SEED
Buffer Overflow Lab

Outline

Principle

1. **High Level Picture**
2. **Program Memory Layout**
3. **Function Stack Layout**
4. **Function Call Chain**

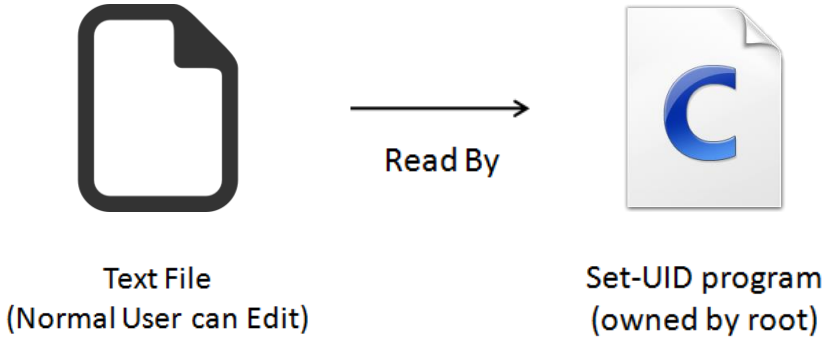
Practice

1. **Vulnerable Program**
2. **Task Breakdown**
3. **Environment Setup**
4. **Run Tasks**
5. **Run the Exploit**

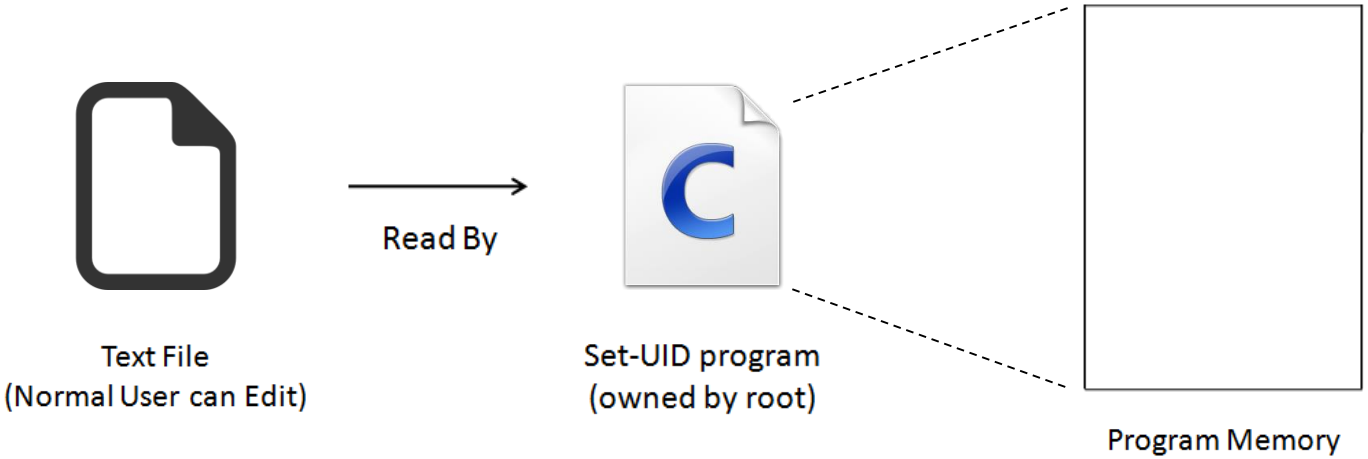
High Level Picture

Set-UID program
(owned by root)

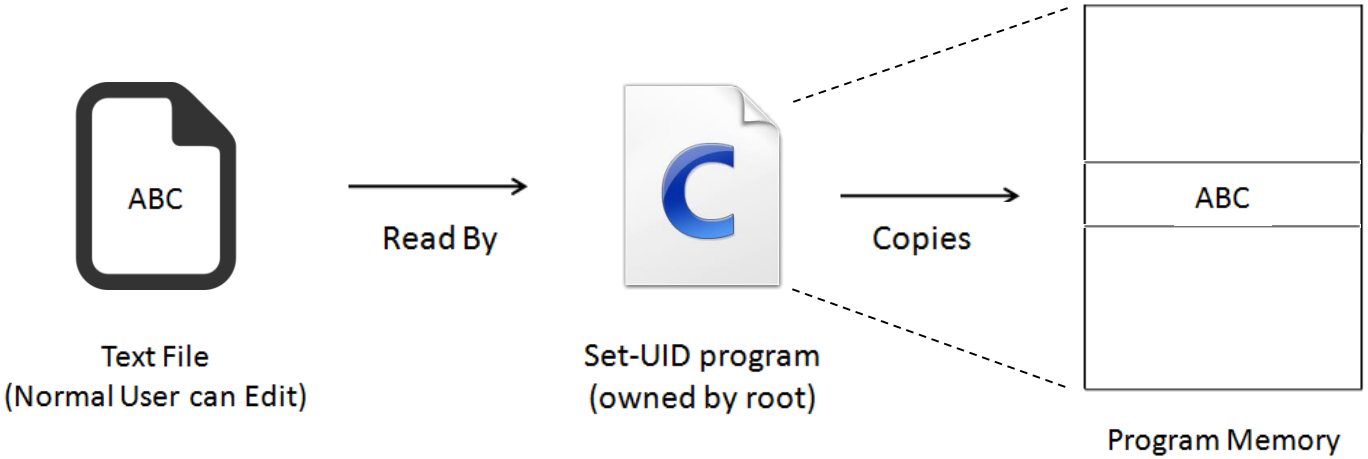
High Level Picture



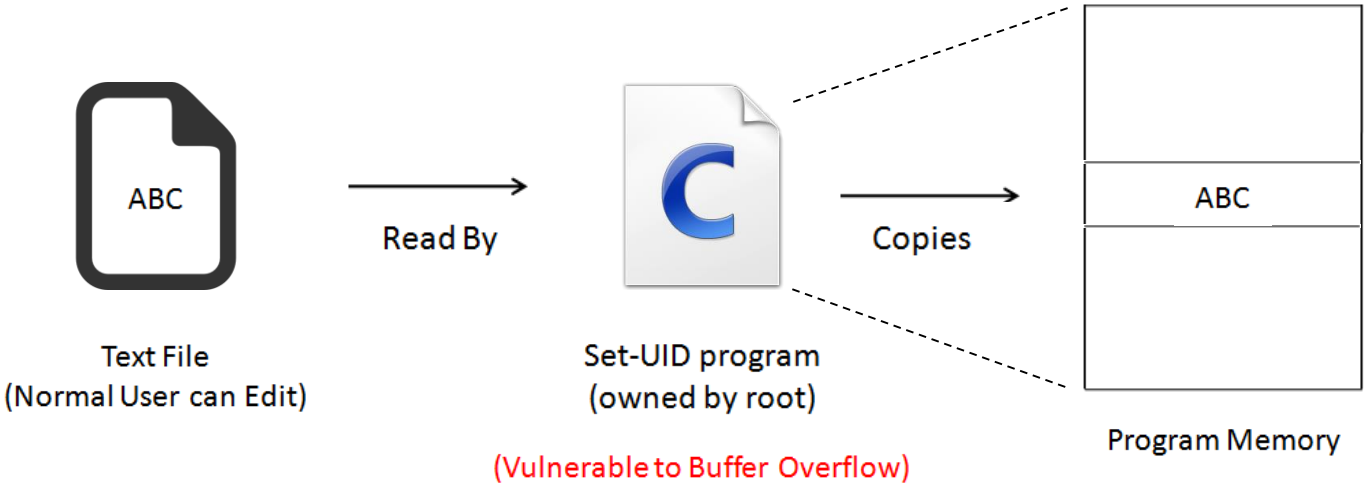
High Level Picture



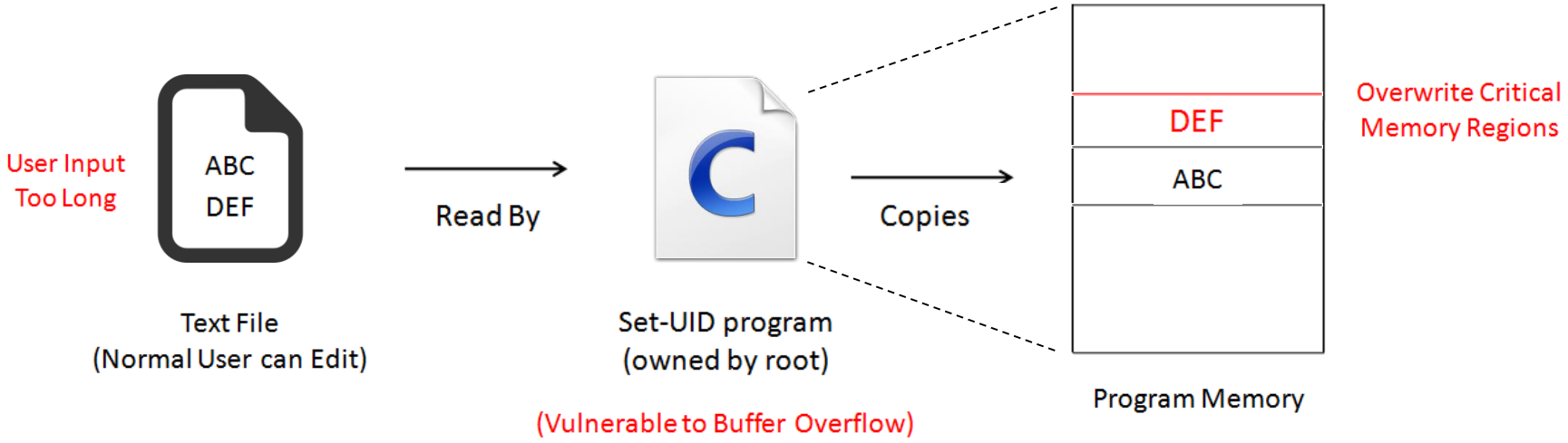
High Level Picture



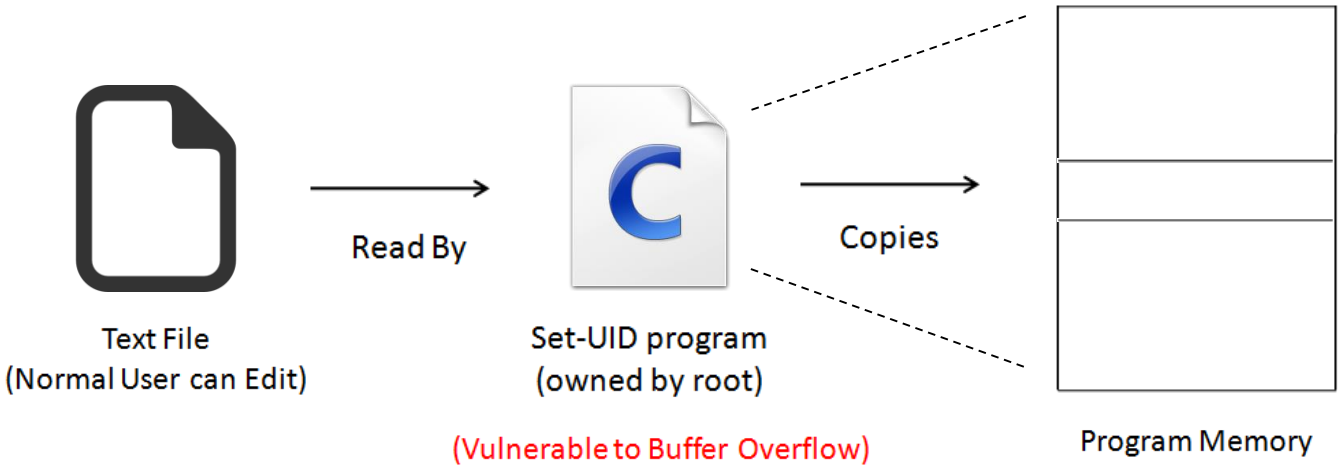
High Level Picture



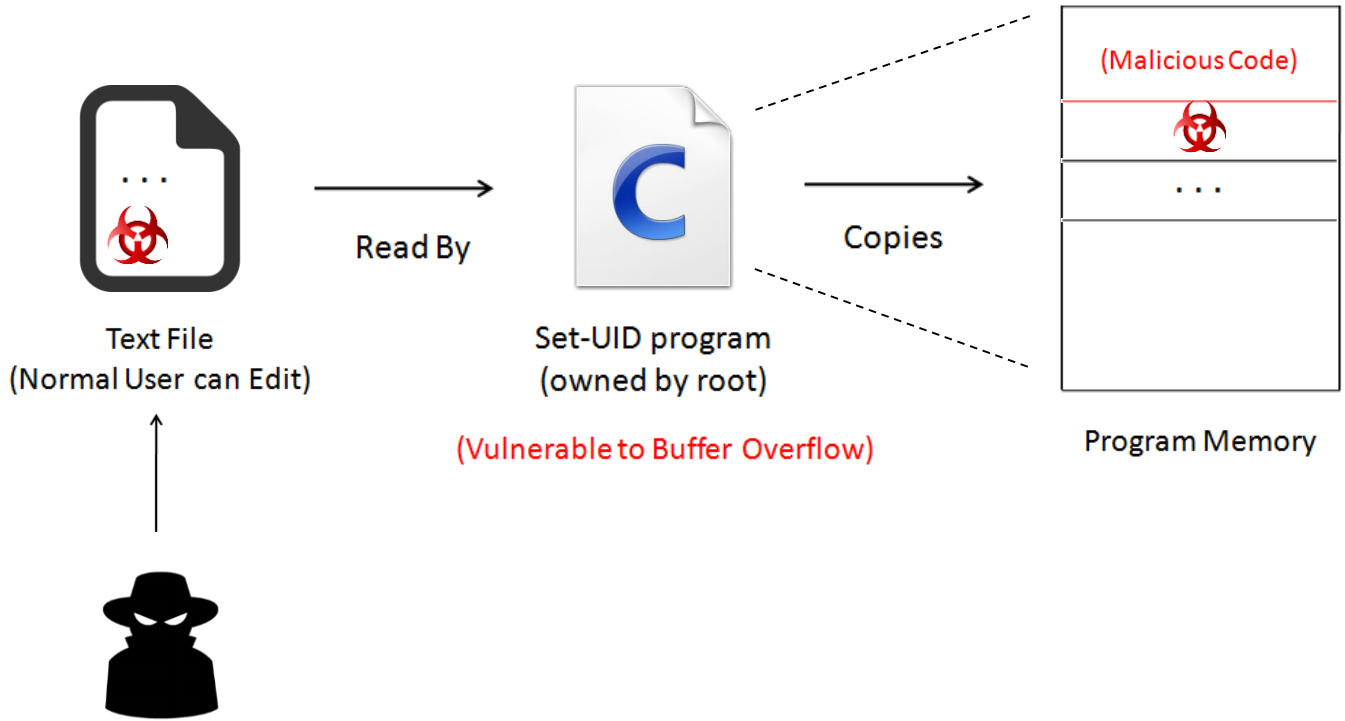
High Level Picture



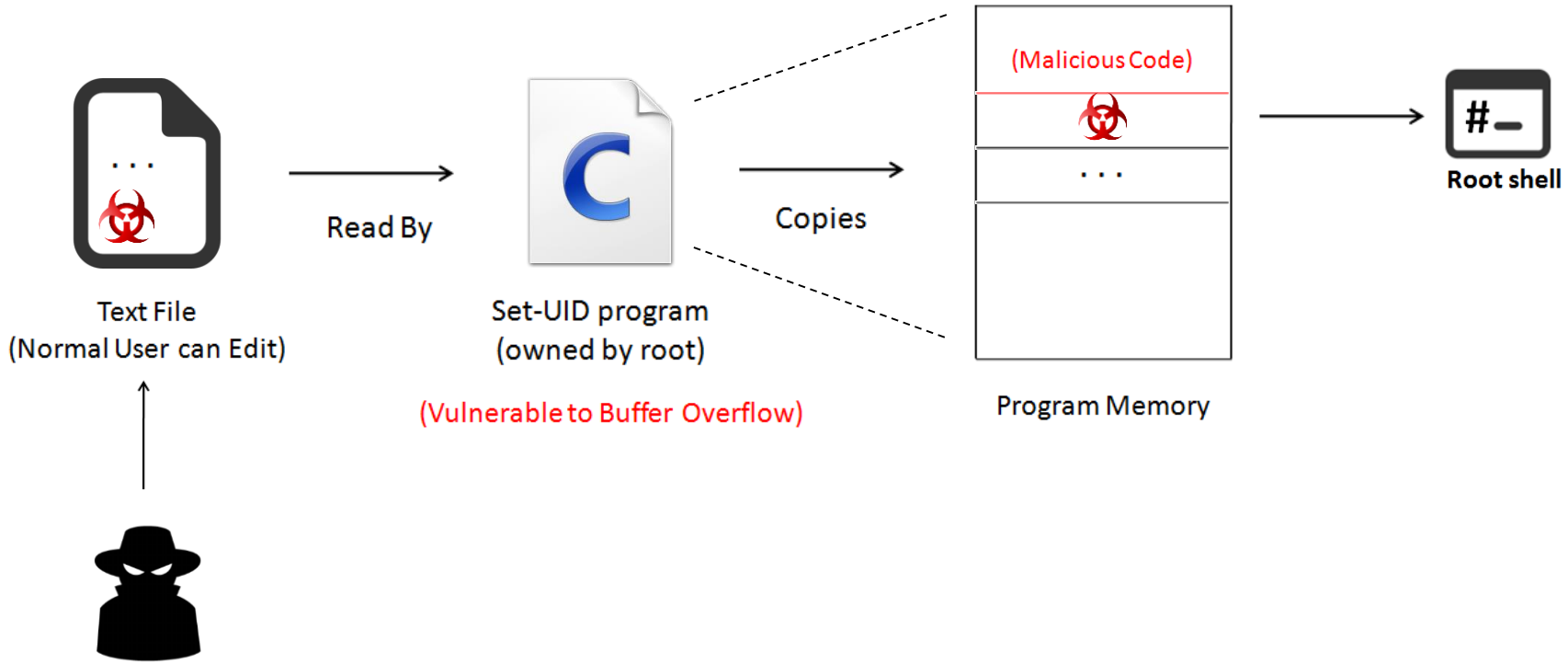
High Level Picture



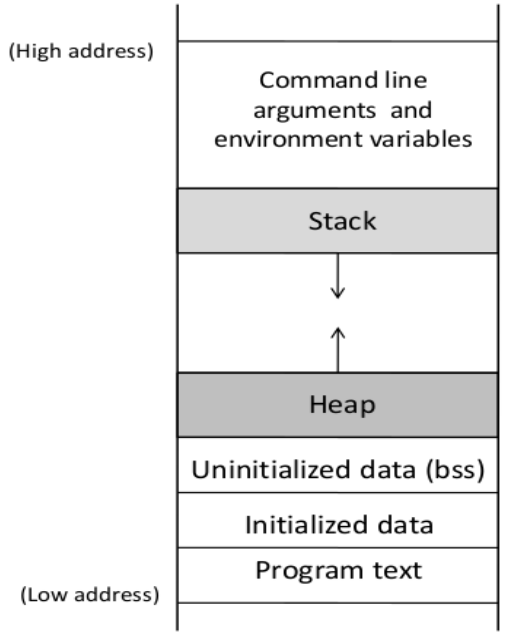
High Level Picture



High Level Picture

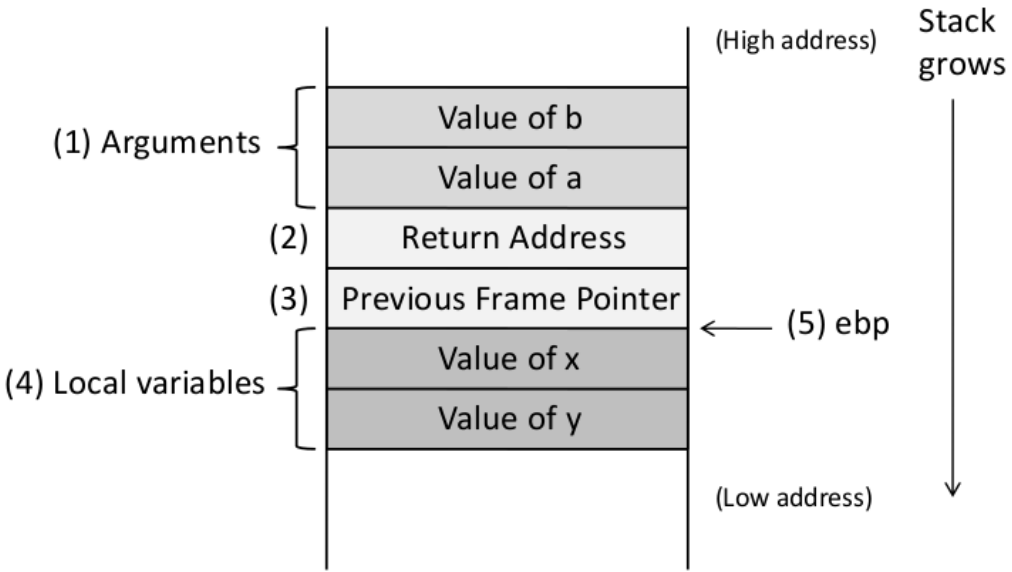


Principle
Program Memory Layout



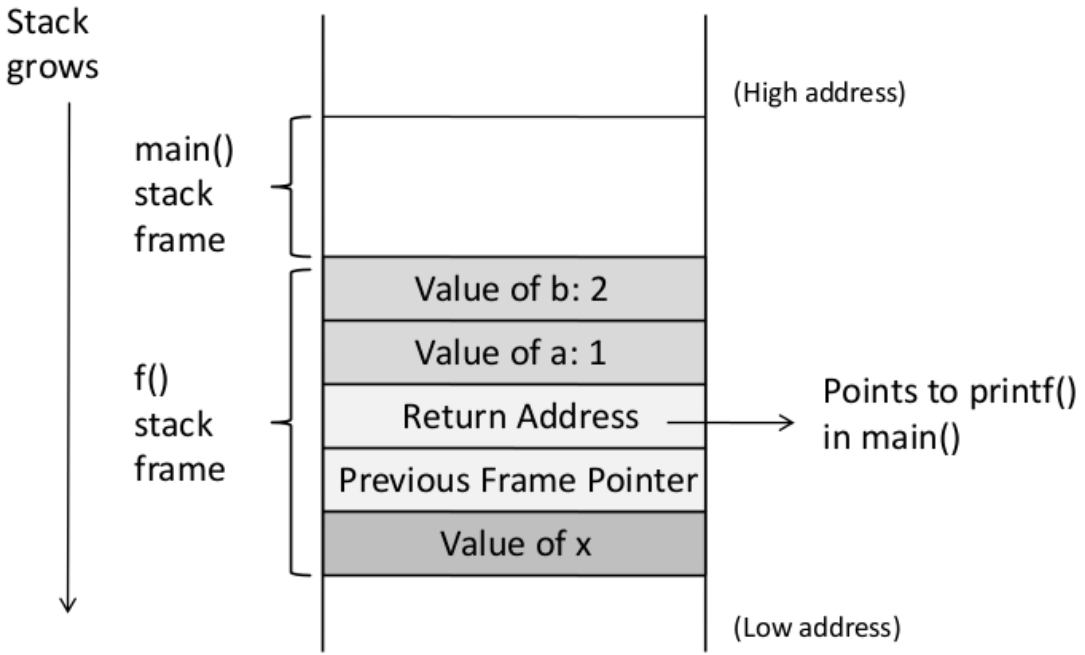
Function Stack Layout

```
void func(int a, int b)
{
   int x,y ;
}
```



Function Call Chain

```
void f(int a, int b)
{
  int x;
}
void main()
{
  f(1,2);
  printf("hello world");
}
```



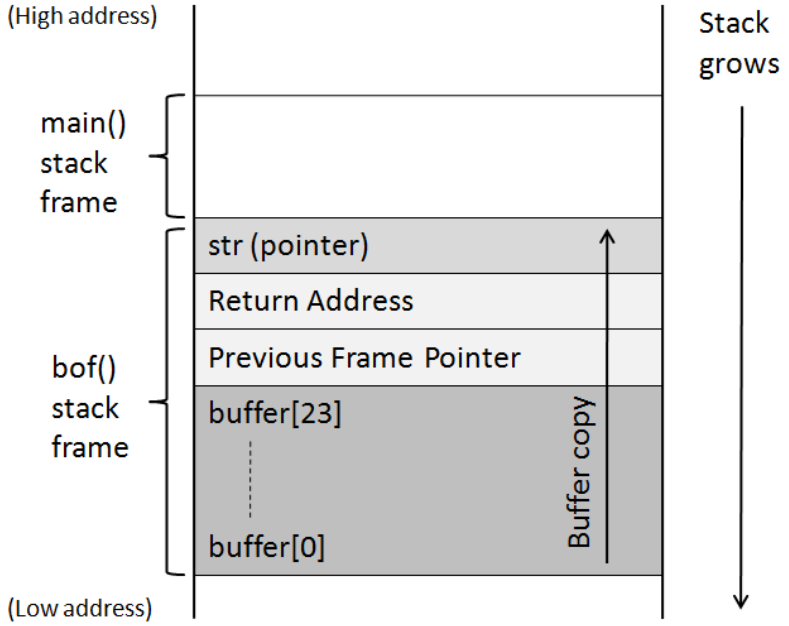
Practice
Vulnerable Program (stack.c)

```
int main(int argc, char **argv)
{   char str[517];
    FILE *badfile;
    // 1. Opens badfile
    badfile = fopen("badfile", "r");
    // 2. Reads upto 517 bytes from badfile
    fread(str, sizeof(char), 517, badfile);
    // 3. Call vulnerable function
    bof(str);
    printf("Returned Properly\n");
    return 1;
}
```

```
int bof(char *str)
{
    char buffer[24];
    // 4. Copy argument into buffer
    // (Possible Buffer Overflow)
    strcpy(buffer, str);
    return 1;
}
```

Buffer Overflow in stack.c

```
int bof(char *str)
{
    char buffer[24];
    // 4. Copy argument into buffer
    // (Possible Buffer Overflow)
    strcpy(buffer, str);
    return 1;
}
```

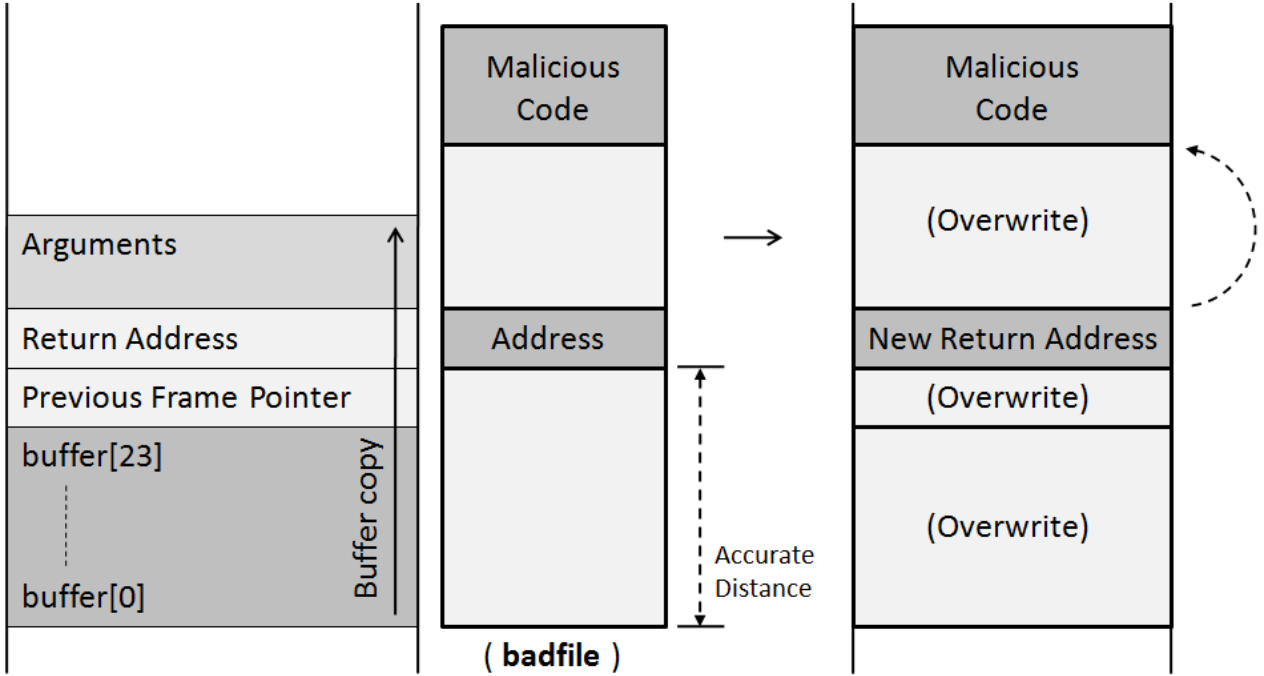


Program Behavior

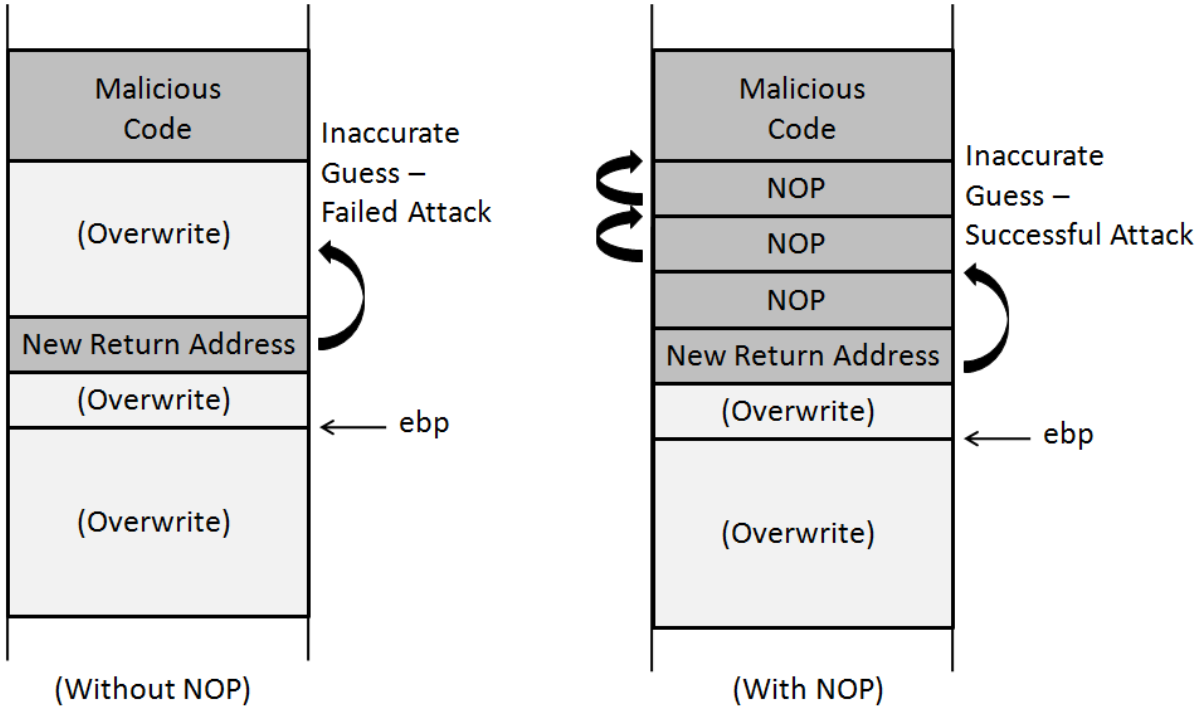
Show program behavior for badfile of length:

- **< 24 bytes**
- **> 24 bytes**

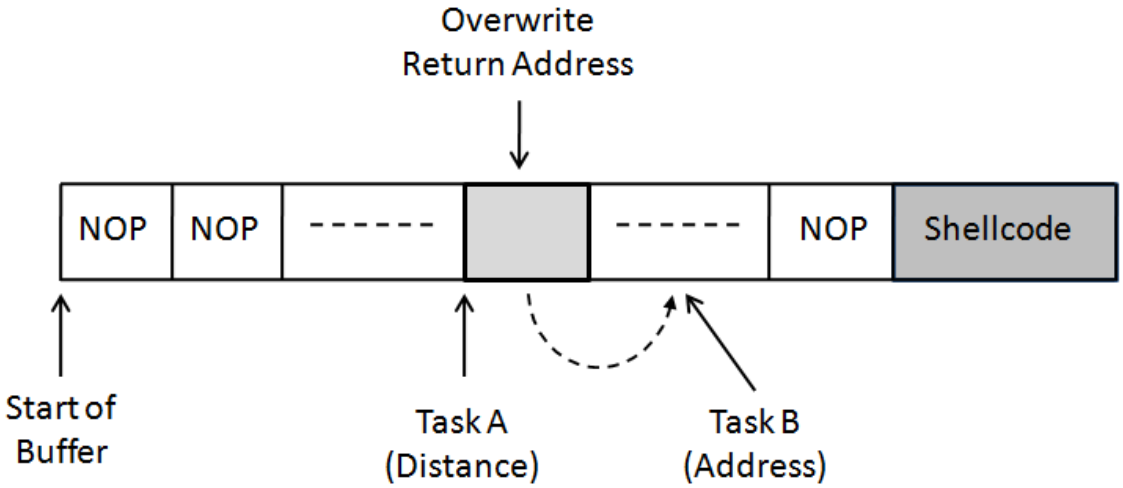
Goal


Arguments
Return Address
Previous Frame Pointer
buffer[23]
buffer[0]
Buffer copy
Malicious Code
Address
Accurate Distance
(badfile)
Malicious Code
(Overwrite)
New Return Address
(Overwrite)
(Overwrite)

Use of NOP's


Malicious Code
(Overwrite)
New Return Address
(Overwrite)
(Overwrite)
Inaccurate Guess – Failed Attack
ebp
(Without NOP)
Malicious Code
NOP
NOP
NOP
New Return Address
(Overwrite)
(Overwrite)
Inaccurate Guess – Successful Attack
ebp
(With NOP)

Task Breakdown - Prepare “badfile”



Environment Setup for Tasks

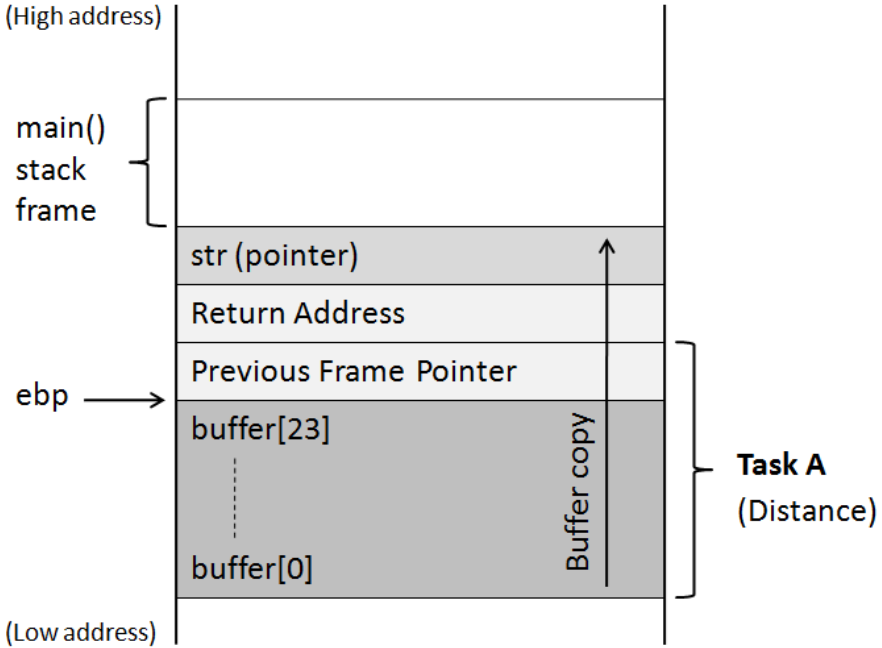
1. **Turn off address randomization (countermeasure)**

```
% sudo sysctl -w kernel.randomize_va_space=0
```

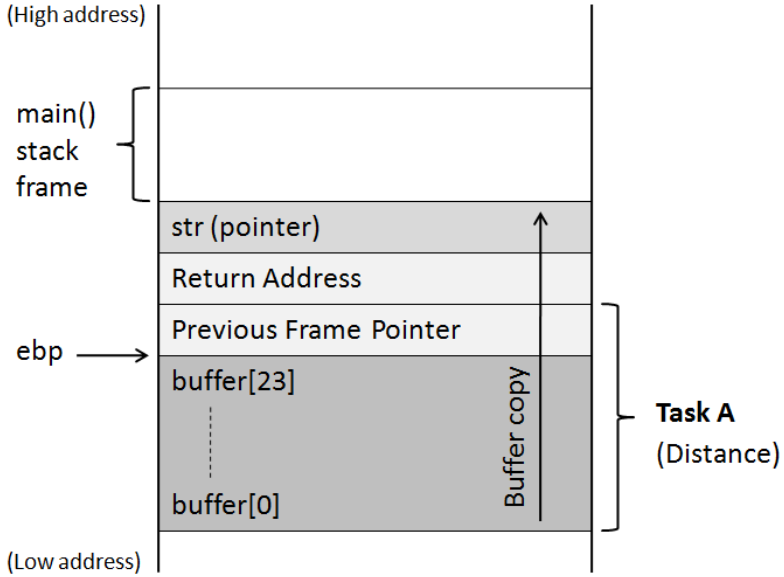
1. **Compile set-uid root version of stack.c**

```
% gcc -o stack -z execstack -fno-stack-protector stack.c
% sudo chown root stack
% sudo chmod 4755 stack
```

Goal - Task A

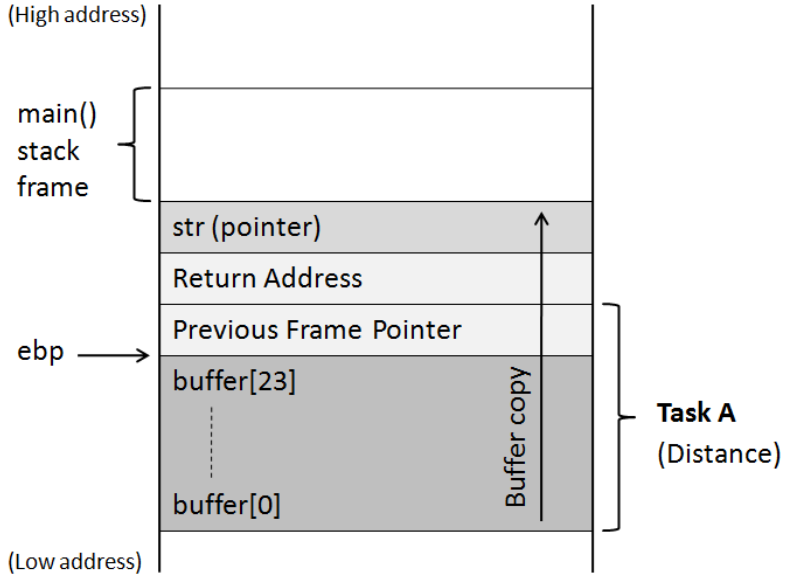

(High address)
main()
stack
frame
str (pointer)
Return Address
Previous Frame Pointer
ebp
buffer[23]
buffer[0]
Buffer copy
Task A
(Distance)
(Low address)

Goal - Task A


(High address)
main()
stack
frame
str (pointer)
Return Address
Previous Frame Pointer
ebp
buffer[23]
buffer[0]
Buffer copy
Task A
(Distance)
(Low address)

Goal - Task A

1. **Need for debugging**
   a. Buffer size may exceed 24 bytes at run time
   b. Need accurate buffer size

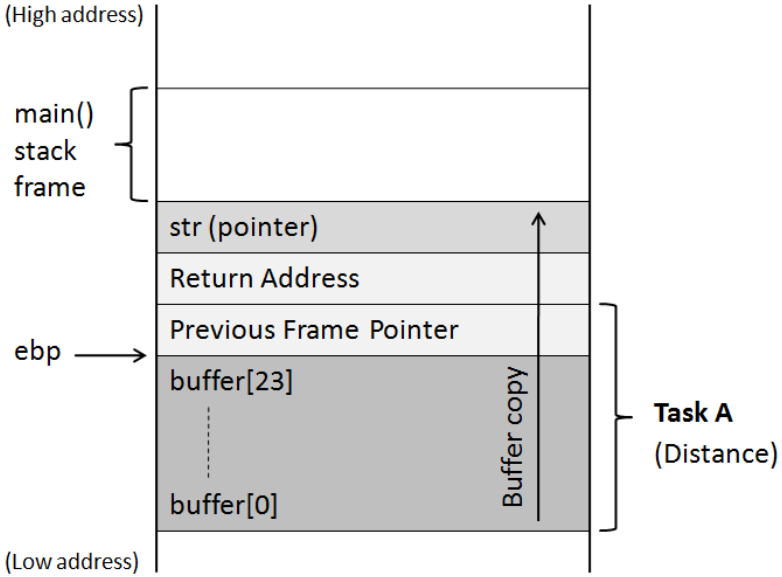


Goal - Task A

1. **Need for debugging**
   a. Buffer size may exceed 24 bytes at run time
   b. Need accurate buffer size

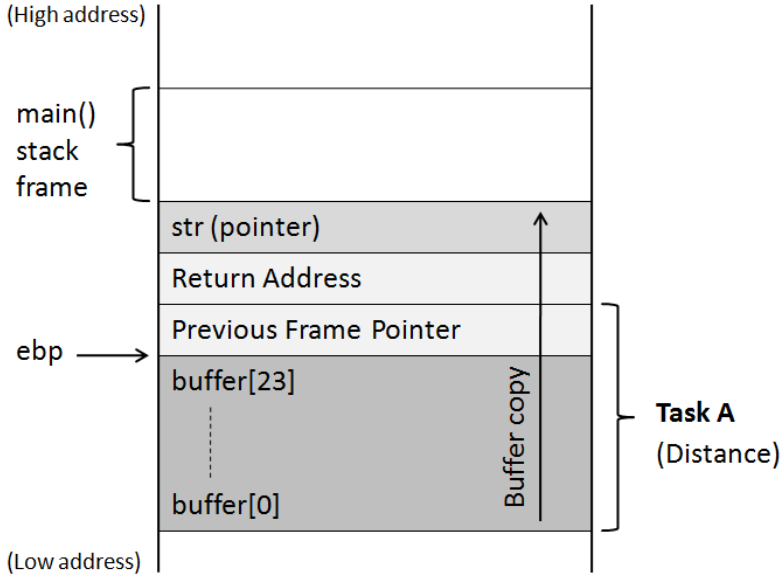
1. **Compile debug version of stack.c**

```
% gcc -z execstack -fno-stack-protector
       -g -o stack_dbg stack.c
```

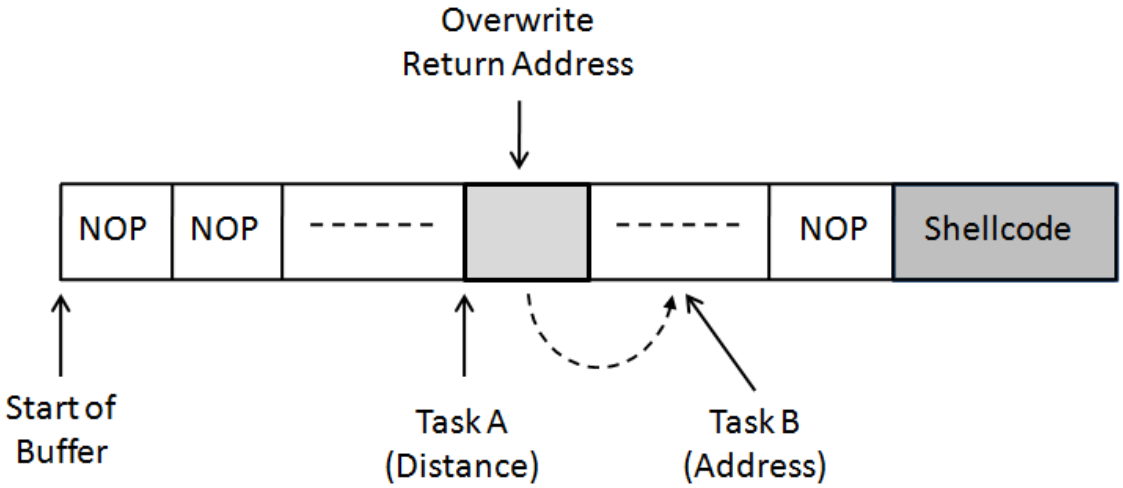


Task A

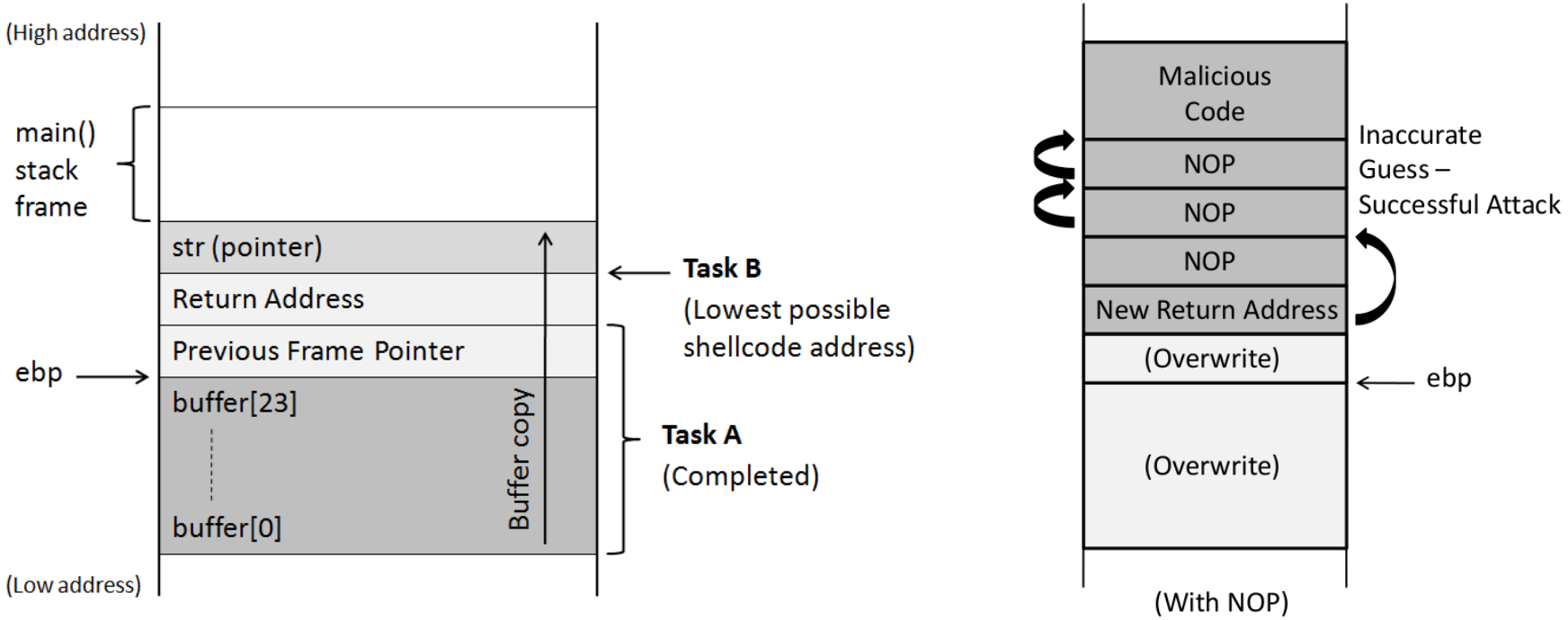
1. **Start debugging using gdb**
2. **Set breakpoint**
3. **Print buffer address**
4. **Print frame pointer address**
5. **Calculate distance**



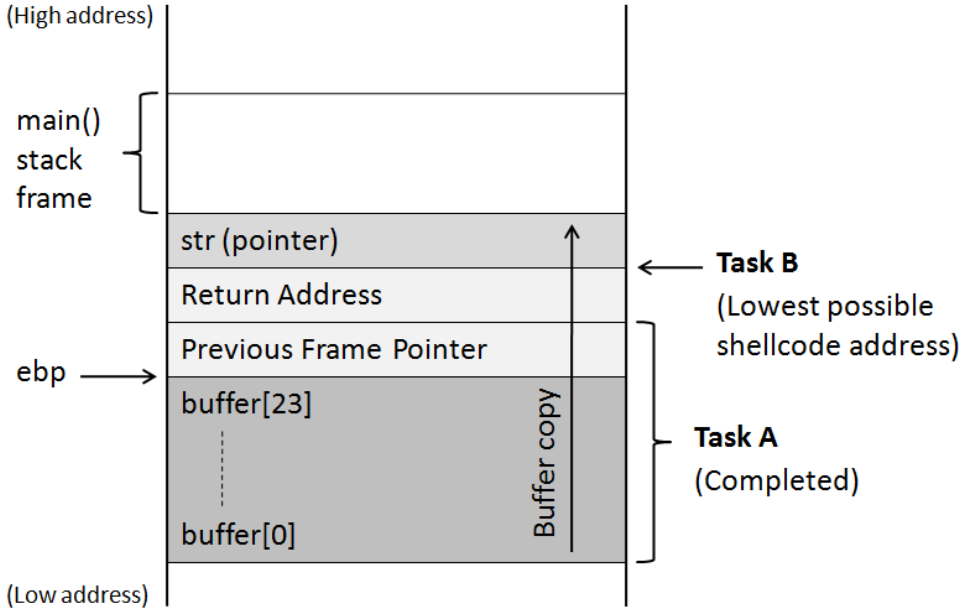
Task Breakdown - Prepare “badfile”



Goal - Task B

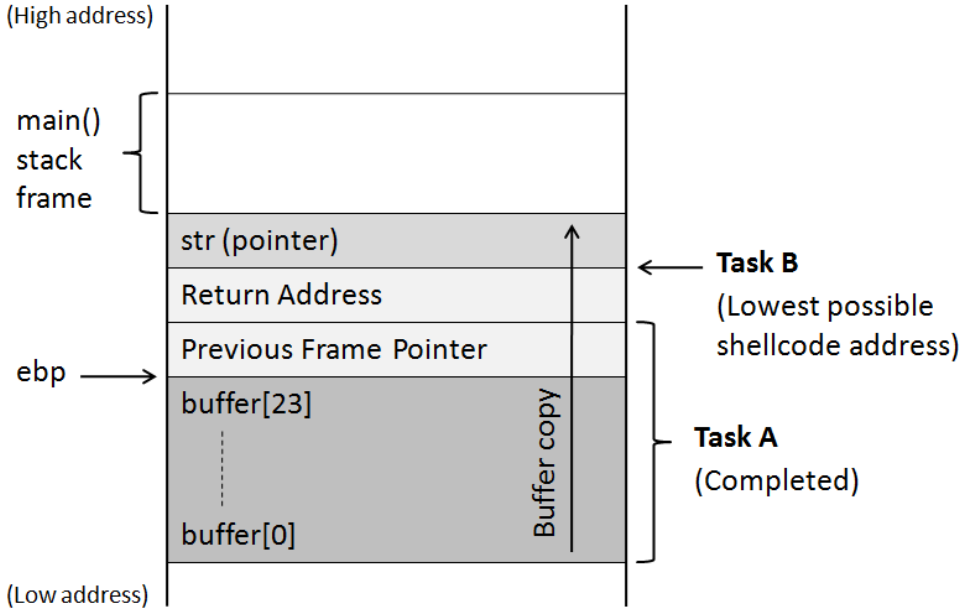

(High address)
main()
stack
frame
str (pointer)
Return Address
Previous Frame Pointer
ebp
buffer[23]
buffer[0]
Buffer copy
(Low address)
Task B
(Lowest possible
shellcode address)
Task A
(Completed)
Malicious
Code
NOP
NOP
NOP
New Return Address
(Overwrite)
(Overwrite)
ebp
Inaccurate
Guess –
Successful Attack
(With NOP)

Task B

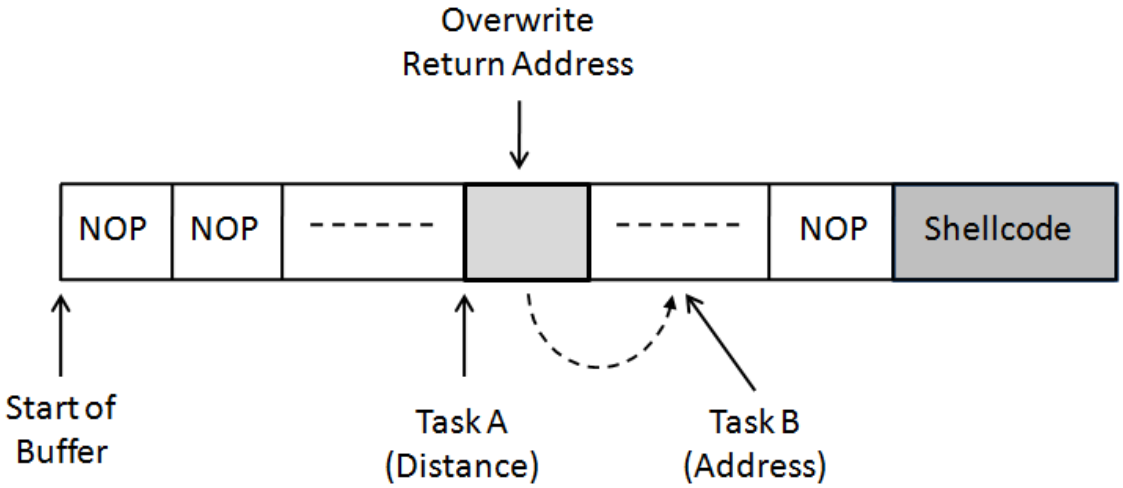

(High address)
main()
stack
frame
str (pointer)
Return Address
Previous Frame Pointer
ebp
buffer[23]
buffer[0]
Buffer copy
(Low address)
Task B
(Lowest possible
shellcode address)
Task A
(Completed)

Task B

1. **Calculate lowest address for shellcod**
2. **Add offset**



Task Breakdown - Prepare “badfile”



Construct the badfile - exploit.c

```
void main(int argc, char **argv)
{
    // Initialize buffer with 0x90 (NOP instruction)
    memset(&buffer, 0x90, 517);

    // From tasks A and B
    *((long *) (buffer + <distance - task A>)) = <address - task B>;

    // Place the shellcode towards the end of buffer
    memcpy(buffer + sizeof(buffer) - sizeof(shellcode), shellcode, sizeof(shellcode));

}
```

Run the exploit

- **Compile and run exploit.c to generate badfile**
- **Run set-uid root compiled stack.c**

Countermeasures

- **ASLR**
- **StackGuard**
- **Non-Executable (NX) Stack**

- **For each of these, refer to lab description or research.**