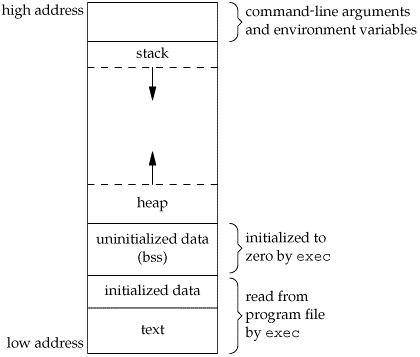
**MEMORY FOR A RUNNING PROCESS**

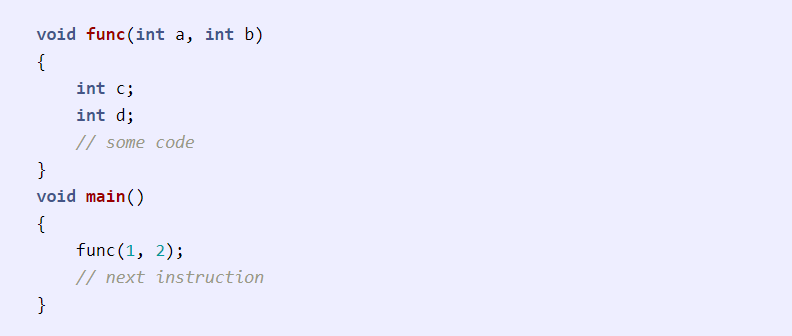
[](https://github.com/Mucahit3/Security-Exercises/blob/master/Screenshots/1.gif)

1. **Stack**: This is the place where all the function parameters, return addresses and the local variables of the function are stored. It's a LIFO structure. It grows downward in memory(from higher address space to lower address space) as new function calls are made. We will examine the stack in more detail later.
2. **Heap**: All the dynamically allocated memory resides here. Whenever we use malloc to get memory dynamically, it is allocated from the heap. The heap grows upwards in memory(from lower to higher memory addresses) as more and more memory is required.
3. **Command line arguments and environment variables**: The arguments passed to a program before running and the environment variables are stored in this section.
4. **Uninitialized data(Bss Segment)**: All the uninitialized data is stored here. This consists of all global and static variables which are not initialized by the programmer. The kernel initializes them to arithmetic 0 by default.
5. **Initialized data(Data Segment)**: All the initialized data is stored here. This constists of all global and static variables which are initialised by the programmer.
6. **Text**: This is the section where the executable code is stored. The loader loads instructions from here and executes them. It is often read only.

**SOME IMPORTANT REGISTERS**

1. **%rip**: The **Instruction pointer register**. It stores the address of the next instruction to be executed. After every instruction execution it's value is incremented depending upon the size of an instrution.
2. **%rsp**: The **Stack pointer register**. It stores the address of the top of the stack. This is the address of the last element on the stack. The stack grows downward in memory(from higher address values to lower address values). So the %rsp points to the value in stack at the lowest memory address.
3. **%rbp**: The **Base pointer register**. The %rbp register usually set to %rsp at the start of the function. This is done to keep tab of function parameters and local variables. Local variables are accessed by subtracting offsets from %rbp and function parameters are accessed by adding offsets to it as you shall see in the next section.

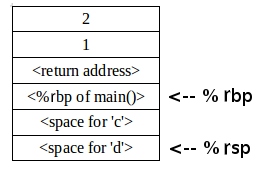
**MEMORY MANAGEMENT DURING FUNCTION CALLS**

[](https://github.com/Mucahit3/Security-Exercises/blob/master/Screenshots/2.png)

Assume our **%rip** is pointing to the **func** call in **main**. The following steps would be taken:

1. A function call is found, push parameters on the stack from right to left(in reverse order). So **2** will be pushed first and then **1**.
2. We need to know where to return after **func** is completed, so push the address of the next instruction on the stack.
3. Find the address of **func** and set **%rip** to that value. The control has been transferred to **func()**.
4. As we are in a new function we need to update **%rbp**. Before updating we save it on the stack so that we can return later back to **main**. So **%rbp** is pushed on the stack.
5. Set **%rbp** to be equal to **%rsp**. **%rbp** now points to current stack pointer.
6. Push local variables onto the stack/reserver space for them on stack. **%rsp** will be changed in this step.
7. After **func** gets over we need to reset the previous stack frame. So set **%rsp** back to **%rbp**. Then pop the earlier **%rbp** from stack, store it back in **%rbp**. So the base pointer register points back to where it pointed in **main**.
8. Pop the return address from stack and set **%rip** to it. The control flow comes back to **main**, just after the **func** function call.

This is how the stack would look while in **func**.

[](https://github.com/Mucahit3/Security-Exercises/blob/master/Screenshots/3.png)

Demo

**Disable canary:**

gcc vuln.c -o vuln\_disable\_canary -fno-stack-protector

**Disable DEP:**

gcc vuln.c -o vuln\_disable\_dep -z execstack

**Disable PIE:**

gcc vuln.c -o vuln\_disable\_pie -no-pie

gcc vulnerable.c -o vuln\_disable\_all -fno-stack-protector -z execstack -no-pie

objdump -d vul

nano payloadGenerator.py

python3 payloadGenerator.py > t

./vuln\_disable\_all < t