Vulnerable Code

```c
#include <string.h>

void vulnerable_function(char* input) {
    char buffer[10];
    strcpy(buffer, input);
}

int main() {
    char input_string[] = "This input is too long!";
    vulnerable_function(input_string);
    return 0;
}
```

Stack 1

c

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


char buffer[20];

int copy_buffer(char *input_buffer) {
    strcpy(buffer, argv[1]);

    printf("DEBUG: strcpy() executed...\n");
}


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




   if (argc < 2) {
      printf("Syntax error\n");
      printf("Syntax: %s <characters>\n", argv[0]);
      exit(0);
   }

    copy_buffer(argv[1]);
```

```
    printf("buffer content= %s\n", buffer);
    return 0;
}
```

Walkthrough Windbg

```
cl.exe /Zi /GS- /EH-
# debugging symbols, no stack security, no seh (not working??)

bp $exentry (if no symbols)
bp example1!main
# F5
# debug -> make sure that source mode is unchecked

# x86 calling convention: parameter 1-3 on stack
# x86 function prolog: push ebp, mov ebp, esp
# [ebp+8] == local variable

# dd esp -> first address on stack == return pointer
# dd ebp -> old base pointer
# dd poi(ebp+8)
```

Stack 2

```
titan.net2k@protonmail.ch
show me an example of a buffer overflow for windows in c that uses
commandline input and a seperate function

Sure! Here's an example of a buffer overflow vulnerability in C for Windows
that uses command-line input and a separate function:

c
Copy code
#include <stdio.h>
#include <string.h>

void vulnerable_function(char *input) {
    char buffer[10];
    strcpy(buffer, input);
}

int main(int argc, char *argv[]) {
    vulnerable_function(argv[1]);
    printf("You entered: %s\n", argv[1]);
```

```
    return 0;
}
In this example, the program takes a string argument from the command line and passes it to the vulnerable_function function, which copies it into a fixed-size buffer of size 10 using the strcpy function. If the argument passed in is longer than the buffer size, the strcpy function will write beyond the bounds of the buffer and overwrite other data on the stack.

To exploit this vulnerability, an attacker could pass in a string argument longer than the buffer size that contains code or data that they want to execute or modify.

To prevent this type of vulnerability, it's important to use functions that limit the amount of data written to a buffer, such as strncpy, and to validate input to ensure that it does not exceed the buffer size. Additionally, it's important to use secure programming practices like input validation and data sanitization. One way to fix this vulnerability is to use a buffer of sufficient size for the input and to use functions that limit the amount of data written to the buffer. Another solution is to use a buffer overflow protection mechanism like Microsoft's /GS (Buffer Security Check) or Address Space Layout Randomization (ASLR) to prevent stack buffer overflows.
```

Heap 2

```
#
https://infosecwriteups.com/stack-based-buffer-overflow-practical-for-windows-vulnserver-8d2be7321af5

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

int main(void)
{
    char buff[15];
    int pass = 0;
    printf("\n Enter the password : \n");
    gets(buff);
    if(strcmp(buff, "P@ssw0rd")) {
        printf("\n Wrong Password \n");
    }
    else {
        printf("\n Correct Password \n");
        pass = 1;
    }
    if(pass) {
        printf("\n Execute guarded command \n");
        char command[50];
```

```
        strcpy(command, "ls -l" );
        system(command);
    }
return 0;
}
```

Heap2-2

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

// Use after free example, original code from https://exploit.education

#define SERVICE_SIZE 32

struct AuthStruct {
  char name[32];
  int is_authenticated;
};

struct AuthStruct *auth;
char *service;

int main(int argc, char **argv)
{
  char line[128];

  while(1) {
    printf("[ auth = %p, service = %p ]\n", auth, service);
    if(fgets(line, sizeof(line), stdin) == NULL) break;

    if(strncmp(line, "user ", 5) == 0) {
      auth = malloc(sizeof(*auth));
      memset(auth, 0, sizeof(*auth));
      if(strlen(line + 5) < 31) {
        strcpy(auth->name, line + 5);
      }
    }
    if(strncmp(line, "reset", 5) == 0) {
      free(auth);
    }
    if(strncmp(line, "service", 6) == 0) {
      service = malloc(SERVICE_SIZE);
      strcpy(service, line+7);
    }
```

```
    if(strncmp(line, "login", 5) == 0) {
      if(auth->is_authenticated) {
        printf("you have logged in already!\n");
      } else {
        printf("please enter your password\n");
      }
    }
  }
}
```