



the binary itself. However, upon closer inspection, I noticed the binary lacked section headers, which made it challenging to analyze initially.

```
(base) dd@dd-spr:~/Desktop/ctfmid/reverse_eg/packer$ ls
notes out
(base) dd@dd-spr:~/Desktop/ctfmid/reverse_eg/packer$ file out
out: ELF 64-bit LSB executable, x86-64, version 1 (GNU/Linux), statically linked, no section header
```

Step 3: Checking Strings

Next, I used the strings command on the binary. This revealed that the binary was packed using the **UPX packer**. UPX (Ultimate Packer for Executables) compresses binaries and includes decompression stubs to restore them during execution.

For more details about UPX, you can check their [official website](http://upx.sourceforge.net).

```
ELF
USORH
PROT_EXEC|PROT_WRITE failed.
sInfo: This file is packed with the UPX executable packer http://upx.sf.net $
SID: UPX 3.95 Copyright (C) 1996-2018 the UPX Team. All Rights Reserved. $
j-X
PAX
```

Step 4: Initial Debugging

I loaded the binary into gdb, but due to the UPX compression, I couldn't retrieve any valuable information. At this stage, the packed binary was still obfuscated.

```
(base) dd@dd-spr:~/Desktop/ctfmid/reverse_eg/packer$ upx -d out
Ultimate Packer for eXecutables
Copyright (C) 1996 - 2024
UPX 4.2.2 Markus Oberhumer, Laszlo Molnar & John Reiser Jan 3rd 2024
-----
File size      Ratio      Format      Name
-----
[WARNING] bad b_info at 0x4b718
[WARNING] ... recovery at 0x4b714
877724 <- 336520 38.34% linux/amd64 out
Unpacked 1 file.
(base) dd@dd-spr:~/Desktop/ctfmid/reverse_eg/packer$ file out
out: ELF 64-bit LSB executable, x86-64, version 1 (GNU/Linux), statically linked, BuildID[sha1]=2e06e54d34a6d4b0c7ef71b3e1ce17f6db, for GNU/Linux 3.2.0, not stripped
```

Step 5: Unpacking the Binary

After some research, I discovered that UPX offers a built-in unpacking feature. Using the upx package, I unpacked the binary with the following command:

```
upx -d <binary_name>
```

This process successfully restored the binary's section headers, making it much easier to analyze.

```
int64_t var_88;
__builtin_strncpy(&var_88, "7069636f4354467b5539585f556e5034636b314e365f42316e34526933353f65313930633366337d", 0x64);
_IO_printf("Enter the password to unlock this-", 0);
_IO_fgets(rsp, var_a0, stdin);
_IO_printf("You entered: %s\n", 0);
if (sub_4010d0(rsp, &var_88, var_a0, &var_88) != 0)
    IO_puts("Access denied");
```