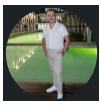
# Emotet Unpacking:. Writer: Ilan Duhin | by Ilan Duhin | Jan, 2023

medium.com/@Ilandu/emotet-unpacking-35bbe2980cfb

Ilan Duhin January 26, 2023

```
75AA6870 <kernel32.VirtualAlloc>     8B FF          mov edi,edi                          VirtualAlloc
75AA6872                             55             push ebp
75AA6873                             8B EC          mov ebp,esp
75AA6875                             5D             pop ebp
75AA6876 <kernel32.VirtualAlloc>     FF 25 0C 12 B0 75  jmp dword ptr ds:[<&VirtualAlloc>]  VirtualAlloc
75AA687C                             CC             int3
75AA687D                             CC             int3
75AA687E                             CC             int3
75AA687F                             CC             int3
75AA6880                             CC             int3
75AA6881                             CC             int3
75AA6882                             CC             int3
75AA6883                             CC             int3
75AA6884                             CC             int3
75AA6885                             CC             int3
75AA6886                             CC             int3
75AA6887                             CC             int3
75AA6888                             CC             int3
75AA6889                             CC             int3
75AA688A                             CC             int3
75AA688B                             CC             int3
75AA688C                             CC             int3
75AA688D                             CC             int3
75AA688E                             CC             int3
75AA688F                             CC             int3
75AA6890 <kernel32.VirtualAllocEx>   8B FF          mov edi,edi                          VirtualAllocEx
75AA6892                             55             push ebp
75AA6893                             8B EC          mov ebp,esp
75AA6895                             5D             pop ebp
75AA6896 <kernel32.VirtualAllocEx>   FF 25 14 12 B0 75  jmp dword ptr ds:[<&VirtualAllocEx>]  VirtualAllocEx
```



Ilan Duhin

Jan 26

.

5 min read

## Emotet Unpacking:

Writer: Ilan Duhin

## Executive Summary:

Emotet is an advanced, self-propagating and modular Trojan. Emotet was once a banking Trojan, but recently has been used as a downloader for other malware or malicious campaigns. It uses multiple methods for maintaining persistence and Evasion techniques **like packing**. In addition, it can be spread through phishing spam emails containing malicious attachments or links.

Emotet uses a number of malicious techniques when he locates on the victim's computer such as: **allocating memory for process injection, and creating new processes/threads to make persistence.**

Starting with **x32dbg & Running our sample** until the Entry point.

We should now search for API call of VirtualAlloc (we saw earlier the allocation memory on process hacker in the dynamic investigation).

So we need to search in the debugger (**Ctrl + G)=VirtualAlloc** and press OK so we can put **BP** on her and see which arguments contain the original code.

When we get to the beginning of the function, our goal is to see the "**ret"** that describe the end of the function so we can put **BP.**

To see the end we need to press **Enter.**

Now we put **BP.** To make sure that the BP is set we can look at Breakpoint tab.

Now Debug & Run + **Step over** (F8) the function (the debugger take now us to the original code).

When we press **step over** we jump to the next function below. **But if we scroll up one function** we see that the register **edi** store the VirtualAlloc function.

In situations like this when we found the call that contains our function we need to check the arguments that it pushes into it for searching our **MZ Header.**

The two arguments we see between the "calls" that used by the function to push them into her are: **[esp+28]** & **[esp+2C].**

Now we need to check there hex values **of unpacked code** by clicking "follow in dump". **The first check is empty** (we don't see any clue about **MZ Header**).

**Also the second one.**

In this situation when we **don't** find any hex values of unpacked code we need to **Run + Step over** one more time to search for more functions call of VirtualAlloc so we can search there if they have arguments that contain the original unpacked code.

So when we Run again we jumped to the **ret 10** instruction again.

**When we press (F8) — step over** we jump into the function that calls **ebp** register that contains VirtualAlloc function and a number of arguments we need to explore.

Similar to the first arguments, here we have another one: **[edi+54]**.

Click “follow in dump” and let’s see the results. When we scroll up we see that the hex strings looks like **executable** with description of **“This program cannot run in DOS mode”,** it is perfect for us because every PE header starts like this. **Now what is left to find out is where the MZ.**

Little bit scrolling up and we see the **MZ Header** that is probably the **unpacked code!**

In this stage of analysis (after finding the unpacked code) we should go to **Memory map** TAB to locate the specific address that contains **Execute permissions** and dump her into **new file**.

To do this, **right click** on the hex values table **& “follow in memory map”.**

Its automatically points us to the address that the malware doing it executable capabilities.

All we have left to do is dump the address of unpacked code into new file like I said earlier.

To do this, **right click** on memory address **& “Dump memory to file”.**

The most fun part for me in unpacking is when you drag the **unpacked file** into **HxD** and clean all the beginning before the **MZ.**

First, we search the MZ string with **Ctrl+F**, when we locate him, we **erase** all strings before him so we can save it into a cleaned **PE File.**