Unpacking malicious software using IDA Pro extensions

A paper by Dennis Elser

In almost all cases of today‘s malicious software, executable packers or -crypters are used in order to obfuscate code and data. In some cases unpackers and dumpers are available. In very few cases they actually work on packed malware executables due to modifications of internal structures such as the PE header.

In the following example an unknown binary is loaded into IDA Pro[1]. The code at the entry point of the executable looks like this:

```
UPX1:01015360
UPX1:01015360                   public start
UPX1:01015360 start             proc near
UPX1:01015360                   pusha
UPX1:01015361                   mov     esi, offset dword_1011000
UPX1:01015366                   lea     edi, [esi-10000h]
UPX1:0101536C                   push    edi
UPX1:0101536D                   or      ebp, 0FFFFFFFFh
UPX1:01015370                   jmp     short loc_1015382
```

Fig. 1

A segment named “UPX1”, an invalid import address table and an empty list of strings are an indicator for a packed file. UPX[2] however, can not unpack the file because internal structures have been modified. This technique often is used by malware authors to make unpacking and reverse engineering harder.

The first step now is to obtain a readable representation of the packed executable. A good and quick start in achieving this is to run the executable and dump the previously packed segment(s), once they have been unpacked. Preferably, the dump should be made right after the executable has been completely unpacked in memory. This often is the case after the original entry point (OEP) has been reached. Finding the OEP isn‘t always trivial and can be a time consuming process because you need to single step through the code. Using the IDA Pro SDK, a plugin named EPF[3] (Entry Point Finder) has been created, aimed towards automating the process of finding the original entry point.

An isolated environment (a virtual machine for example) is used to carefully

```
UPX1:01015360 public start
UPX1:01015360 start proc near
EIP UPX1:01015360 pusha
UPX1:01015361 mov     esi, offset dword_1011000
UPX1:01015366 lea     edi, [esi-10000h]
UPX1:0101536C push    edi
UPX1:0101536D or      ebp, 0FFFFFFFFh
UPX1:01015370 jmp     short loc_1015382
```

Fig. 2

run the executable in IDA Pro‘s debugger. Figure 2 shows the extended instruction

pointer (EIP) pointing to a “pusha” mnemonic. This statement is used as the first instruction to “back up” the content of all standard registers. Many executable compressors use a “popa” instruction at the end of their code to restore the previously saved state. This behavior can be exploited by the EPF plugin; the plugin offers an option to let the IDA Pro debugger trace code until a specific mnemonic is reached.

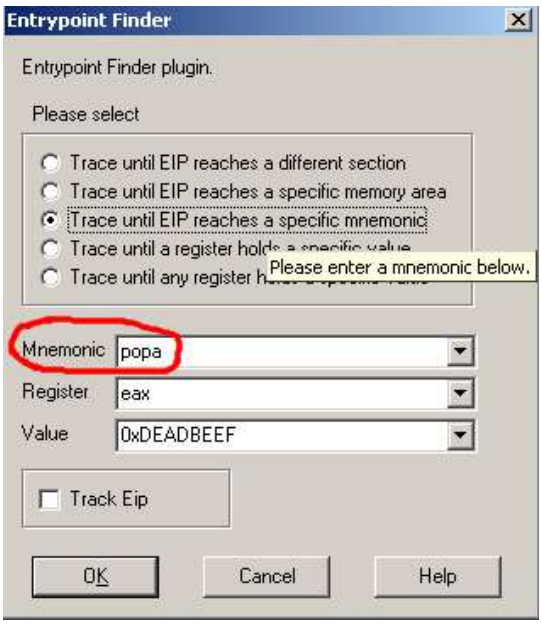

Fig. 3

After the EPF plugin has been started and configured, the process can be resumed (be careful, don‘t run malware on your host system!). After a few seconds, the process is paused and EPF turned off. The following message appears:

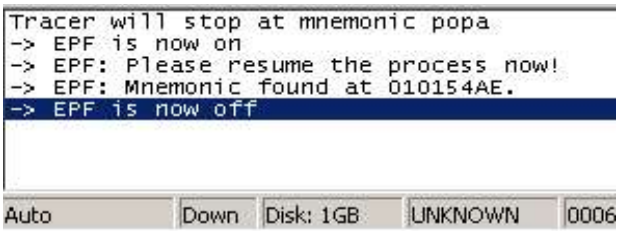

Fig. 4

The code at EIP points to a “popa” mnemonic followed by a jump and the end of

Fig. 5

the “start” procedure. Single stepping over the jump leads to the following message box:

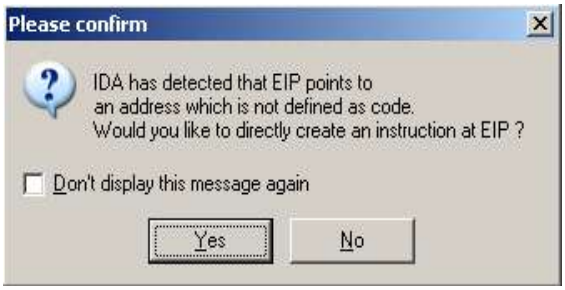

Fig. 6

Choosing “Yes” creates an instruction at EIP and IDA Pro begins to analyze control flow.

Fig. 7

EIP now points to a segment labeled “UPX0”. This is very likely the original entry point. It is reasonable to make a dump of the segment now. The DumpSeg[4] plugin can list and dump all segments available.

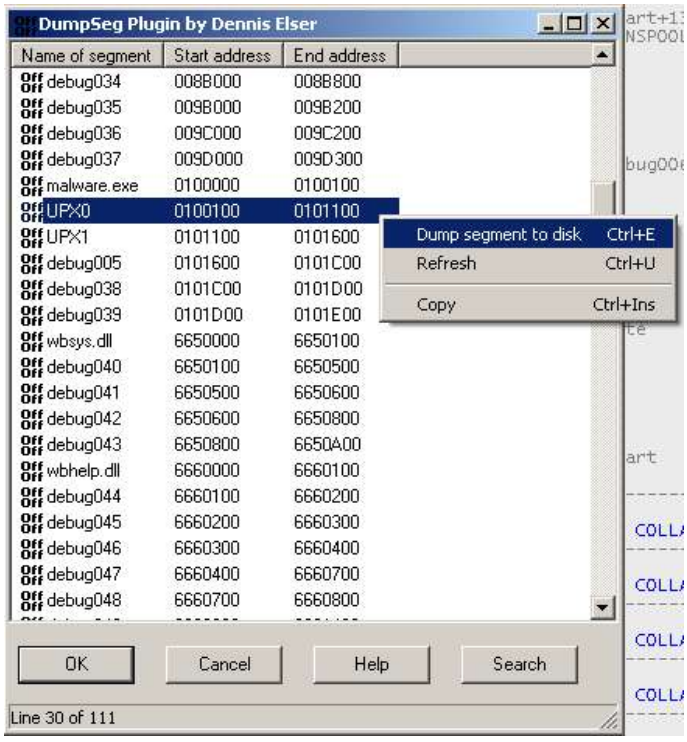

Fig. 8

The dumped segment can then be reloaded and analyzed by IDA Pro.

References

1.) IDA Pro, DateRescue (http://www.datarescue.com/idabase)

2.) UPX, Markus F.X.J. Oberhumer & Laszlo Molnar

3.) EPF, Dennis Elser (http://www.backtrace.de)

4.) DumpSeg, Dennis Elser (http://www.backtrace.de)