University of BRISTOL

# Computer System B -Security

## Introduction to Software Vulnerabilities Part 2 – Buffer Overflow

Sanjay Rawat

# Recap…

```
int func(arg1, arg2)
{
int var1;
int var2;
char arr[20];
int var3;
…
…
return 0;}

int main()
{
int x;
X= func(4, 10);
…
…
}
```

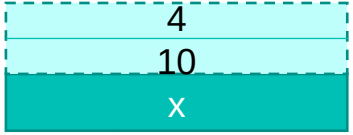
# Recap…

```
int func(arg1, arg2)
{
int var1;
int var2;
char arr[20];
int var3;
…
…
return 0;}

int main()
{
int x;
X= func(4, 10);
…
…
}
```


main’s stack

# Recap…

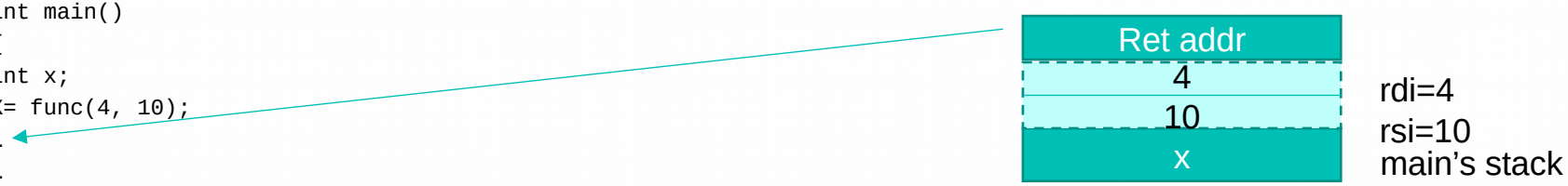
```
int func(arg1, arg2)
{
int var1;
int var2;
char arr[20];
int var3;
…
…
return 0;}

int main()
{
int x;
X= func(4, 10);
…
…
}
```




rdi=4
rsi=10
main’s stack

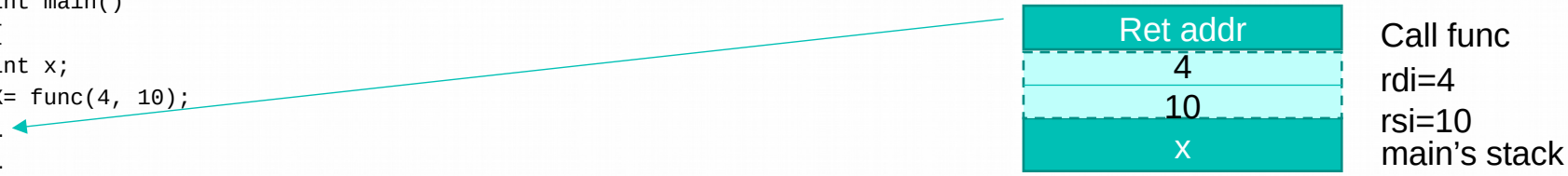
# Recap…

```
int func(arg1, arg2)
{
int var1;
int var2;
char arr[20];
int var3;
…
…
return 0;}

int main()
{
int x;
X= func(4, 10);
…
…
}
```



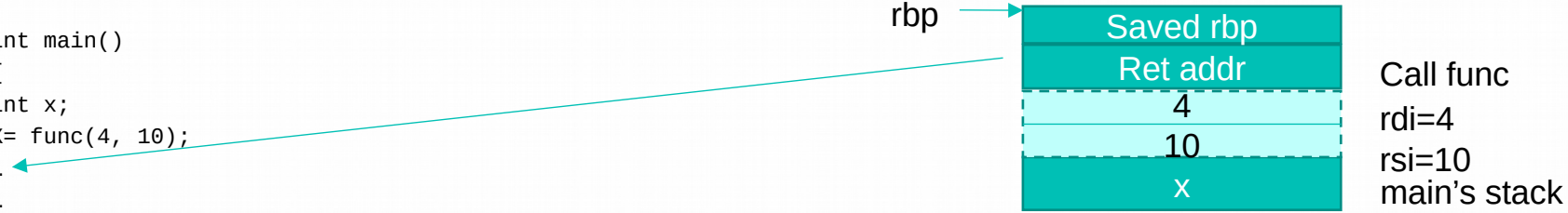
# Recap…

```
int func(arg1, arg2)
{
int var1;
int var2;
char arr[20];
int var3;
…
…
return 0;}

int main()
{
int x;
X= func(4, 10);
…
…
}
```



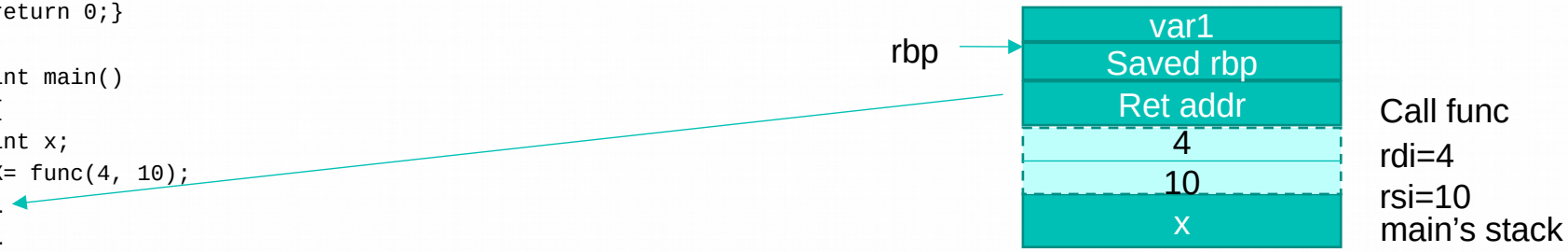
# Recap…

```
int func(arg1, arg2)
{
int var1;
int var2;
char arr[20];
int var3;
…
…
return 0;}

int main()
{
int x;
X= func(4, 10);
…
…
}
```



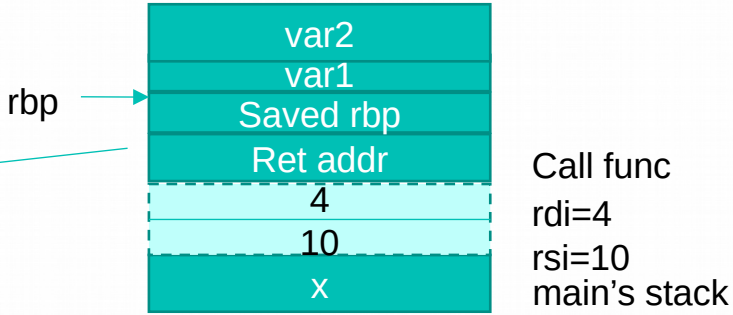
# Recap…

```
int func(arg1, arg2)
{
int var1;
int var2;
char arr[20];
int var3;
…
…
return 0;}

int main()
{
int x;
X= func(4, 10);
…
…
}
```



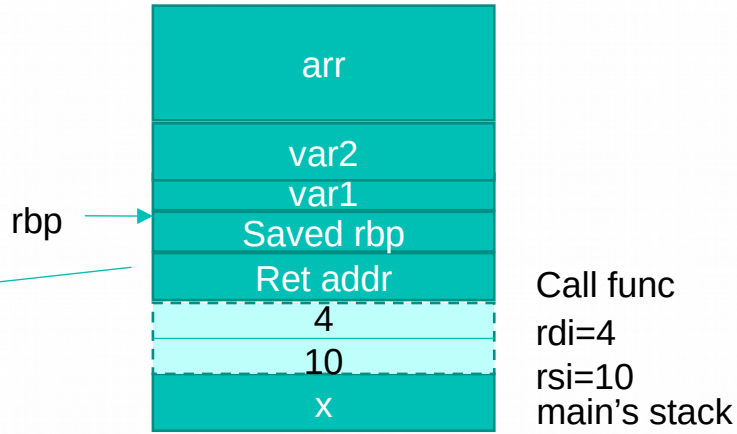
# Recap…

```
int func(arg1, arg2)
{
int var1;
int var2;
char arr[20];
int var3;
…
…
return 0;}

int main()
{
int x;
X= func(4, 10);
…
…
}
```



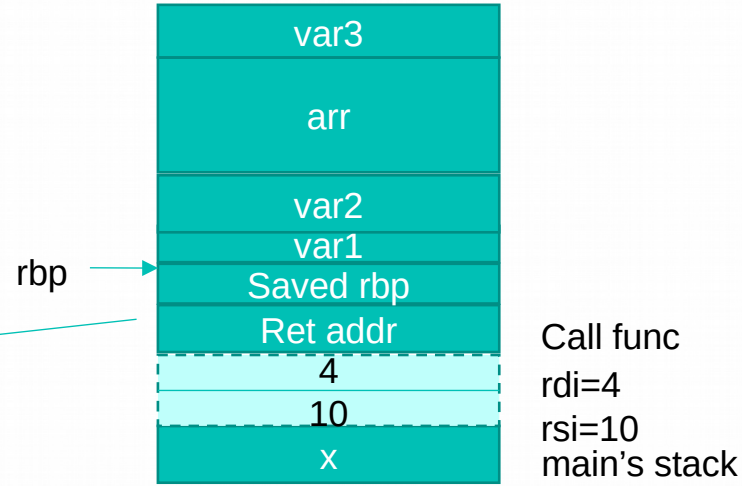
# Recap…

```
int func(arg1, arg2)
{
int var1;
int var2;
char arr[20];
int var3;
…
…
return 0;}

int main()
{
int x;
X= func(4, 10);
…
…
}
```



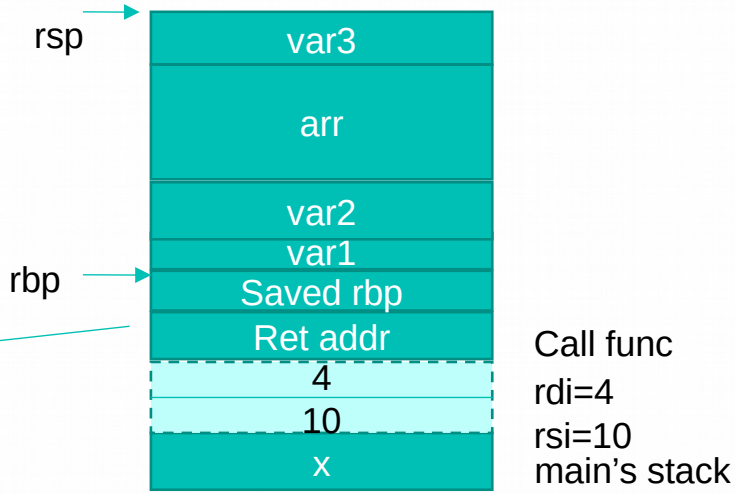
# Recap…

```
int func(arg1, arg2)
{
int var1;
int var2;
char arr[20];
int var3;
…
…
return 0;}

int main()
{
int x;
X= func(4, 10);
…
…
}
```



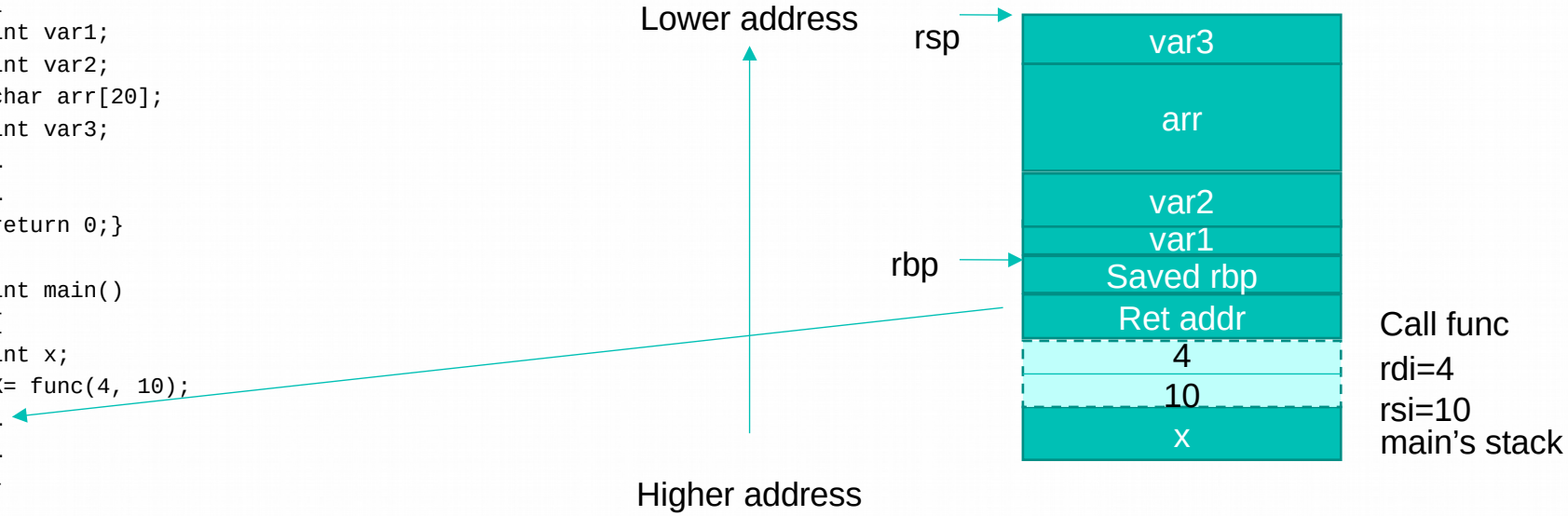
# Recap…

```
int func(arg1, arg2)
{
int var1;
int var2;
char arr[20];
int var3;
…
…
return 0;}

int main()
{
int x;
X= func(4, 10);
…
…
}
```



# Recap…

```
int func(arg1, arg2)
{
int var1;
int var2;
char arr[20];
int var3;
…
…
return 0;}

int main()
{
int x;
X= func(4, 10);
…
…
}
```



# Lecture Agenda

- Buffer Overflow, in general
  - Stack Overflow
  - Heap Overflow
- Code patterns leading to such bugs.
- Book Ref.
  - Our text book (intro Comp Sec.): Chapter 4- section 4.1, 4.2, 4.3
  - SANS report: https://www.sans.org/reading-room/whitepapers/threats/buffer-overflows-dummies-481

# Buffer Overflow

- Several decades old problem (still appears in SANS TOP 25 Software errors!!)
- Main cause: putting more data than *intended*!!
- Consequences: memory corruption (can be very dangerous!)

# Buffer Overflow

- Several decades old problem (still appears in SANS TOP 25 Software errors!!)
- Main cause: putting more data than *intended*!!
- Consequences: memory corruption (can be very dangerous!)

# Stack based BoF

- Cause
  - Stack grows downward
  - Local buffers are allocated onto the stack
  - With no memory protection, these variables can overflow!
- Effect- security vulnerability
  - At CALL, return address is saved on the stack
  - Return address is POPed into the RIP
  - RIP can point to anywhere in the memory!

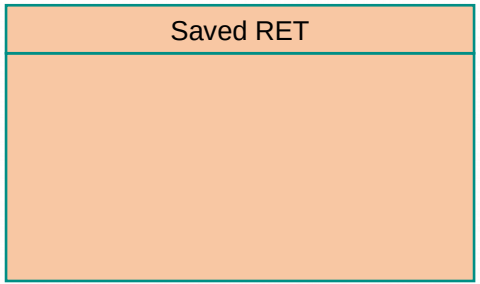
# Illustration- How it happens?

```
buf(char *string) {
    char buffer[20];
     strcpy(buffer,
     string);
       return 1;
        }
```

main

# Illustration- How it happens?

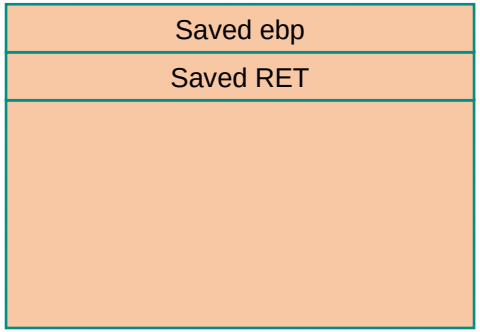
```
buf(char *string) {
    char buffer[20];
     strcpy(buffer,
     string);
       return 1;
        }
```




main

# Illustration- How it happens?

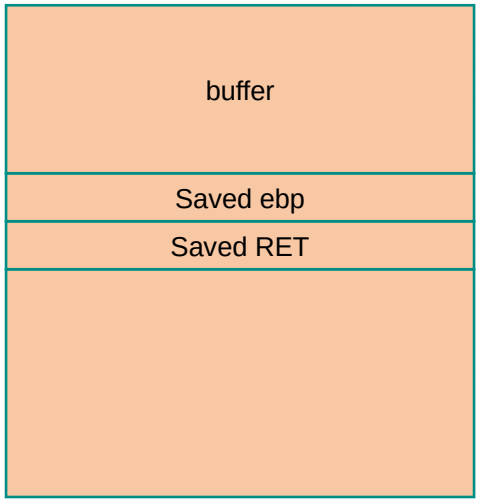
```
buf(char *string) {
    char buffer[20];
     strcpy(buffer,
     string);
       return 1;
        }
```




main

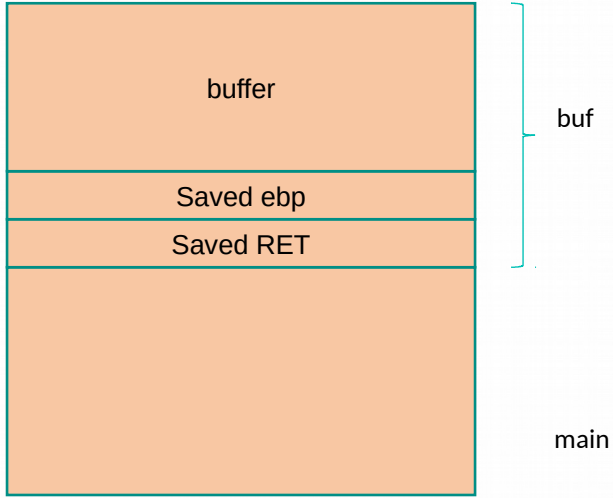
# Illustration- How it happens?

```
buf(char *string) {
    char buffer[20];
     strcpy(buffer,
     string);
       return 1;
        }
```



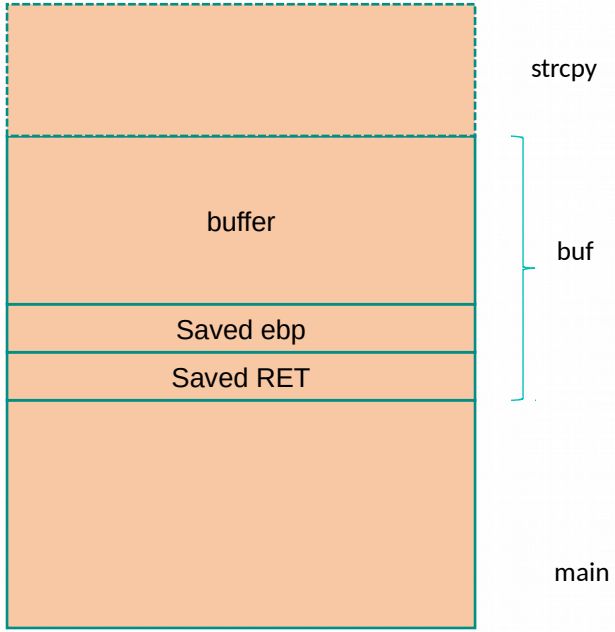
# Illustration- How it happens?

```
buf(char *string) {
    char buffer[20];
     strcpy(buffer,
     string);
       return 1;
        }
```



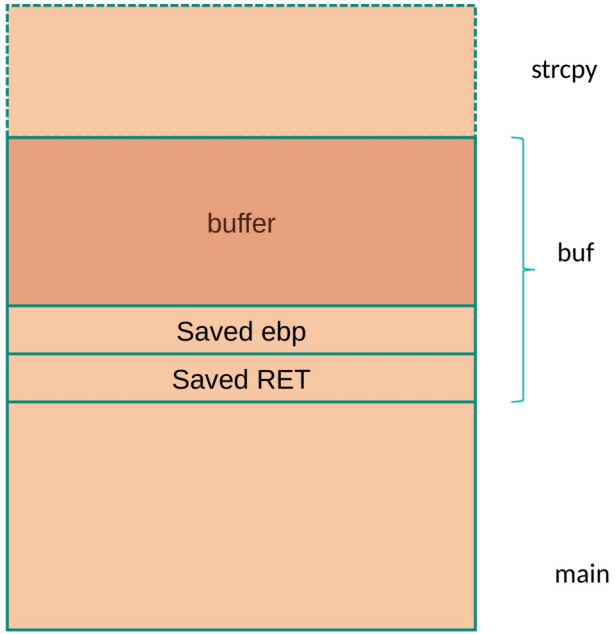
# Illustration- How it happens?

```
buf(char *string) {
    char buffer[20];
     strcpy(buffer,
     string);
       return 1;
        }
```



# Illustration- How it happens?

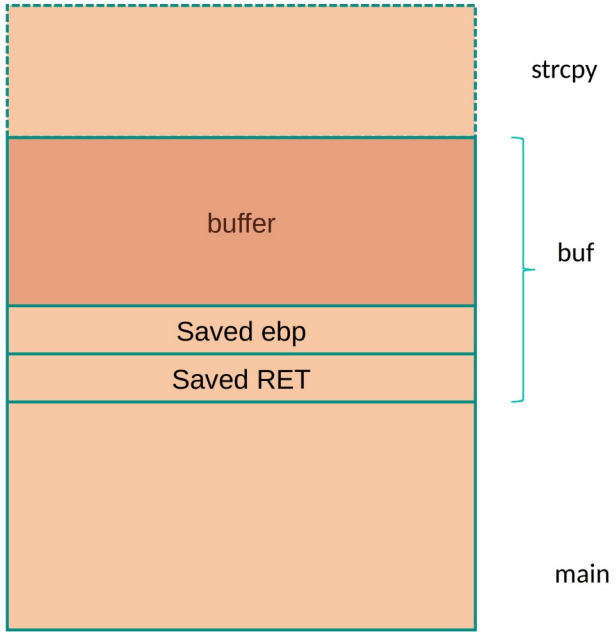
```
buf(char *string) {
    char buffer[20];
     strcpy(buffer,
     string);
       return 1;
        }
```



# Illustration- How it happens?

```
buf(char *string) {
    char buffer[20];
     strcpy(buffer,
     string);
       return 1;
        }
```

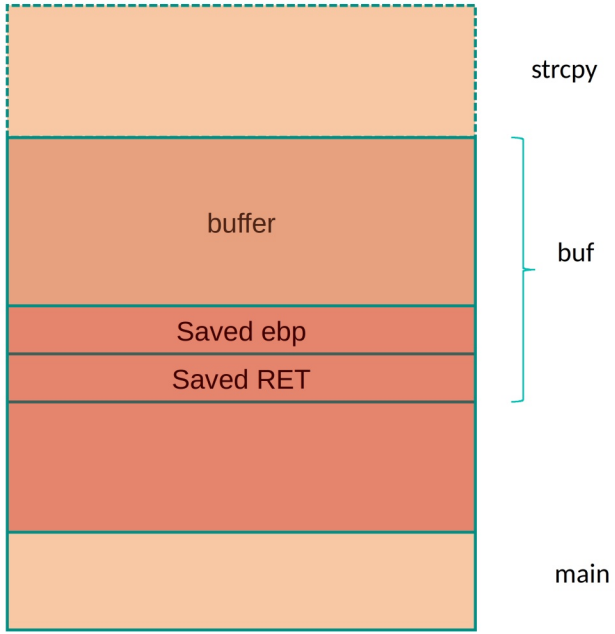
String > buffer??



# Illustration- How it happens?

```
buf(char *string) {
    char buffer[20];
     strcpy(buffer,
     string);
       return 1;
        }
```

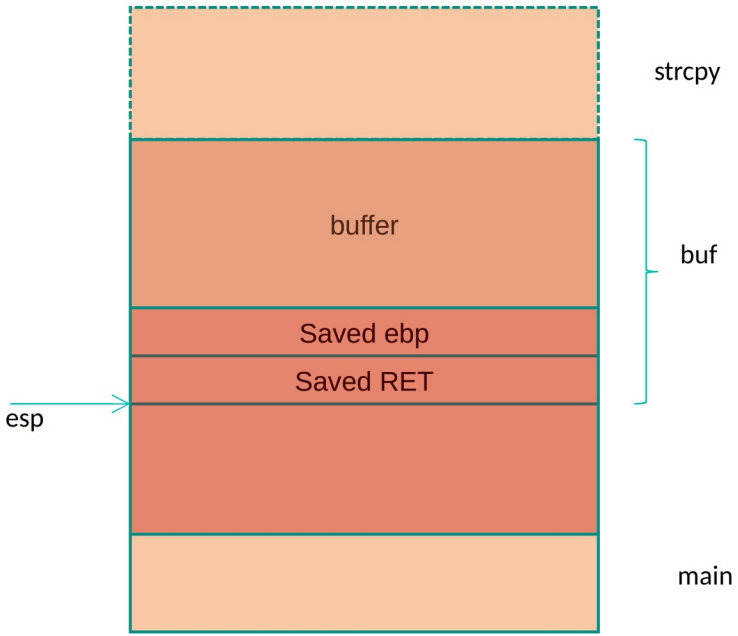
String > buffer??



# Illustration- How it happens?

```
buf(char *string) {
    char buffer[20];
     strcpy(buffer,
     string);
       return 1;
        }
```

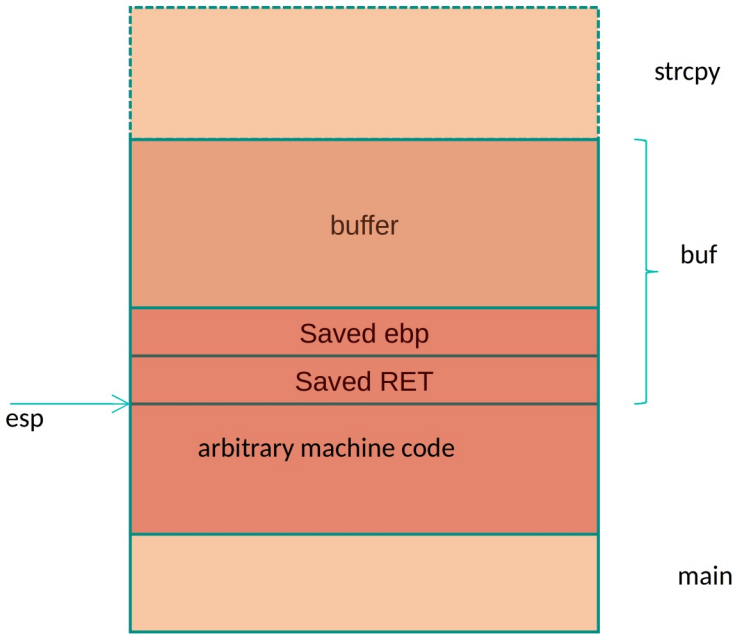
String > buffer??



# Illustration- How it happens?

```
buf(char *string) {
    char buffer[20];
     strcpy(buffer,
     string);
       return 1;
        }
```

String > buffer??

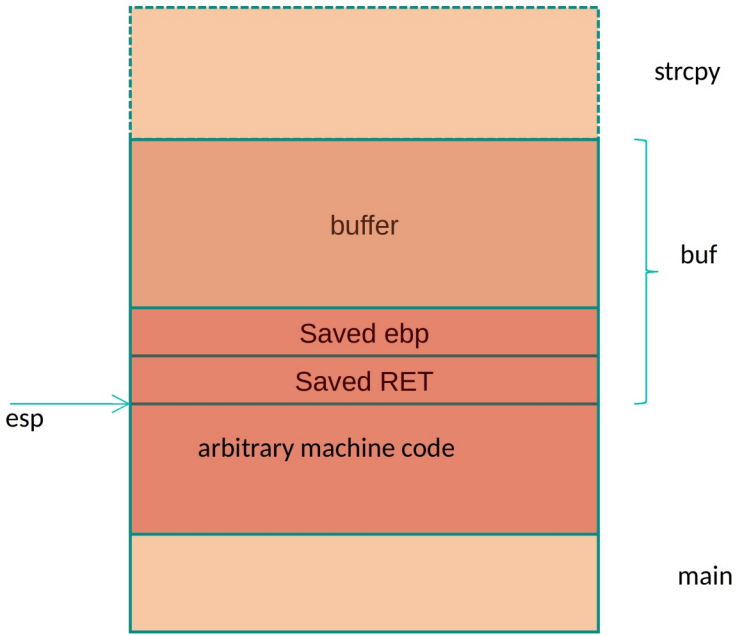


# Illustration- How it happens?

```
buf(char *string) {
    char buffer[20];
     strcpy(buffer,
     string);
       return 1;
        }
```

String > buffer??

Find address to jmp esp (XYZ)

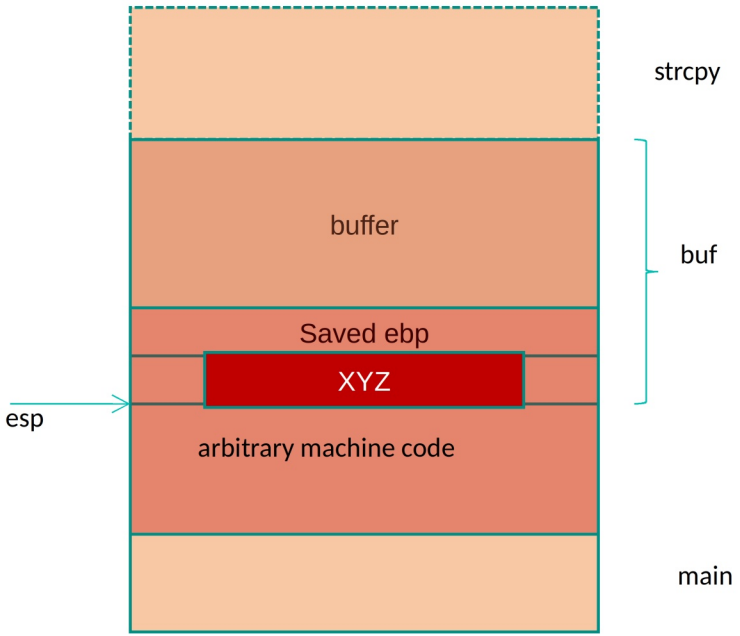


# Illustration- How it happens?

```
buf(char *string) {
    char buffer[20];
     strcpy(buffer,
     string);
       return 1;
        }
```

String > buffer??

Find address to jmp esp (XYZ)

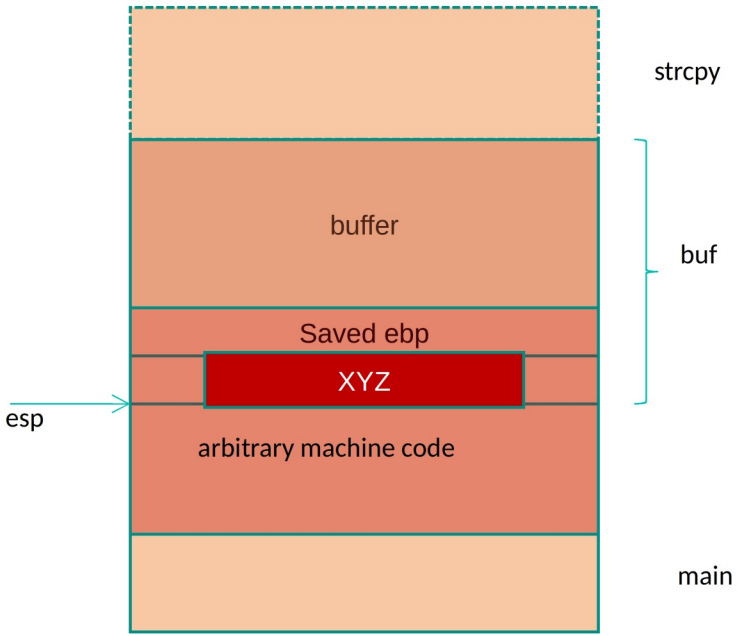


# Illustration- How it happens?

```
buf(char *string) {
    char buffer[20];
     strcpy(buffer,
     string);
       return 1;
        }
```

String > buffer??

Find address to jmp esp (XYZ)




Payload = junk data + return addr overflow + **shellcode**

# Few examples

Ex#1

```
int main ()
{
int x=1;
char array[20];
int y;
for (y=0;y<200;y++)
{
array[y]='A';
}
…
….
}
```

# Ex#3

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int get_cookie()
{
    return rand();
}
int main()
{
     int check=10;
    int guess;
    char name[20];
    guess=get_cookie();
    printf("Enter your name..\n");
    gets(name);
    if(guess == 0x41424344)
        printf("You win %s\n",name);
    else printf("Better luck next time %s :(\
n",name);
    if (check!= 10)
    printf("\n hmmmm....!\n");
    return 0;
}
```

## Ex#2

```
int main(int argc, char *argv[])
{
     int index=100;
     char name[20]
    char welcome[20]="Welcome to the CSB\n";
     printf("1. Index is: %d\n",index);
     strcpy(name, argv[1]);
     printf ("[*] Hi %s\n",name);
     printf("[*] %s\n", welcome);
     printf("2. Index is: %d\n",index);
     return 0;
}
```

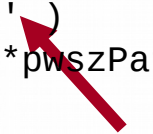
# Spot the defect!

```
HRESULT GetMachineName(WCHAR *pwszPath,
     WCHAR  wszMachineName[N + 1])
{
     LPWSTR pwszServerName = wszMachineName;
     while (*pwszPath != L'\\' )
         *pwszServerName++ = *pwszPath++;
     ...
}
```

**Loop termination (exploited by Blaster worm)**

# Spot the defect!

```
HRESULT GetMachineName(WCHAR *pwszPath,
     WCHAR  wszMachineName[N + 1])
{
     LPWSTR pwszServerName = wszMachineName;
     while (*pwszPath != L'\\' )
         *pwszServerName++ = *pwszPath++;
     ...
}
```



**what if there is no ‘/’ in the URL?**

**Loop termination (exploited by Blaster worm)**

# Another Simple Example

```
void bar(){
char buf[256];
int i;
for(i = 0; i <= 256; i++)
buf[i] = getchar();
/* some other statements */
... ... ... ...
}
```

# Another Simple Example

```
void bar(){
char buf[256];
int i;
for(i = 0; i <= 256; i++)
buf[i] = getchar();
/* some other statements */
... ... ... ...
}
```

# Another Simple Example

```
void bar(){
char buf[256];
int i;
for(i = 0; i <= 256; i++)
buf[i] = getchar();
/* some other statements */
... ... ... ...
}
```

# Another Simple Example

```
void bar(){
char buf[256];
int i;
for(i = 0; i <= 256; i++)
buf[i] = getchar();
/* some other statements */
... ... ... ...
}
```

- Loop iterated 256+1 times
- buf overflows by 1 byte
- It will corrupt saved EBP
- On return, ESP will be different!

# Code level Mitigation

# Code level Mitigation

- We know why buffer overflow happens → out-of-bound access.

# Code level Mitigation

- We know why buffer overflow happens → out-of-bound access.
- Use safe version of C APIs
  - `Gets → fgets`
  - `Strcpy → strncpy`
  - `...`

# Code level Mitigation

- We know why buffer overflow happens → out-of-bound access.
- Use safe version of C APIs
  - `Gets → fgets`
  - `Strcpy → strncpy`
  - `...`
- No assumption on user inputs → sanitization

# Code level Mitigation

- We know why buffer overflow happens → out-of-bound access.
- Use safe version of C APIs
  - `Gets → fgets`
  - `Strcpy → strncpy`
  - `...`
- No assumption on user inputs → sanitization
- A quick read: https://security.web.cern.ch/recommendations/en/codetools/c.shtml