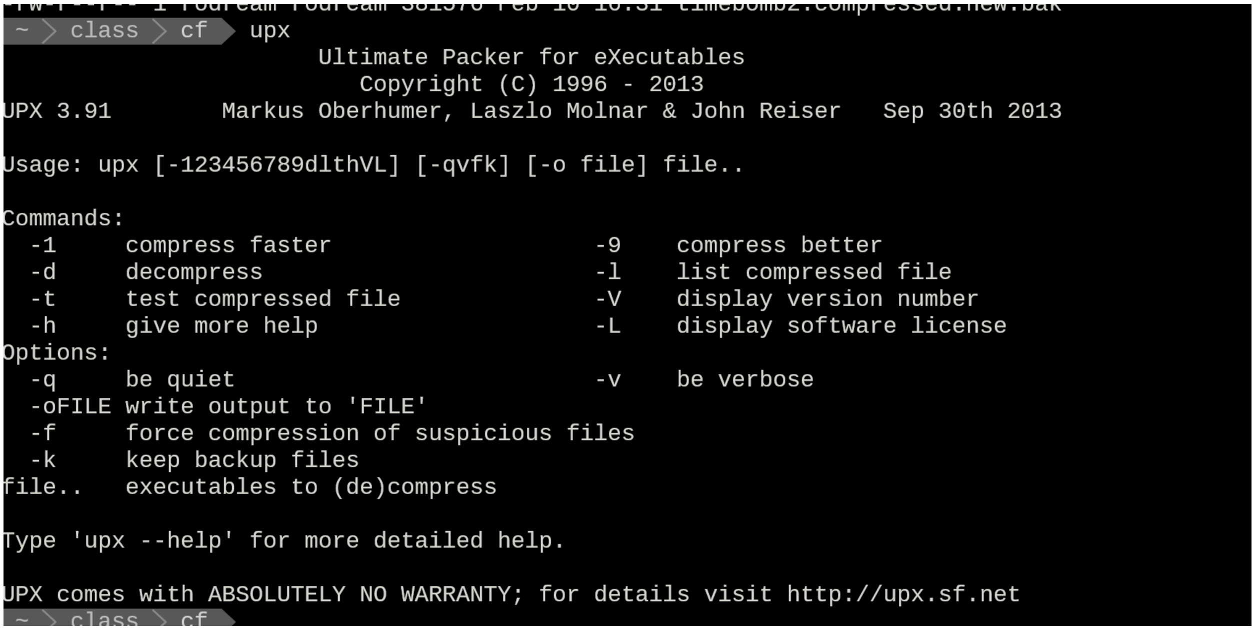
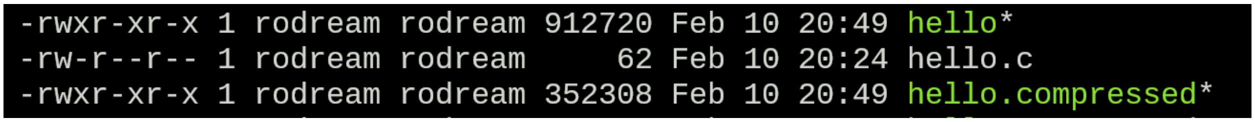
1. UPX compression



```
-rw-r--r-- 1 rodream rodream 381376 Feb 10 10:31 timebomb2.compressed.new.bak
 ~ 〉 class 〉 cf 〉 upx
                       Ultimate Packer for eXecutables
                          Copyright (C) 1996 - 2013
UPX 3.91        Markus Oberhumer, Laszlo Molnar & John Reiser   Sep 30th 2013

Usage: upx [-123456789dlthVL] [-qvfk] [-o file] file..

Commands:
  -1     compress faster                   -9    compress better
  -d     decompress                        -l    list compressed file
  -t     test compressed file              -V    display version number
  -h     give more help                    -L    display software license
Options:
  -q     be quiet                          -v    be verbose
  -oFILE write output to 'FILE'
  -f     force compression of suspicious files
  -k     keep backup files
file..   executables to (de)compress

Type 'upx --help' for more detailed help.

UPX comes with ABSOLUTELY NO WARRANTY; for details visit http://upx.sf.net
 ~ 〉 class 〉 cf 〉
```

UPX is a program that can compress a binary file to reduce the size of the executable file.



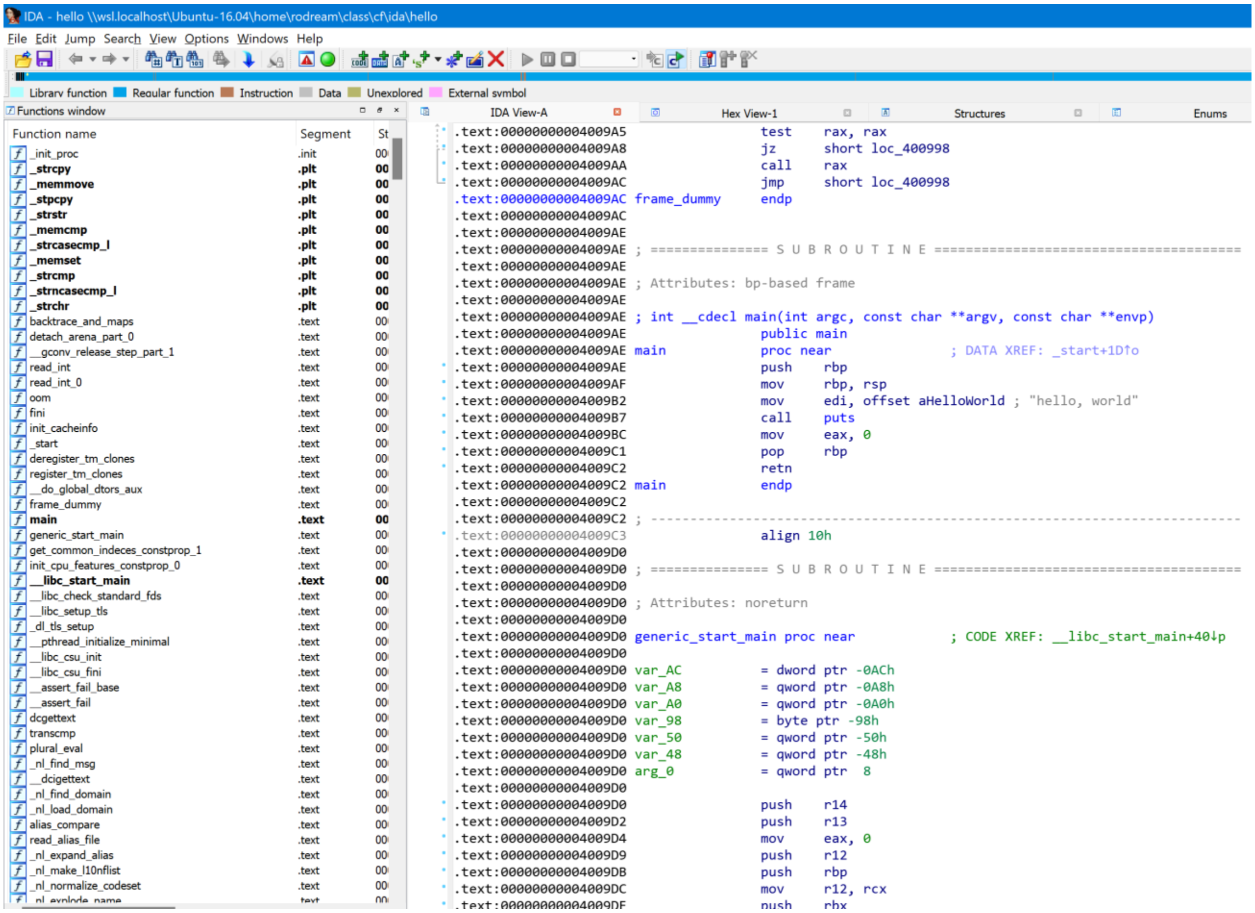
```
-rwxr-xr-x 1 rodream rodream 912720 Feb 10 20:49 hello*
-rw-r--r-- 1 rodream rodream     62 Feb 10 20:24 hello.c
-rwxr-xr-x 1 rodream rodream 352308 Feb 10 20:49 hello.compressed*
```

Here is an example. “hello” program is a simple hello world program, compiled from hello.c. The binary file’s size is about 912K. I statically compiled the program so that it would have a large binary size.

Then, to compress the binary file, I use UPX to get “hello.compressed”. You see, the file size is 352K (less than half of the original size).

It is done by running the following command: **“upx -o hello.compressed ./hello”**

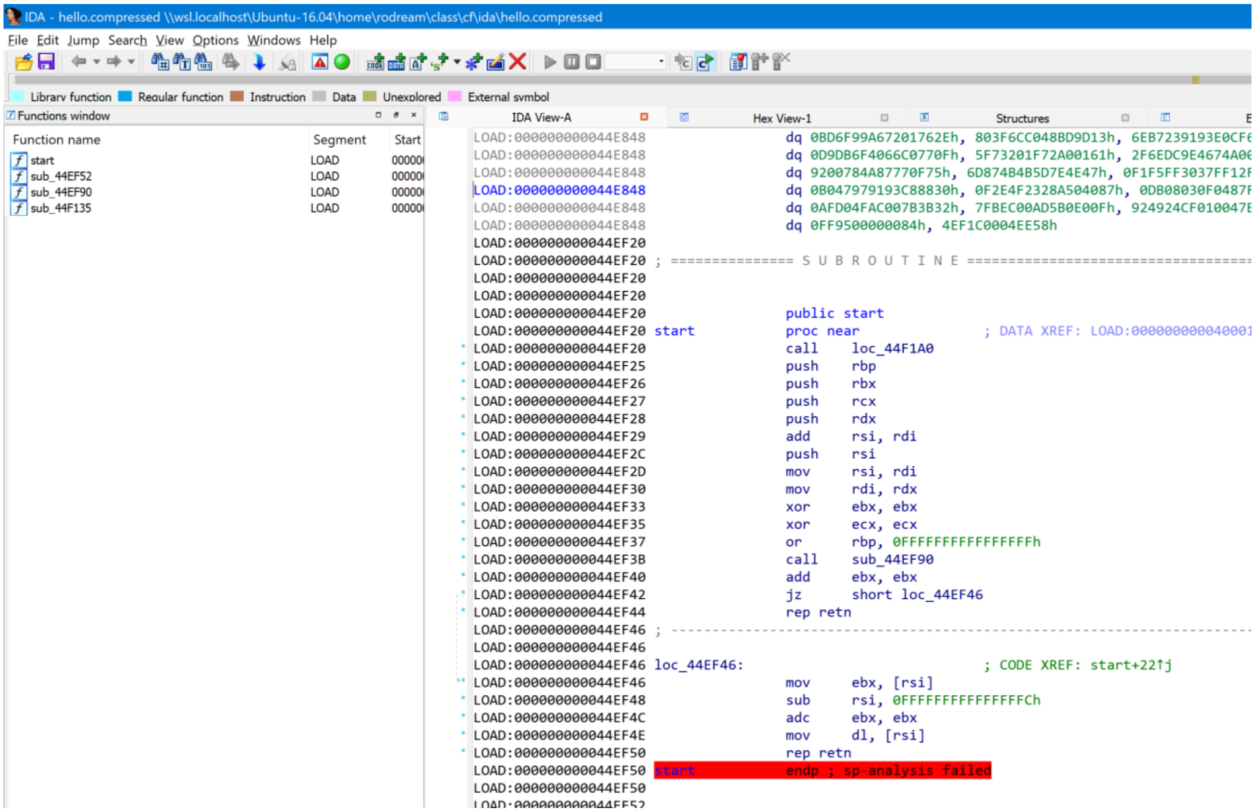
2. Original vs Compressed version



Function name	Segment	St
_init_proc	.init	00
_strcpy	**.plt**	**00**
_memmove	**.plt**	**00**
_stpcpy	**.plt**	**00**
_strstr	**.plt**	**00**
_memcmp	**.plt**	**00**
_strcasecmp_l	**.plt**	**00**
_memset	**.plt**	**00**
_strcmp	**.plt**	**00**
_strncasecmp_l	**.plt**	**00**
_strchr	**.plt**	**00**
backtrace_and_maps	.text	00
detach_arena_part_0	.text	00
__gconv_release_step_part_1	.text	00
read_int	.text	00
read_int_0	.text	00
oom	.text	00
fini	.text	00
init_cacheinfo	.text	00
_start	.text	00
deregister_tm_clones	.text	00
register_tm_clones	.text	00
__do_global_dtors_aux	.text	00
frame_dummy	.text	00
main	**.text**	**00**
generic_start_main	.text	00
get_common_indeces_constprop_1	.text	00
init_cpu_features_constprop_0	.text	00
__libc_start_main	**.text**	**00**
__libc_check_standard_fds	.text	00
__libc_setup_tls	.text	00
_dl_tls_setup	.text	00
__pthread_initialize_minimal	.text	00
__libc_csu_init	.text	00
__libc_csu_fini	.text	00
__assert_fail_base	.text	00
__assert_fail	.text	00
dcgettext	.text	00
transcmp	.text	00
plural_eval	.text	00
_nl_find_msg	.text	00
__dcigettext	.text	00
_nl_find_domain	.text	00
_nl_load_domain	.text	00
alias_compare	.text	00
read_alias_file	.text	00
_nl_expand_alias	.text	00
_nl_make_l10nflist	.text	00
_nl_normalize_codeset	.text	00
_nl_explode_name	.text	00

```
.text:00000000004009A5                 test    rax, rax
.text:00000000004009A8                 jz      short loc_400998
.text:00000000004009AA                 call    rax
.text:00000000004009AC                 jmp     short loc_400998
.text:00000000004009AC frame_dummy     endp
.text:00000000004009AC
.text:00000000004009AE
.text:00000000004009AE ; =============== S U B R O U T I N E =======================================
.text:00000000004009AE
.text:00000000004009AE ; Attributes: bp-based frame
.text:00000000004009AE
.text:00000000004009AE ; int __cdecl main(int argc, const char **argv, const char **envp)
.text:00000000004009AE                 public main
.text:00000000004009AE main            proc near               ; DATA XREF: _start+1D↑o
.text:00000000004009AE                 push    rbp
.text:00000000004009AF                 mov     rbp, rsp
.text:00000000004009B2                 mov     edi, offset aHelloWorld ; "hello, world"
.text:00000000004009B7                 call    puts
.text:00000000004009BC                 mov     eax, 0
.text:00000000004009C1                 pop     rbp
.text:00000000004009C2                 retn
.text:00000000004009C2 main            endp
.text:00000000004009C2
.text:00000000004009C2 ; ---------------------------------------------------------------------------
.text:00000000004009C3                 align 10h
.text:00000000004009D0
.text:00000000004009D0 ; =============== S U B R O U T I N E =======================================
.text:00000000004009D0
.text:00000000004009D0 ; Attributes: noreturn
.text:00000000004009D0
.text:00000000004009D0 generic_start_main proc near           ; CODE XREF: __libc_start_main+40↓p
.text:00000000004009D0
.text:00000000004009D0 var_AC          = dword ptr -0ACh
.text:00000000004009D0 var_A8          = qword ptr -0A8h
.text:00000000004009D0 var_A0          = qword ptr -0A0h
.text:00000000004009D0 var_98          = byte ptr -98h
.text:00000000004009D0 var_50          = qword ptr -50h
.text:00000000004009D0 var_48          = qword ptr -48h
.text:00000000004009D0 arg_0           = qword ptr  8
.text:00000000004009D0
.text:00000000004009D0                 push    r14
.text:00000000004009D2                 push    r13
.text:00000000004009D4                 mov     eax, 0
.text:00000000004009D9                 push    r12
.text:00000000004009DB                 push    rbp
.text:00000000004009DC                 mov     r12, rcx
.text:00000000004009DF                 push    rbx
```

This is the original program’s disassembled code. On the left-hand side, you can see a lot of library functions, and on the right-hand side, you can see the main function with “hello, world” which is basically the functionality of the program.



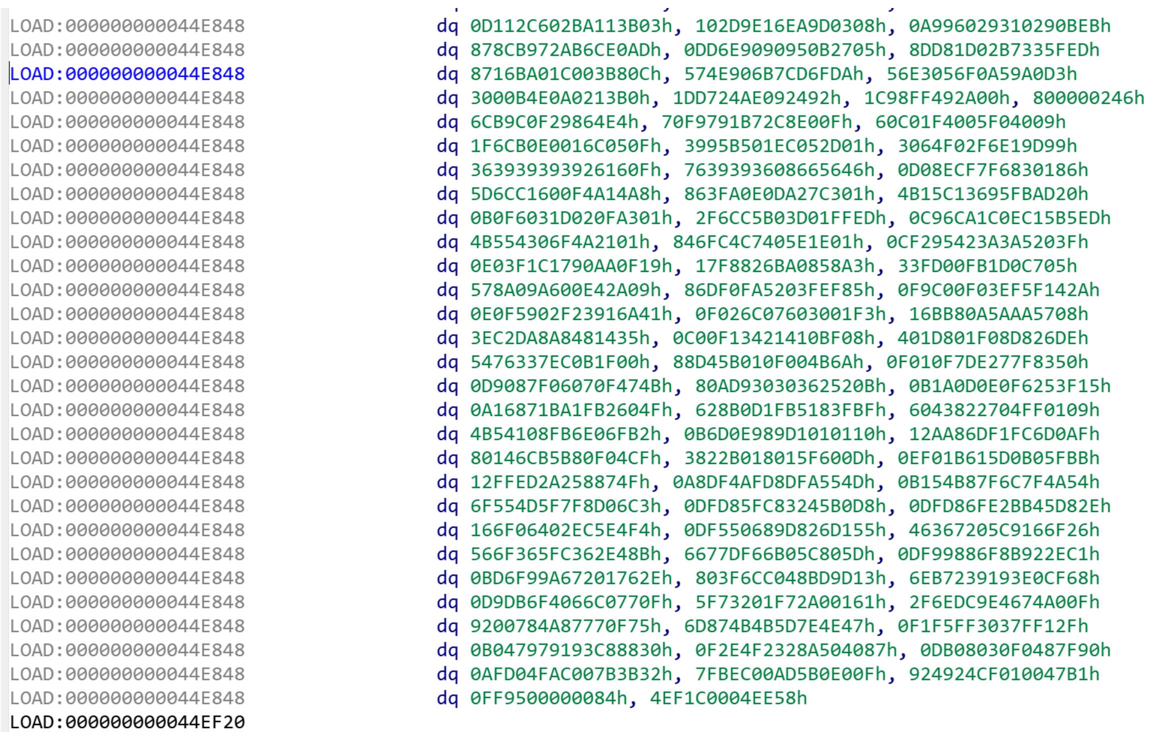
IDA - hello.compressed \\wsl.localhost\Ubuntu-16.04\home\rodream\class\cf\ida\hello.compressed
File Edit Jump Search View Options Windows Help
Library function
Regular function
Instruction
Data
Unexplored
External symbol
Functions window
IDA View-A
Hex View-1
Structures
E

Function name	Segment	Start
start	LOAD	00000
sub_44EF52	LOAD	00000
sub_44EF90	LOAD	00000
sub_44F135	LOAD	00000

```
LOAD:000000000044E848                 dq 0BD6F99A67201762Eh, 803F6CC048BD9D13h, 6EB7239193E0CF6
LOAD:000000000044E848                 dq 0D9DB6F4066C0770Fh, 5F73201F72A00161h, 2F6EDC9E4674A06
LOAD:000000000044E848                 dq 9200784A87770F75h, 6D874B4B5D7E4E47h, 0F1F5FF3037FF12F
LOAD:000000000044E848                 dq 0B047979193C88830h, 0F2E4F2328A504087h, 0DB08030F0487F
LOAD:000000000044E848                 dq 0AFD04FAC007B3B32h, 7FBEC00AD5B0E00Fh, 924924CF010047E
LOAD:000000000044E848                 dq 0FF9500000084h, 4EF1C0004EE58h
LOAD:000000000044EF20
LOAD:000000000044EF20 ; =============== S U B R O U T I N E =======================================
LOAD:000000000044EF20
LOAD:000000000044EF20
LOAD:000000000044EF20                 public start
LOAD:000000000044EF20 start           proc near               ; DATA XREF: LOAD:000000000040001
LOAD:000000000044EF20                 call    loc_44F1A0
LOAD:000000000044EF25                 push    rbp
LOAD:000000000044EF26                 push    rbx
LOAD:000000000044EF27                 push    rcx
LOAD:000000000044EF28                 push    rdx
LOAD:000000000044EF29                 add     rsi, rdi
LOAD:000000000044EF2C                 push    rsi
LOAD:000000000044EF2D                 mov     rsi, rdi
LOAD:000000000044EF30                 mov     rdi, rdx
LOAD:000000000044EF33                 xor     ebx, ebx
LOAD:000000000044EF35                 xor     ecx, ecx
LOAD:000000000044EF37                 or      rbp, 0FFFFFFFFFFFFFFFFh
LOAD:000000000044EF3B                 call    sub_44EF90
LOAD:000000000044EF40                 add     ebx, ebx
LOAD:000000000044EF42                 jz      short loc_44EF46
LOAD:000000000044EF44                 rep retn
LOAD:000000000044EF46 ; ---------------------------------------------------------------------------
LOAD:000000000044EF46
LOAD:000000000044EF46 loc_44EF46:                             ; CODE XREF: start+22↑j
LOAD:000000000044EF46                 mov     ebx, [rsi]
LOAD:000000000044EF48                 sub     rsi, 0FFFFFFFFFFFFFFFCh
LOAD:000000000044EF4C                 adc     ebx, ebx
LOAD:000000000044EF4E                 mov     dl, [rsi]
LOAD:000000000044EF50                 rep retn
LOAD:000000000044EF50 start           endp ; sp-analysis failed
LOAD:000000000044EF50
LOAD:000000000044EF52
```

This is the compressed program’s disassembled code. Observe that we only have four functions now, and the start function (the entry point; the function that is executed right after the program execution) does not have any original functionality of the program (i.e., printing out “hello, world”).

Also, you may observe there are a lot of data above the start function.

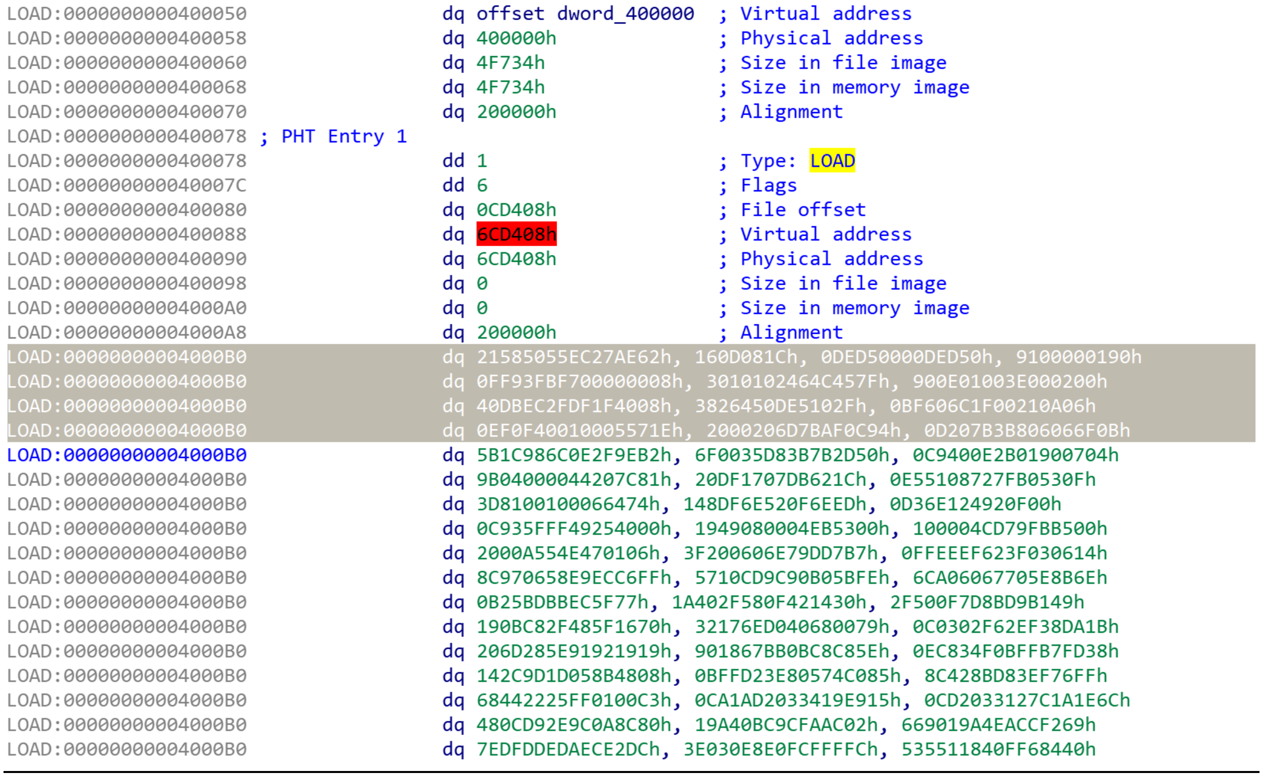


```
LOAD:000000000044E848                 dq 0D112C602BA113B03h, 102D9E16EA9D0308h, 0A996029310290BEBh
LOAD:000000000044E848                 dq 878CB972AB6CE0ADh, 0DD6E9090950B2705h, 8DD81D02B7335FEDh
LOAD:000000000044E848                 dq 8716BA01C003B80Ch, 574E906B7CD6FDAh, 56E3056F0A59A0D3h
LOAD:000000000044E848                 dq 3000B4E0A0213B0h, 1DD724AE092492h, 1C98FF492A00h, 800000246h
LOAD:000000000044E848                 dq 6CB9C0F29864E4h, 70F9791B72C8E00Fh, 60C01F4005F04009h
LOAD:000000000044E848                 dq 1F6CB0E0016C050Fh, 3995B501EC052D01h, 3064F02F6E19D99h
LOAD:000000000044E848                 dq 363939393926160Fh, 7639393608665646h, 0D08ECF7F6830186h
LOAD:000000000044E848                 dq 5D6CC1600F4A14A8h, 863FA0E0DA27C301h, 4B15C13695FBAD20h
LOAD:000000000044E848                 dq 0B0F6031D020FA301h, 2F6CC5B03D01FFEDh, 0C96CA1C0EC15B5EDh
LOAD:000000000044E848                 dq 4B554306F4A2101h, 846FC4C7405E1E01h, 0CF295423A3A5203Fh
LOAD:000000000044E848                 dq 0E03F1C1790AA0F19h, 17F8826BA0858A3h, 33FD00FB1D0C705h
LOAD:000000000044E848                 dq 578A09A600E42A09h, 86DF0FA5203FEF85h, 0F9C00F03EF5F142Ah
LOAD:000000000044E848                 dq 0E0F5902F23916A41h, 0F026C07603001F3h, 16BB80A5AAA5708h
LOAD:000000000044E848                 dq 3EC2DA8A8481435h, 0C00F13421410BF08h, 401D801F08D826DEh
LOAD:000000000044E848                 dq 5476337EC0B1F00h, 88D45B010F004B6Ah, 0F010F7DE277F8350h
LOAD:000000000044E848                 dq 0D9087F06070F474Bh, 80AD93030362520Bh, 0B1A0D0E0F6253F15h
LOAD:000000000044E848                 dq 0A16871BA1FB2604Fh, 628B0D1FB5183FBFh, 6043822704FF0109h
LOAD:000000000044E848                 dq 4B54108FB6E06FB2h, 0B6D0E989D1010110h, 12AA86DF1FC6D0AFh
LOAD:000000000044E848                 dq 80146CB5B80F04CFh, 3822B018015F600Dh, 0EF01B615D0B05FBBh
LOAD:000000000044E848                 dq 12FFED2A258874Fh, 0A8DF4AFD8DFA554Dh, 0B154B87F6C7F4A54h
LOAD:000000000044E848                 dq 6F554D5F7F8D06C3h, 0DFD85FC83245B0D8h, 0DFD86FE2BB45D82Eh
LOAD:000000000044E848                 dq 166F06402EC5E4F4h, 0DF550689D826D155h, 46367205C9166F26h
LOAD:000000000044E848                 dq 566F365FC362E48Bh, 6677DF66B05C805Dh, 0DF99886F8B922EC1h
LOAD:000000000044E848                 dq 0BD6F99A67201762Eh, 803F6CC048BD9D13h, 6EB7239193E0CF68h
LOAD:000000000044E848                 dq 0D9DB6F4066C0770Fh, 5F73201F72A00161h, 2F6EDC9E4674A00Fh
LOAD:000000000044E848                 dq 9200784A87770F75h, 6D874B4B5D7E4E47h, 0F1F5FF3037FF12Fh
LOAD:000000000044E848                 dq 0B047979193C88830h, 0F2E4F2328A504087h, 0DB08030F0487F90h
LOAD:000000000044E848                 dq 0AFD04FAC007B3B32h, 7FBEC00AD5B0E00Fh, 924924CF010047B1h
LOAD:000000000044E848                 dq 0FF9500000084h, 4EF1C0004EE58h
LOAD:000000000044EF20
```

What are they?

They are essentially the compressed data of the original program’s code.

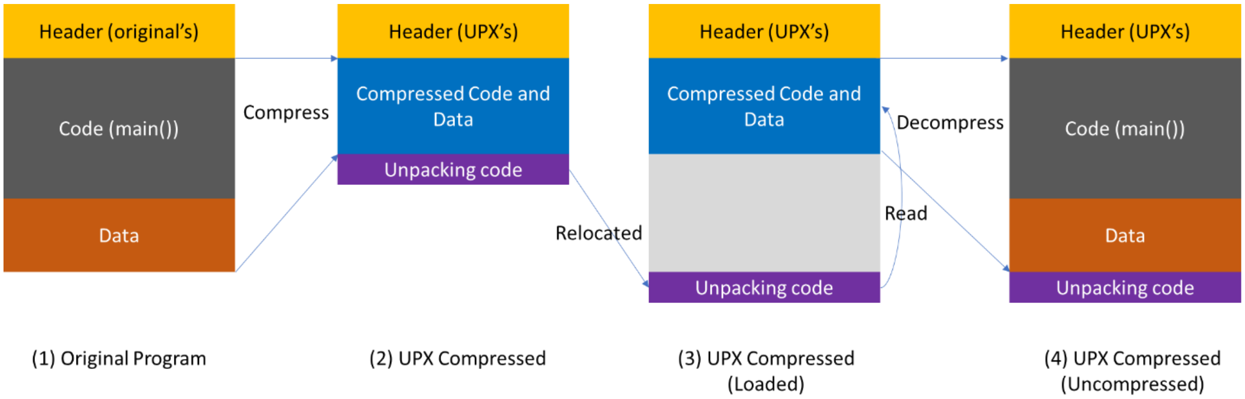
And, look at the addresses a little bit. Now the start function is at 44EF20. Remember that the original program’s main function is at 4009AE. Now see what’s in the 4009AE in the compressed program.



```
LOAD:0000000000400050                    dq offset dword_400000  ; Virtual address
LOAD:0000000000400058                    dq 400000h              ; Physical address
LOAD:0000000000400060                    dq 4F734h               ; Size in file image
LOAD:0000000000400068                    dq 4F734h               ; Size in memory image
LOAD:0000000000400070                    dq 200000h              ; Alignment
LOAD:0000000000400078 ; PHT Entry 1
LOAD:0000000000400078                    dd 1                    ; Type: LOAD
LOAD:000000000040007C                    dd 6                    ; Flags
LOAD:0000000000400080                    dq 0CD408h              ; File offset
LOAD:0000000000400088                    dq 6CD408h              ; Virtual address
LOAD:0000000000400090                    dq 6CD408h              ; Physical address
LOAD:0000000000400098                    dq 0                    ; Size in file image
LOAD:00000000004000A0                    dq 0                    ; Size in memory image
LOAD:00000000004000A8                    dq 200000h              ; Alignment
LOAD:00000000004000B0                    dq 21585055EC27AE62h, 160D081Ch, 0DED50000DED50h, 9100000190h
LOAD:00000000004000B0                    dq 0FF93FBF700000008h, 3010102464C457Fh, 900E01003E000200h
LOAD:00000000004000B0                    dq 40DBEC2FDF1F4008h, 3826450DE5102Fh, 0BF606C1F00210A06h
LOAD:00000000004000B0                    dq 0EF0F40010005571Eh, 2000206D7BAF0C94h, 0D207B3B806066F0Bh
LOAD:00000000004000B0                    dq 5B1C986C0E2F9EB2h, 6F0035D83B7B2D50h, 0C9400E2B01900704h
LOAD:00000000004000B0                    dq 9B04000044207C81h, 20DF1707DB621Ch, 0E55108727FB0530Fh
LOAD:00000000004000B0                    dq 3D8100100066474h, 148DF6E520F6EEDh, 0D36E124920F00h
LOAD:00000000004000B0                    dq 0C935FFF49254000h, 1949080004EB5300h, 100004CD79FBB500h
LOAD:00000000004000B0                    dq 2000A554E470106h, 3F200606E79DD7B7h, 0FFEEEF623F030614h
LOAD:00000000004000B0                    dq 8C970658E9ECC6FFh, 5710CD9C90B05BFEh, 6CA06067705E8B6Eh
LOAD:00000000004000B0                    dq 0B25BDBBEC5F77h, 1A402F580F421430h, 2F500F7D8BD9B149h
LOAD:00000000004000B0                    dq 190BC82F485F1670h, 32176ED040680079h, 0C0302F62EF38DA1Bh
LOAD:00000000004000B0                    dq 206D285E91921919h, 901867BB0BC8C85Eh, 0EC834F0BFFB7FD38h
LOAD:00000000004000B0                    dq 142C9D1D058B4808h, 0BFFD23E80574C085h, 8C428BD83EF76FFh
LOAD:00000000004000B0                    dq 68442225FF0100C3h, 0CA1AD2033419E915h, 0CD2033127C1A1E6Ch
LOAD:00000000004000B0                    dq 480CD92E9C0A8C80h, 19A40BC9CFAAC02h, 669019A4EACCF269h
LOAD:00000000004000B0                    dq 7EDFDDEDAECE2DCh, 3E030E8E0FCFFFFCh, 535511840FF68440h
```

What is happening? To give you the answer, UPX compressed the original code at the original address and stored them. Then, at runtime, they uncompress those data at exactly the same address of the original program.

Here is some visualization to help you to understand.



(1) Original Program

(2) UPX Compressed

(3) UPX Compressed (Loaded)

(4) UPX Compressed (Uncompressed)

First, the original program (1) is compressed to (2). The compressed binary (2) contains the compressed code and data and a code snippet that can unpack the code at runtime.

Then, when the (2) is executed, the unpacking code is relocated (this is done by specifying the virtual address of the unpacking code at the ELF file's header) so that the unpacking code does not use the virtual address of any of the original program's code and data. It ends up with the memory layout of (3).

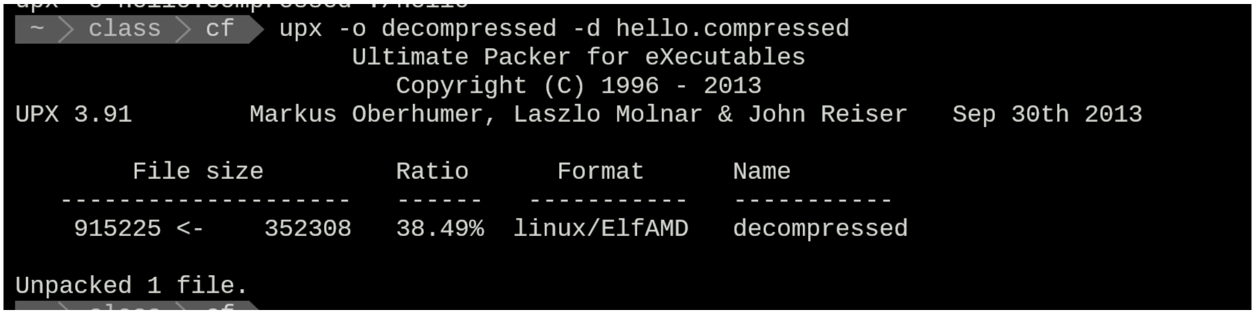
Finally, it runs the unpacking code. It reads the compressed code and data and then unpacks them to the memory in the exact same layout of the original program's code and data, resulting in the memory layout like (4).

Once the decompression is done, the unpacking code jumps to the original program's main() function (that is just decompressed). From now on, the program execution is exactly the same as the original program. The only difference is that we now have unnecessary unpacking code at (4) on the virtual memory.

This is how the UPX packer works at high-level, compressing the original program but ensuring that the compressed program functions exactly the same as the original program.

3. UPX decompression

A normal upx compressed binary can be decompressed by running the following command: "upx -o FILENAME -d [INPUTFILE]"



```
 ~ > class > cf > upx -o decompressed -d hello.compressed
                       Ultimate Packer for eXecutables
                          Copyright (C) 1996 - 2013
UPX 3.91        Markus Oberhumer, Laszlo Molnar & John Reiser   Sep 30th 2013

        File size         Ratio      Format      Name
   --------------------   ------   -----------   -----------
    915225 <-    352308   38.49%  linux/ElfAMD   decompressed

Unpacked 1 file.
```

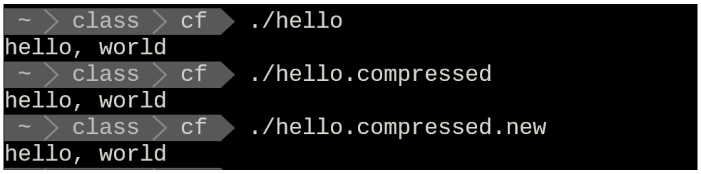
The above screenshot shows the example. So, this would be handy if you find upx compressed binaries (some malware are using those). The decompressed binary is the original binary, making it easy for you to analyze.

Of course, malware writers can prevent you from using the decompression functionality. This can be done by "corrupting some parts of the compressed file." If the parts are not needed for the execution but needed for the decompression, you can break the upx's decompression functionality.

4. Breaking UPX decompression

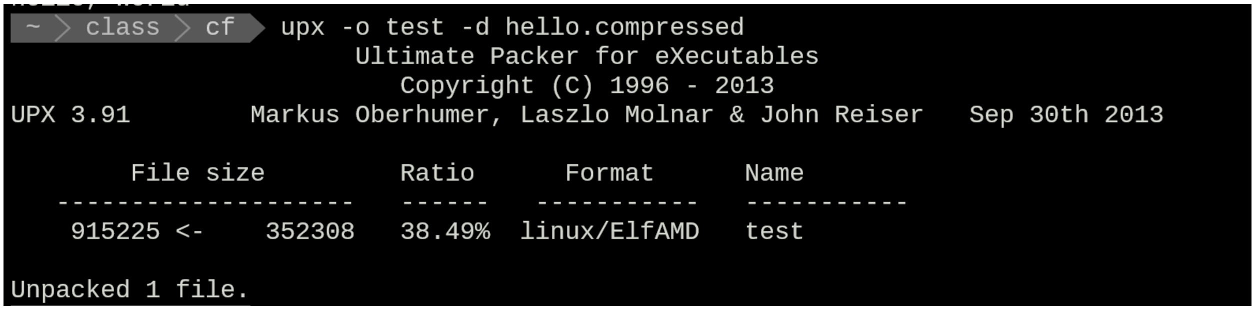
Then what is needed for the decompression but not required for the execution? I would like to leave this part for you to figure out. Here are some hints. First, you may download the UPX program's source code. It is open-source so that you can see and find out what they are checking and refuse to decompress. For example, they check some values in the UPX's header, and if the values do not make sense, the decompression function simply returns without doing it. However, some of those values may not be needed for decompression (but the decompression functionality would just check it for a sanity check).

Ok, now I will show the result.



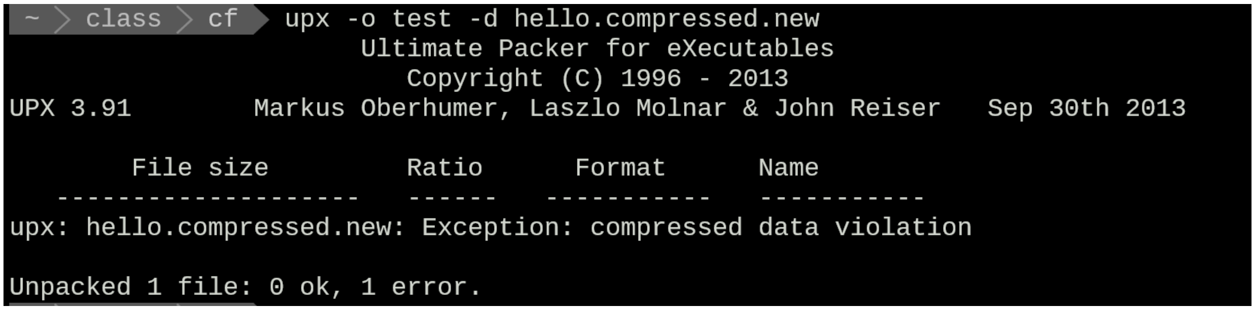
```
 ~ > class > cf >  ./hello
hello, world
 ~ > class > cf >  ./hello.compressed
hello, world
 ~ > class > cf >  ./hello.compressed.new
hello, world
```

I have three files: “hello” is the original binary, and “hello.compressed” is the upx compressed file. “hello.compressed.new” is the file I corrupt a value that will be checked by the upx’s decompression functionality (-d option) but ignored when it is executed. See the above screenshot that all of the three programs run perfectly.



```
 ~ > class > cf >  upx -o test -d hello.compressed
                       Ultimate Packer for eXecutables
                          Copyright (C) 1996 - 2013
UPX 3.91        Markus Oberhumer, Laszlo Molnar & John Reiser   Sep 30th 2013

        File size         Ratio      Format      Name
   --------------------   ------   -----------   -----------
    915225 <-    352308   38.49%  linux/ElfAMD   test

Unpacked 1 file.
```

Here is an example of running the decompression functionality. It successfully unpacked one file.



```
 ~ > class > cf >  upx -o test -d hello.compressed.new
                       Ultimate Packer for eXecutables
                          Copyright (C) 1996 - 2013
UPX 3.91        Markus Oberhumer, Laszlo Molnar & John Reiser   Sep 30th 2013

        File size         Ratio      Format      Name
   --------------------   ------   -----------   -----------
upx: hello.compressed.new: Exception: compressed data violation

Unpacked 1 file: 0 ok, 1 error.
```

And this is the new file that cannot be decompressed. I will give you all three files, and you should be able to figure out what happened.

For the last hint, you may want to compare “hello.compressed” and “hello.compressed.new”.

Read this for more information (I didn’t use the exact same method, but it will give you more idea of what “upx header” look file, and you should be able to understand “what I corrupt in the hello.compressed.new file”): https://medium.com/dark-sky-technology/repairing-a-damaged-upx-header-169e49cb5d0