

# Quick reference for manual unpacking

2012-04-01

Abhishek Singh

FireEye, USA

**Editor:** Helen Martin

## Abstract

By packing their malicious executables, malware authors can be sure that when they are opened in a disassembler they will not show the correct sequence of instructions, thus making malware analysis a more lengthy and difficult process. Abhishek Singh provides a quick reference guide for unpacking malware from some of the most commonly used packers.

*Copyright © 2012 Virus Bulletin*

## Table of contents

### Manual unpacking

ASPack

KKrunchy

PECompact v2.x

NSPack

FSG 1.33

FSG 2.0

UPX

PEDiminisher

MEW

### Conclusion

Malware authors utilize packers to make it difficult for their malware to be reversed – the packers encode the original instructions. By packing a malicious executable, its author can be sure that when it is opened in a disassembler it will not show the correct sequence of instructions. Packers add some instructions at the top of the binary to unpack the executable. The process of decryption is performed in memory at run time, and the state of the application is restored. Since packers work on a compiled executable, the unpacking module must be independent of the original application.

One of the methods that can be used to locate the original entry point (OEP) of the file is to apply break points on the following APIs:

```
GetLoadLibraryA  
GetVersionExA  
GetEnvironmentA  
LoadLibraryA  
GetProcAddress  
IniHeap
```

These APIs are called by the packers' start-up routines in order to set up the execution environment. When a breakpoint is applied to these routines, we are close to the OEP. When the break point triggers, we can use step-by-step tracing to locate the initialization of the stack frame. The start of the function can be recognized by the initialization of the stack frame.

```
push ebp  
mov  ebp, esp
```

The instructions shown above denote the start of the stack frame. Once these instructions are located, the debugged process can be dumped to obtain the unpacked version of the file.

In the following sections we describe some common packers and the assembly instructions that can be used to locate the OEP.

## Manual unpacking

The purpose of this section is to provide a quick reference guide that will assist malware analysts in the unpacking of malware and reduce the response time for malware analysis – the full technical details of each packer have therefore been omitted.

### ASPack

ASPack is an advanced *Windows 32* executable compressor capable of reducing the file size of 32-bit *Windows (95/98/ME/NT/2000/XP/2003/Vista/7)* programs by as much as 70%. It is also used by some hackers to protect their programs.

To unpack ASPack, follow the first `jmp`, and follow `JMP EAX`. Later in the code you will find the following instructions:

```
mov  eax,1  
retn 0C  
push 0  
retn
```

Once these instructions have been identified, as shown in **Figure 1**, a break point should be put on `RETN`. When the break point triggers, we are at the OEP. The process can be dumped at this stage, leaving us with the unpacked executable.

0038B415	61	POPAD
0038B416	75 00	JMP SHORT crackn_1.003
0038B417	8B 01000000	MOV EAX, 1
0038B418	C2 0C00	RETN 0C
0038B420	68 00000000	PUSH 0
0038B421	C3	RETN
0038B426	8B85 00000000	MOV EAX, DWORD PTR SS:[
0038B42C	8D8D A1040000	LEA ECX, DWORD PTR SS:[
0038B432	51	PUSH ECX
0038B433	50	PUSH EAX
0038B434	FF95 A50F0000	CALL DWORD PTR SS:[EBP
0038B43A	8985 B1050000	MOV DWORD PTR SS:[EBP+
0038B440	8D85 AD040000	LEA EAX, DWORD PTR SS:[
0038B446	50	PUSH EAX
0038B447	FF95 AD0F0000	CALL DWORD PTR SS:[EBP
0038B44D	8985 90040000	MOV DWORD PTR SS:[EBP+
0038B453	8D8D B8040000	LEA ECX, DWORD PTR SS:[
0038B459	51	PUSH ECX
0038B45A	50	PUSH EAX
0038B45B	FF95 A50FA000	CALL DWORD PTR SS:[EBP

**Figure 1. Instructions before the code is unpacked.**

OlyScript code for the automatic unpacking of ASPack is shown in **Figure 2**. The instruction 'findop eip, #6800000000#' locates the PUSH 0 instruction in a debugged process packed with ASPack. Once this instruction is located, the debugger steps once to reach the RETN instruction. The debugger then steps again to reach the OEP instruction. Once the OEP instruction is located the debugger steps once more to reach the OEP. The debugged process can now be dumped to get the unpacked version of the file.

```
findop eip, #6800000000#
go $RESULT
sti
sti
msg "OEP is found for ASPack "
run
```

**Figure 2. OllyScript code used to locate the OEP for ASPack.**

## KKrunchy

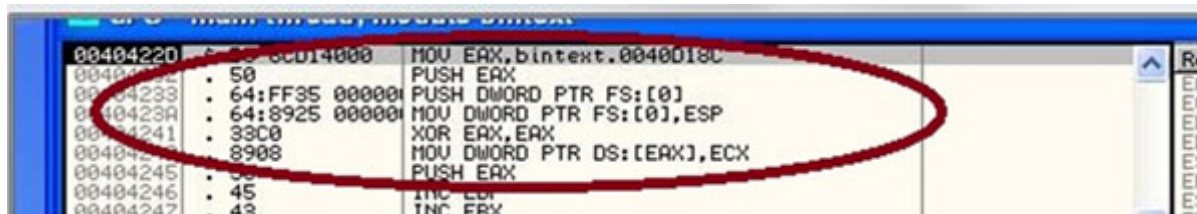
KKrunchy [1] is a small executable packer intended for 64k intros. It does not try to pack DLLs and cannot handle exports or TLS. It performs a transform on the input code to allow it to compress better. It will fill uninitialized data sections with zeros and then pack them together with the rest of the code. KKrunchy is often used by malware authors to prevent AV analysts from reversing their code.

In order to unpack KKrunchy, put a break point on LoadLibraryA. When the break point triggers, step the debugger and search for the initialization of the stack frame. Once the stack frame initialization is complete, dump the debugged process. The dumped process is the unpacked version of the executable.

## PECompact v2.x

PECompact [2] is fully compatible with DEP and code signing, and provides support for *Windows 7* and *Windows 2008*. It provides a good compression ratio compared to other compressors such as ASPack. The PECompact [3] loader consists of three components. The first is the SEH entry, which transfers control to the second component, the loader decoder. The loader decoder decodes the code and invokes the third component, the primary loader. The loader decoder is stored in the last section (or the second-to-last section if relocations have been preserved). The primary loader exists in uncompressed form at runtime in dynamically allocated memory. To hide the transfer of control, an SEH frame is set up and there is an exception. The exception handler then modifies the code at the exception address to a JMP and continues execution.

Figure 3 shows PECompact's exception handler. The instruction sequence 'PUSH EAX, PUSH DWORD PTR FS:[0], MOV DWORD PTR FS:[0], ESP' sets up the SEH frame. The instruction 'XOR EAX, EAX' sets the value in EAX to zero. The instruction 'MOV DWORD PTR DS:[EAX], ECX' triggers the exception.




```

00404220 55 8CD14000 MOV EAX,bintext.0040018C
00404222 50          PUSH EAX
00404223 64:FF35 000000 PUSH DWORD PTR FS:[0]
0040422A 64:8925 000000 MOV DWORD PTR FS:[0],ESP
00404241 33C0       XOR EAX,EAX
00404242 8908       MOV DWORD PTR DS:[EAX],ECX
00404245 50          PUSH EAX
00404246 45          INC EBP
00404247 43          INC EBX
  
```

Figure 3. The PECompact exception handler.

To unpack PECompact, follow the exception and step through the code until the instructions shown in Figure 4 are observed. JMP EAX is the jump to the OEP.

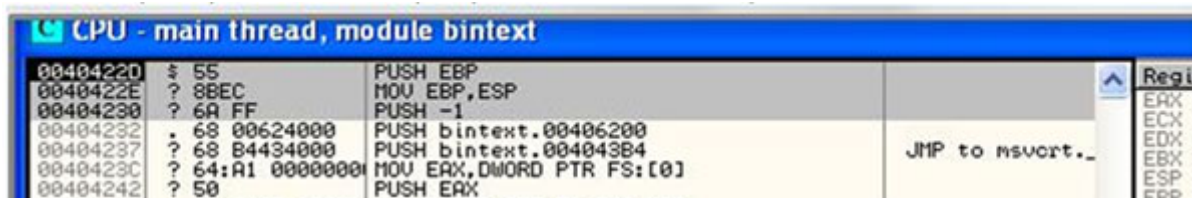


```

00404246 8B77       MOV EAX,ESI
00404249 5A          POP EDX
0040424A 5E          POP ESI
0040424B 5F          POP EDI
00404248 59          POP ECX
0040424C 5B          POP EBX
0040424E 5D          POP EBP
0040424E FFEB       JMP EAX
00404250 2D 42400000 CMP EAX,42400000
00404255 74 00000000 JNC EAX+00000000
  
```

Figure 4. PECompact instructions before unpacking.

Set a break point on JMP EAX, step once, and observe the initialization of the stack frame as shown in Figure 5. Dump the process. The dumped process will be the unpacked executable.



```

CPU - main thread, module bintext
00404220 55          PUSH EBP
0040422E 8BEC       MOV EBP,ESP
00404230 6A FF     PUSH -1
00404232 68 00624000 PUSH bintext.00406200
00404237 68 B4434000 PUSH bintext.004043B4
0040423C 64:A1 00000000 MOV EAX,DWORD PTR FS:[0]
00404242 50          PUSH EAX
  
```

Figure 5. Initialization of the stack frame.

The logic shown in Figure 4 can be converted into script such as that shown in Figure 6 (the script is available from Open RCE [4]).

```

sto
sto
sto
sto
sto
sto
esto
find eip, #8BC65A5E5F595B5DFFE0#
add $RESULT,08
bp $RESULT
run
bc $RESULT
sto
cmt eip, "This is a OEP!"
msg "OEP found, Dumped and fix IAT now!"
ret
  
```

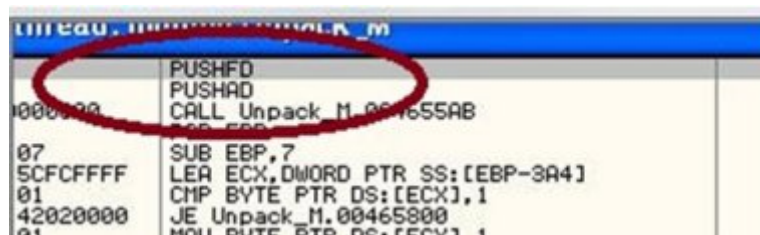
Figure 6. OllyScript for PECompact.

The instruction 'find eip, #8BC65A5E5F595B5DFFFE0#' locates the instructions 'MOV EAX ESI, POP EDX, POP ESI, POP EDI, POP ECX, POP EBX, POP EBP, JMP EAX'. Once these are located, the script steps once at the JMP instruction and the debugger is at the OEP. The debugged process now can be dumped to obtain the unpacked version of the file.

## NSPack

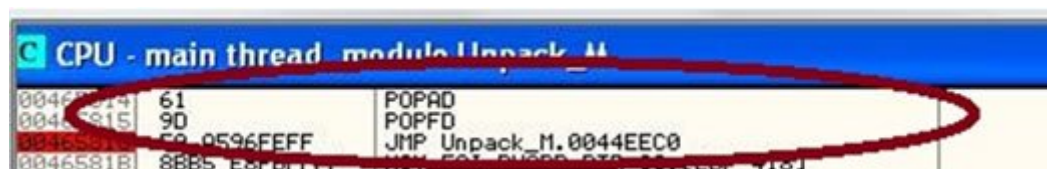
NSPack [5] is capable of compressing EXE, DLL, OCX and SCR files. It also has the ability to compress 64-bit executables. It provides support to compress files packed by other packers such as UPX, ASPack and PECompact. It supports direct compression of directories or multiple files. This packer is quite commonly used by malware authors.

As shown in Figure 7, the packer starts with the instructions PUSHFD, PUSHAD.



**Figure 7. The starting instructions for NSPack.**

Check for equivalent POPAD and POPFD instructions, as shown in Figure 8. The JMP instruction follows. Put a break point on the JMP instruction. When the break point triggers, step once and dump the process to obtain the unpacked file.



**Figure 8. NSPack instructions before unpacking.**

The abovementioned logic can be converted into the OllyScript shown in Figure 9. The instruction 'find eip, #619DE9#' locates the instruction POPAD, followed by POPFD, followed by a JMP instruction. Once these are located, the code is debugged, step by step, until the JMP instruction is executed – the debugger has then reached the OEP instruction. By using a plug-in like OllyDump, the process can be dumped to obtain the unpacked version of the file.

```
sti
find eip, #619DE9#
go $RESULT
sto
sto
sto
msg "OEP for NSPack "
```

**Figure 9. OllyScript used to locate the OEP for NSPack.**

## FSG 1.33

FSG stands for Fast Small and Good, and is currently used to pack various malware. It was originally created to pack assembly demos. Since it has a small loader, it is one of the most desirable packers for small executables.

In order to obtain the unpacked executable file for FSG 1.33, put a break point on the LoadLibraryA function, as shown in **Figure 10**.

```

7C801D78 8BFF      MOV EDI,EDI
7C801D7D 55       PUSH EBP
7C801D7E 8BEC     MOV EBP,ESP
7C801D80 837D 08 00 CMP DWORD PTR SS:[EBP+8],0
7C801D84 53      PUSH EBX
7C801D85 56      PUSH ESI
7C801D86 v74 14   JE SHORT kernel32.7C801D9C
7C801D88 68 50E1807C PUSH kernel32.7C80E150
7C801D8D FF75 08  PUSH DWORD PTR SS:[EBP+8]
7C801D90 FF15 A813807C CALL DWORD PTR DS:[kernel32.ntdll._stricmp]
7C801D96 85C0     TEST EAX,EAX
7C801D98 59      POP ECX
7C801D99 59      POP ECX
7C801D9A v74 12   JE SHORT kernel32.7C801DAE
7C801D9C 6A 00   PUSH 0
7C801D9E 6A 00   PUSH 0
7C801DA0 FF75 08  PUSH DWORD PTR SS:[EBP+8]
7C801DA3 E8 ABFFFFFF CALL kernel32.LoadLibraryExA
7C801DA8 5E      POP ESI
7C801DA9 5B      POP EBX
7C801DAA 5D      POP EBP
7C801DAB C2 0400  RFTN 4

```

**Figure 10. The LoadLibraryA function in FSG 1.33.**

When the break point triggers, step a few instructions below until the following instructions are seen:

```

dec byte ptr [esi]
jz xxxxxxxx
PUSH ESI
PUSH EDP
CALL DWORD PTR DS:[EBX+4]

```

When JE Address triggers, we can observe the initialization of the stack frame. We are at the OEP, so dump the process to get the unpacked version of the file.

```

00478046 ^75 FB   JNZ SHORT unpackme.00478043
00478048 FE0E    DEC BYTE PTR DS:[ESI]
0047804A ^74 F0   JE SHORT unpackme.0047803C
0047804C v79 05   JNS SHORT unpackme.00478053
0047804E 46     INC ESI
0047804F AD     LODS DWORD PTR DS:[ESI]
00478050 50     PUSH EAX
00478051 vEB 09   JNZ SHORT unpackme.0047805C
00478053 FE0E    DEC BYTE PTR DS:[ESI]
00478055 -0F84 A582FCFF JE unpackme.00440300
00478058 56     PUSH ESI
0047805C 55     PUSH EBP
0047805E FF53 04 CALL DWORD PTR DS:[EBX+4]
00478060 AB     STOS DWORD PTR ES:[EDI]
00478061 ^EB E0   JNZ SHORT unpackme.00478043
00478063 33C9   XOR ECX,ECX
00478065 41     INC ECX
00478066 FF13   CALL DWORD PTR DS:[EBX]
00478068 1300   INC ECX

```

**Figure 11. Instructions denoting the end of FSG.**

## FSG 2.0

For version 2.0 of the FSG packer, the instructions that indicate the end of the FSG stub are as follows:

```

move eax (edi)
inc eax
js address
jnz address
jmp dword ptr [ebx+0Ch]

```

In order to manually unpack a file packed with FSG 2.0, put a break point on LoadLibraryA and execute the compressed file. When it breaks, clear the break point and execute until return (Ctrl -f9). Step through the debugged application until the instructions shown in **Figure 12** are reached.

Address	Disassembly	Registers (FPU)
004001C3	97 XCHG EAX,EDI	EAX 00000000
004001C4	AD LODS DWORD PTR DS:[ESI]	ECX 7C917DE9 ntdll.7C917DE9
004001C5	50 PUSH EAX	EDX 7C97B178 ntdll.7C97B178
004001C6	FF53 10 CALL DWORD PTR DS:[EBX+10]	EBX 00411358 bintext.00411358
004001C9	95 MOV EAX,EBX	ESP 012FFC4
004001CA	40 MOV EAX,DWORD PTR DS:[EDI]	EBP 77C00000 nsvort.77C00000
004001CB	40 INC EAX	ESI 00406294 bintext.00406294
004001CD	78 F3 JS SHORT bintext.004001C2	EDI 00405114 bintext.00405114
004001CE	75 03 JNZ SHORT bintext.004001D4	EIP 004001D1 bintext.004001D1
004001D1	FF53 10 JMP DWORD PTR DS:[EBX+C]	C 0 ES 0023 32bit 0(FFFFFFFF)
004001D4	50 PUSH EBP	P 1 CS 001B 32bit 0(FFFFFFFF)
004001D5	55 PUSH EBP	A 1 SS 0023 32bit 0(FFFFFFFF)
004001D6	FF53 14 CALL DWORD PTR DS:[EBX+14]	Z 1 DS 0023 32bit 0(FFFFFFFF)
004001D9	AB STOS DWORD PTR ES:[EDI]	S 0 FS 003B 32bit 7FDD0000(FI)
004001DA	EB EE JMP SHORT bintext.004001CA	T 0 GS 0000 NULL
004001DB	33C9 XOR ECX,ECX	O 0
004001DC	41 INC ECX	D 0
004001DE	FF13 CALL DWORD PTR DS:[EBX]	O 0 LastErr ERROR_MOD_NOT_FOUND
004001DF	13C9 ADC ECX,ECX	EFL 00000256 (NO,NB,E,BE,NS,PI)
004001E1	13C9 ADC ECX,ECX	
004001E3	FF13 CALL DWORD PTR DS:[EBX]	
004001E5	72 F8 JB SHORT bintext.004001DF	

**Figure 12. Instructions reached before unpacking FSG 2.0.**

Here, 'JMP DWORD PTR DS: PTR [ebx+0Ch]' is the jump to OEP. Once the JMP instruction is executed, dump the process to get the unpacked version of the file.

## UPX

UPX [6] stands for Ultimate Packer for eXecutables. It offers an excellent compression ratio which is better than WinZip, Zip and GZIP. It also maintains a checksum for both compressed and uncompressed files. It uses compression algorithms like UCL [7]. UCL has the inherent advantage that the decompressor can be implemented in a few hundred bytes of code. Many malware families such as Qakbot are packed using UPX. It offers very fast compression and decompression speeds: ~10MB/s on a *Pentium* 133. It also offers support for LZMA compression and has support for BSD. LZMA decompression is disabled on the 16-bit platform due to the slow decompression speed on older platforms. It also provides support for two types of decompression routines. The first is the in-place technique, which decompresses the executable in memory. In-place decompression is possible only for some platforms. The extraction of a temporary file, even though it uses extra overhead, allows any executable file format to be packed.

In order to unpack UPX using a manual approach, the end of the UPX routine must be identified. The end of the UPX routine can be identified by the instructions CALL, POPAD and JMP, as shown in **Figure 13**. Put a break point on the JMP instruction. The JMP instruction will lead to initialization of the stack frame. After the JMP instruction has executed, dump the process by using a plug-in such as OllyDump, and the program is unpacked.

Address	Disassembly	Registers (FPU)
00889F	74 07 JE SHORT tnnbtib.004088A8	EAX 00000000
0088A1	8903 MOV DWORD PTR DS:[EBX],EAX	ECX 0012FFB0
0088A3	83C3 04 ADD EBX,4	EDX 7C90E4F4 ntdll.K
0088A6	EB 51 JMP SHORT tnnbtib.00408899	EBX 7FFDC000
0088A9	FF96 94800000 CALL DWORD PTR DS:[ESI+8071]	ESP 0012FFC4
0088AE	> 61 POPAD	EBP 0012FFF0
0088AF	-E9 1789FFFF JMP tnnbtib.004011CB	ESI FFFFFFFF
0088B0	DB 00 DB 00	EDI 7C910208 ntdll.7
0088B5	DB 00 DB 00	
0088B6	DB 00 DB 00	

**Figure 13. UPX end of routine instructions.**

```
var BPforPOPAD
var BPforJMP
findop eip, #61#
mov BPforPOPAD, $RESULT
bp BPforPOPAD
run
findop eip, #E9?????????#
mov BPforJMP, $RESULT
bp BPforJMP
run
sti
msg "OEP for UPX. Dump the Process"
```

**Figure 14. The OllyScript used to unpack UPX.**

The script shown in **Figure 14** is the implementation of the logic used to locate the OEP. The instruction ‘findop eip, #61#’ locates the assembly instruction POPAD, sets a break point on it, and then executes the code packed with UPX. Once the break point is triggered, the instruction ‘findop eip, #E9?????????#’ locates the JMP instruction and sets a break point on it. When the break point triggers, the debugger steps once in the code and is at the OEP. The debugged process can be dumped to get the unpacked version of the file.

## PEDiminisher

PEDiminisher is a simple PE packer. It uses the aplib compression/decompression library. Many AV engines have the ability to unpack files packed with PEDiminisher to check for malicious content.

The end routine for PEDiminisher is shown below:

```
pop EBP
POP EDI
POP ESI
POP EDX
POP ECX
POP EBX
JMP EAX
```

For unpacking, the end instructions must first be located in the packed file (as shown in **Figure 15**). JMP EAX is the jump to the OEP. Set a break point at the JMP instruction, step once and then dump the process to get the unpacked version of the file.



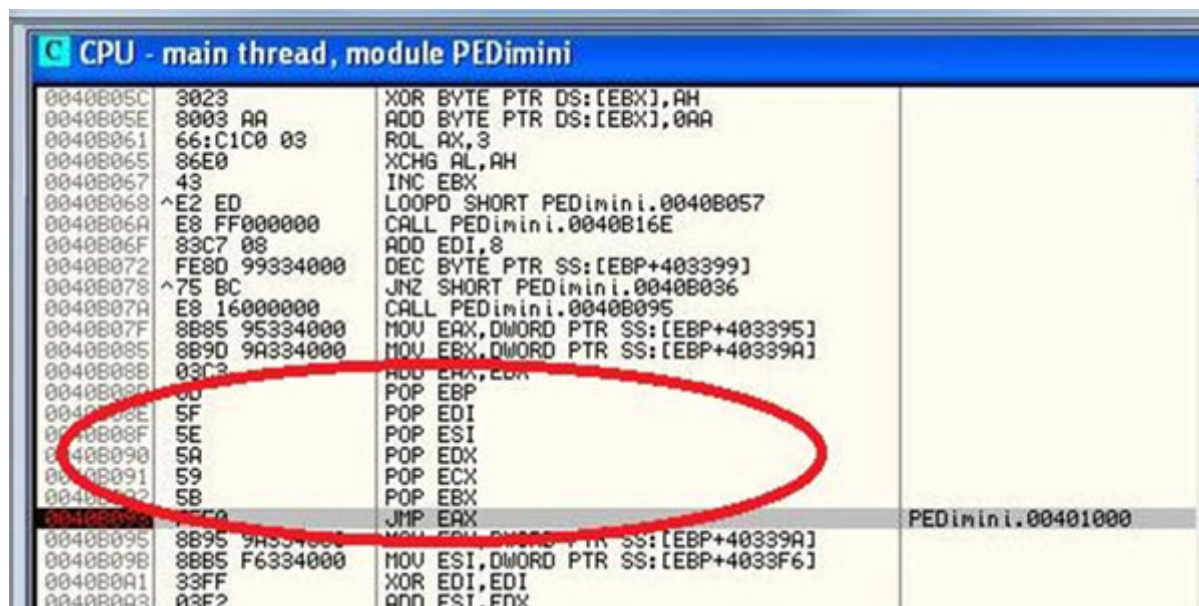


Figure 15. The end instruction for PEDiminisher.

```
find eip, #5D5F5E5A95BFFE0#
bp $RESULT
run
sti
sti
sti
sti
sti
sti
sti
msg "OEP found for PEDiminisher, Dump now!"
```

Figure 16. The OllyScript used to unpack PEDiminisher.

The instruction 'find eip, #5D5F5E5A95BFFE0#' locates the instructions 'POP EBP, POP EDI, POP ESI, POP EDX, POP ECX, POP EBX, JMP EAX'. The script then steps through the debugger until it reaches JMP EAX. Once it is at JMP EAX, the code steps once and is at the OEP. The OllyDump plug-in can be used to dump the process and we are left with the unpacked version of the executable file.

## MEW

MEW [8] is an executable tool which was designed to handle small files. It works on 32-bit workstations and uses the LZMA algorithm. It strips reloc tables, Delphi resources, and unused resources. Even though it was designed to handle small files, it can compress large files as well.

The last instruction in the MEW stub, as shown in Figure 17, is RETN. After this instruction a jump to the OEP takes place. Set a break point on the RETN instruction. When the break point is triggered, as shown in Figure 17, step once and then dump the process to get the unpacked version of the file.

```

004001E6 40      INC EAX
004001E7 59      POP ECX
004001E8 ^74 EC  JE SHORT XORSearch.004001D6
004001EA v79 07  JNS SHORT XORSearch.004001F3
004001EC AC      LODS BYTE PTR DS:[ESI]
004001ED 3C 00   CMP AL,0
004001EF ^75 FB  JNZ SHORT XORSearch.004001EC
004001F1 91      XCHG EAX,ECX
004001F2 40      INC EAX
004001F3 50      PUSH EAX
004001F4 55      CALL DWORD PTR DS:[EBX-C]
004001F8 AB      STOS DWORD PTR ES:[EDI]
004001F9 85C0   TEST EAX,EAX
004001FB ^75 E5  JNZ SHORT XORSearch.004001E2
004001FD 00      RETN
004001FE 0000   ADD BYTE PTR DS:[EAX],AL
00400200 0000   ADD BYTE PTR DS:[EAX],AL
00400202 0000   ADD BYTE PTR DS:[EAX],AL
00400204 0000   ADD BYTE PTR DS:[EAX],AL
00400206 0000   ADD BYTE PTR DS:[EAX],AL
00400208 0000   ADD BYTE PTR DS:[EAX],AL
0040020A 0000   ADD BYTE PTR DS:[EAX],AL
0040020C 0000   ADD BYTE PTR DS:[EAX],AL
0040020E 0000   ADD BYTE PTR DS:[EAX],AL
00400210 0000   ADD BYTE PTR DS:[EAX],AL
00400212 0000   ADD BYTE PTR DS:[EAX],AL
00400214 0000   ADD BYTE PTR DS:[EAX],AL

```

**Figure 17. The last instructions for the MEW packer.**

The logic used to locate the OEP for MEW is shown in **Figure 18**. The code ‘findop eip, #C3#’ locates the RETN instruction in the debugged process packed with the MEW packer. Once the RETN instruction is located, the debugger steps once and is at the OEP. The OllyDump plug-in can be used to dump the process and we are left with the unpacked version of the executable file.

```

sti
findop eip, #C3#
go $RESULT
sto
sto
msg "OEP found for MEW"

```

**Figure 18. The OllyScript used to unpack MEW.**

## Conclusion

Reducing the time it takes to perform malware analysis is very important. For static analysis of malware it is important that the malware is unpacked. There are many approaches to unpacking a piece of malware – for example, it can be executed in a virtual environment and then we can capture a memory snapshot of the executing malware. Once we get the snapshot, we can dump the unpacked malware directly from memory. However, it is possible that not all of the code of the unpacked malware will be in memory, so dumping a process from memory might not be an effective unpacking method. Loading a packed malicious executable and executing step by step instructions in a debugger is one of the best ways to locate the OEP and execute the malware. In this article we have provided assembly instructions for the most commonly used packers which can be used to quickly unpack malware. We have also provided OllyScripts for the logic to manually unpack the malware. This can further aid in reducing response time for malware analysis.

## Bibliography

- [1] <http://www.farbrausch.de/~fg/kkrunchy/> (<http://www.farbrausch.de/~fg/kkrunchy/>).
- [2] <http://pecompact.com/pecompact.php> (<http://pecompact.com/pecompact.php>).
- [3] <http://www.bitsum.com/pec2av.htm> (<http://www.bitsum.com/pec2av.htm>).

[4] [http://www.openrce.org/downloads/details/156/PECompact\\_v.2.40\\_-\\_OEP\\_finder](http://www.openrce.org/downloads/details/156/PECompact_v.2.40_-_OEP_finder) ([http://www.openrce.org/downloads/details/156/PECompact\\_v.2.40\\_-\\_OEP\\_finder](http://www.openrce.org/downloads/details/156/PECompact_v.2.40_-_OEP_finder)).

[5] <http://nspack.download-230-13103.programsbase.com/> (<http://nspack.download-230-13103.programsbase.com/>).

[6] <http://upx.sourceforge.net/> (<http://upx.sourceforge.net/>).

[7] <http://www.oberhumer.com/opensource/ucl/> (<http://www.oberhumer.com/opensource/ucl/>).

[8] <http://www.softpedia.com/get/Programming/Packers-Crypters-Protectors/MEW-SE.shtml> (<http://www.softpedia.com/get/Programming/Packers-Crypters-Protectors/MEW-SE.shtml>).



(<https://twitter.com/virusbulletin>) (<https://www.facebook.com/virusbulletin/>) (<https://www.linkedin.com/company/virusbulletin/>) (<https://plus.google.com/+virusbulletin>) (<https://www.reddit.com/r/virusbulletin/>)

text=Quick reference for manual unpacking&url=https://www.virusbulletin.com/virusbulletin/2012/04/quick-reference-manual-unpacking) u=https://www.virusbulletin.com/virusbulletin/2012/04/quick-reference-manual-unpacking) &title=Quick reference-manual-unpacking) (https://www.virusbulletin.com/virusbulletin/2012/04/quick-reference-manual-unpacking) (https://www.virusbulletin.com/virusbulletin/2012/04/quick-reference-manual-unpacking) (https://www.virusbulletin.com/virusbulletin/2012/04/quick-reference-manual-unpacking)

## Latest articles:

### Throwback Thursday: Olympic Games

(/virusbulletin/2016/august/throwback-thursday-olympic-games/)

In 1994, along with the Olympic Games came an Olympic virus, from a group of Swedish virus authors calling themselves ‘Immortal Riot’. Mikko Hyppönen had the details.

### Throwback Thursday: Holding the Bady

(/virusbulletin/2016/07/throwback-thursday-holding-bady/)

In 2001, ‘Code Red’ caused White House administrators to change the IP address of the official White House website, and even penetrated the mighty Microsoft’s own IIS servers. In August 2001, Costin Raiu analysed the Win32/Bady.worm,

### The Journey of Evasion Enters Behavioural Phase

(/virusbulletin/2016/07/journey-evasion-enters-behavioural-phase/)

No malware author wants their piece of code to be easy to detect. Over time, several different approaches have been put into action to detect malware, and in response, malware authors have put into action different methods of evading them. This paper...

### Throwback Thursday: You Are the Weakest Link, Goodbye! -

Passwords, Malware and You (/virusbulletin/2016/07/throwback-thursday-you-are-weakest-link-goodbye-passwords-malware-and-you/)

Have you heard the one about the computer user who used their pet’s name as their password? Just like jokes, it seems the old ones and the obvious ones are considered the best when it comes to users

selecting their passwords. Martin Overton looks at...

## New Keylogger on the Block (/virusbulletin/2016/07/new-keylogger-block/)

This paper provides an overview of the KeyBase trojan, both the keylogger itself and the server-side management component. Additionally, we will look at an example of when this trojan was used.

---

**About us** (/about-vb/about-us/)

**Contact us** (/about-vb/contact-us/)

**Advisory board** (/about-vb/advisory-board/)

**Press information** (/about-vb/press/)

**Security events calendar** (/resources/calendar/)

**Security jobs** (/resources/jobs/)

**Testing** (/testing/)

**VB100** (/testing/vb100/)

**VBSpam** (/testing/vbspam/)

**VBWeb** (/testing/vbweb/)

**Consultancy services** (/testing/consultancy-services/)

**Spammers' Compendium** (/resources/spammerscompendium/)

**VB2016 (Denver)** (/conference/vb2016/)

**VB2015 (Prague)** (/conference/vb2015/)

**VB2014 (Seattle)** (/conference/vb2014/)

**VB2013 (Berlin)** (/conference/vb2013/)

**VB2012 (Dallas)** (/conference/vb2012/)

**Older conferences** (/conference/vb-conference-archive/)



6827080883932) (<https://www.facebook.com/virusbulletin>) (<https://www.youtube.com/user/virusbtn>)

©1989-2016 Virus Bulletin. [Privacy policy](/about-vb/privacy-policy/) [Cookies](/about-vb/privacy-policy/cookies/) [Terms and Conditions](/about-vb/terms-and-conditions/)