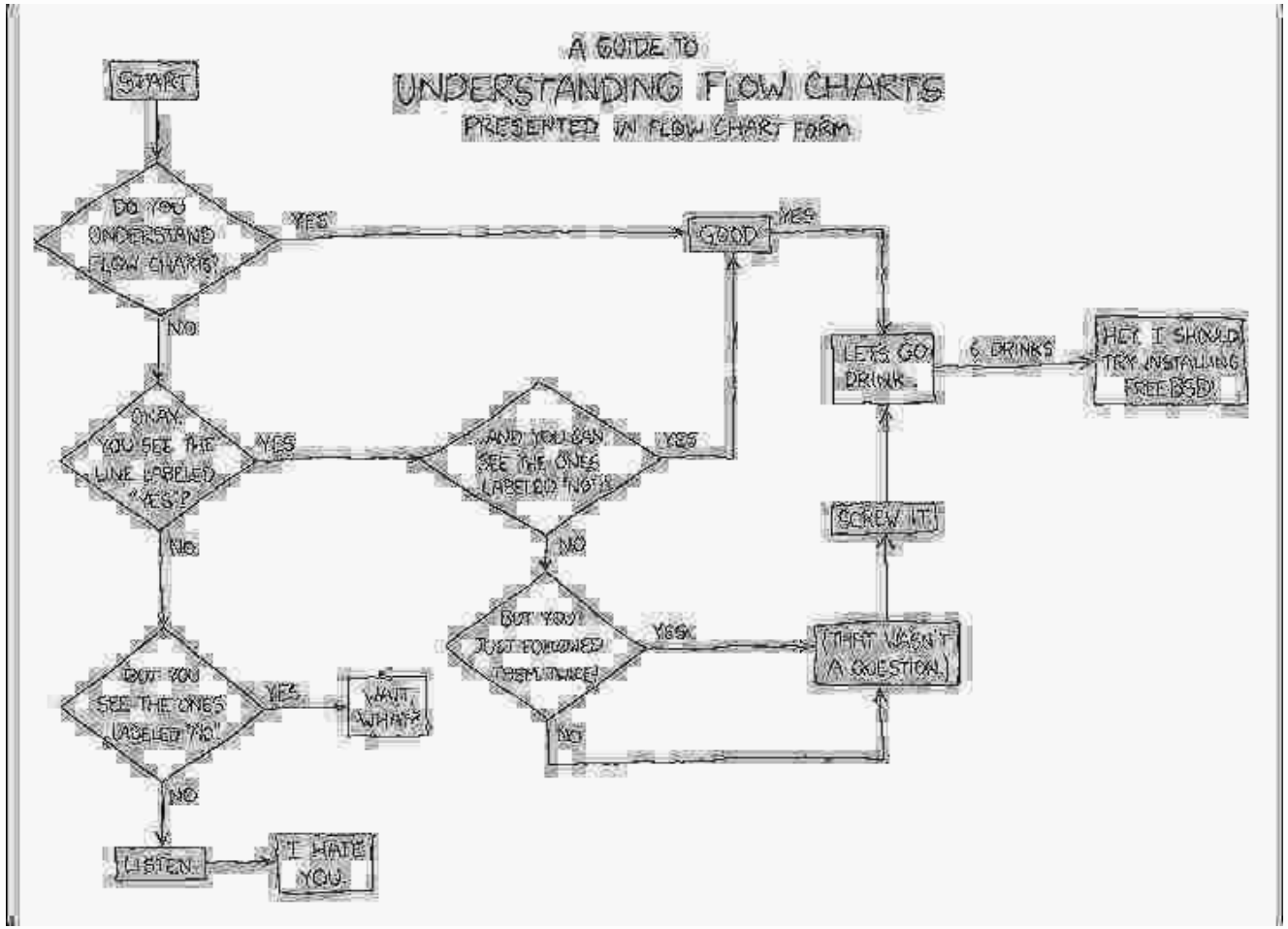

nick@neutrino: ~
nick@neutrino:~$ listdlls.exe | grep -v Command | grep -v pid | grep basetest.exe
0x00000000000f0000 0x6000 C:\basetest.exe
0x00000000000f0000 0x6000 C:\basetest.exe
0x00000000000f0000 0x6000 C:\basetest.exe
0x00000000000f0000 0x6000 C:\basetest.exe
0x00000000000f0000 0x6000 C:\basetest.exe
0x00000000000f0000 0x6000 C:\basetest.exe
0x00000000000f0000 0x6000 C:\basetest.exe
0x00000000000f0000 0x6000 C:\basetest.exe
0x00000000000f0000 0x6000 C:\basetest.exe
0x00000000000f0000 0x6000 C:\basetest.exe
nick@neutrino:~$ mv /mnt/c/basetest.exe /mnt/c/old.exe
nick@neutrino:~$ cat /mnt/c/old.exe >/mnt/c/basetest.exe
nick@neutrino:~$ listdlls.exe | grep -v Command | grep -v pid | grep basetest.exe
0x00000000000a0000 0x6000 C:\basetest.exe
0x00000000000a0000 0x6000 C:\basetest.exe
0x00000000000a0000 0x6000 C:\basetest.exe
0x00000000000a0000 0x6000 C:\basetest.exe
0x00000000000a0000 0x6000 C:\basetest.exe
0x00000000000a0000 0x6000 C:\basetest.exe
0x00000000000a0000 0x6000 C:\basetest.exe
0x00000000000a0000 0x6000 C:\basetest.exe
nick@neutrino:~$

Preselection via File Mapping Invalidation

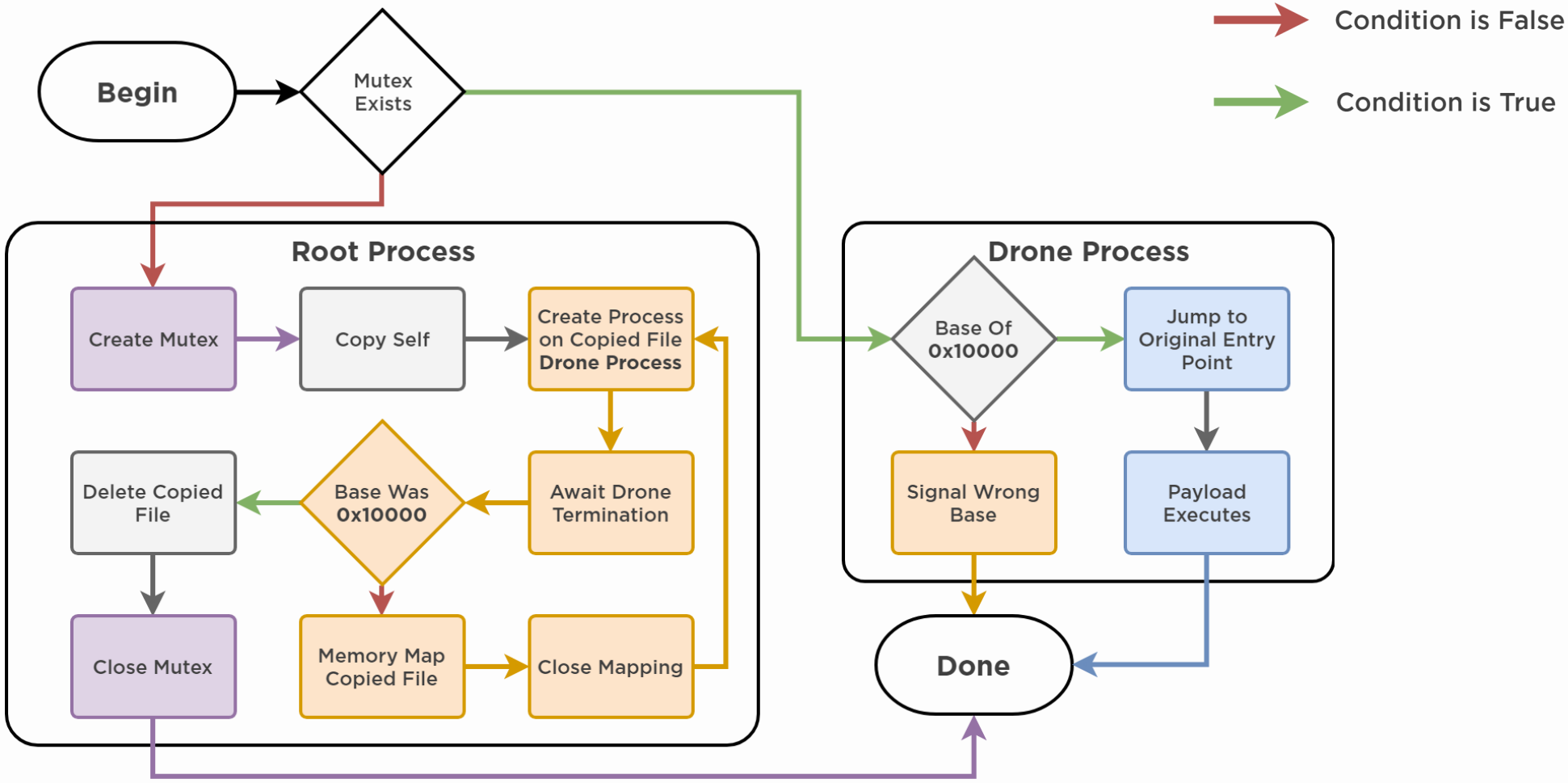
Preselection via File Mapping Invalidation



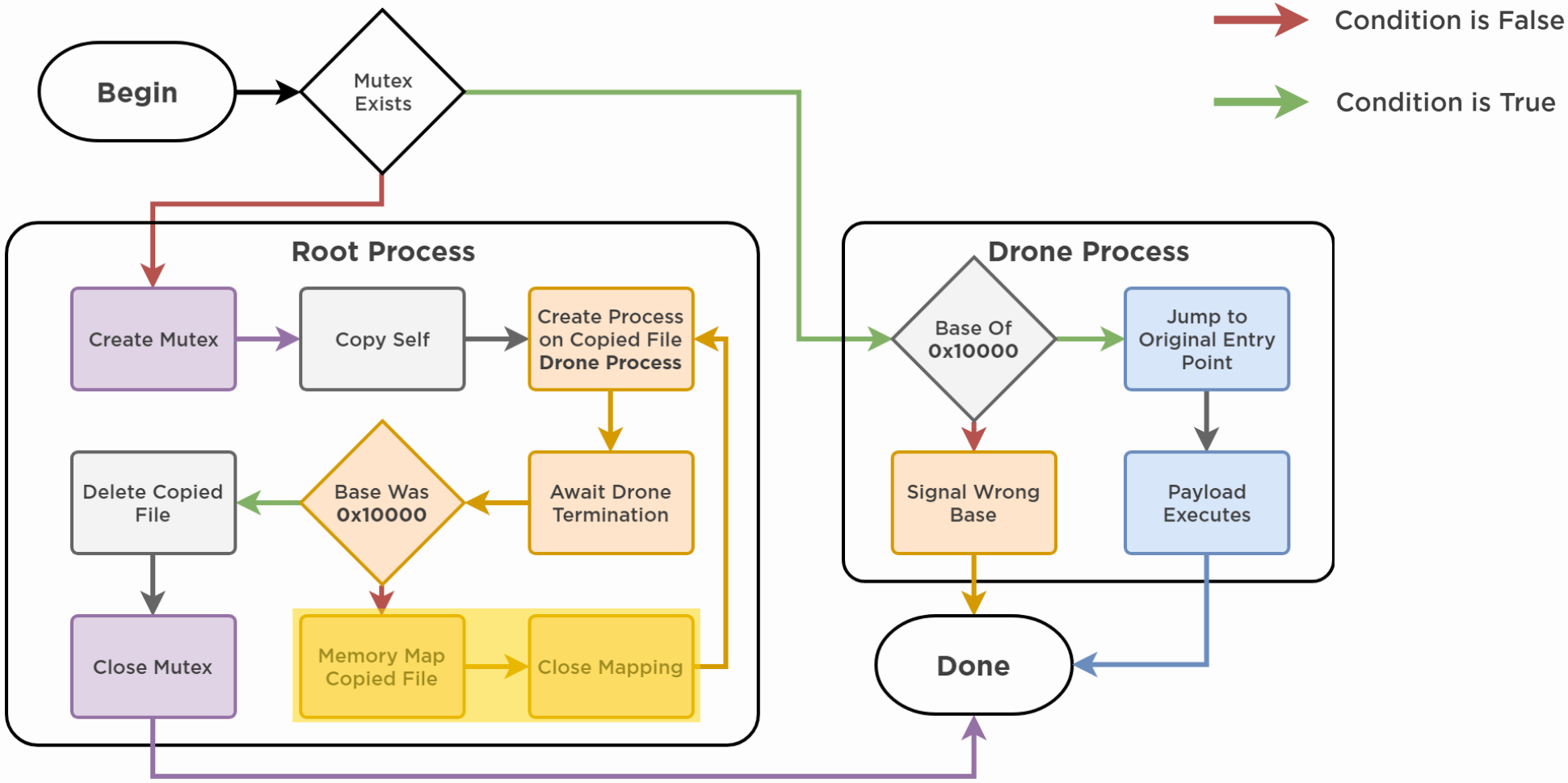
Preselection via File Mapping Invalidation



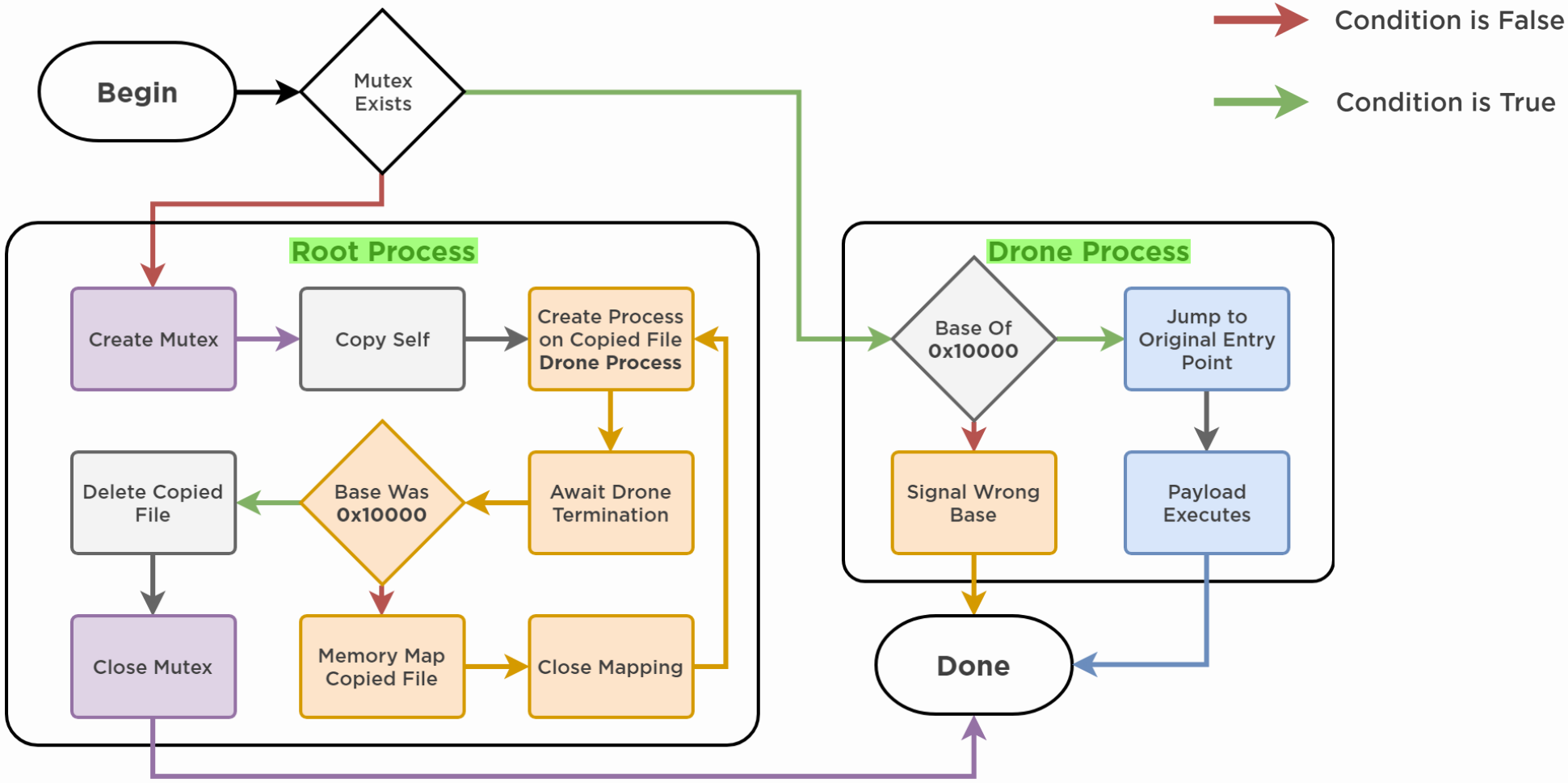
Preselection via File Mapping Invalidation



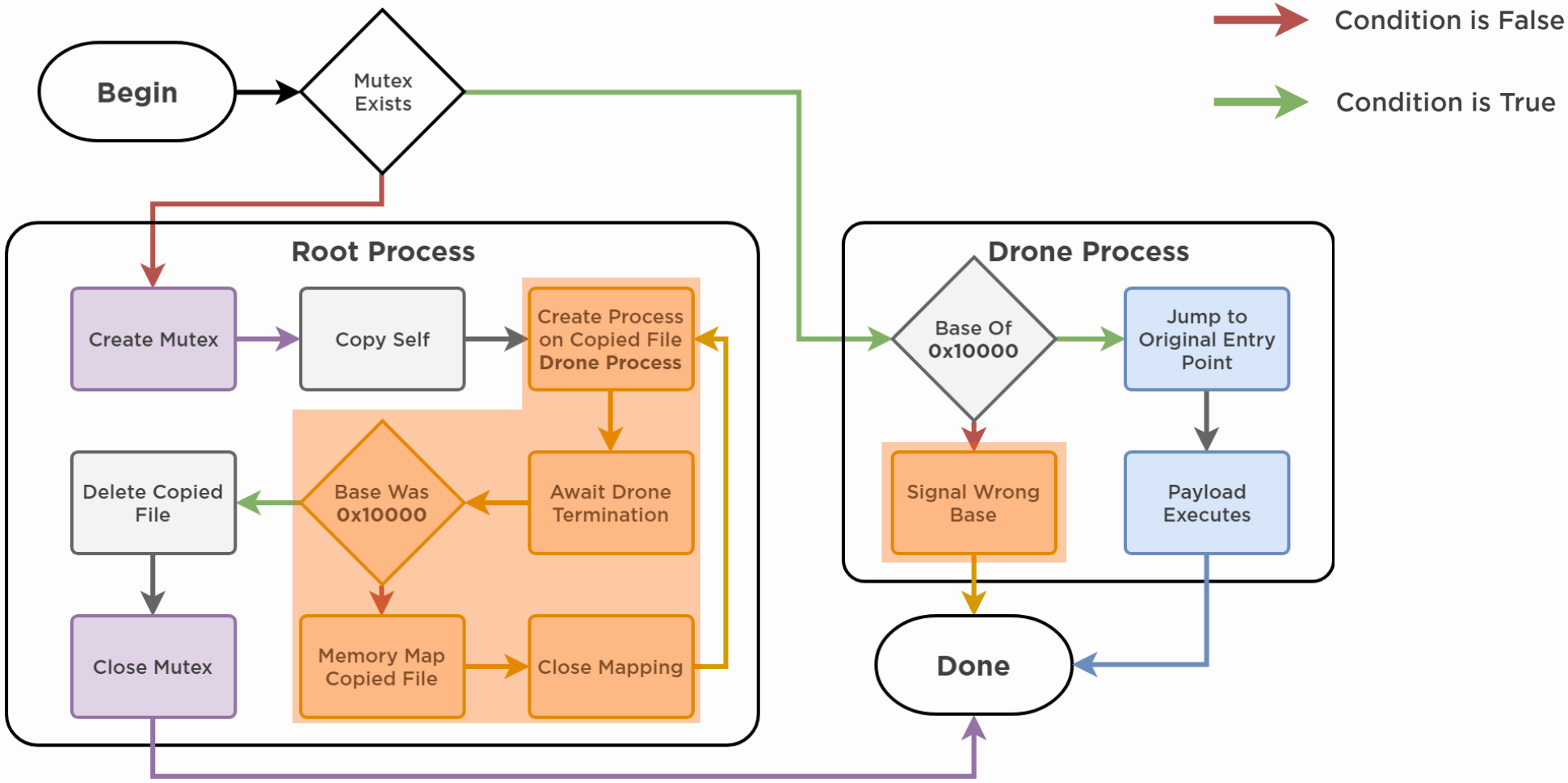
Preselection via File Mapping Invalidation



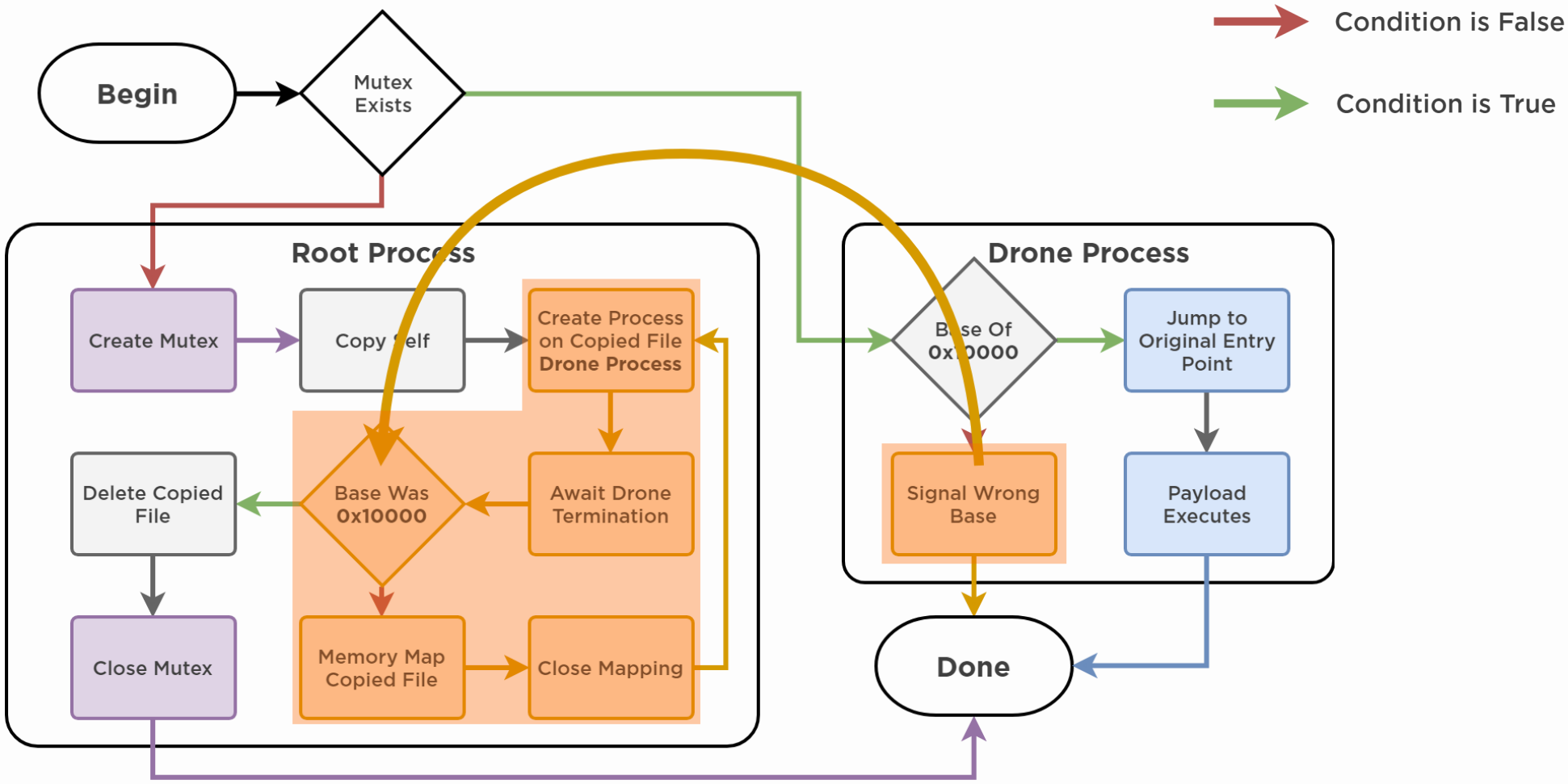
Preselection via File Mapping Invalidation



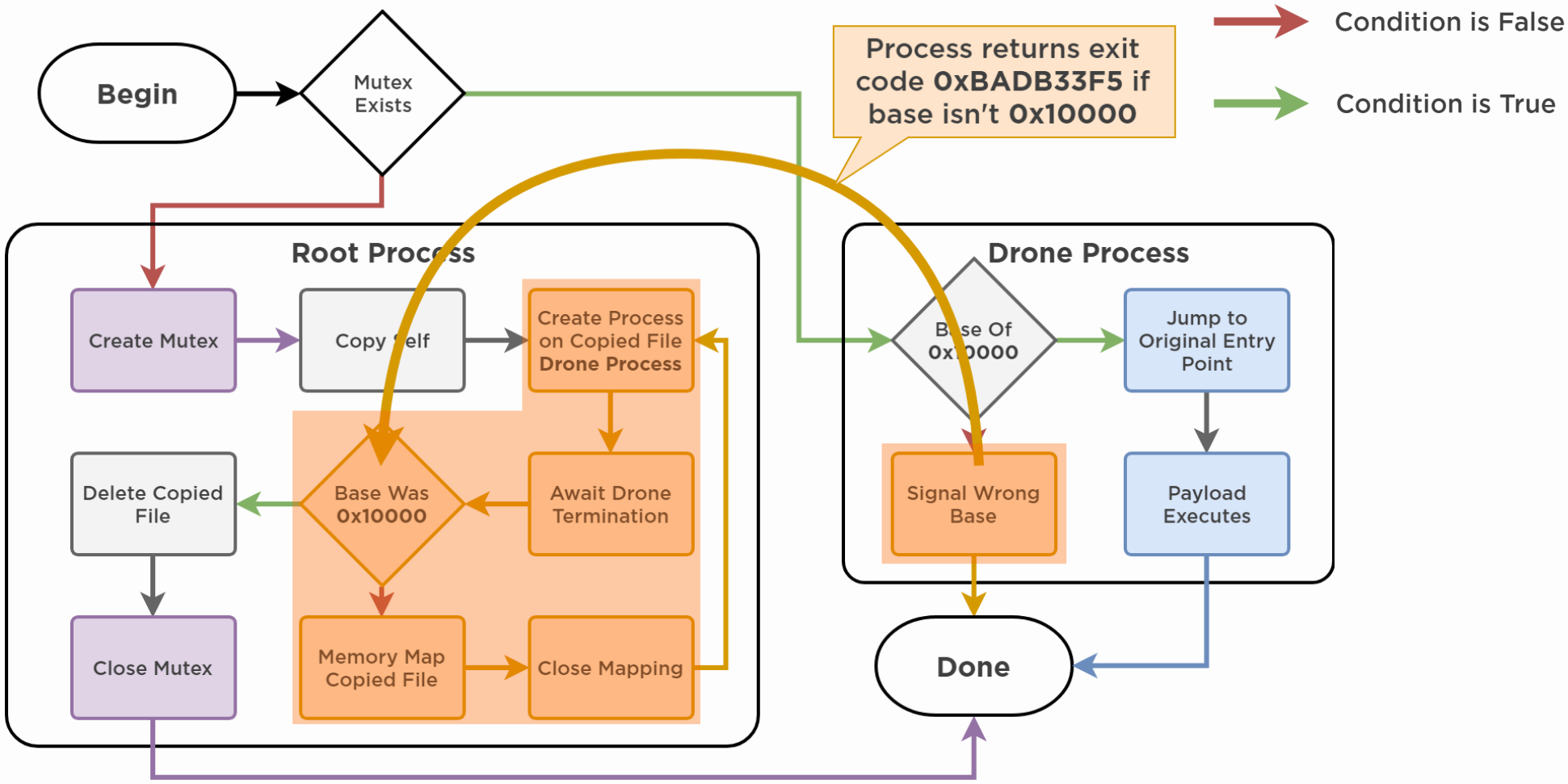
Preselection via File Mapping Invalidation



Preselection via File Mapping Invalidation



Preselection via File Mapping Invalidation



Weaponization

Weaponization

The tool must:

Weaponization

The tool must:

- Create a new section with enough room for the code

Weaponization

The tool must:

- Create a new section with enough room for the code
- Embed the code inside of this new section

Weaponization

The tool must:

- Create a new section with enough room for the code
- Embed the code inside of this new section
- Inform the embedded code of the true `EntryPoint`

Weaponization

The tool must:

- Create a new section with enough room for the code
- Embed the code inside of this new section
- Inform the embedded code of the true `EntryPoint`
- Overwrite `EntryPoint` to point to the embedded code

Weaponization

The tool must:

- Create a new section with enough room for the code
- Embed the code inside of this new section
- Inform the embedded code of the true `EntryPoint`
- Overwrite `EntryPoint` to point to the embedded code

Weaponization

The tool must:

- Create a new section with enough room for the code
- Embed the code inside of this new section
- Inform the embedded code of the true `EntryPoint`
- Overwrite `EntryPoint` to point to the embedded code

For this to work, the ASLR preselection code must be:

Weaponization

The tool must:

- Create a new section with enough room for the code
- Embed the code inside of this new section
- Inform the embedded code of the true `EntryPoint`
- Overwrite `EntryPoint` to point to the embedded code

For this to work, the ASLR preselection code must be:

- Position-agnostic

Weaponization

The tool must:

- Create a new section with enough room for the code
- Embed the code inside of this new section
- Inform the embedded code of the true `EntryPoint`
- Overwrite `EntryPoint` to point to the embedded code

For this to work, the ASLR preselection code must be:

- Position-agnostic
- Generically embeddable in any PE

Weaponization

The tool must:

- Create a new section with enough room for the code
- Embed the code inside of this new section
- Inform the embedded code of the true `EntryPoint`
- Overwrite `EntryPoint` to point to the embedded code

For this to work, the ASLR preselection code must be:

- Position-agnostic
- Generically embeddable in any PE

```c
// HELPERS FOR VARIADIC MACROS
#define EXPAND( x ) x
#define __ARG_N( \
      _1, _2, _3, _4, _5, _6, _7, _8, _9,_10, \
     _11,_12,_13,_14,_15,_16,_17,_18,_19,_20, \
     _21,_22,_23,_24,_25,_26,_27,_28,_29,_30, \
     _31,_32,_33,_34,_35,_36,_37,_38,_39,_40, \
     _41,_42,_43,_44,_45,_46,_47,_48,_49,_50, \
     _51,_52,_53,_54,_55,_56,_57,_58,_59,_60, \
     _61,_62,_63,N,...) N
#define __RSEQ_N() \
     63,62,61,60,                   \
     59,58,57,56,55,54,53,52,51,50, \
     49,48,47,46,45,44,43,42,41,40, \
     39,38,37,36,35,34,33,32,31,30, \
     29,28,27,26,25,24,23,22,21,20, \
     19,18,17,16,15,14,13,12,11,10, \
     9,8,7,6,5,4,3,2,1,0
#define __NARG_I_(...) EXPAND(__ARG_N(__VA_ARGS__))
#define __NARG__(...) __NARG_I_(__VA_ARGS__,__RSEQ_N())
#define __NARG_PLUS_TWO__(...) __NARG__(0, 0, __VA_ARGS__)

#define _VFUNC_(name, n) name##n
#define _VFUNC(name, n) _VFUNC_(name, n)
#define VFUNC(func, ...) EXPAND(_VFUNC(func, __NARG__(__VA_ARGS__)) (__VA_ARGS__))

// GetModuleHandle as a macro
#define LOWER(c) ((((char)c) <= 0x5A && ((char)c) >= 0x41) ? ((char)c) + 0x20 : ((char)c))
#define LDR_GET_MODULE(varName, len, firstChar, lastCharBeforeExt) \
    PVOID varName = NULL; \
    auto module##varName = (PLDR_MODULE)LoaderData->InLoadOrderModuleList.Flink; \
    do { \
        auto checkLen = module##varName->FullDllName.Length / 2; \
        /*wprintf(L"Mod (len %d): %s\n", checkLen, module##varName->FullDllName.Buffer);*/ \
        if (checkLen > len) { \
            if (LOWER(module##varName->FullDllName.Buffer[checkLen - len - 1]) == '\\' && \
                LOWER(module##varName->FullDllName.Buffer[checkLen - len]) == firstChar && \
                LOWER(module##varName->FullDllName.Buffer[checkLen - 5]) == lastCharBeforeExt) { \
                    varName = module##varName->BaseAddress; \
                    break; \
            } \
        } \
        module##varName = (PLDR_MODULE)module##varName->InLoadOrderModuleList.Flink; \
    } while ((PVOID)module##varName != &LoaderData->InLoadOrderModuleList && module##varName->BaseAddress); \
    if (varName == NULL) /*return 0xBADB33F5;*/ {__asm int 3} \
    //printf("%s at 0x%p\n", #varName, varName);

// GetProcAddress as a macro
#define LDR_STR_SIG1(c1) ((int)c1 * (1 << (1 % 16)))
#define LDR_STR_SIG2(c1, c2) (LDR_STR_SIG1(c1) + (int)c2 * (1 << (2 % 16)))
#define LDR_STR_SIG3(c1, c2, c3) (LDR_STR_SIG2(c1, c2) + (int)c3 * (1 << (3 % 16)))
#define LDR_STR_SIG4(c1, c2, c3, c4) (LDR_STR_SIG3(c1, c2, c3) + (int)c4 * (1 << (4 % 16)))
#define LDR_STR_SIG5(c1, c2, c3, c4, c5) (LDR_STR_SIG4(c1, c2, c3, c4) + (int)c5 * (1 << (5 % 16)))
#define LDR_STR_SIG6(c1, c2, c3, c4, c5, c6) (LDR_STR_SIG5(c1, c2, c3, c4, c5) + (int)c6 * (1 << (6 % 16)))
#define LDR_STR_SIG7(c1, c2, c3, c4, c5, c6, c7) (LDR_STR_SIG6(c1, c2, c3, c4, c5, c6) + (int)c7 * (1 << (7 % 16)))
#define LDR_STR_SIG8(c1, c2, c3, c4, c5, c6, c7, c8) (LDR_STR_SIG7(c1, c2, c3, c4, c5, c6, c7) + (int)c8 * (1 << (8 % 16)))
#define LDR_STR_SIG9(c1, c2, c3, c4, c5, c6, c7, c8, c9) (LDR_STR_SIG8(c1, c2, c3, c4, c5, c6, c7, c8) + (int)c9 * (1 << (9 % 16)))
#define LDR_STR_SIG10(c1, c2, c3, c4, c5, c6, c7, c8, c9, c10) (LDR_STR_SIG9(c1, c2, c3, c4, c5, c6, c7, c8, c9) + (int)c10 * (1 << (10 % 16)))
#define LDR_STR_SIG11(c1, c2, c3, c4, c5, c6, c7, c8, c9, c10, c11) (LDR_STR_SIG10(c1, c2, c3, c4, c5, c6, c7, c8, c9, c10) + (int)c11 * (1 << (11 % 16)))
#define LDR_STR_SIG12(c1, c2, c3, c4, c5, c6, c7, c8, c9, c10, c11, c12) (LDR_STR_SIG11(c1, c2, c3, c4, c5, c6, c7, c8, c9, c10, c11) + (int)c12 * (1 << (12 % 16)))
#define LDR_STR_SIG13(c1, c2, c3, c4, c5, c6, c7, c8, c9, c10, c11, c12, c13) (LDR_STR_SIG12(c1, c2, c3, c4, c5, c6, c7, c8, c9, c10, c11, c12) + (int)c13 * (1 << (13 % 16)))
#define LDR_STR_SIG14(c1, c2, c3, c4, c5, c6, c7, c8, c9, c10, c11, c12, c13, c14) (LDR_STR_SIG13(c1, c2, c3, c4, c5, c6, c7, c8, c9, c10, c11, c12, c13) + (int)c14 * (1 << (14 % 16)))
#define LDR_STR_SIG15(c1, c2, c3, c4, c5, c6, c7, c8, c9, c10, c11, c12, c13, c14, c15) (LDR_STR_SIG14(c1, c2, c3, c4, c5, c6, c7, c8, c9, c10, c11, c12, c13, c14) + (int)c15 * (1 << (15 % 16)))
#define LDR_STR_SIG16(c1, c2, c3, c4, c5, c6, c7, c8, c9, c10, c11, c12, c13, c14, c15, c16) (LDR_STR_SIG15(c1, c2, c3, c4, c5, c6, c7, c8, c9, c10, c11, c12, c13, c14, c15) + (int)c16 * (1 << (16 % 16)))
#define LDR_STR_SIG17(c1, c2, c3, c4, c5, c6, c7, c8, c9, c10, c11, c12, c13, c14, c15, c16, c17) (LDR_STR_SIG16(c1, c2, c3, c4, c5, c6, c7, c8, c9, c10, c11, c12, c13, c14, c15, c16) + (int)c17 * (1 << (17 % 16)))
#define LDR_STR_SIG18(c1, c2, c3, c4, c5, c6, c7, c8, c9, c10, c11, c12, c13, c14, c15, c16, c17, c18) (LDR_STR_SIG17(c1, c2, c3, c4, c5, c6, c7, c8, c9, c10, c11, c12, c13, c14, c15, c16, c17) + (int)c18 * (1 << (18 % 16)))
#define LDR_STR_SIG19(c1, c2, c3, c4, c5, c6, c7, c8, c9, c10, c11, c12, c13, c14, c15, c16, c17, c18, c19) (LDR_STR_SIG18(c1, c2, c3, c4, c5, c6, c7, c8, c9, c10, c11, c12, c13, c14, c15, c16, c17, c18) + (int)c19 * (1 << (19 % 16)))
#define LDR_STR_SIG20(c1, c2, c3, c4, c5, c6, c7, c8, c9, c10, c11, c12, c13, c14, c15, c16, c17, c18, c19, c20) (LDR_STR_SIG19(c1, c2, c3, c4, c5, c6, c7, c8, c9, c10, c11, c12, c13, c14, c15, c16, c17, c18, c19) + (int)c20 * (1 << (20 % 16)))
#define LDR_STR_SIG21(c1, c2, c3, c4, c5, c6, c7, c8, c9, c10, c11, c12, c13, c14, c15, c16, c17, c18, c19, c20, c21) (LDR_STR_SIG20(c1, c2, c3, c4, c5, c6, c7, c8, c9, c10, c11, c12, c13, c14, c15, c16, c17, c18, c19, c20) + (int)c21 * (1 << (21 % 16)))
#define LDR_STR_SIG22(c1, c2, c3, c4, c5, c6, c7, c8, c9, c10, c11, c12, c13, c14, c15, c16, c17, c18, c19, c20, c21, c22) (LDR_STR_SIG21(c1, c2, c3, c4, c5, c6, c7, c8, c9, c10, c11, c12, c13, c14, c15, c16, c17, c18, c19, c20, c21) + (int)c22 * (1 << (22 % 16)))
#define LDR_STR_SIG23(c1, c2, c3, c4, c5, c6, c7, c8, c9, c10, c11, c12, c13, c14, c15, c16, c17, c18, c19, c20, c21, c22, c23) (LDR_STR_SIG22(c1, c2, c3, c4, c5, c6, c7, c8, c9, c10, c11, c12, c13, c14, c15, c16, c17, c18, c19, c20, c21, c22) + (int)c23 * (1 << (23 % 16)))
#define LDR_STR_SIG24(c1, c2, c3, c4, c5, c6, c7, c8, c9, c10, c11, c12, c13, c14, c15, c16, c17, c18, c19, c20, c21, c22, c23, c24) (LDR_STR_SIG23(c1, c2, c3, c4, c5, c6, c7, c8, c9, c10, c11, c12, c13, c14, c15, c16, c17, c18, c19, c20, c21, c22, c23) + (int)c24 * (1 << (24 % 16)))
#define LDR_STR_SIG25(c1, c2, c3, c4, c5, c6, c7, c8, c9, c10, c11, c12, c13, c14, c15, c16, c17, c18, c19, c20, c21, c22, c23, c24, c25) (LDR_STR_SIG24(c1, c2, c3, c4, c5, c6, c7, c8, c9, c10, c11, c12, c13, c14, c15, c16, c17, c18, c19, c20, c21, c22, c23, c24) + (int)c25 * (1 << (25 % 16)))
#define LDR_STR_SIG26(c1, c2, c3, c4, c5, c6, c7, c8, c9, c10, c11, c12, c13, c14, c15, c16, c17, c18, c19, c20, c21, c22, c23, c24, c25, c26) (LDR_STR_SIG25(c1, c2, c3, c4, c5, c6, c7, c8, c9, c10, c11, c12, c13, c14, c15, c16, c17, c18, c19, c20, c21, c22, c23, c24, c25) + (int)c26 * (1 << (26 % 16)))
#define LDR_STR_SIG27(c1, c2, c3, c4, c5, c6, c7, c8, c9, c10, c11, c12, c13, c14, c15, c16, c17, c18, c19, c20, c21, c22, c23, c24, c25, c26, c27) (LDR_STR_SIG26(c1, c2, c3, c4, c5, c6, c7, c8, c9, c10, c11, c12, c13, c14, c15, c16, c17, c18, c19, c20, c21, c22, c23, c24, c25, c26) + (int)c27 * (1 << (27 % 16)))
#define LDR_STR_SIG28(c1, c2, c3, c4, c5, c6, c7, c8, c9, c10, c11, c12, c13, c14, c15, c16, c17, c18, c19, c20, c21, c22, c23, c24, c25, c26, c27, c28) (LDR_STR_SIG27(c1, c2, c3, c4, c5, c6, c7, c8, c9, c10, c11, c12, c13, c14, c15, c16, c17, c18, c19, c20, c21, c22, c23, c24, c25, c26, c27) + (int)c28 * (1 << (28 % 16)))
#define LDR_STR_SIG29(c1, c2, c3, c4, c5, c6, c7, c8, c9, c10, c11, c12, c13, c14, c15, c16, c17, c18, c19, c20, c21, c22, c23, c24, c25, c26, c27, c28, c29) (LDR_STR_SIG28(c1, c2, c3, c4, c5, c6, c7, c8, c9, c10, c11, c12, c13, c14, c15, c16, c17, c18, c19, c20, c21, c22, c23, c24, c25, c26, c27, c28) + (int)c29 * (1 << (29 % 16)))
#define LDR_STR_SIG(...) __NARG__(__VA_ARGS__), VFUNC(LDR_STR_SIG, __VA_ARGS__)

#define LDR_GET_PROC_INTERNAL(varName, module, typeCast, len, sig) \
    auto varName = (typeCast)NULL; \
    auto base##varName = (HINSTANCE)module; \
    auto dosHeader##varName = (PIMAGE_DOS_HEADER)base##varName; \
    auto ntHeader##varName = (PIMAGE_NT_HEADERS)((LONG)dosHeader##varName + dosHeader##varName->e_lfanew); \
    auto exportOffset##varName = ntHeader##varName->OptionalHeader.DataDirectory[IMAGE_DIRECTORY_ENTRY_EXPORT].VirtualAddress; \
    auto exportSize##varName = ntHeader##varName->OptionalHeader.DataDirectory[IMAGE_DIRECTORY_ENTRY_EXPORT].Size; \
    auto exportDir##varName = (PIMAGE_EXPORT_DIRECTORY)((DWORD)base##varName + exportOffset##varName); \
    auto functionArray##varName = (PDWORD)((LPBYTE)base##varName + exportDir##varName->AddressOfFunctions); \
    auto functionNameArray##varName = (PDWORD)((LPBYTE)base##varName + exportDir##varName->AddressOfNames); \
    auto ordinal##varName = (PWORD)((LPBYTE)base##varName + exportDir##varName->AddressOfNameOrdinals); \
    for (DWORD i = 0; i < exportDir##varName->NumberOfFunctions; i++) { \
        if (functionNameArray##varName[i] == NULL) continue; \
        auto funcName = (char*)base##varName+functionNameArray##varName[i]; \
        int nameLen = 0; \
        int nameSig = 0; \
        for (; funcName[nameLen] != '\x00'; nameLen++) \
            nameSig += (int)funcName[nameLen] * (1 << ((nameLen + 1) % 16)); \
        /*printf("Func (len %02d, sig %08d): %s\n", nameLen, nameSig, funcName);*/ \
        if (nameLen == len && nameSig == sig) { \
            varName = (typeCast)((LPBYTE)base##varName+functionArray##varName[ordinal##varName[i]]); \
            /*printf("%s %08d = %s %08d\n", #varName, sig, funcName, nameSig);*/ \
            break; \
        } \
    } \
    if (varName == NULL) /*return 0xBADB33F5;*/ {__asm int 3} \
    //printf("%s at 0x%p\n", #varName, varName);

#define LDR_GET_PROC(...) EXPAND(LDR_GET_PROC_INTERNAL(__VA_ARGS__))

// String definitions in inline assembly as a macro
#define EMIT1(c1) __asm _emit c1
#define EMIT2(c1, c2) EMIT1(c1)  __asm _emit c2
#define EMIT3(c1, c2, c3) EMIT2(c1, c2)  __asm _emit c3
#define EMIT4(c1, c2, c3, c4) EMIT3(c1, c2, c3)  __asm _emit c4
#define EMIT5(c1, c2, c3, c4, c5) EMIT4(c1, c2, c3, c4)  __asm _emit c5
#define EMIT6(c1, c2, c3, c4, c5, c6) EMIT5(c1, c2, c3, c4, c5)  __asm _emit c6
#define EMIT7(c1, c2, c3, c4, c5, c6, c7) EMIT6(c1, c2, c3, c4, c5, c6)  __asm _emit c7
#define EMIT8(c1, c2, c3, c4, c5, c6, c7, c8) EMIT7(c1, c2, c3, c4, c5, c6, c7)  __asm _emit c8
#define EMIT9(c1, c2, c3, c4, c5, c6, c7, c8, c9) EMIT8(c1, c2, c3, c4, c5, c6, c7, c8)  __asm _emit c9
#define EMIT10(c1, c2, c3, c4, c5, c6, c7, c8, c9, c10) EMIT9(c1, c2, c3, c4, c5, c6, c7, c8, c9)  __asm _emit c10
#define EMIT11(c1, c2, c3, c4, c5, c6, c7, c8, c9, c10, c11) EMIT10(c1, c2, c3, c4, c5, c6, c7, c8, c9, c10)  __asm _emit c11
#define EMIT12(c1, c2, c3, c4, c5, c6, c7, c8, c9, c10, c11, c12) EMIT11(c1, c2, c3, c4, c5, c6, c7, c8, c9, c10, c11)  __asm _emit c12
#define EMIT13(c1, c2, c3, c4, c5, c6, c7, c8, c9, c10, c11, c12, c13) EMIT12(c1, c2, c3, c4, c5, c6, c7, c8, c9, c10, c11, c12)  __asm _emit c13
#define EMIT14(c1, c2, c3, c4, c5, c6, c7, c8, c9, c10, c11, c12, c13, c14) EMIT13(c1, c2, c3, c4, c5, c6, c7, c8, c9, c10, c11, c12, c13)  __asm _emit c14
#define EMIT15(c1, c2, c3, c4, c5, c6, c7, c8, c9, c10, c11, c12, c13, c14, c15) EMIT14(c1, c2, c3, c4, c5, c6, c7, c8, c9, c10, c11, c12, c13, c14)  __asm _emit c15
#define EMIT16(c1, c2, c3, c4, c5, c6, c7, c8, c9, c10, c11, c12, c13, c14, c15, c16) EMIT15(c1, c2, c3, c4, c5, c6, c7, c8, c9, c10, c11, c12, c13, c14, c15)  __asm _emit c16
#define EMIT17(c1, c2, c3, c4, c5, c6, c7, c8, c9, c10, c11, c12, c13, c14, c15, c16, c17) EMIT16(c1, c2, c3, c4, c5, c6, c7, c8, c9, c10, c11, c12, c13, c14, c15, c16)  __asm _emit c17
#define EMIT18(c1, c2, c3, c4, c5, c6, c7, c8, c9, c10, c11, c12, c13, c14, c15, c16, c17, c18) EMIT17(c1, c2, c3, c4, c5, c6, c7, c8, c9, c10, c11, c12, c13, c14, c15, c16, c17)  __asm _emit c18
#define EMIT19(c1, c2, c3, c4, c5, c6, c7, c8, c9, c10, c11, c12, c13, c14, c15, c16, c17, c18, c19) EMIT18(c1, c2, c3, c4, c5, c6, c7, c8, c9, c10, c11, c12, c13, c14, c15, c16, c17, c18)  __asm _emit c19
#define EMIT20(c1, c2, c3, c4, c5, c6, c7, c8, c9, c10, c11, c12, c13, c14, c15, c16, c17, c18, c19, c20) EMIT19(c1, c2, c3, c4, c5, c6, c7, c8, c9, c10, c11, c12, c13, c14, c15, c16, c17, c18, c19)  __asm _emit c20
#define EMIT21(c1, c2, c3, c4, c5, c6, c7, c8, c9, c10, c11, c12, c13, c14, c15, c16, c17, c18, c19, c20, c21) EMIT20(c1, c2, c3, c4, c5, c6, c7, c8, c9, c10, c11, c12, c13, c14, c15, c16, c17, c18, c19, c20)  __asm _emit c21

#define DEFINE_STR(type, varName, ...) \
    type varName = NULL; \
    __asm _emit 0xE8 __asm _emit 00 __asm _emit 00 __asm _emit 00 __asm  _emit 00 /* CALL <next instruction> */ \
    __asm POP EAX \
    __asm ADD EAX, 0x06 \
    __asm _emit 0xEB __asm _emit __NARG_PLUS_TWO__(__VA_ARGS__)                  /* jmp over string */ \
    VFUNC(EMIT, __VA_ARGS__)                                                     /* emit string */ \
    __asm _emit '\0' __asm _emit '\0'                                            /* emit nullterm */ \
    __asm MOV varName, EAX

#define ZERO_MEM(ptr, size) \
    for (size_t zerosize = 0; zerosize < size; zerosize++) { \
        ((char*)ptr)[zerosize] = 0x00; \
    }
```

```
LDR_GET_PROC(f_GetModuleHandle,      kernel32Handle, HMODULE(__stdcall *)(PUCHAR), LDR_STR_SIG('G', 'e', 't', 'M', 'o', 'd', 'u', 'l', 'e', 'H', 'a', 'n', 'd', 'l', 'e', 'A'));
LDR_GET_PROC(f_CreateMutexA,         kernel32Handle, HANDLE(__stdcall *)(PVOID, BOOL, LPCSTR), LDR_STR_SIG('C', 'r', 'e', 'a', 't', 'e', 'M', 'u', 't', 'e', 'x', 'A'));
LDR_GET_PROC(f_GetLastError,         kernel32Handle, DWORD(__stdcall *)(), LDR_STR_SIG('G', 'e', 't', 'L', 'a', 's', 't', 'E', 'r', 'r', 'o', 'r'));
LDR_GET_PROC(f_CloseHandle,          kernel32Handle, BOOL(__stdcall *)(HANDLE), LDR_STR_SIG('C', 'l', 'o', 's', 'e', 'H', 'a', 'n', 'd', 'l', 'e'));
LDR_GET_PROC(f_GetCurrentProcessId,  kernel32Handle, DWORD(__stdcall *)(), LDR_STR_SIG('G', 'e', 't', 'C', 'u', 'r', 'r', 'e', 'n', 't', 'P', 'r', 'o', 'c', 'e', 's', 's', 'I', 'd'))
LDR_GET_PROC(f_GetModuleFileNameW,   kernel32Handle, DWORD(__stdcall *)(HMODULE, LPWSTR, DWORD), LDR_STR_SIG('G', 'e', 't', 'M', 'o', 'd', 'u', 'l', 'e', 'F', 'i', 'l', 'e', 'N', 'a
LDR_GET_PROC(f_CreateProcessW,       kernel32Handle, BOOL(__stdcall *)(LPCWSTR, LPWSTR, PVOID, PVOID, BOOL, DWORD, LPVOID, LPCWSTR, LPSTARTUPINFOW, LPPROCESS_INFORMATION), LDR_STR_SI
LDR_GET_PROC(f_WaitForSingleObject,  kernel32Handle, DWORD(__stdcall *)(HANDLE, DWORD), LDR_STR_SIG('W', 'a', 'i', 't', 'F', 'o', 'r', 'S', 'i', 'n', 'g', 'l', 'e', 'O', 'b', 'j', '
LDR_GET_PROC(f_GetExitCodeProcess,   kernel32Handle, BOOL(__stdcall *)(HANDLE, LPDWORD), LDR_STR_SIG('G', 'e', 't', 'E', 'x', 'i', 't', 'C', 'o', 'd', 'e', 'P', 'r', 'o', 'c', 'e',
LDR_GET_PROC(f_CreateFileW,          kernel32Handle, HANDLE(__stdcall *)(LPCWSTR, DWORD, DWORD, LPSECURITY_ATTRIBUTES, DWORD, DWORD, HANDLE), LDR_STR_SIG('C', 'r', 'e', 'a', 't', 'e
LDR_GET_PROC(f_CreateFileMappingW,   kernel32Handle, HANDLE(__stdcall *)(HANDLE, LPSECURITY_ATTRIBUTES, DWORD, DWORD, DWORD, LPCWSTR), LDR_STR_SIG('C', 'r', 'e', 'a', 't', 'e', 'F',
LDR_GET_PROC(f_MapViewOfFileEx,      kernel32Handle, LPVOID(__stdcall *)(HANDLE, DWORD, DWORD, DWORD, SIZE_T, LPVOID), LDR_STR_SIG('M', 'a', 'p', 'V', 'i', 'e', 'w', 'O', 'f', 'F',
LDR_GET_PROC(f_UnmapViewOfFile,      kernel32Handle, BOOL(__stdcall *)(LPCVOID), LDR_STR_SIG('U', 'n', 'm', 'a', 'p', 'V', 'i', 'e', 'w', 'O', 'f', 'F', 'i', 'l', 'e'));
LDR_GET_PROC(f_CopyFileW,            kernel32Handle, BOOL(__stdcall *)(LPCWSTR, LPCWSTR, BOOL), LDR_STR_SIG('C', 'o', 'p', 'y', 'F', 'i', 'l', 'e', 'W'));
LDR_GET_PROC(f_DeleteFileW,          kernel32Handle, BOOL(__stdcall *)(LPCWSTR), LDR_STR_SIG('D', 'e', 'l', 'e', 't', 'e', 'F', 'i', 'l', 'e', 'W'));
LDR_GET_PROC(f_lstrcatW,             kernel32Handle, LPWSTR(__stdcall *)(LPWSTR, LPCWSTR), LDR_STR_SIG('l', 's', 't', 'r', 'c', 'a', 't', 'W'));
LDR_GET_PROC(f_ExitProcess,          kernel32Handle, VOID(__stdcall *)(DWORD), LDR_STR_SIG('E', 'x', 'i', 't', 'P', 'r', 'o', 'c', 'e', 's', 's'));

// define strings
DEFINE_STR(const char*, mutexName, 'r', 'e', 'l', 'o', 'c', '_', 'p', 'a', 'c', 'k', '_', 'm', 'u', 't');
DEFINE_STR(const wchar_t*, copySuffix, '2', 0x00, '.', 0x00, 'e', 0x00, 'x', 0x00, 'e', 0x00);
```

Weaponization

Weaponization

It works!

Weaponization

It works!

Caveats

Weaponization

It works!

Caveats

- It can be slow, averaging of 200 iterations to land

Weaponization

It works!

Caveats

- It can be slow, averaging of 200 iterations to land
- Imports can't be obfuscated

Weaponization

It works!

Caveats

- It can be slow, averaging of 200 iterations to land
- Imports can't be obfuscated

Advantages

Weaponization

It works!

Caveats

- It can be slow, averaging of 200 iterations to land
- Imports can't be obfuscated

Advantages

- Base can be anything, not just `0x00010000`!

Weaponization

It works!

Caveats

- It can be slow, averaging of 200 iterations to land
- Imports can't be obfuscated

Advantages

- Base can be anything, not just `0x00010000`!
- As a side effect, some form of symbolic execution is needed to discover the intended base in order to fix up the file for analysis.

ROUND TWO

FIGHT

Use Cases

Use Cases

- Annoying analysts

Use Cases

- Annoying analysts
- Breaking automated static analysis systems

Use Cases

- Annoying analysts
- Breaking automated static analysis systems
- Breaking tools

Use Cases

- Annoying analysts
- Breaking automated static analysis systems
- Breaking tools
- Breaking AV Parsers

Potential Improvements

Potential Improvements

- More obfuscations

Potential Improvements

- More obfuscations
  - New targets

Potential Improvements

- More obfuscations
  - New targets
  - Multiple passes

Potential Improvements

- More obfuscations
  - New targets
  - Multiple passes
- Header Scrambling

Potential Improvements

- More obfuscations
  - New targets
  - Multiple passes
- Header Scrambling
- Combining with runtime packers

Potential Improvements

- More obfuscations
  - New targets
  - Multiple passes
- Header Scrambling
- Combining with runtime packers
- Support for 64bit binaries

Potential Improvements

- More obfuscations
  - New targets
  - Multiple passes
- Header Scrambling
- Combining with runtime packers
- Support for 64bit binaries
- Support for DLLs

Potential Improvements

- More obfuscations
  - New targets
  - Multiple passes
- Header Scrambling
- Combining with runtime packers
- Support for 64bit binaries
- Support for DLLs
- Selective obfuscations

THE END

Find Me

https://nickcano.com

https://github.com/nickcano

https://twitter.com/nickcano93

https://nostarch.com/gamehacking

https://pluralsight.com/authors/nick-cano

Source Code

https://github.com/nickcano/RelocBonus

https://github.com/nickcano/RelocBonusSlides

Resources

https://msdn.microsoft.com/en-us/library/ms809762.aspx

https://github.com/corkami/pocs/tree/master/PE