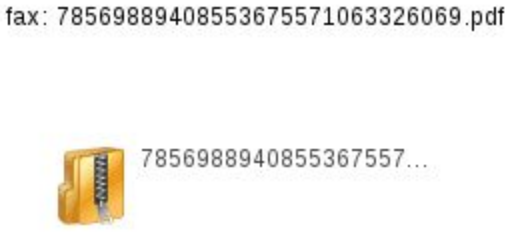
Unpacking Fraudulent “Fax”: Dyreza Malware from Spam

October 19, 2015 by hasherezade

Last updated: October 16, 2016

This post describes the process of unpacking a malware delivered in a spam campaign. The described sample has been delivered on 1 October 2015 at 17:33 CEST.

E-mail content:



Fragment of message headers:

Received: from spamfilter.jpenergypartners.com (84.95.205.45.forward.012.net.il [84.95.205.45])
	(using TLSv1 with cipher ECDHE-RSA-AES256-SHA (256/256 bits))
	(No client certificate requested)
(...)
Received: by jpep.com with SMTP
 id oSMlcE7; Thu, 01 Oct 2015 11:35:28 -0400
Received: from spamfilter.jpenergypartners.com (spamfilter.jpenergypartners.com [74.7.154.89])
 by spamfilter.jpenergypartners.com with ESMTP id ixqVA32rotPPduej2c;
 Thu, 01 Oct 2015 11:33:28 -0400
Message-ID: <560D52C0.D8CF643D@jpep.com>
Date: Thu, 01 Oct 2015 11:33:28 -0400
From: "fax" <sstandley@jpep.com>
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:31.0) Gecko/20100101 Thunderbird/31.7.0
MIME-Version: 1.0
(...)
Subject:
Content-Type: multipart/mixed;

The malicious sample is shipped as a ZIP-compressed executable, pretending to be a PDF document:

After several stages of unpacking, we receive the payload belonging to the Dyreza malware family, a credential stealer.

Elements involved

There are 4 different executables involved in a chain of execution, leading to deployment of core malicious functions:

- 841ca39312bdbb4a0010807a773961d6 – ***391476564923647.exe*** – the main file, which we receive directly after unpacking the original ZIP
- cc5d0acba5c7e0d62dd547641d9da1a1 – ***zatribmet.exe*** – an intermediate executable, which is dropped into the **%TEMP%** folder

- 2 payloads, which are loaded by the RunPE technique, substituting in memory the 2 above samples. Note that the payloads' hashes (below) may vary, depending on the chosen method of dumping:
    - 9a4a171db069af2b15d6f88759b08db0 – **Payload #1** (Upatre – obfuscated & self modifying downloader, prepares code and injects it into *svchost.exe*)
    - ff3d706015b7b142ee0a8f0ad7ea2911 – **Payload #2** (Dyreza agent)

The following focuses on retrieving these payload files.

Flow:

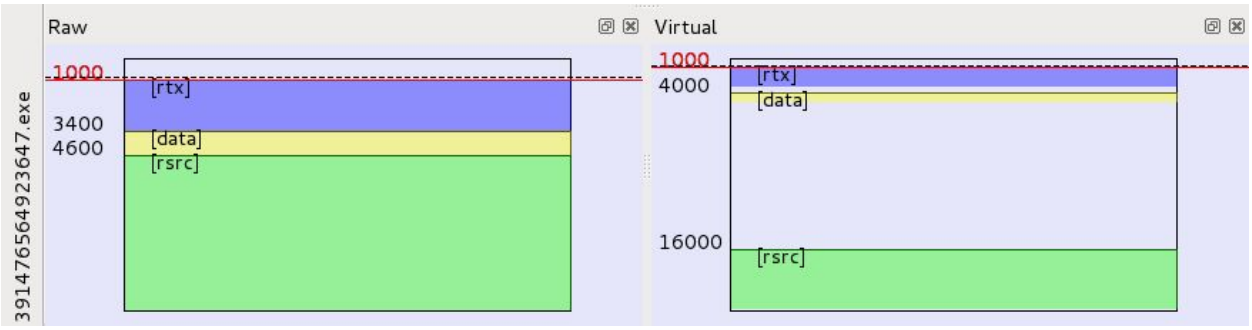
391476564923647.exe -> RunPE : ***Payload#1*** -> injects code into ***svchost.exe*** -> downloads: ***zatribmet.exe*** -> RunPE: ***Payload #2***

Unpacking

The 2 samples, ***391476564923647.exe*** and ***zatribmet.exe***, are carriers of payloads. Both are packed by the same custom packer containing obfuscating techniques.

Due to the fact, that unpacking of both files follows analogical steps, the first file: ***391476564923647.exe*** will be used as an example.

Sections in the original sample:



*First section (code) renamed to **rtx**. Section **data**, that is relatively small in the raw image, gets enlarged in the virtual image, to fit the uncompressed data. Entry point is at the beginning of the first section (offset 0x1000).*

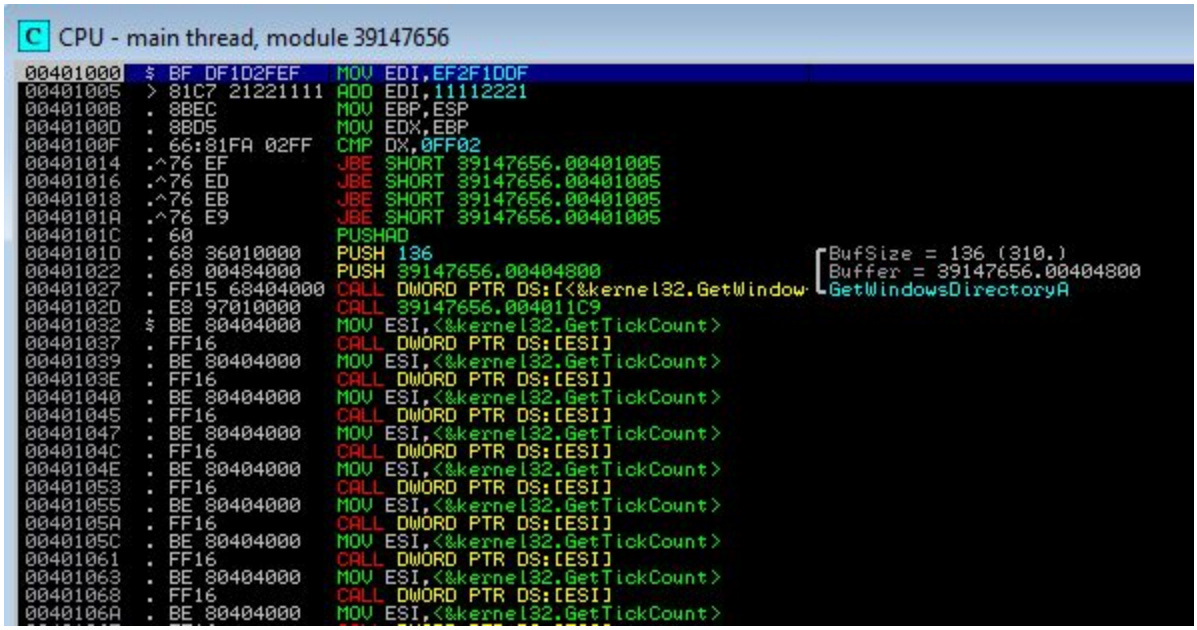
Unpacking follows several stages:

1. In **rtx** section: unpacking code into **data** section and jumping there
2. In **data** section: loading imports, copying a piece of position-independent code (that will be referred as ***shellcode***) from **data** into newly allocated memory

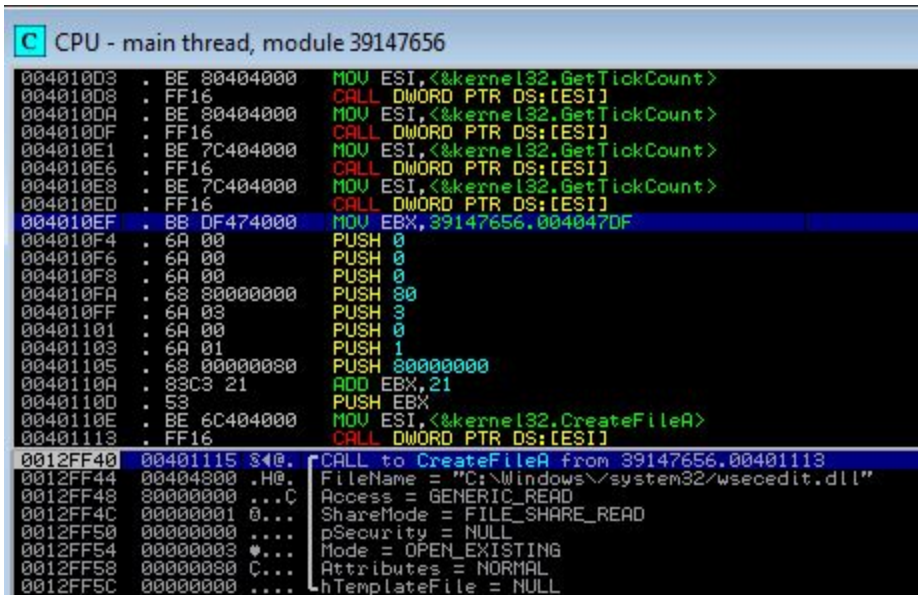
3. In ***shellcode***: opening the file of running executable and reading, piece by piece, the code to be decrypted into memory. It forms a new PE file, that is, after that loaded by a RunPE technique into the memory, substituting the currently running executable.

Stage 1:

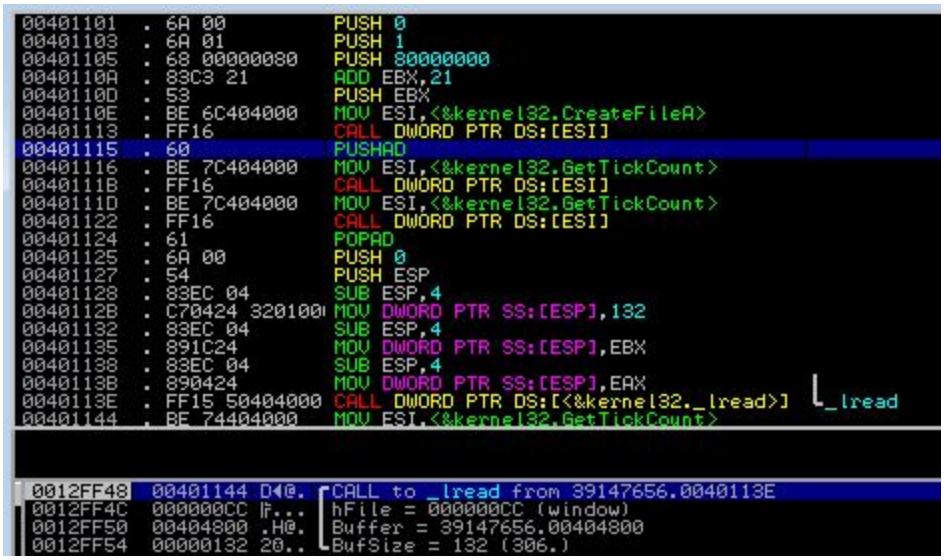
Entry Point of ***391476564923647.exe***:



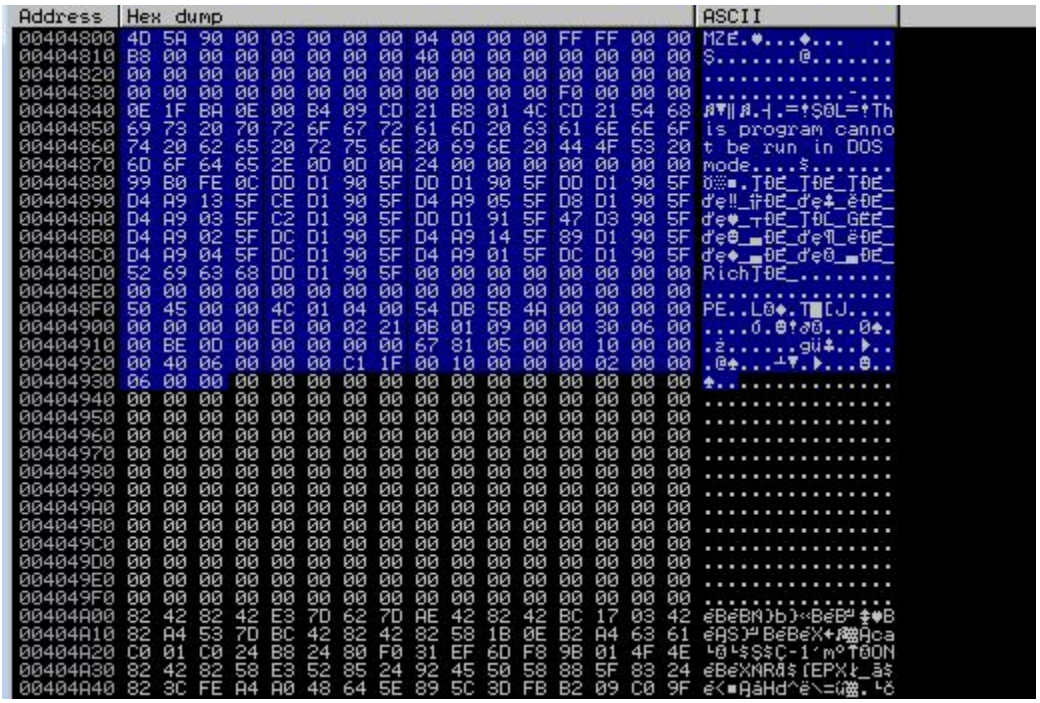
First, the ***wsecedit.dll (C:Windows/system32/wsecedit.dl)*** is opened:



Its headers are read into memory:

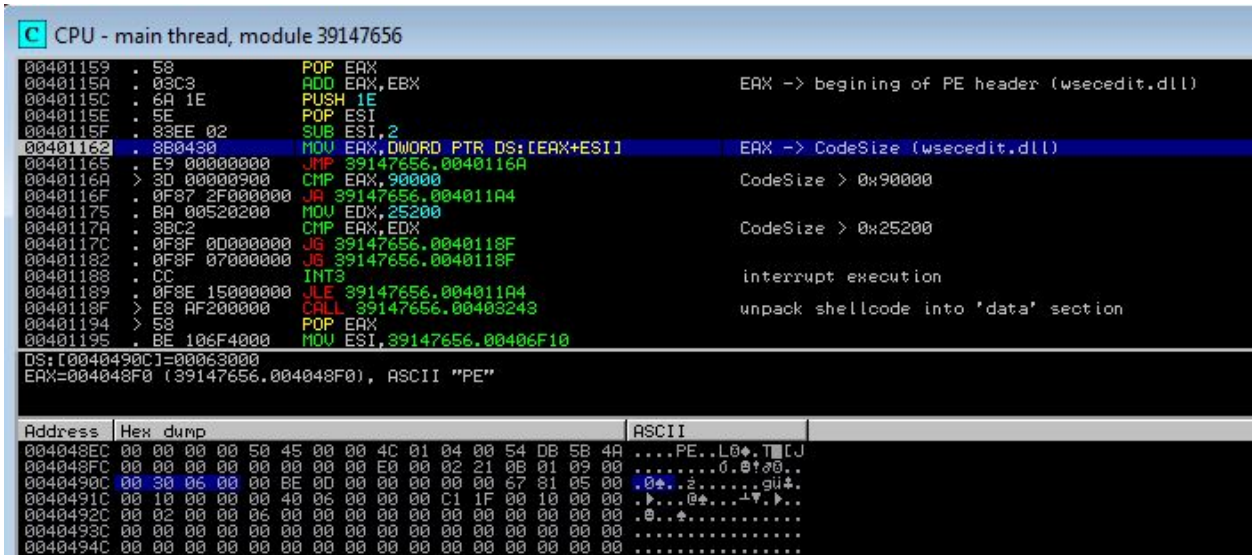


First 0x132 bytes of ***wsecedit.dll*** :

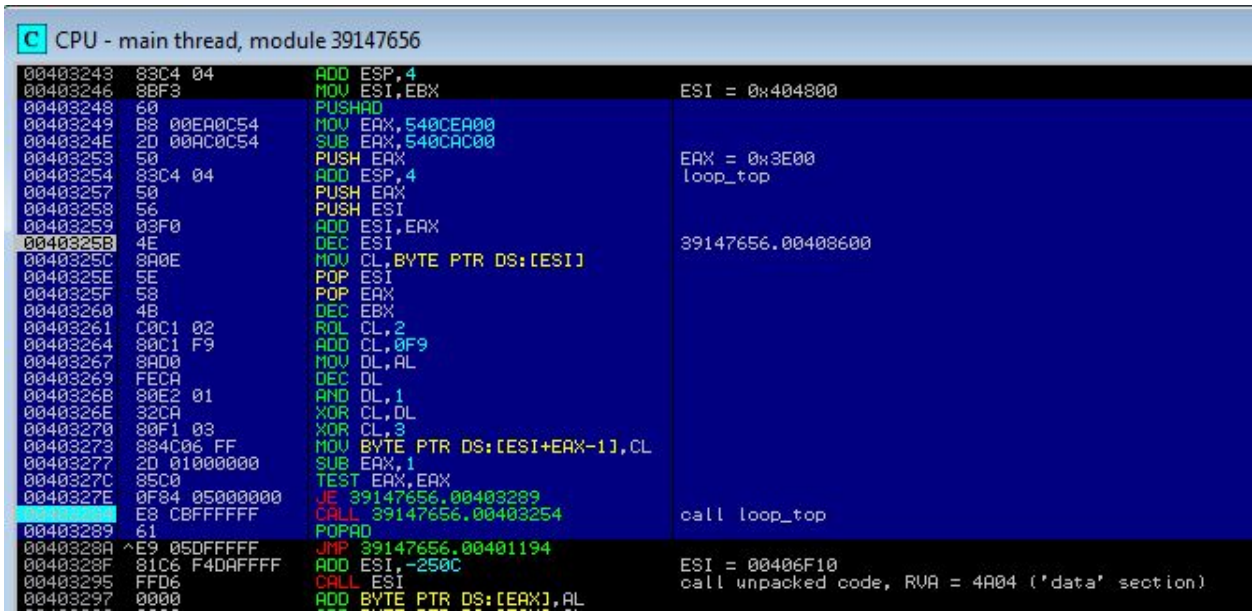


After that, its CodeSize is retrieved and compared to some hard-coded values. (**This is the only role that malware authors had for this DLL – it is not used anywhere else, neither in the process of unpacking*).

If it doesn't satisfy the conditions, the execution is interrupted:



Otherwise, the programs proceed to unpack the data section:

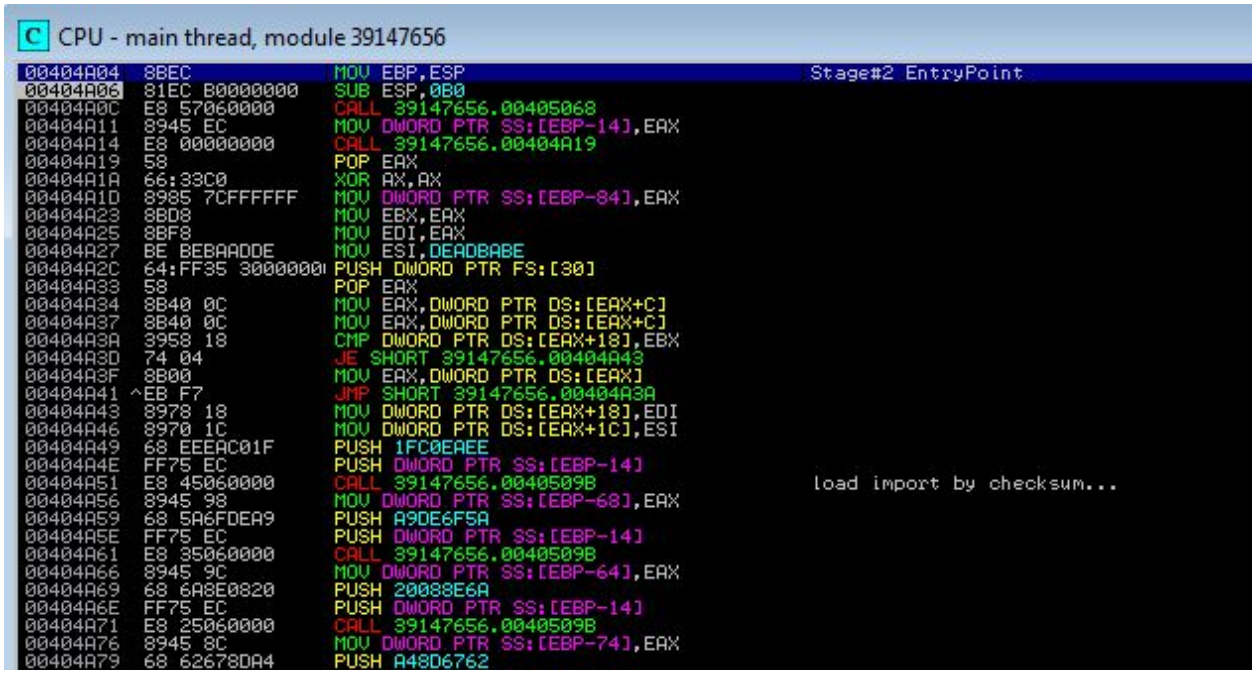


… and the execution is redirected there.

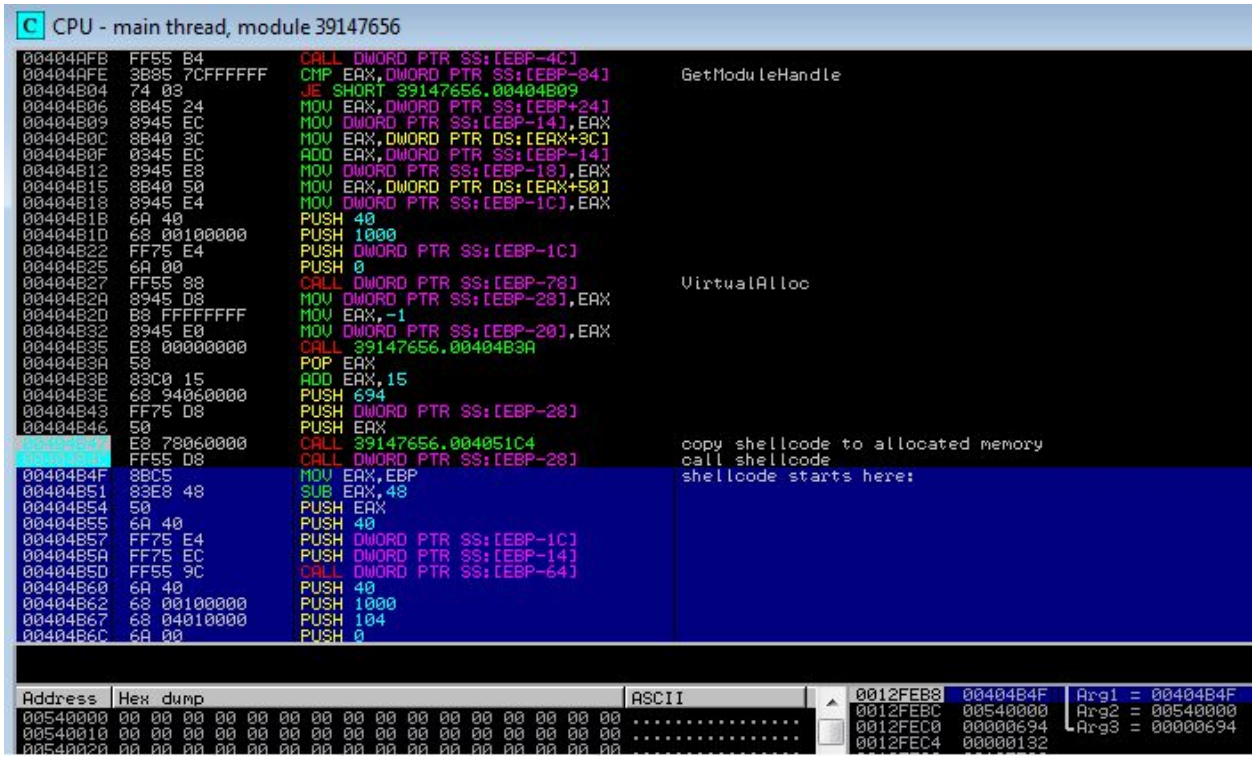
Stage 2:

Inside the data section:

First, additional imports are loaded by their checksums:



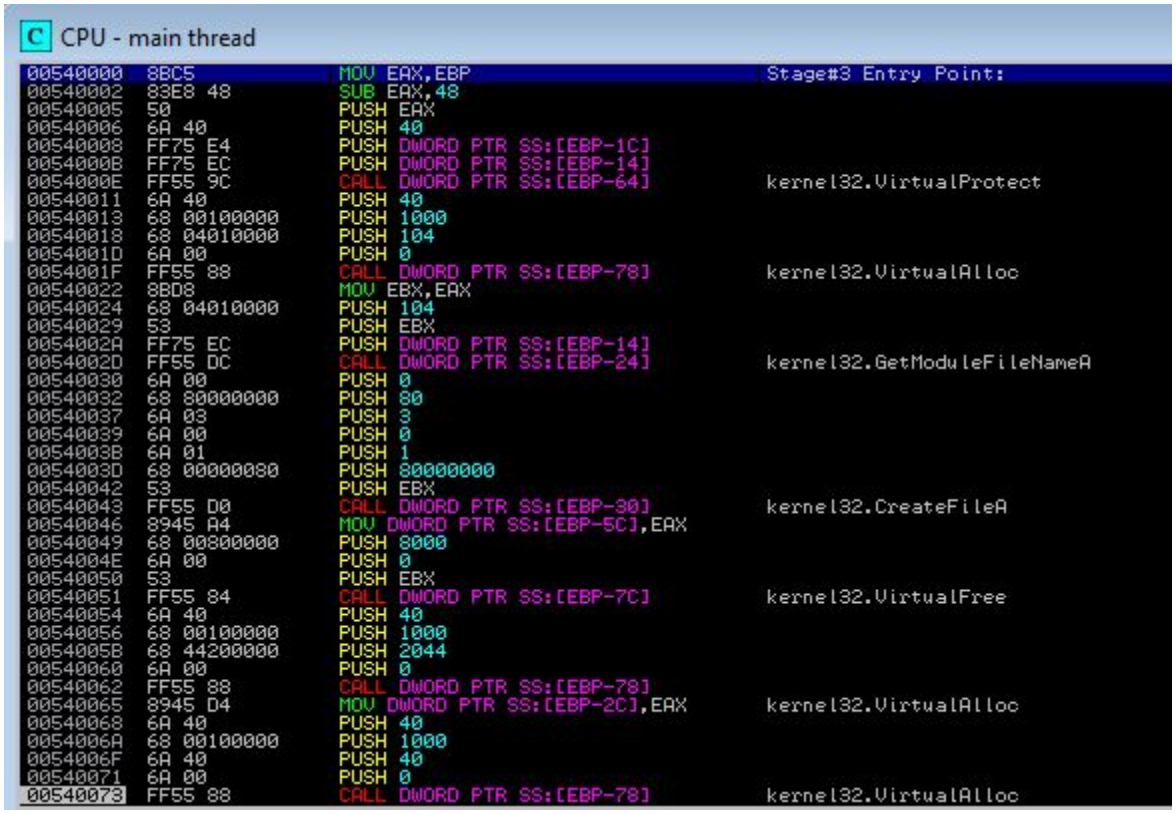
Then, a part of the code from data section (0x694 bytes, beginning marked blue on below illustration) is copied into a newly allocated memory (for the convenience I called it ***shellcode***). This step is necessary in order to have execution redirected outside the mapped image – further, the full image is cleared for the purpose of mapping the payload on its place (RunPE technique).



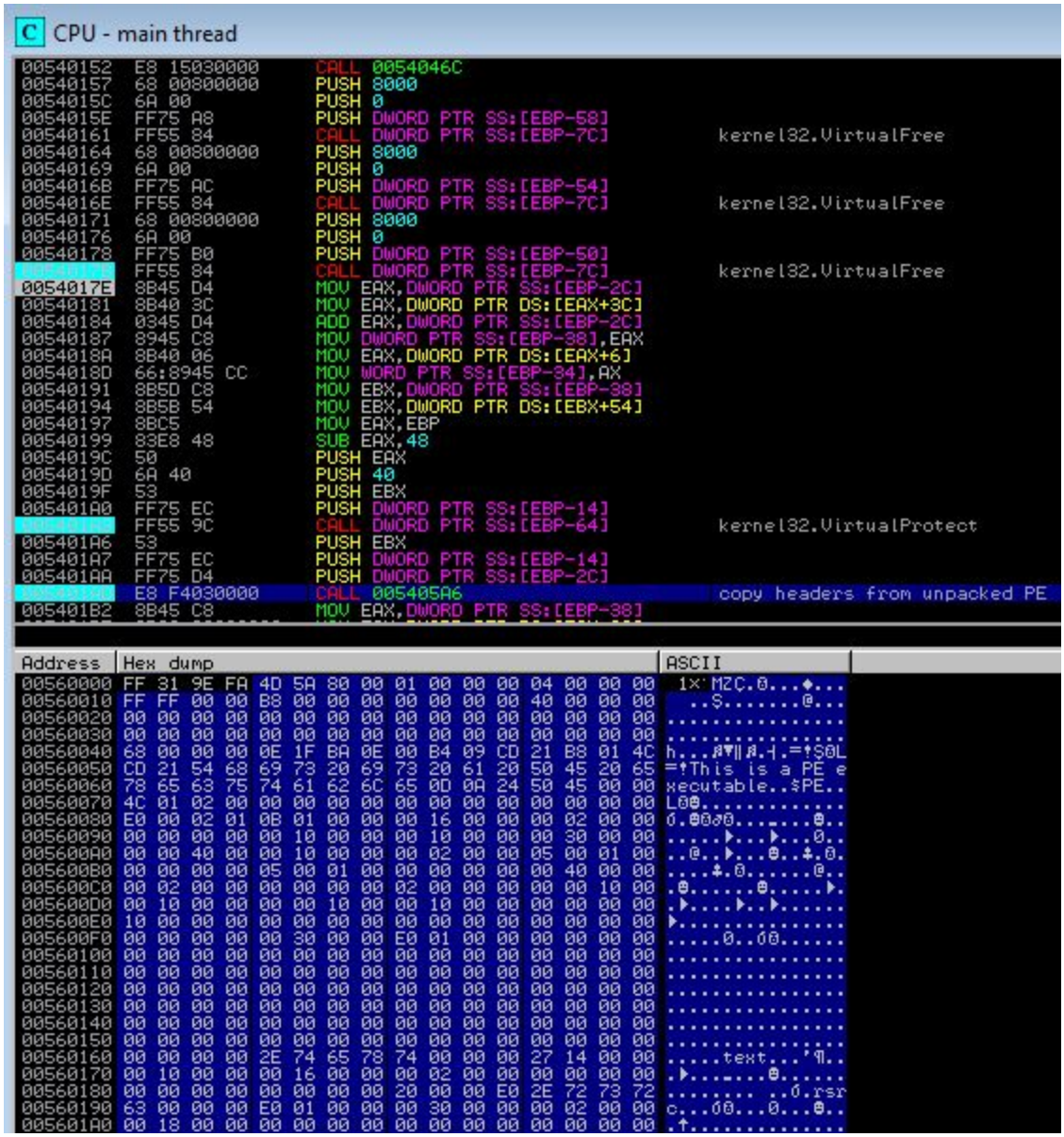
The execution is then redirected to that part of memory (outside the mapped PE sections).

Stage 3:

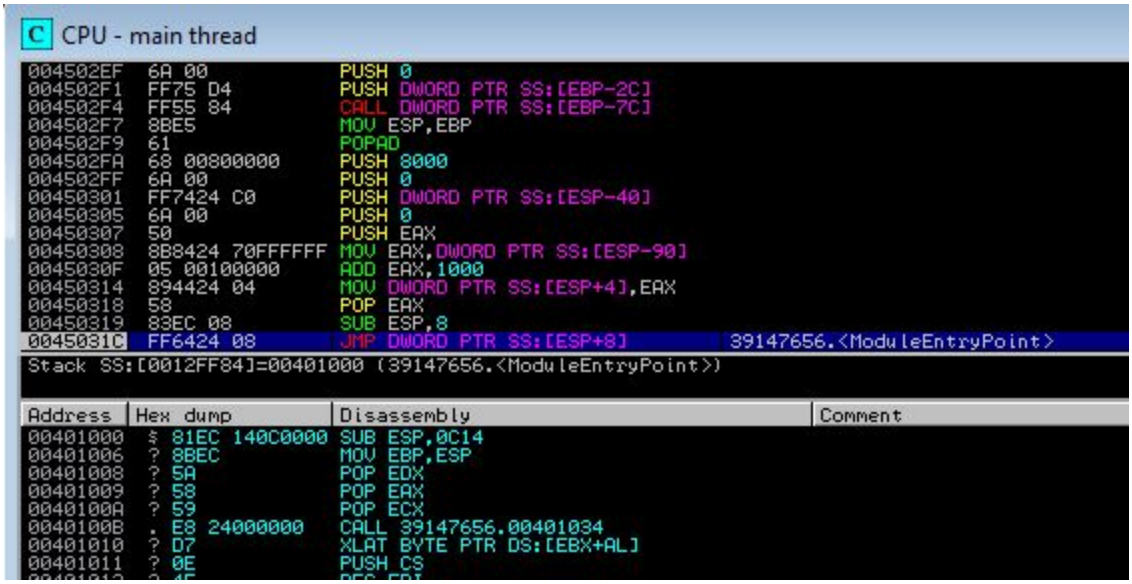
Inside the ***shellcode***, the file of the current executable is opened from the disk; particular chunks are then read and decrypted. It uses imports loaded in the Stage 2.



This results in the emergence of a new PE (payload) in memory:



Then, this new payload is mapped onto memory in place of the current executable. After finishing the substitution operation, ***shellcode*** jumps to the first section (offset 0x1000), which is now filled with a completely new code:



Payloads

Payloads are not intended to be a topic of this post, but let's take just a brief look.

Payload#1 is an obfuscated downloader – Upatre. It communicates with C&C over SSL and downloads a second malware (***zatribmet.exe***). After that it deletes itself.

Payload #2, which comes from ***zatribmet.exe***, is a persistent Dyreza agent. It starts from its anti-VM technique by checking the number of processors of the machine it is executed on:

When the above condition is satisfied, it then copy itself into 2 locations: **%APPDATA%** and **C:\Windows** (under pseudo-random names, following the pattern **[a-zA-Z]{15}.exe** (e.g. *HdIQpVwEebXreDK.exe)*. After this, it creates a task in the scheduler for persistence and communicates with a C&C server, which is randomly picked from a predefined pool, over SSL.

See also: A Technical Look At Dyreza