Binary Reverse Engineering And Analysis

Course 5: Stack frames 101

Caragea Radu

March 16, 2021

Recap: stack micro-operations (demo 1)

```
POP RAX                    ; rax = *(int64_t*)rsp; rsp += 8
PUSH RAX                   ; rsp -= 8; *(int64_t*)rsp = rax;

CALL 0x12345               ; PUSH RIP; JMP 0x12345

RET                        ; POP RIP
```

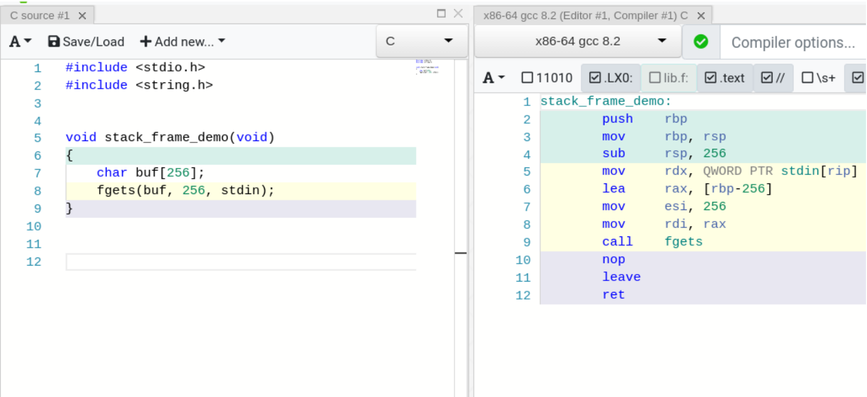
Recap: stack macro-operations

```
PUSH RBP                        ; save previous frame base
MOV RBP, RSP                    ; move frame base to current top
SUB RSP, 100                    ; allocate 100 bytes on the stack
                                ; "push new stack frame"

MOV RBX, [RBP - 0x20]           ; rbx = *(int64_t*)(rbp-0x20)
                                ; use the allocated space for storage

LEAVE                           ; MOV RSP, RBP ; POP RBP
                                ; "pop current stack frame"
```

Today

- Better understanding of stack variable allocation
- Better understanding of function calls
- Common vulnerabilities
- Ways to exploit
- Next time: ways to prevent

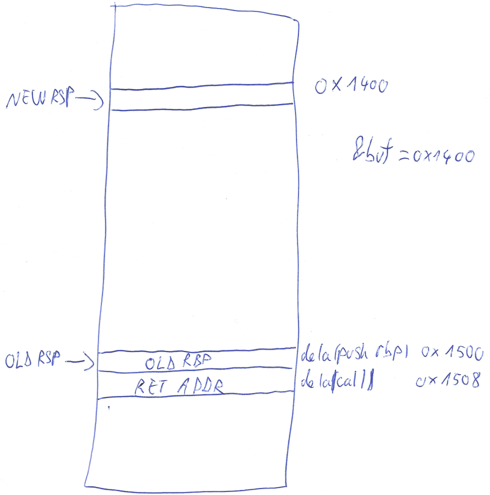
Stack visualization: 1 buffer



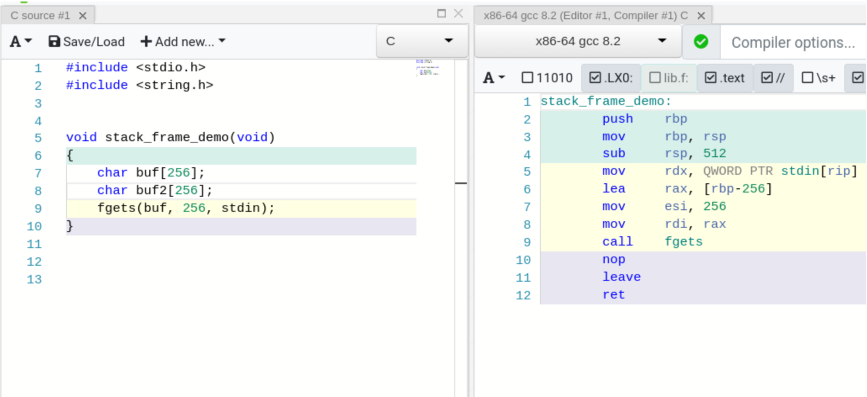
```c
#include <stdio.h>
#include <string.h>


void stack_frame_demo(void)
{
    char buf[256];
    fgets(buf, 256, stdin);
}


```

```asm
stack_frame_demo:
        push    rbp
        mov     rbp, rsp
        sub     rsp, 256
        mov     rdx, QWORD PTR stdin[rip]
        lea     rax, [rbp-256]
        mov     esi, 256
        mov     rdi, rax
        call    fgets
        nop
        leave
        ret
```

Stack visualization: 1 buffer



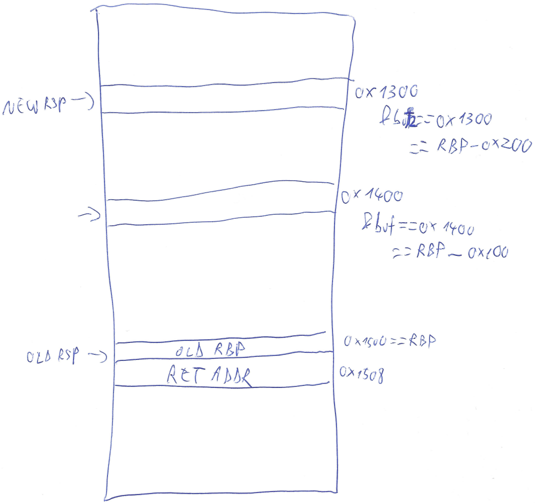
Stack visualization: 2 buffers (1/2)



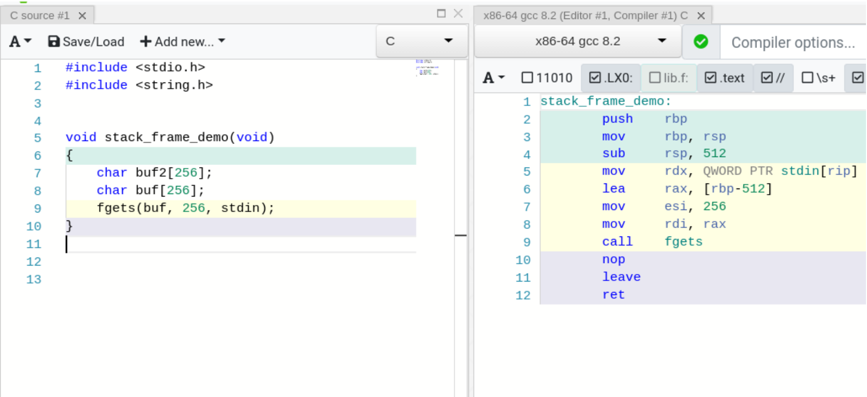
```c
#include <stdio.h>
#include <string.h>


void stack_frame_demo(void)
{
    char buf[256];
    char buf2[256];
    fgets(buf, 256, stdin);
}
```

```
stack_frame_demo:
        push    rbp
        mov     rbp, rsp
        sub     rsp, 512
        mov     rdx, QWORD PTR stdin[rip]
        lea     rax, [rbp-256]
        mov     esi, 256
        mov     rdi, rax
        call    fgets
        nop
        leave
        ret
```

Stack visualization: 2 buffers (1/2)



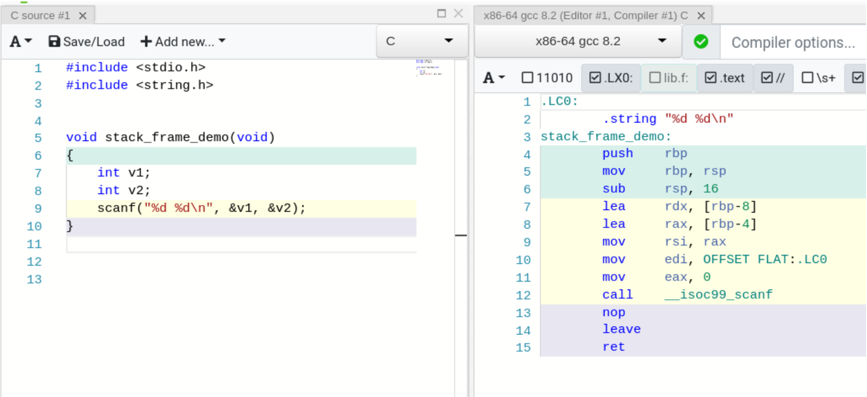
Stack visualization: 2 buffers (2/2)


C source #1 ×
A ▾ Save/Load + Add new... ▾
C ▾
x86-64 gcc 8.2 (Editor #1, Compiler #1) C ×
x86-64 gcc 8.2 ▾
Compiler options...
A ▾ □ 11010 ☑ .LX0: □ lib.f: ☑ .text ☑ // □ \s+ ☑

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


void stack_frame_demo(void)
{
    char buf2[256];
    char buf[256];
    fgets(buf, 256, stdin);
}
```

```
stack_frame_demo:
        push    rbp
        mov     rbp, rsp
        sub     rsp, 512
        mov     rdx, QWORD PTR stdin[rip]
        lea     rax, [rbp-512]
        mov     esi, 256
        mov     rdi, rax
        call    fgets
        nop
        leave
        ret
```

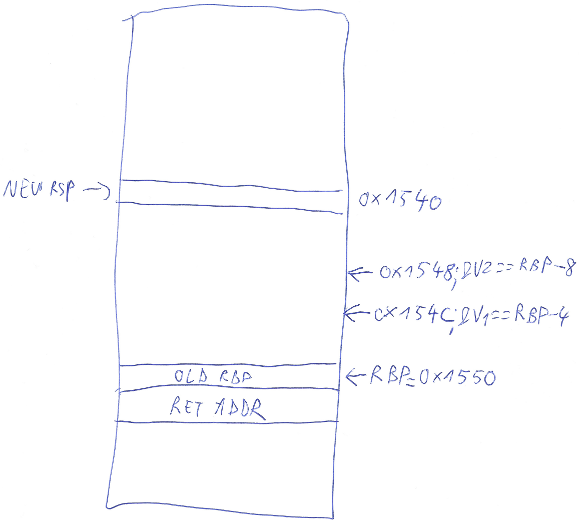
Stack visualization: 2 ints (1/2)



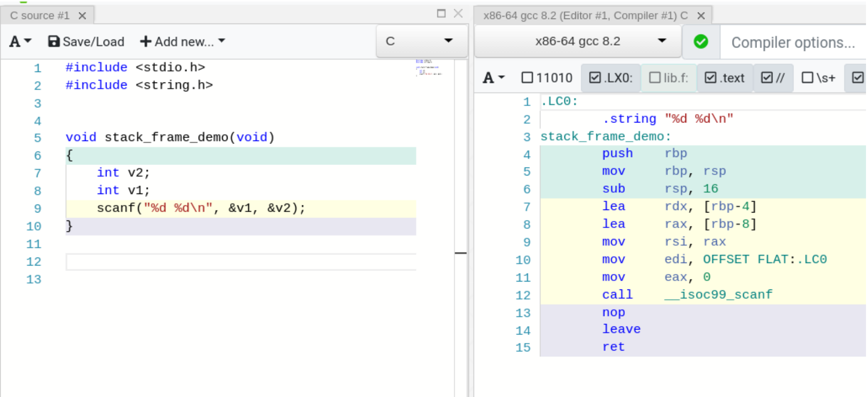
```c
#include <stdio.h>
#include <string.h>


void stack_frame_demo(void)
{
    int v1;
    int v2;
    scanf("%d %d\n", &v1, &v2);
}
```

```
.LC0:
        .string "%d %d\n"
stack_frame_demo:
        push    rbp
        mov     rbp, rsp
        sub     rsp, 16
        lea     rdx, [rbp-8]
        lea     rax, [rbp-4]
        mov     rsi, rax
        mov     edi, OFFSET FLAT:.LC0
        mov     eax, 0
        call    __isoc99_scanf
        nop
        leave
        ret
```

Stack visualization: 2 ints (1/2)



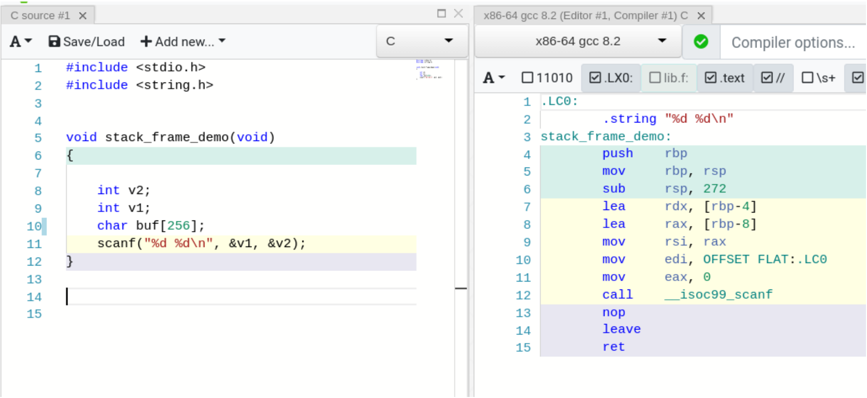
Stack visualization: 2 ints (2/2)



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


void stack_frame_demo(void)
{
    int v2;
    int v1;
    scanf("%d %d\n", &v1, &v2);
}


```

```asm
.LC0:
        .string "%d %d\n"
stack_frame_demo:
        push    rbp
        mov     rbp, rsp
        sub     rsp, 16
        lea     rdx, [rbp-4]
        lea     rax, [rbp-8]
        mov     rsi, rax
        mov     edi, OFFSET FLAT:.LC0
        mov     eax, 0
        call    __isoc99_scanf
        nop
        leave
        ret
```

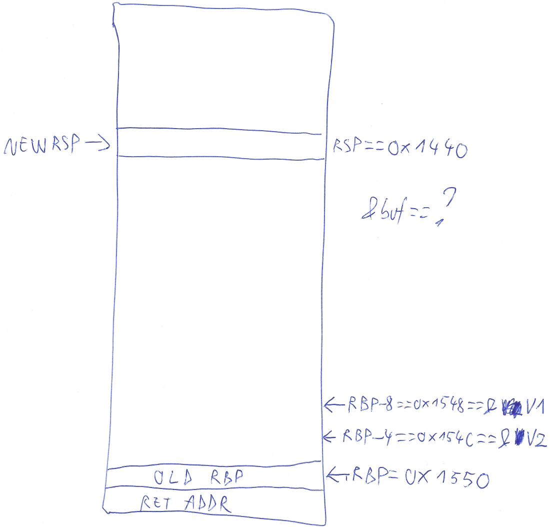
Stack visualization: combination (1/2)



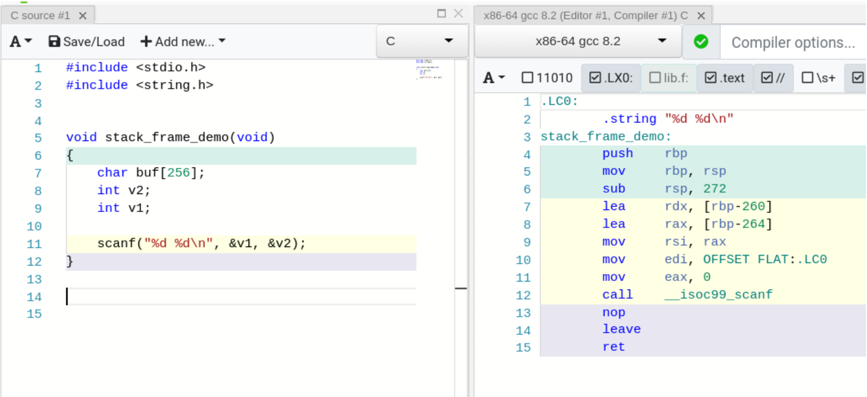
```c
#include <stdio.h>
#include <string.h>


void stack_frame_demo(void)
{

    int v2;
    int v1;
    char buf[256];
    scanf("%d %d\n", &v1, &v2);
}

```

```
.LC0:
        .string "%d %d\n"
stack_frame_demo:
        push    rbp
        mov     rbp, rsp
        sub     rsp, 272
        lea     rdx, [rbp-4]
        lea     rax, [rbp-8]
        mov     rsi, rax
        mov     edi, OFFSET FLAT:.LC0
        mov     eax, 0
        call    __isoc99_scanf
        nop
        leave
        ret
```

Stack visualization: combination (1/2)



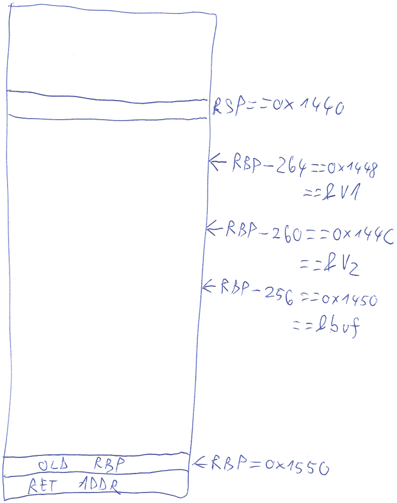
Stack visualization: combination (2/2)



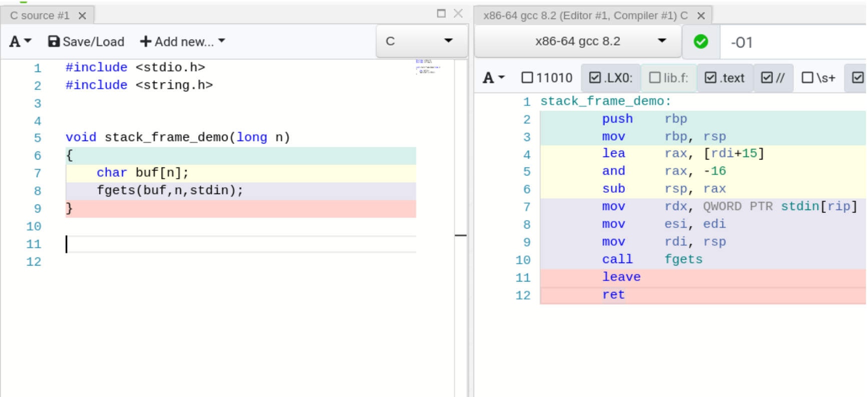
```c
#include <stdio.h>
#include <string.h>


void stack_frame_demo(void)
{
    char buf[256];
    int v2;
    int v1;

    scanf("%d %d\n", &v1, &v2);
}


```

```
.LC0:
        .string "%d %d\n"
stack_frame_demo:
        push    rbp
        mov     rbp, rsp
        sub     rsp, 272
        lea     rdx, [rbp-260]
        lea     rax, [rbp-264]
        mov     rsi, rax
        mov     edi, OFFSET FLAT:.LC0
        mov     eax, 0
        call    __isoc99_scanf
        nop
        leave
        ret
```

Stack visualization: combination (2/2)



Stack visualization: variable length buffers



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


void stack_frame_demo(long n)
{
    char buf[n];
    fgets(buf,n,stdin);
}


```

```
stack_frame_demo:
        push    rbp
        mov     rbp, rsp
        lea     rax, [rdi+15]
        and     rax, -16
        sub     rsp, rax
        mov     rdx, QWORD PTR stdin[rip]
        mov     esi, edi
        mov     rdi, rsp
        call    fgets
        leave
        ret
```

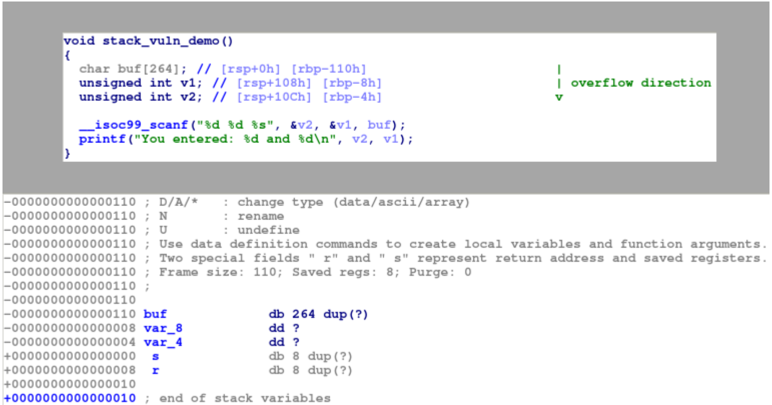
Vulnerability 1: locality (demo 1)

- Since all variables are ”packed”, mishaps can happen

Vulnerability 1: locality (demo 1)

- Since all variables are ”packed”, mishaps can happen
- Buffers read improperly can overflow (spill) into adjacent variables
- In extreme cases, the overflow can hijack the execution
- Let’s see a DEMO!

Demo 1 key takeaway



```
void stack_vuln_demo()
{
  char buf[264]; // [rsp+0h] [rbp-110h]                         |
  unsigned int v1; // [rsp+108h] [rbp-8h]                       | overflow direction
  unsigned int v2; // [rsp+10Ch] [rbp-4h]                       v

  __isoc99_scanf("%d %d %s", &v2, &v1, buf);
  printf("You entered: %d and %d\n", v2, v1);
}
```

```
-0000000000000110 ; D/A/*    : change type (data/ascii/array)
-0000000000000110 ; N        : rename
-0000000000000110 ; U        : undefine
-0000000000000110 ; Use data definition commands to create local variables and function arguments.
-0000000000000110 ; Two special fields " r" and " s" represent return address and saved registers.
-0000000000000110 ; Frame size: 110; Saved regs: 8; Purge: 0
-0000000000000110 ;
-0000000000000110
-0000000000000110 buf             db 264 dup(?)
-0000000000000008 var_8           dd ?
-0000000000000004 var_4           dd ?
+0000000000000000  s              db 8 dup(?)
+0000000000000008  r              db 8 dup(?)
+0000000000000010
+0000000000000010 ; end of stack variables
```

Function call recap (demo 2)

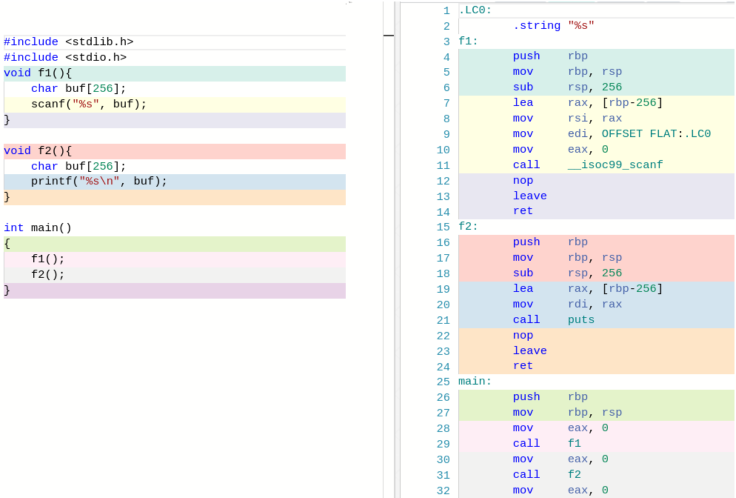
- We now know a bit about debuggers
- Let’s see a function call DEMO

Function return hijack (demo 3)

- Functions (usually) return to the call site
- The call site (return address) is stored on the stack
- When other variables cannot be overflown: ret addr
- Let's see another DEMO

Vulnerability 2: Reuse (demo 4)

- https://godbolt.org/z/92rh_U



```c
#include <stdlib.h>
#include <stdio.h>
void f1(){
    char buf[256];
    scanf("%s", buf);
}

void f2(){
    char buf[256];
    printf("%s\n", buf);
}

int main()
{
    f1();
    f2();
}
```

```asm
.LC0:
        .string "%s"
f1:
        push    rbp
        mov     rbp, rsp
        sub     rsp, 256
        lea     rax, [rbp-256]
        mov     rsi, rax
        mov     edi, OFFSET FLAT:.LC0
        mov     eax, 0
        call    __isoc99_scanf
        nop
        leave
        ret
f2:
        push    rbp
        mov     rbp, rsp
        sub     rsp, 256
        lea     rax, [rbp-256]
        mov     rdi, rax
        call    puts
        nop
        leave
        ret
main:
        push    rbp
        mov     rbp, rsp
        mov     eax, 0
        call    f1
        mov     eax, 0
        call    f2
        mov     eax, 0
```

Practice

- Any Questions?
- `http://pwnthybytes.ro/unibuc_re/05-lab.html`