Loading...

IDA Pro crash course

(don’t tell pancake)

Introduction to unpacking using r2

Agenda

Introduction

Useful r2 commands for unpacking

Techniques and samples

Dealing with the IAT

Introduction

Introduction

- About Us
- About You
- About Packers
- About PEs

About Us

About You

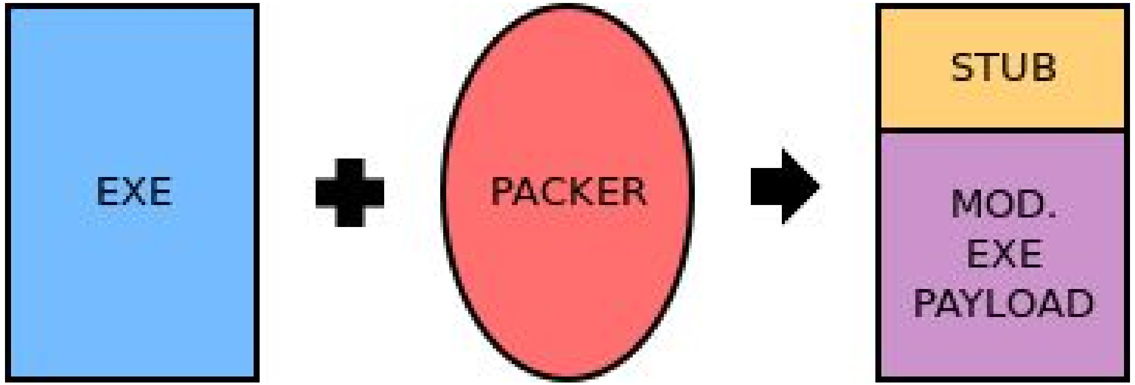
1. Who's dealt with malware unpacking?

2. Who's has unpacked malware with r2?

3. Who has successfully unpacked malware with r2?

About Packers

What a packer is



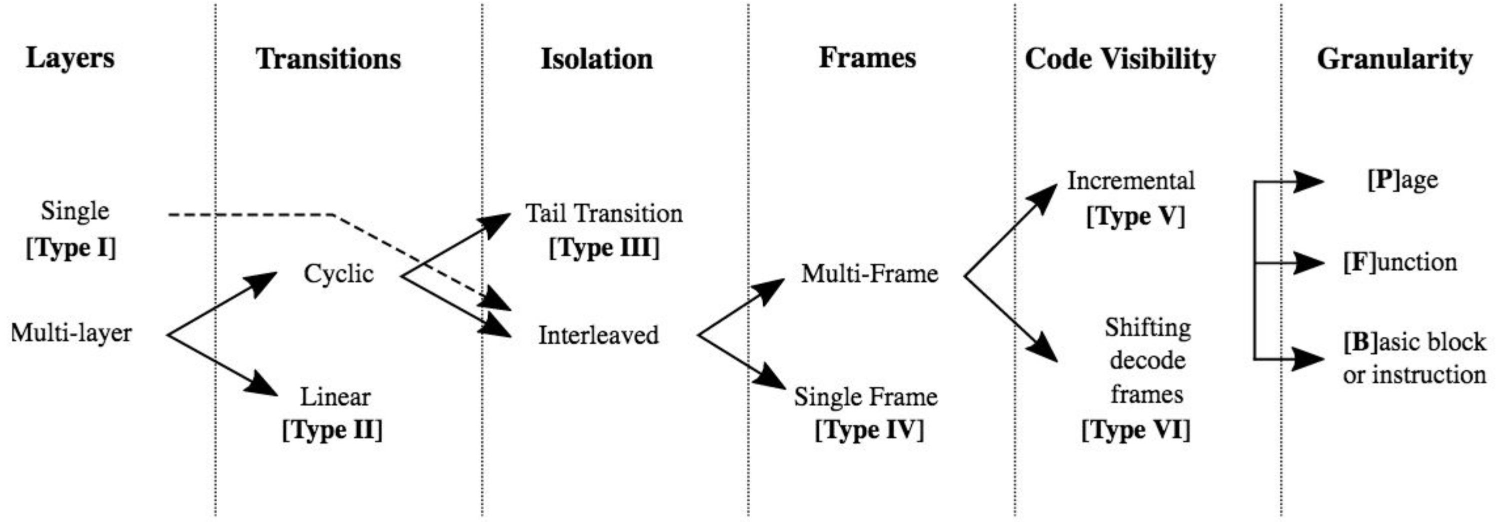
About Packers

Packer transformations

- Compressors
- Crypters
- Protectors

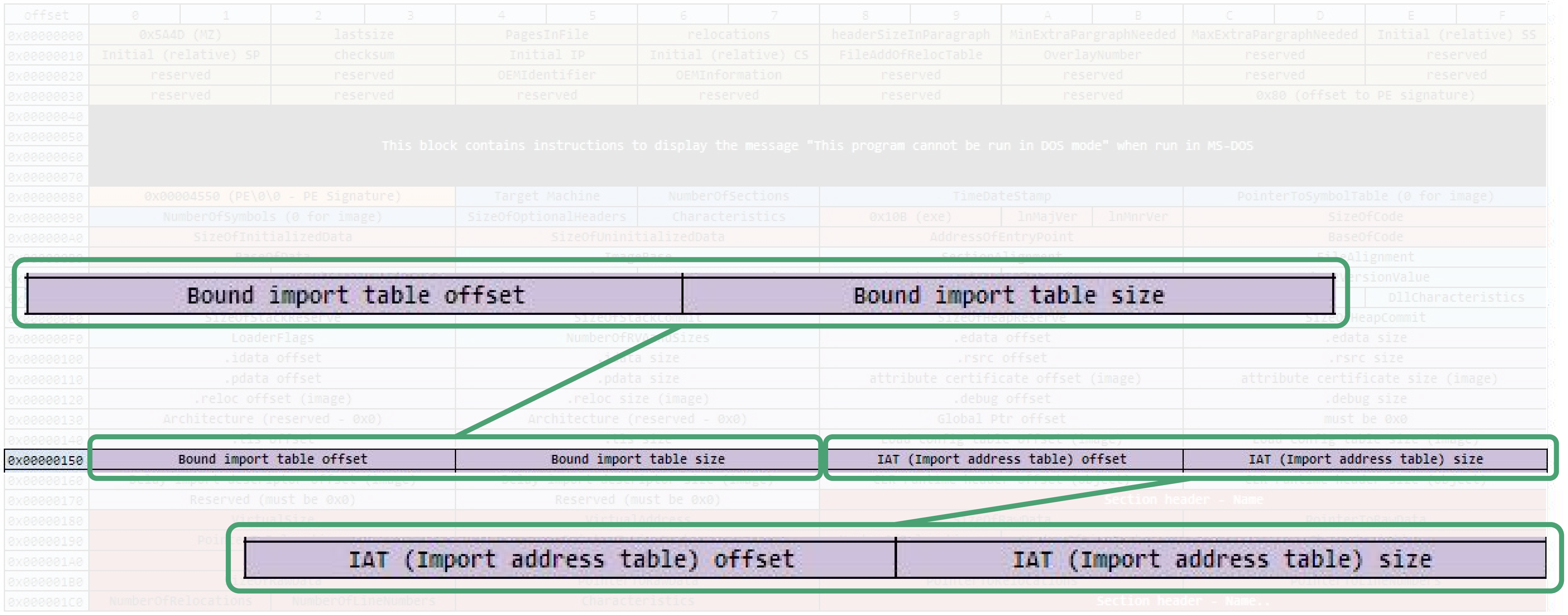
About Packers

Packer complexity types



img source: www.packerinspector.com/reference

About PEs



offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0x00000000	0x5A4D (MZ)		lastsize		PagesInFile		relocations		headerSizeInParagraph		MinExtraPargraphNeeded		MaxExtraPargraphNeeded		Initial (relative) SS	
0x00000010	Initial (relative) SP		checksum		Initial IP		Initial (relative) CS		FileAddOfRelocTable		OverlayNumber		reserved		reserved	
0x00000020	reserved		reserved		OEMIdentifier		OEMInformation		reserved		reserved		reserved		reserved	
0x00000030	reserved		reserved		reserved		reserved		reserved		reserved		0x80 (offset to PE signature)			
0x00000040	This block contains instructions to display the message "This program cannot be run in DOS mode" when run in MS-DOS															
0x00000050																
0x00000060																
0x00000070																
0x00000080	0x00004550 (PE\0\0 - PE Signature)				Target Machine		NumberOfSections		TimeDateStamp				PointerToSymbolTable (0 for image)			
0x00000090	NumberOfSymbols (0 for image)				SizeOfOptionalHeaders		Characteristics		0x10B (exe)		lnMajVer	lnMnrVer	SizeOfCode			
0x000000A0	SizeOfInitializedData				SizeOfUninitializedData				AddressOfEntryPoint				BaseOfCode			
0x000000B0	BaseOfData				ImageBase				SectionAlignment				FileAlignment			
[illegible]	[illegible]				[illegible]				[illegible]				[illegible]VersionValue			
[illegible]	[illegible]				[illegible]				[illegible]				[illegible]		DllCharacteristics	
0x000000E0	SizeOfStackReserve				SizeOfStackCommit				SizeOfHeapReserve				SizeOfHeapCommit			
0x000000F0	LoaderFlags				NumberOfRvaAndSizes				.edata offset				.edata size			
0x00000100	.idata offset				.idata size				.rsrc offset				.rsrc size			
0x00000110	.pdata offset				.pdata size				attribute certificate offset (image)				attribute certificate size (image)			
0x00000120	.reloc offset (image)				.reloc size (image)				.debug offset				.debug size			
0x00000130	Architecture (reserved - 0x0)				Architecture (reserved - 0x0)				Global Ptr offset				must be 0x0			
0x00000140	.tls offset				.tls size				Load config table offset (image)				Load config table size (image)			
0x00000150	Bound import table offset				Bound import table size				IAT (Import address table) offset				IAT (Import address table) size			
0x00000160	Delay import descriptor offset (image)				Delay import descriptor size (image)				CLR runtime header offset (object)				CLR runtime header size (object)			
0x00000170	Reserved (must be 0x0)				Reserved (must be 0x0)				Section header - Name							
0x00000180	VirtualSize				VirtualAddress				SizeOfRawData				PointerToRawData			
0x00000190	Poi[illegible]				[illegible]				[illegible]				[illegible]			
0x000001A0	[illegible]				[illegible]				[illegible]				[illegible]			
0x000001B0	SizeOfRawData				PointerToRawData				PointerToRelocations				PointerToLineNumbers			
0x000001C0	NumberOfRelocations		NumberOfLineNumbers		Characteristics				Section header - Name..							

Bound import table offset
Bound import table size
IAT (Import address table) offset
IAT (Import address table) size

img source: stackoverflow

About PEs

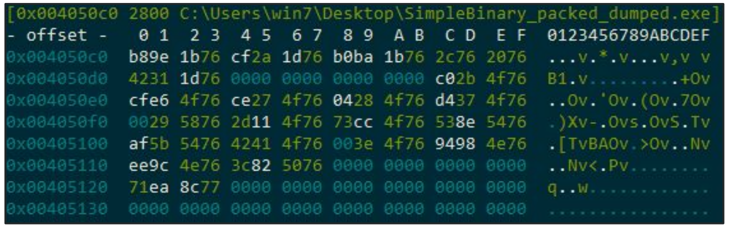
Import Table (IT)

It is a structure defined within the PE header that defines which modules and functions will be imported, and thus used by the process.

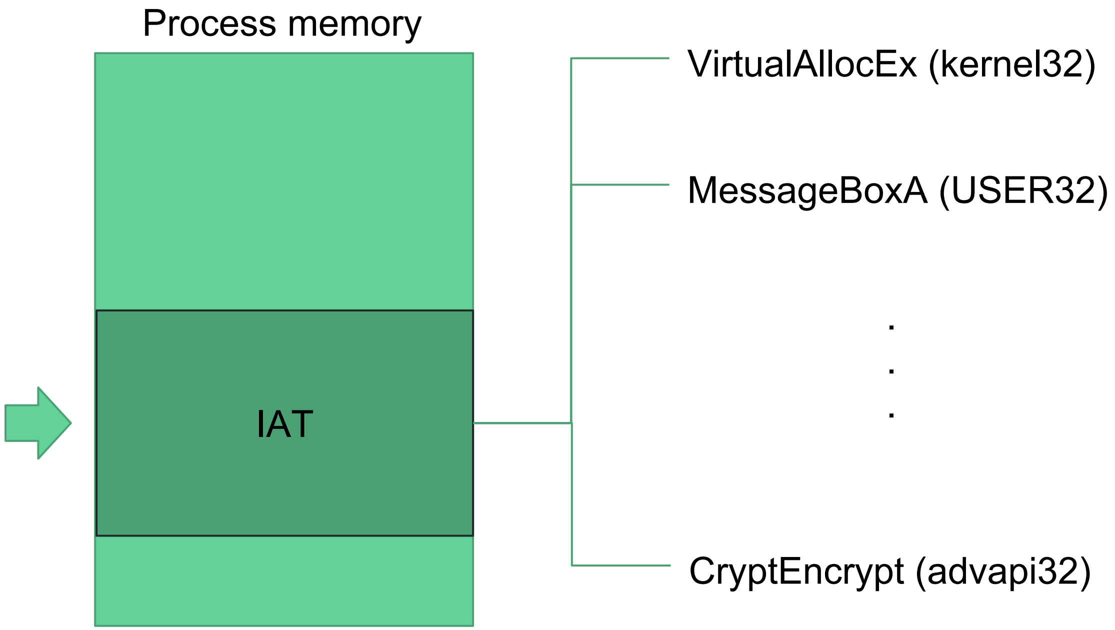
Import Address Table (IAT)

It is a table that the Windows Loader fills up with the virtual addresses where the API functions used by the PE reside into memory (they will not be mapped always to the same addresses). It can be found within the process mapped memory.

About PEs



```
[0x004050c0 2800 C:\Users\win7\Desktop\SimpleBinary_packed_dumped.exe]
- offset -   0 1  2 3  4 5  6 7  8 9  A B  C D  E F  0123456789ABCDEF
0x004050c0  b89e 1b76 cf2a 1d76 b0ba 1b76 2c76 2076  ...v.*.v...v,v v
0x004050d0  4231 1d76 0000 0000 0000 0000 c02b 4f76  B1.v.........+Ov
0x004050e0  cfe6 4f76 ce27 4f76 0428 4f76 d437 4f76  ..Ov.'Ov.(Ov.7Ov
0x004050f0  0029 5876 2d11 4f76 73cc 4f76 538e 5476  .)Xv-.Ovs.OvS.Tv
0x00405100  af5b 5476 4241 4f76 003e 4f76 9498 4e76  .[TvBAOv.>Ov..Nv
0x00405110  ee9c 4e76 3c82 5076 0000 0000 0000 0000  ..Nv<.Pv........
0x00405120  71ea 8c77 0000 0000 0000 0000 0000 0000  q..w............
0x00405130  0000 0000 0000 0000 0000 0000 0000 0000  ................
```



Useful r2 commands for unpacking

Useful r2 commands for unpacking

Follow Execution Flow

r2 -d <file_path>: opens file in r2 for debugging

dc: continues execution

ds: single step

dcr: continues until stack frame return

dr <register>: shows the value of specified register

Useful r2 commands for unpacking

Breakpoints

db <address>: sets software breakpoint on specified address

dbc <address> <r2_cmd>: assigns r2 command to be executed when bp on address is triggered

drx <number> <address> <len> <rwx>: sets hardware breakpoint on specified address range (by address and len) when accessed for read, write or execute

db- <address>: removes software breakpoint on specified address

drx- <number>: removes hardware breakpoint on specified address

Useful r2 commands for unpacking

Memory Maps

dm: lists memory map of target process

dmi <address | libname>: lists loaded DLL symbols

Useful r2 commands for unpacking

Memory Dumping

wtf <filename> <size> @<starting_address>: saves memory region to file

dmd <filename> @<address>: dumps memory map at given address to a file

Useful r2 commands for unpacking

Misc

/A <opcode>: searches for specified opcode on current section

pf p4 @<address>: retrieves 4 bytes of content from pointer at given address

*<address>: equivalent to pf p4 @ <address>

“e cmd.vprompt=px@esp**”**: In visual mode, it shows the stack on top of the view

Useful r2 commands for unpacking

R2 Command Parsing

~<text>**:**<row>**[**<column>**]**: Filters lines matching text, then selects the given row and finally returns the specified column

<cmd>**j**: Several r2 commands support this ending character to output data in json

Techniques and samples

Locky

Locky

Useful Steps to Unpack Locky

- Debug execution flow commands
- Hardware breakpoints
- Regular breakpoints
- List loaded DLL imports

Locky

Useful Commands to Unpack Locky

- dcr
- drx 0 <address> 4 w
- db `<symbol_from_loaded_DLL>`
- dmi kernel32~<symbol>:0[1]
- ds

Locky

VirtualAlloc

Locky

Idea

In order to write the PE into memory, Locky will reserve memory space. To do so, it uses the Windows API function **VirtualAlloc**.

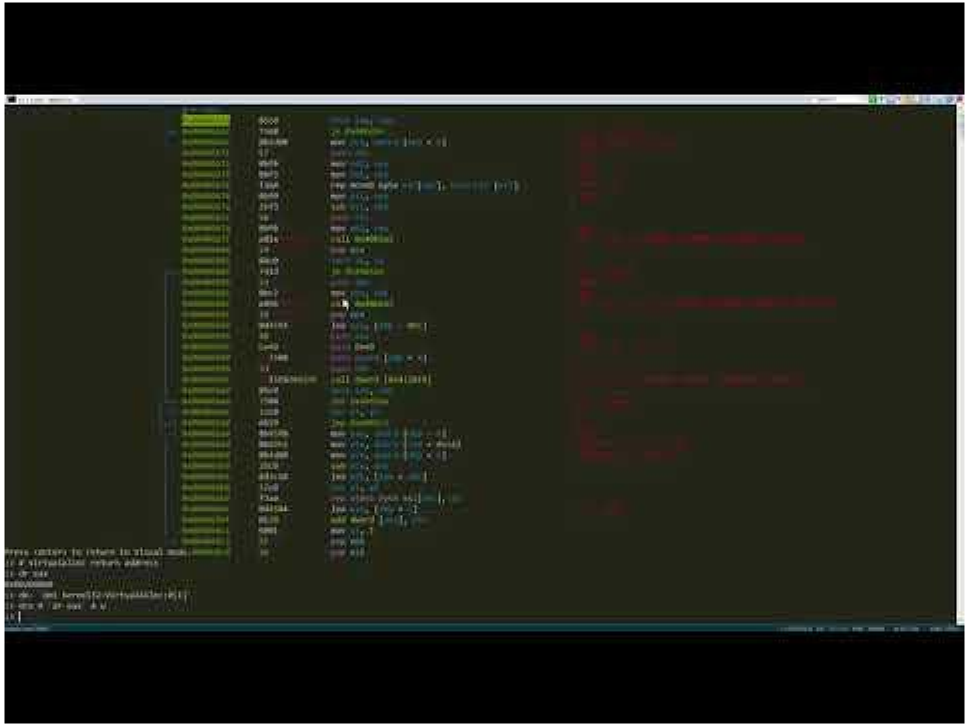
We will put a breakpoint on this function and check if its parameters are appropriate to fit lengthy data (size).

For each VirtualAlloc “suspicious” call, we will set a (write) hardware breakpoint on its return memory address. We will check if the PE header (starting with “MZ”) is written starting at that address.

Locky

```
dc
db `dmi kernel32~VirtualAlloc:0[1]`
dc ──────────────────────────────────→ 10 times
V
[0x76202fb6]> pf p4 @ esp + 8~[1]
0x0001ae00 ──────────────────────────→ VirtualAlloc Size
[0x76202fb6]> *esp + 16
0x00000004 ──────────────────────────→ VirtualAlloc Permissions
dcr
s
drx 0 `dr eax` 4 w ────────→ eax contains VirtualAlloc returned memory address
dc
```

Locky



```
Press <enter> to return to Visual mode.
:> # VirtualAlloc return address
:> dr eax
[illegible]
:> [illegible]
:> [illegible]
:>
```

Dridex Dropper

Dridex Dropper

Useful Steps to Unpack Dridex Dropper

- Debug execution flow commands
- Regular breakpoints
- List loaded DLL imports

Dridex Dropper

Useful Commands to Unpack Dridex Dropper

- dc
- db `<symbol_from_loaded_DLL>`
- dmi ntdll~<symbol>:0[1]
- *<register+offset>

Dridex Dropper

RtlDecompressBuffer

Dridex Dropper

Idea

As with Locky, the Dridex Dropper uses VirtualAlloc in order to reserve memory space where the code will write the PE.

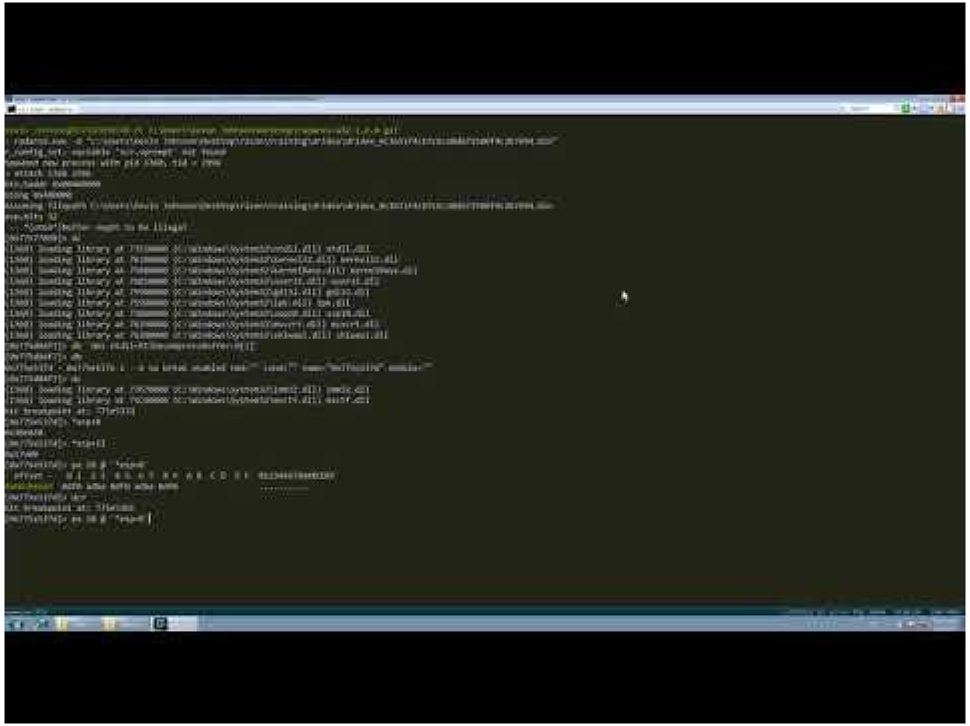
However the Windows API function **RtlDecompressBuffer** is used to fill the buffer that will contain the PE, so there is no need to search on all memory regions reserved by VirtualAlloc as shown with Locky.

This procedure is valid for unpacking Locky too.

Dridex Dropper

```
dc
db `dmi ntdll ~RtlDecompressBuffer:0[1]`
dc
dcr
*esp+8
*esp+12
wtf rtl_dump.exe `*esp+12`@`*esp+8`
```

Dridex Dropper



```
[illegible]
```

UPX #1

UPX #1

Useful Steps to Unpack UPX #1

- Context switching instructions
- Hardware breakpoints on read

UPX #1

Useful Commands to Unpack UPX #1

- drx 0 <address> 4 r
- db <address>
- drx-
- dc

UPX #1

pushal

UPX #1

Idea

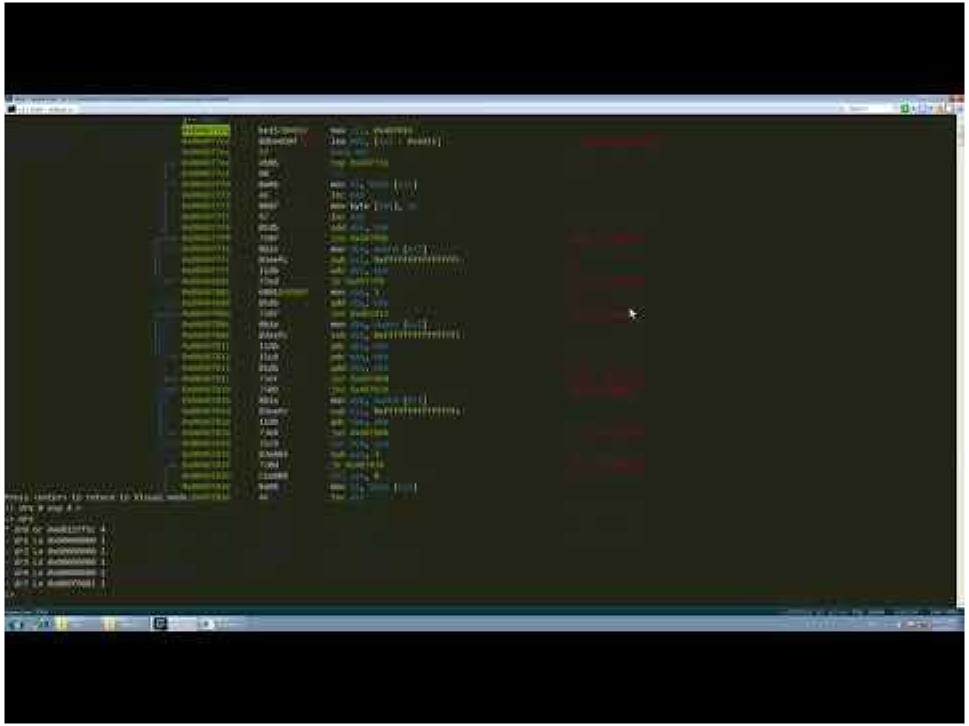
Having a context switching instruction (i.e. **pushal**) at the beginning of the stub code is suspicious.

The **pushal** instruction stores all the registers into the stack so probably, at some point near the original code, it will restore those values back into the registers.

UPX #1

```
dc
db entry0
dc
ds
drx 0 esp 4 r
dc
db 0x00407953
drx-
dc
ds
```

UPX #1



```
[illegible]
```

UPX #2

UPX #2

Useful Hints to Unpack UPX #2

- Long jumps (not always though)
- Jumps with nonsense code afterwards

UPX #2

Useful Commands to Unpack UPX #2

- /x <bytes>
- e search.in = io.section

UPX #2

Spotting the tail jump

UPX #2

Idea

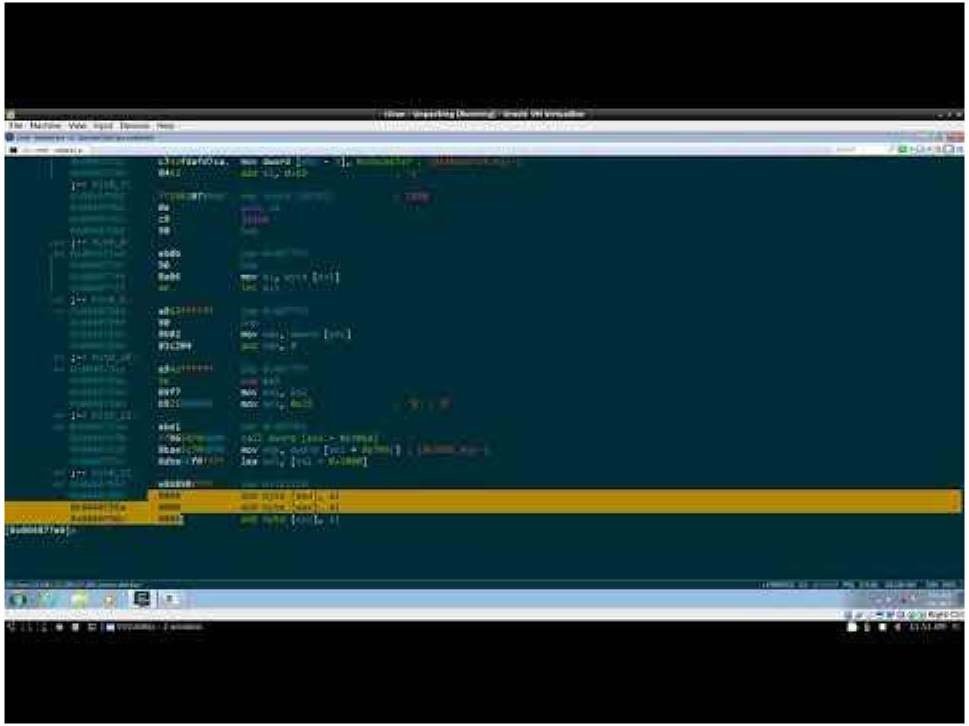
The tail jump is the last instruction (a jump) executed within the stub before executing the original decompressed code.

Usually, the jump to the original code will be a long jump, and probably it will have a bunch of nonsense code afterwards, since it will never be executed.

UPX #2

```
dc
db entry0
dc
/A jmp
pd 4 @ hit*
<look for a jump with nonsense opcodes after>
```

UPX #2



UPX #3

UPX #3

Useful Hints to Unpack UPX #3

- API functions called by the original malware
- Breakpoints on API functions of loaded DLLs
- Function prologue

UPX #3

Useful Commands to Unpack UPX #3

- db <address>
- dmi <dll_name>~<symbol>:0[1]
- dcr

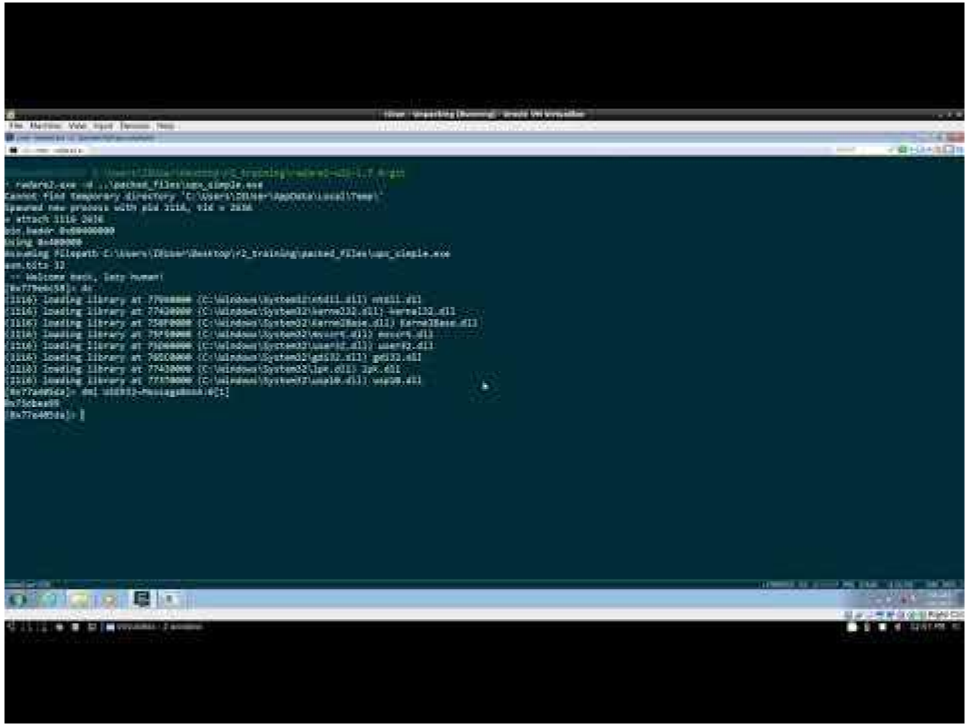
UPX #3

Breaking on common imports

UPX #3

```
dc
db `dmi USER32~MessageBox:0[1]`
dc
dcr
ds
<spot the function prologue>
```

UPX #3



```
[illegible] C:\Users\IEUser\Desktop\r2_training\radare2-w32-1.7.0-git
> radare2.exe -d ..\packed_files\upx_simple.exe
Cannot find temporary directory 'C:\Users\IEUser\AppData\Local\Temp\'
Spawned new process with pid 1116, tid = 2636
= attach 1116 2636
bin.baddr 0x00400000
Using 0x400000
asm.bits 32
 -- Welcome back, lazy human!
[0x779eb158]> dc
(1116) loading library at 77000000 (C:\Windows\System32\ntdll.dll) ntdll.dll
(1116) loading library at 77420000 (C:\Windows\System32\kernel32.dll) kernel32.dll
(1116) loading library at 750F0000 (C:\Windows\System32\KernelBase.dll) KernelBase.dll
(1116) loading library at 75F50000 (C:\Windows\System32\msvcrt.dll) msvcrt.dll
(1116) loading library at 75D00000 (C:\Windows\System32\user32.dll) user32.dll
(1116) loading library at 765C0000 (C:\Windows\System32\gdi32.dll) gdi32.dll
(1116) loading library at 77430000 (C:\Windows\System32\lpk.dll) lpk.dll
(1116) loading library at 77370000 (C:\Windows\System32\usp10.dll) usp10.dll
[0x77a405da]> dml user32-MessageBoxA:0[1]
0x75cbea95
[0x77a405da]>
```

Dealing with the IAT

Dealing with the IAT

Problem

UPX and other packers stubs do not decompress the whole PE into memory. Instead they just decompressed all the sections of the PE, and they resolve all the API calls the original code used (i.e. they load the necessary modules, fix the references. etc).

To obtain a valid PE with the “original” code we need to:

- Find the IAT
- Dump the process
- Fix the dump (Fix IAT and Rebuild PE)

Finding the IAT

Once the OEP has been spotted, the IAT will be somewhere in the memory.

Since the IAT contains references to API functions, the following steps could be followed:

- Get all the modules that the process has loaded (and the addresses of the functions they expose)
- Find an API pointer within the code (either call or jmp instructions)
- We now have a pointer to an API reference within the IAT
- Scan backwards and forward until no more API references are found, and we've found the start and end of the IAT (and also the size).

Dumping the process

Once the IAT was found, we need to dump the process memory to disk.

The dump has to be a PE (not necessarily valid yet), so we can use the original PE structure, and dump the memory into it.

Alternatively, we could create a new PE if we have that capability.

Fixing the dump

Once we have a PE, we need to make it valid:

- Update all references on the code to API functions to point to the new IAT (Fix IAT)
- Realign (Rebuild PE)
- Rebase (Rebuild PE)
- Update header checksum (Rebuild PE)

All this process is tedious and would be repetitive, so tools like Scylla now come handy

Scylla

It is well known import reconstruction tool for x86/x64 (https://github.com/NtQuery/Scylla)

It automatizes all the process already described, and it’s GPLv3 licensed!

However, it’s full GUI based and does not integrate very well with the radare2 workflow.

Scylla

R2SCYLLA

R2SCYLLA

Scylla plugin for radare2

It allows full use of Scylla without the need of the Scylla GUI, integrating with the radare2 workflow.

Written in Python, uses the DLL compiled version of Scylla via CTypes.

R2SCYLLA

Install

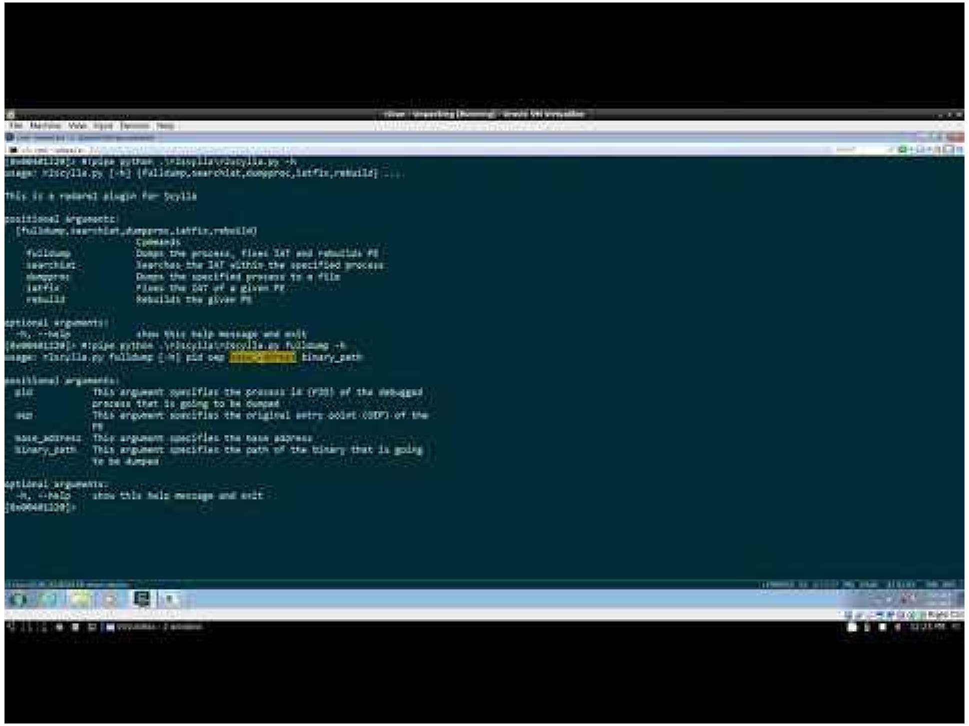
git clone https://github.com/zlowram/r2scylla

Usage

```
#!pipe python r2scylla.py fulldump -h
```

```
usage: r2scylla.py fulldump [-h] pid oep base_address binary_path
```

R2SCYLLA



```
[0x00401120]> #!pipe python .\r2scylla\r2scylla.py -h
usage: r2scylla.py [-h] {fulldump,searchiat,dumpproc,iatfix,rebuild} ...

This is a radare2 plugin for Scylla

positional arguments:
  {fulldump,searchiat,dumpproc,iatfix,rebuild}
                        Commands
    fulldump            Dumps the process, fixes IAT and rebuilds PE
    searchiat           Searches the IAT within the specified process
    dumpproc            Dumps the specified process to a file
    iatfix              Fixes the IAT of a given PE
    rebuild             Rebuilds the given PE

optional arguments:
  -h, --help            show this help message and exit
[0x00401120]> #!pipe python .\r2scylla\r2scylla.py fulldump -h
usage: r2scylla.py fulldump [-h] pid oep base_address binary_path

positional arguments:
  pid           This argument specifies the process id (PID) of the debugged
                process that is going to be dumped
  oep           This argument specifies the original entry point (OEP) of the
                PE
  base_address  This argument specifies the base address
  binary_path   This argument specifies the path of the binary that is going
                to be dumped

optional arguments:
  -h, --help    show this help message and exit
[0x00401120]>
```