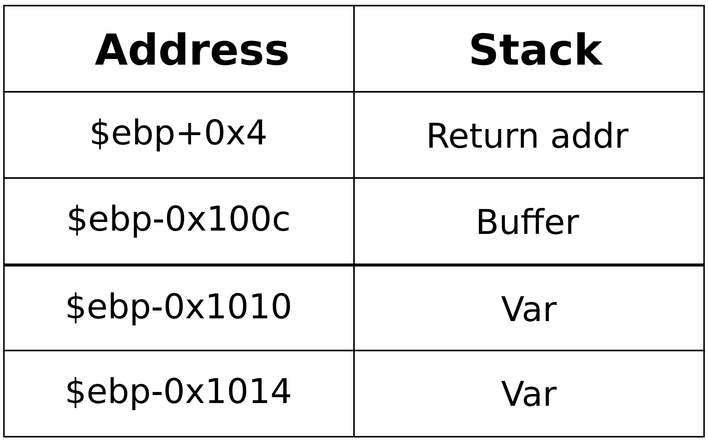
Proactive Computer Security
Assignment 6

Nikolaj Dybdahl Rathcke (rfq695)

June 8, 2016

The exploit is in the function `get_line` and `get_lines` in the `sortfile`. There is a commented disassembly in `src/sortfile.asm`, which comments (it is maybe a bit lacking) the key things that happens in the function. What we can extract from it is that our input begins at `ebp-0x100c` which means the offset for our attack is 4108 bytes. The drawing in 1 shows the stack in the scope of `get_lines`:



Address	Stack
$ebp+0x4	Return addr
$ebp-0x100c	Buffer
$ebp-0x1010	Var
$ebp-0x1014	Var

Figure 1: Stack in the scope of the function `get_lines`

Thus our attack begins at the offset of 4108. What the attack does is that we use `memcpy` to write one byte at a time to `.data`. The entire exploit can be found in `doit_sortfile.py`, but the first part looks like this:

```
exploit  = 'A' * 4108          # Offset
exploit += '\x50\x84\x04\x08' # memcpy
exploit += '\xbd\x89\x04\x08' # pop
exploit += '\x40\xa0\x04\x08' # .data
exploit += '\x54\x81\x04\x08' # /
exploit += '\x01\x00\x00\x00' # 1 byte
```

First we have the offset, then we call `memcpy` at the given address. The return address we have is a ROP gadget we can use - it pops three times, which are the three arguments given to `memcpy`. The arguments are the destination (`.data`), the source (address on a byte we want to use) and the number of bytes (which is 1). This way, we can write one byte at a time to `.data` (we want to write /bin/sh). When we have what we need, we make a system call:

```
exploit += '\x80\x84\x04\x08' # System
exploit += '\x84\x94\x04\x08' # Garbage - return
exploit += '\x40\xa0\x04\x08' # /bin/sh
```

i First we have the address on the call to `system`, the following is the return address, which we do not care about (it can be anything). Then finally we have the address on `.data` where we have /bin/sh. Running `sortfile` with this exploit will give us a shell, which is exactly what we wanted.
The reason we use `.data` is because it is writable and allocatable, which is a means to work around ASLR and DEP.