

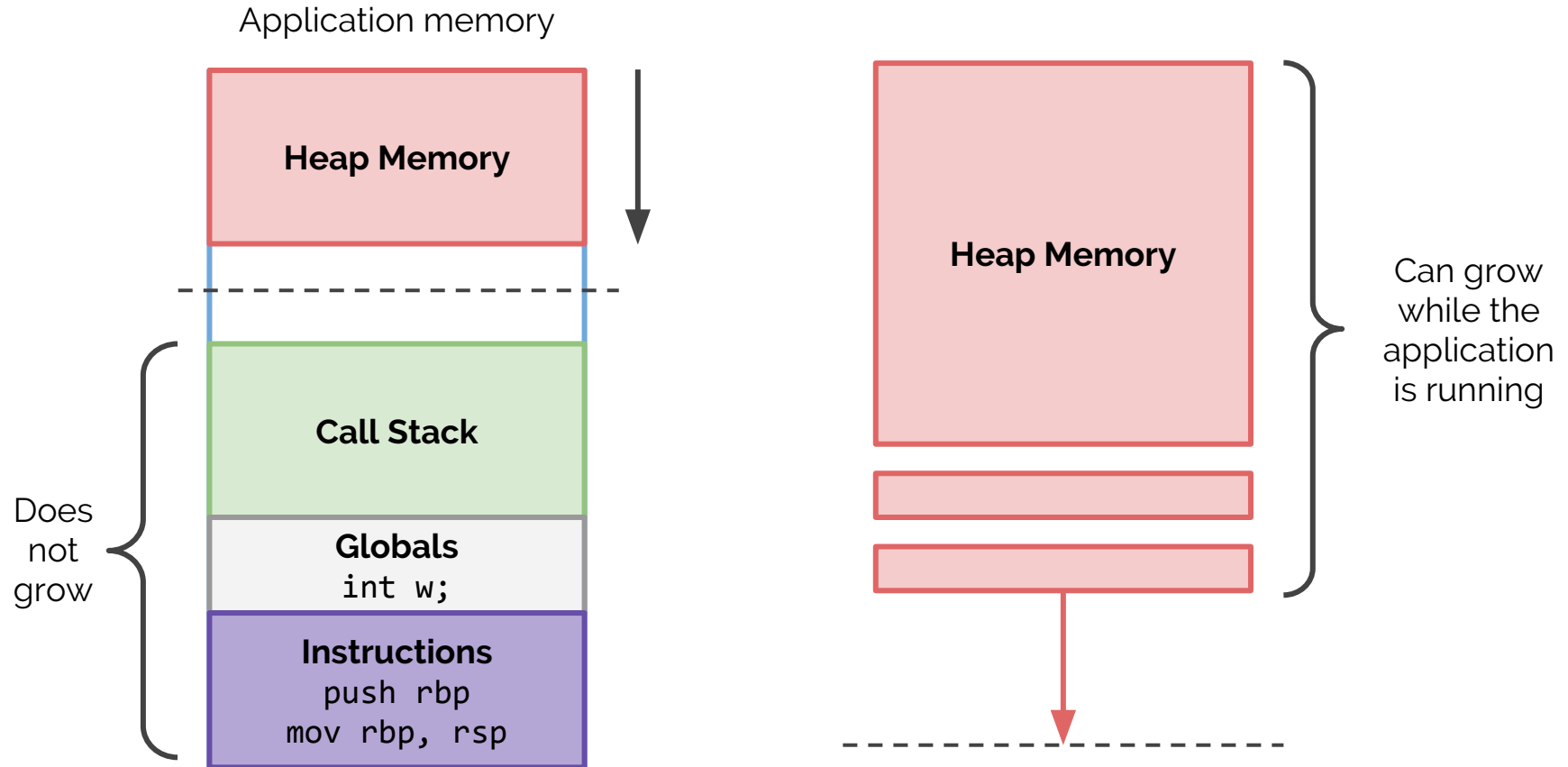
Cyber Security

Software Security: Exploits and Mitigations

Dr Chris Willcocks



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: bbbbbbbbbb
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 **bbbbbbbbb****b**

Buffer Overflows

```
0000000000000072a <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 mov     DWORD PTR [rbp-0x4], 0x0
739: 48 8d 3d d8 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 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 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 lea     rdi, [rip+0xbf]
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

0000000000000072a <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+0xbf]
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

```
0000000000000072a <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 mov     DWORD PTR [rbp-0x4], 0x0
739: 48 8d 3d d8 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 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 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 lea     rdi, [rip+0xbf]
781: e8 5a fe ff ff    call    5e0 <puts@plt>
786: b8 00 00 00 00    mov     eax, 0x0
78b: c9               leave   rbp
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

00000000000072a <main>:

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

```
0000000000000072a <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 mov     DWORD PTR [rbp-0x4],0x0
739: 48 8d 3d d8 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 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 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 lea     rdi,[rip+0xbf] # 840
781: e8 5a fe ff ff    call    5e0 <puts@plt>
786: b8 00 00 00 00    mov     eax,0x0
78b: c9               leave   %rax
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

```
0000000000000072a <main>:
72a: 55                push    rbp
72b: 48 89 e5          mov     rbp,rsb
72e: 48 83 ec 10       sub     rsp,0x10
732: c7 45 fc 00 00 00 mov     DWORD PTR [rbp-0x4],0x0
739: 48 8d 3d d8 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 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 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 lea     rdi,[rip+0xbf] # 840
781: e8 5a fe ff ff    call    5e0 <puts@plt>
786: b8 00 00 00 00    mov     eax,0x0
78b: c9               leave   rbp
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

```
0000000000000072a <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 mov     DWORD PTR [rbp-0x4], 0x0
739: 48 8d 3d d8 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 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 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 lea     rdi, [rip+0xbf] # 840
781: e8 5a fe ff ff    call    5e0 <puts@plt>
786: b8 00 00 00 00    mov     eax, 0x0
78b: c9               leave   rbp
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];

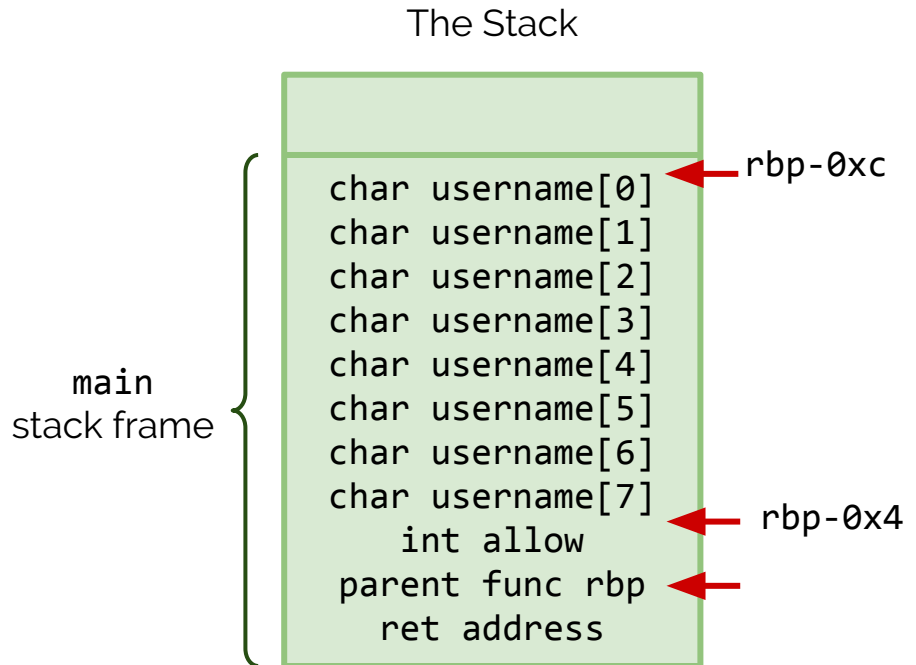
    allow = 0;

    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);

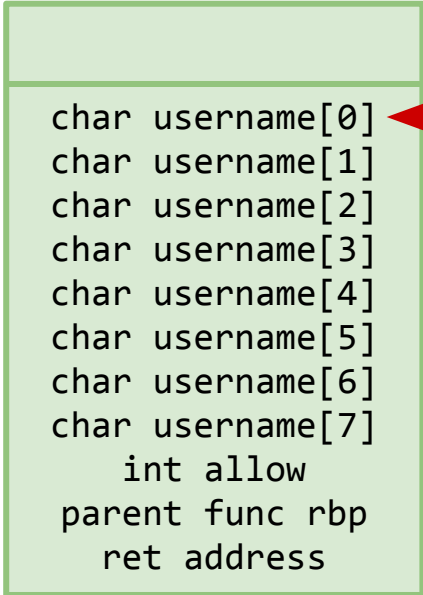
    if (strcmp(username, "chris") == 0)
        allow = 1;

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

    return 0;
}
```

main
stack frame

The Stack



char username[0] ← rax
char username[1]
char username[2]
char username[3]
char username[4]
char username[5]
char username[6]
char username[7]
int allow
parent func rbp
ret address

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;
}
```

main
stack frame

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

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;
}
```

main
stack frame

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;
}
```

main
stack frame

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

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;
}
```

main
stack frame

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

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;
}
```

main
stack frame

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

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;
}
```

main
stack frame

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

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;
}
```

main
stack frame

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

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;
}
```

main
stack frame

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

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;
}
```

main
stack frame

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;
}
```

main
stack frame

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

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;
}
```

main
stack frame

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

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	
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

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;
}
```

main
stack frame

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

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;
}
```

main
stack frame

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

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;
}
```

main
stack frame

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 m
n.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
✗ chris@chris-lab > ~/security % master •
```

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 (and their mitigations)

not examined

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?

The Stack		ASCII	Decimal
main stack frame	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

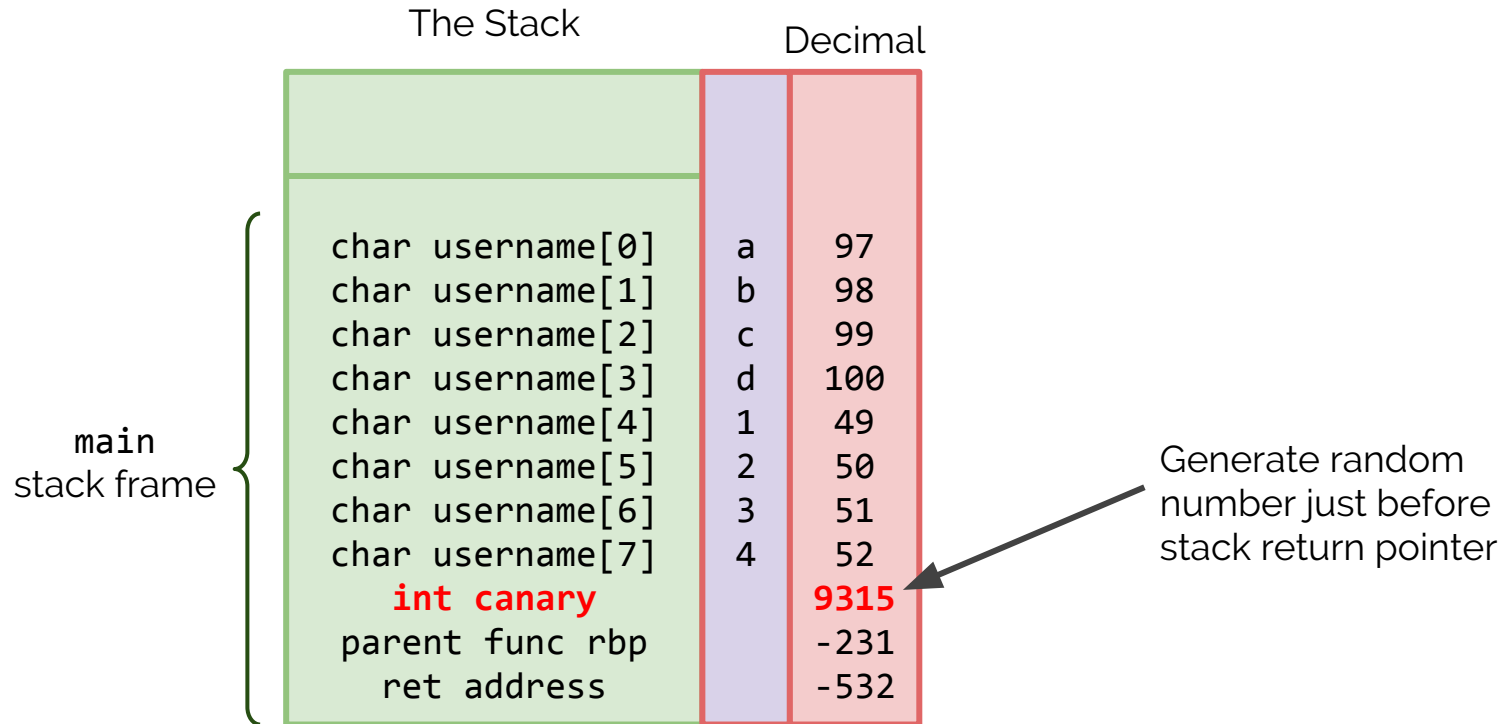
What if we kept going until we reached here?

Choose where we want to jump to...

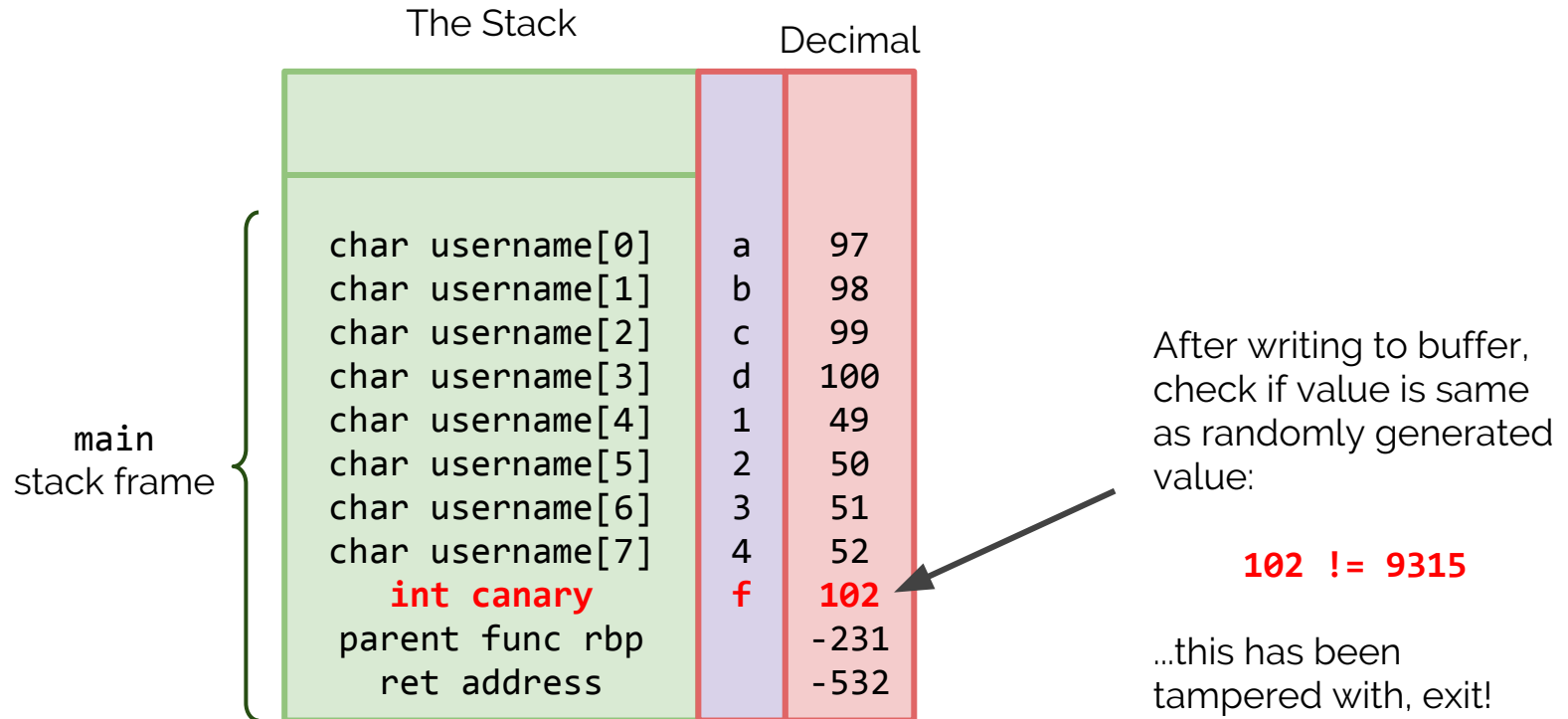
Stack Smashing

- 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 ourself, 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



Canary Value



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 process's 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.

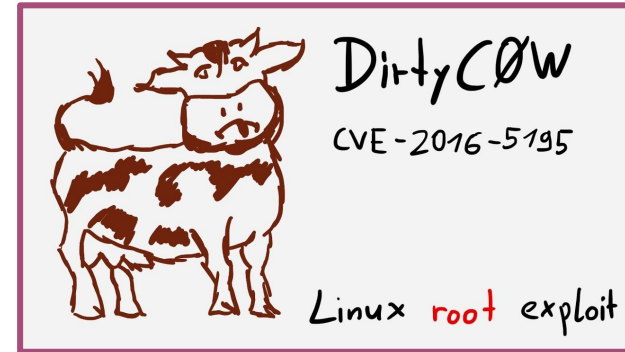
Race Condition Attacks

not examined

- 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 map 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 bruteforce

= 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.
 - 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

Impact of AI on Cyber Security

- Gaining recent research traction (especially in 2018)
- Lots of unethical research taking place in this field
 - Shown to accurately predict sensitive information about people
 - Why are you researching that? "It doesn't matter, it's not personally identifiable." Yes it does!
 - "When a model is trained, you can't reverse engineer it to recover source data, so its ok!"
- Will it advance faster than we can keep up with?

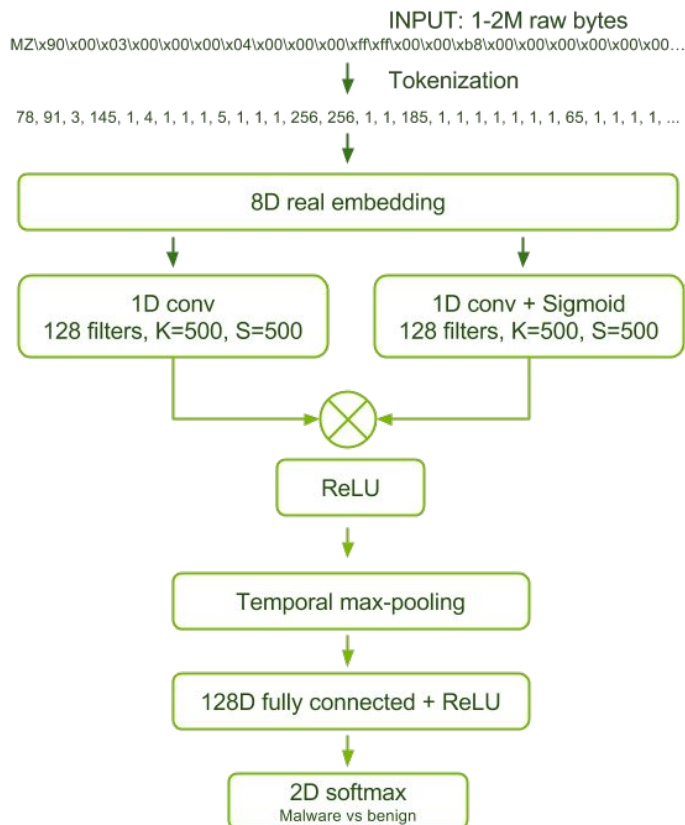
More/bigger datasets becoming available:

<https://www.secrepo.com/>

Questions:

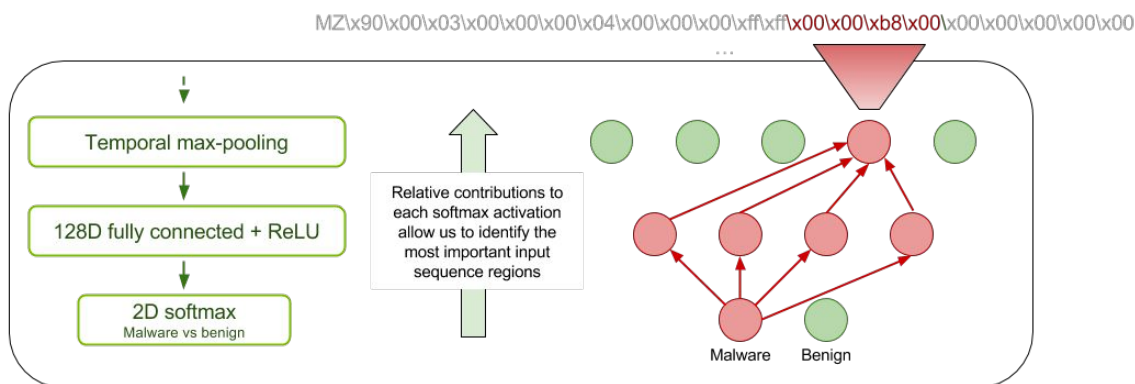
- Will AI **help us write more secure software?**
- ...or will AI discover more vulnerabilities faster than we can patch?

Impact of AI on Cyber Security



- Better anti-virus or...
- Better virus?

Good at analysis of complex ill-defined problems - large systems/datasets



Impact of AI on Cyber Security

Adversarial against Humans:

- Chatbots
 - New types of spear phishing?
- Can't distinguish human from AI voice
- Reinforcement learning (Alpha Go, Starcraft)?
 - Real-world game?
- Deep Learning fools CAPTCHA

[Fraudster Chatbots](#)

Currently mostly focusing on detecting malware and intrusions

[Collection of AI Cyber Security Research Papers](#)

Better Defense Example:

Malware Prediction



Kaggle competition launched 18 hours ago:

<https://www.kaggle.com/c/microsoft-malware-prediction>

- ProductName - Defender state information e.g. win8defender
- EngineVersion - Defender state information e.g. 1.1.12603.0
- AppVersion - Defender state information e.g. 4.9.10586.0
- AvSigVersion - Defender state information e.g. 1.217.1014.0
- IsBeta - Defender state information e.g. false
- AVProductsInstalled - NA
- AVProductsEnabled - NA
- CountryIdentifier - ID for the country the machine is located in
- CityIdentifier - ID for the city the machine is located in
- OrganizationIdentifier - ID for the organization the machine belongs in, organization ID is mapped to both specific companies and broad industries
- GeoNameIdentifier - ID for the geographic region a machine is located in

Better Offense Example:

PassGAN: A Deep Learning Approach for Password Guessing

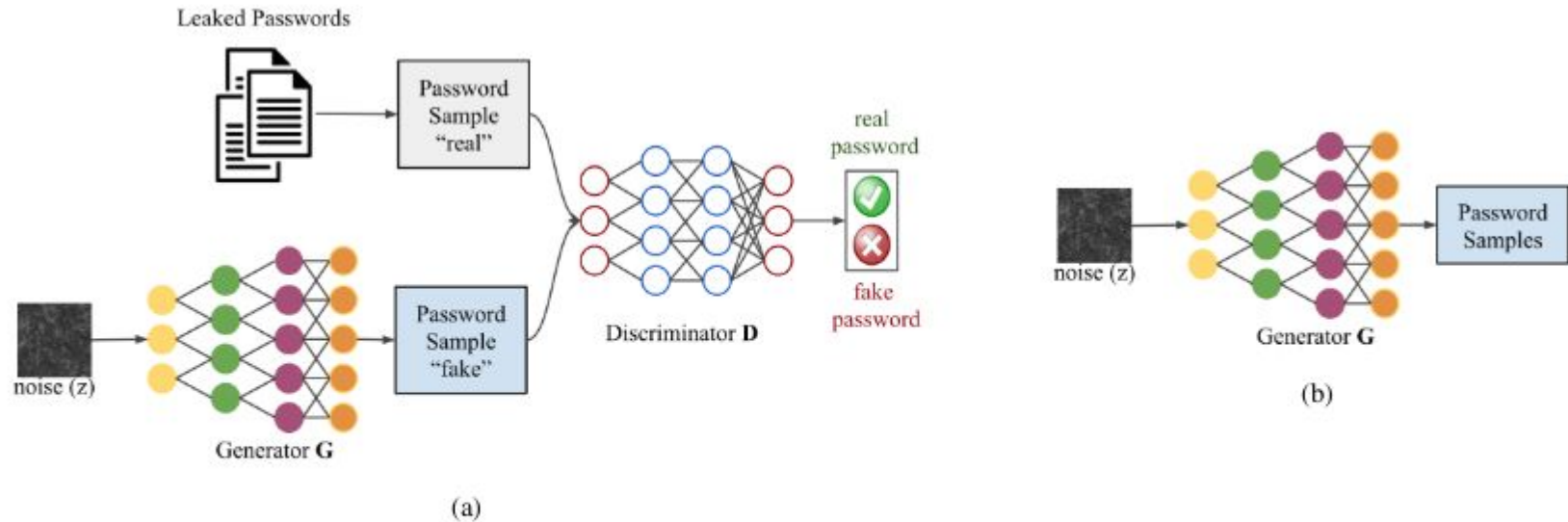
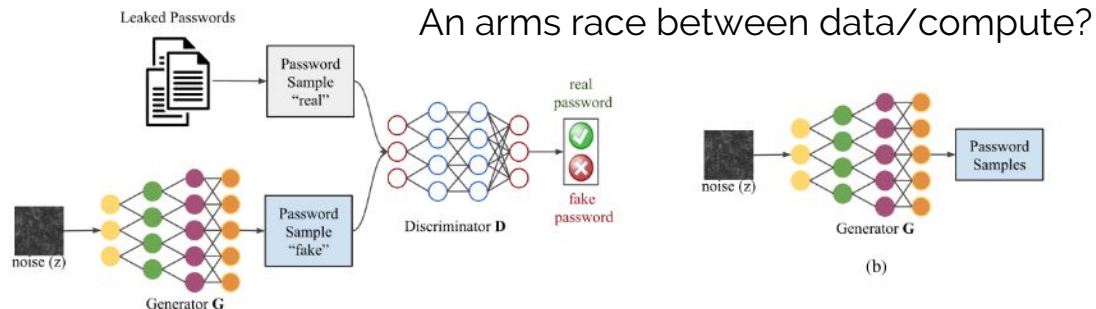


Fig. 1: Summary of PassGAN's Architecture. In the training procedure, shown in (a), the discriminator (D) processes passwords from the training dataset, as well as password samples produced by the generator (G). Based on the feedback from D, G fine-tunes its network to produce password samples that are close to the training set (G has no direct access to the training set). The password generation procedure is shown in (b).

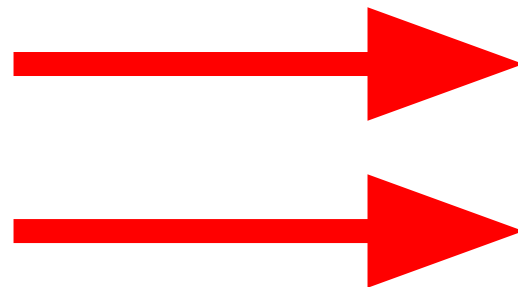
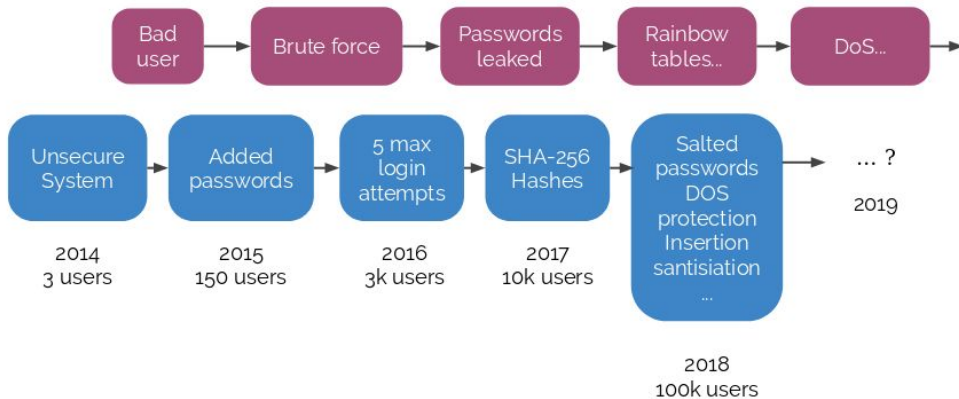
Recall from Lecture 1



image from <https://mile2.com>



An arms race between humans...



Questions?

- What to revise
- Research themes
- Any concepts discussed which were challenging to understand?



Romantic Takeaways

- **Just remember how easy it is to break in with a little time and effort**
 - Don't assume people won't try to break your code
- Security covers all levels and infrastructure of a system
 - The weakest link
 - Check your permissions
 - Use strong passwords
 - Sanitize your inputs
 - Prepare your statements
 - Lock your machine physically
- Understand **the platform**
- Understand **the people**
- Don't be careless or manage in a way that promotes carelessness
- KISS!



[cwkx](https://github.com/cwkx)



[@cwkx](https://twitter.com/cwkx)

Thank you

- **Just remember how easy it is to break in with a little time and effort**
 - Don't assume people won't try to break your code
- Security covers all levels and infrastructure of a system
 - The weakest link
 - Check your permissions
 - Use strong passwords
 - Sanitize your inputs
 - Prepare your statements
 - Lock your machine physically
- Understand **the platform**
- Understand **the people**
- Don't be careless or manage in a way that promotes carelessness
- KISS!



[cwvx](https://github.com/cwvx)



[@cwvx](https://twitter.com/cwvx)

