

Reverse Engineering

0x01 – Abradolf Lincler

0x02 – Mortycrypto

Mitchell Luke Tuck

CYBER SECURITY – AN OFFENSIVE MINDSET 2019

Table of Contents





0X01 – ABRADOLF LINCLER 2

0X02 – MORTYCRYPTO..... 8

REFERENCES..... 11

Reverse Engineering Lab

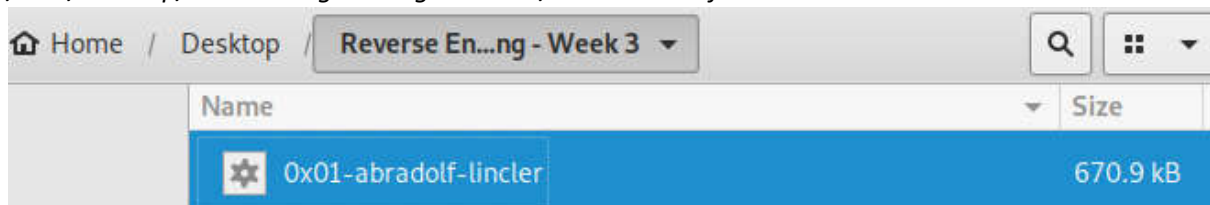
Reverse Engineering and Exploitation - 0x00 Home

 <p>0x01 Abradolf Lincler</p> <p>Although his looks are intimidating defeating this monster is easy.</p> <p>Download Challenge</p>	 <p>0x02 Morty Crypto</p> <p>Although Morty looks rather simple from the outside a little debugging is required to fully understand his little brain.</p> <p>Download Challenge</p>	 <p>0x03 Schmeckles</p> <p>Hmm... you might have to see if you can solve this challenge from memory</p> <p>Download Challenge</p>	 <p>0x04 Snowball</p> <p>Where is my flag Summer School?</p> <p>Download Challenge</p>
--	---	--	--

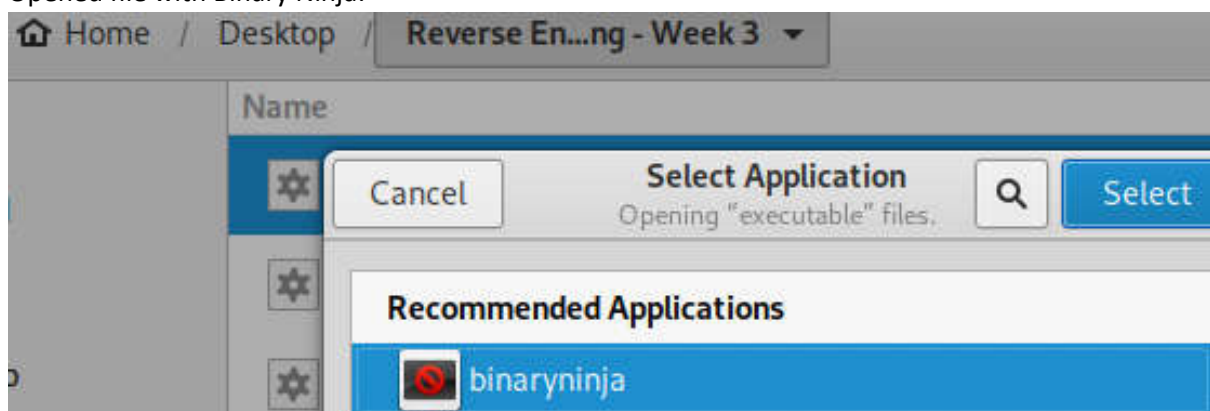
0X01 – ABRADOLF LINCLER

Downloaded and Decrypted Challenge to local file:

/root/Desktop/Reverse Engineering - Week 3/0x01-abradolf-lincler



Opened file with Binary Ninja:



If first time doing Reverse Engineering, downloaded Peda, GDB and PwnTools:

```
root@kali:~# git clone https://github.com/longld/peda.git ~/peda
Cloning into '/root/peda'...
remote: Enumerating objects: 4, done.
remote: Counting objects: 100% (4/4), done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 351 (delta 0), reused 2 (delta 0), pack-reused 347
Receiving objects: 100% (351/351), 331.58 KiB | 547.00 KiB/s, done.
Resolving deltas: 100% (217/217), done.
root@kali:~# echo "source ~/peda/peda.py" >> ~/.gdbinit
root@kali:~# echo "DONE! debug your program with gdb and enjoy"
DONE! debug your program with gdb and enjoy
root@kali:~# sudo -H pip install pwntools
Collecting pwntools
  Downloading https://files.pythonhosted.org/packages/f1/a6/530d4a5044be8d3dacbc42a32f05f8bedab5ad2b450263d/pwntools-3.12.2.tar.gz (2.0MB)
    100% |████████████████████████████████████████| 2.0MB 275kB/s
Collecting capstone>=3.0.5rc2 (from pwntools)
  Downloading https://files.pythonhosted.org/packages/35/0c/74db5b932865b7612658bd796cd02c26b1d567cbc9f0ab6/capstone-4.0.1-py2.py3-non
x86_64.whl (1.9MB)
    100% |████████████████████████████████████████| 1.9MB 142kB/s
```

Change into the directory where the file is located and run GDB:

```
root@kali:~# cd Desktop/
root@kali:~/Desktop# cd Reverse\ Engineering\ -\ Week\ 3/
root@kali:~/Desktop/Reverse Engineering - Week 3# ls
0x01-abradolf-lincler  peda-session-0x01-abradolf-lincler
0x02-mortcrypt0       peda-session-0x02-mortcrypt0.txt
0x03-shmeckles
root@kali:~/Desktop/Reverse Engineering - Week 3# gdb
GNU gdb (Debian 8.2-1) 8.2
Copyright (C) 2018 Free Software Foundation, Inc.
```

Now open file with Binary Ninja:

0x01-abradolf-linler — Binary Ninja

File Edit View Tools Help

0x01-abradolf-linler (ELF Graph) X

Warning: This function has unresolved stack usage. [View graph of stack usage](#)

```

int32_t __fastcall _start(uint32_t arg1,
                          int32_t arg2) __noreturn

_start:
xor     ebp, ebp    {0x0}
pop     esi {__return_addr}
mov     ecx, esp {arg_4}
and     esp, 0xffffffff
push    eax {var_4}
push    esp {var_4} {var_8}
push    edx {var_c}
call    sub_80499b3
add     ebx, 0x92670 {_GLOBAL_OFFSET_TABLE_}
lea     eax, [ebx-0x91700] {__libc_csu_fini}
push    eax {var_10} {__libc_csu_fini}
lea     eax, [ebx-0x917a0] {__libc_csu_init}
push    eax {var_14} {__libc_csu_init}
push    ecx {arg_4} {var_18}
push    esi {var_1c}
mov     eax, main
push    eax {var_20} {main}
call    __libc_start_main
hlt
  
```

Xrefs MiniGraph

08048018 from Data
struct Elf32_Header

Navigate to “{Main}”:

```

0x01-abradolf-llncler (ELF Graph)
Resume.col...
Resume_or_...
coded_value...
coded_value...
cheinfo
99b3
ocate_stati...
et_pc_thunk...
ter_tm_clon...
r_tm_clones
obal_dtors_...
ummy
n2
n1
et_pc_thunk...
mon_indeces...
start_main
et_pc_thunk...
ne_fd
check_stand...
setup_tls
et_pc_thunk...
csu_init
csu_fini
t_fail_base
t_fail
xt
eval
n
int32_t main()
ecx, [esp+0x4 {arg_4}]
esp, 0xffffffff
dword [ecx-0x4 {__return_addr}] {var_4}
ebp {__saved_ebp}
ebp, esp {__saved_ebp}
ebx {__saved_ebx}
ecx {arg_4} {var_10}
__x86.get_pc_thunk.ax
eax, 0x9247f {_GLOBAL_OFFSET_TABLE_}
esp, 0xc
edx, [eax-0x2dff4] {data_80ae00c, "\n          _..._\n          |||"}
edx {var_20} {data_80ae00c, "\n          _..._\n          |||||..."}
ebx, eax {_GLOBAL_OFFSET_TABLE_}
_IO_puts
esp, 0x10
function1
eax, 0x0
esp, [ebp-0x8]
ecx {var_10}
ebx {__saved_ebx}
ebp {__saved_ebp}
esp, [ecx-0x4]

```

Now that we're here, we want to see what happens when we run this code, please note that you **WOULD NOT** do this if you were analysing Malware:

root@kali: ~/Desktop/Reverse Engineering - Week 3

File Edit View Search Terminal Help

root@kali:~/Desktop/Reverse Engineering - Week 3# ./0x01-abradolf-lincler

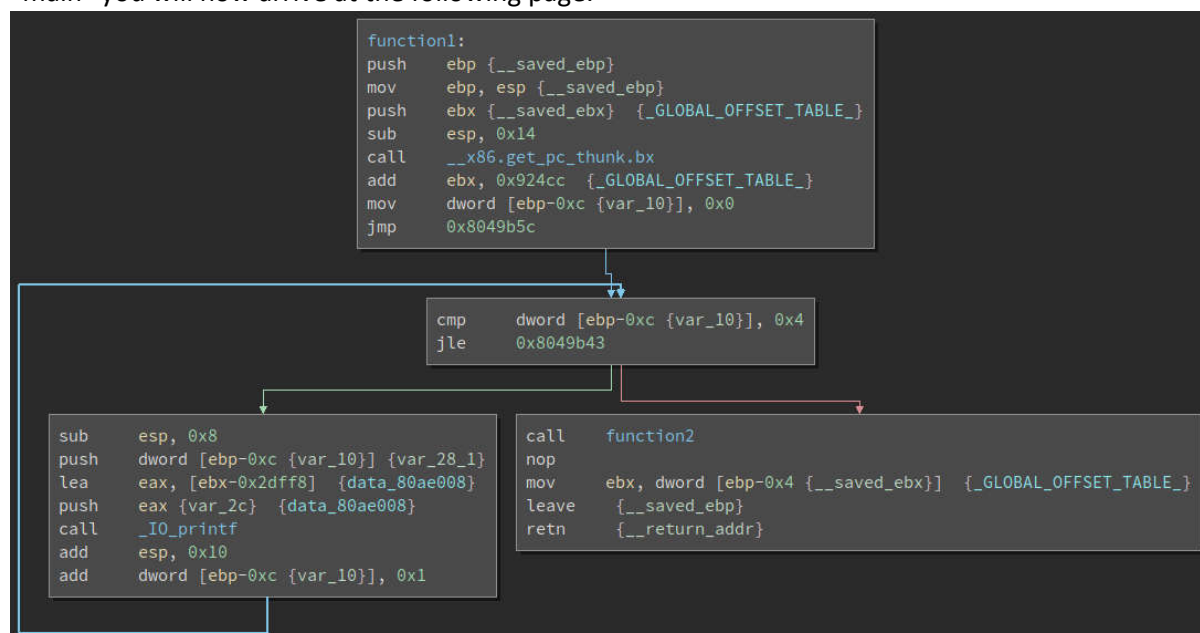
```

GNU gdb (Debian 8.2-1) 8.2
Copyright (C) 2018 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later http://www.gnu.org/licenses/gpl.html.
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration.
For bug reporting instructions, please see:
http://www.gnu.org/software/gdb/.
Type "show" for a list of all commands and other options.
Type "help" for a list of all commands.
Type "help command-name" for details about a command.
Type "show" for a list of all commands and other options.
Type "help" for a list of all commands.
Type "help command-name" for details about a command.

```

Do not think what RE can do for you. But what can you RE?

Now that we've ran it, we can see that the ASCII in the "main" forms an artwork. Now we need to find a way to access this function and find out details about this, do this by double clicking "function1" on "main" you will now arrive at the following page:



From this, we can see that there is a for loop (blue line) that adds 0x1 (1) after each iteration. But what stands out to me is the else statement (red line) that calls function two, so I navigated too it:

```

function2:
push    ebp {__saved_ebp}
mov     ebp, esp {__saved_ebp}
sub     esp, 0x20
call    __x86.get_pc_thunk.ax
add     eax, 0x92500 {_GLOBAL_OFFSET_TABLE_}
mov     dword [ebp-0x11 {var_15}], 0x47414c46
mov     dword [ebp-0xd {var_11}], 0x6234427b
mov     dword [ebp-0x9 {var_d}], 0x31527379
mov     dword [ebp-0x5 {var_9}], 0x7d747372
mov     byte [ebp-0x1 {var_5}], 0x0
nop
leave   {__saved_ebp}
retn    {__return_addr}
  
```

Here I can see that there is a string (mov) that terminates, the Function 2 is terminated by a Null byte (0x0). So, I decided to covert each line of the string to Display as a Character Constant, this revealed the flag:

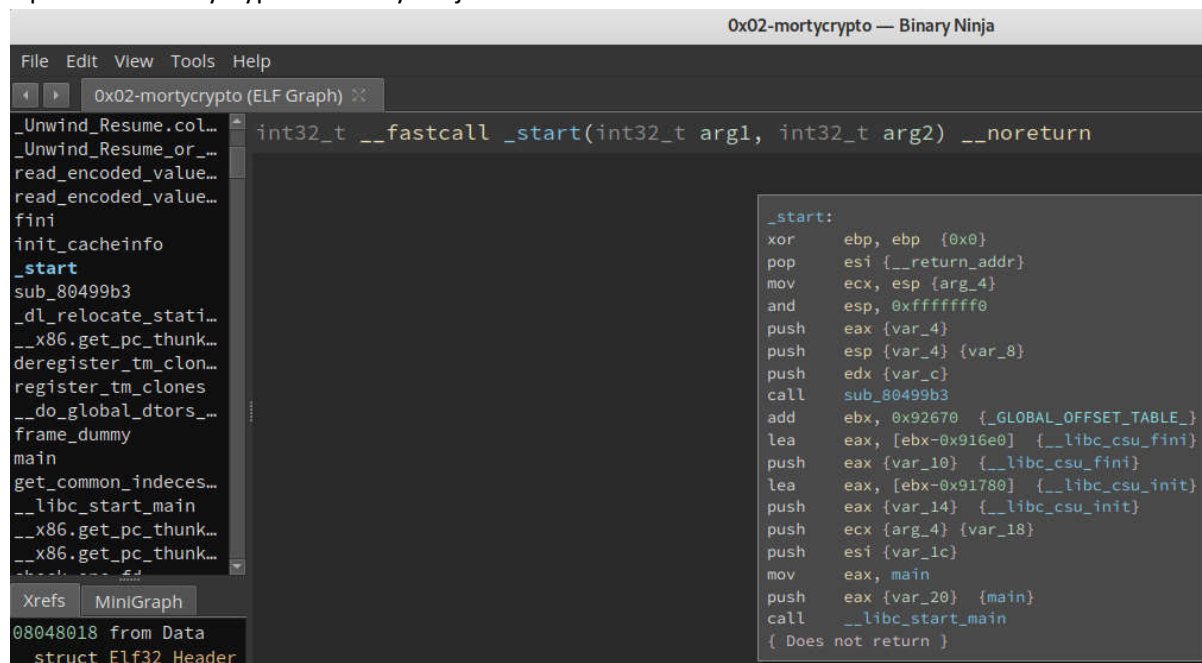
```
function2:
push     ebp {__saved_ebp}
mov      ebp, esp {__saved_ebp}
sub      esp, 0x20
call     __x86.get_pc_thunk.ax
add      eax, 0x92500 {_GLOBAL_OFFSET_TABLE_}
mov      dword [ebp-0x11 {var_15}], 'FLAG'
mov      dword [ebp-0xd {var_11}], '{B4b'
mov      dword [ebp-0x9 {var_d}], 'ysR1'
mov      dword [ebp-0x5 {var_9}], 'rst}'
mov      byte [ebp-0x1 {var_5}], '\x00'
nop
leave    {__saved_ebp}
retn     {__return_addr}
```

I found this to be a very good introduction into one of the most complex fields in IT, I aspire to perfect these techniques as it is a very useful skill to have as an IT Security Professional.

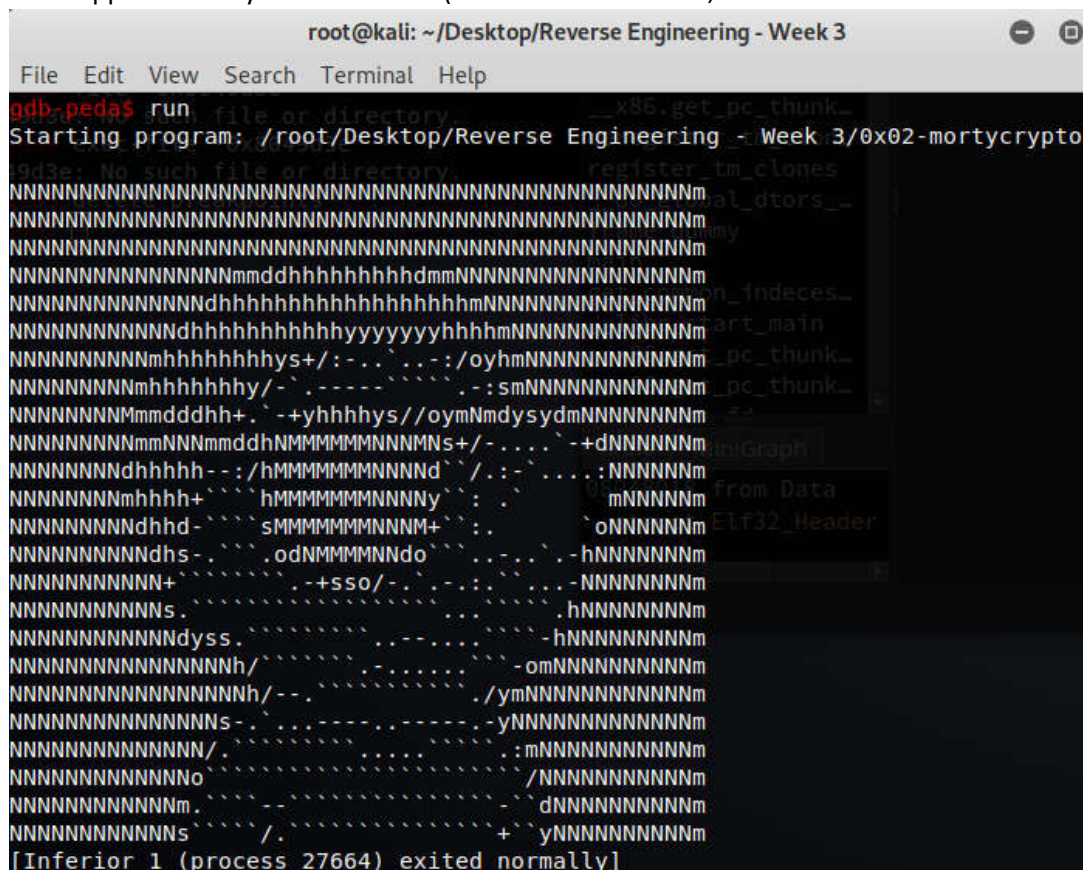
0X02 – MORTYCRYPTO

As we already have all the necessary packages installed from the previous Challenge, we can get straight into this!

Open 0x02-mortycrypto in Binary Ninja:

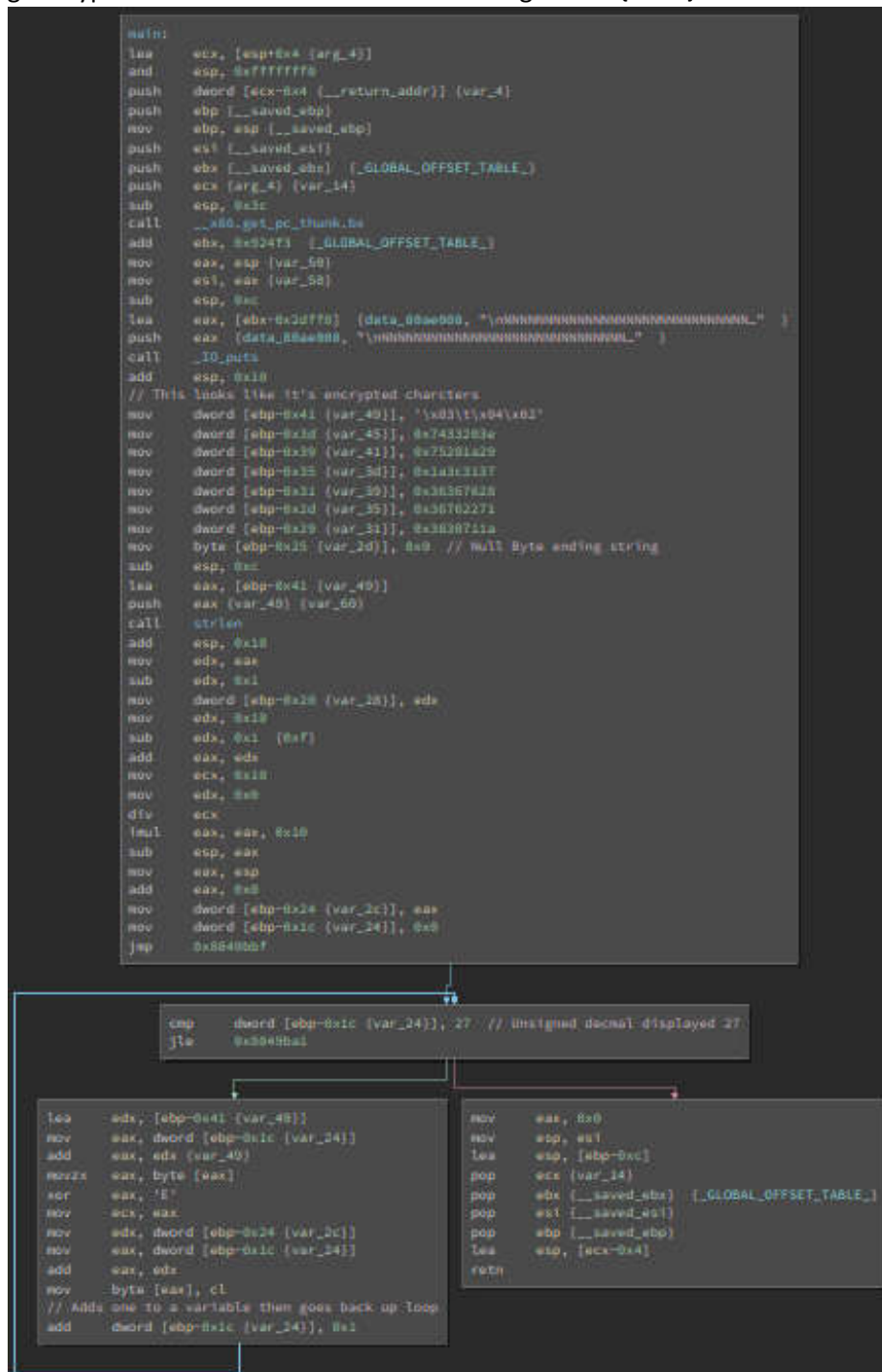


See what happens when you run the file (as mentioned in 0x01, wouldn't do this with Malware):



As we can see, it outputs ASCII Art.

Now in Binary Ninja we want to analyse the `_start` to get an idea of how this code is functioning. The first thing I'd note is that there is an "xor" that forms part of this, xor's are known to be utilised primarily in encryption methods, and with the name "mortycrypto" we can almost guarantee that this code is using encryption of some form. We will now navigate to "{Main}":



We can see again that in main, ASCII is utilised to create the art, and that there are quite a few strings being inserted in and a strlen being called. It then jumps to an if/else statement with a for loop on the if statement. In the bottom left box, we can see this is where the encryption is being utilised (xor) which I converted to find it to be "E", and at the bottom it adds 1 to the variable before looping up to the cmp, this seems to be looping through the encrypted data. In cmp I decided I wanted to see the hex number converted to an unsigned decimal which revealed 27, this shows that the code will count up to =<27.

I identified 0x8049bc5 to be of interest as there seems to be another string that's being entered and being terminated by a null byte, though as mentioned above (converting 0x8049bc5 over shows encryption). I then set a breakpoint pointing at 0x8049bc5:

[illegible]

Which then displayed the following flag/strings:

```

root@kali: ~/Desktop/Reverse Engineering - Week 3
File Edit View Search Terminal Help
EIP: 0x8049baf (<main+186>: mov ecx,eax)
EFLAGS: 0x202 (carry parity adjust zero sign trap INTERRUPT direction overflow)
-----code-----
0x8049ba7 <main+178>: add eax,edx
0x8049ba9 <main+180>: movzx eax,BYTE PTR [eax]
0x8049bac <main+183>: xor eax,0x45
=> 0x8049baf <main+186>: mov ecx,eax
0x8049bb1 <main+188>: mov edx,DWORD PTR [ebp-0x24]
0x8049bb4 <main+191>: mov eax,DWORD PTR [ebp-0x1c]
0x8049bb7 <main+194>: add eax,edx
0x8049bb9 <main+196>: mov BYTE PTR [eax],cl
-----stack-----
0000| 0xfffffd230 ("FLAG{ev1l_m0rty_m3s\377}")
0004| 0xfffffd234 ("{ev1l_m0rty_m3s\377}")
0008| 0xfffffd238 ("l_m0rty_m3s\377")
0012| 0xfffffd23c ("rty_m3s\377")
0016| 0xfffffd240 --> 0xff73336d
0020| 0xfffffd244 --> 0x0
0024| 0xfffffd248 --> 0x3
0028| 0xfffffd24c --> 0x8049b0d (<main+24>: add ebx,0x924f3)
-----
Legend: code, data, rodata, value
0x08049baf in main()
gdb-peda$

```

REFERENCES

Binary Ninja 2019, *Binary Ninja Demo*, viewed 25 February 2019, <<https://binary.ninja/>>.