

# Buffer Overflow Attacks

Tom Chothia

Computer Security, Lecture 15

# Introduction

- A simplified, high-level view of buffer overflow attacks.
  - x86 architecture
  - overflows on the stack
- Exploiting buffer overflows using Metasploit

# Introduction

- In languages like C, you have to tell the compiler how to manage the memory.
  - This is hard.
- If you get it wrong, then an attacker can usually exploit this bug to make your application run *arbitrary code*.
- Countless worms, attacks against SQL servers, Web Servers, iPhone Jailbreak, SSH servers, ...

# USS Yorktown

US Navy Aegis missile cruiser

Dead in the water for 2 and a half hours due to a buffer overflow.



“Because of politics, some things are being forced on us that without political pressure we might not do, like Windows NT. If it were up to me I probably would not have used Windows NT in this particular application.”

Ron Redman, deputy technical director Aegis

# The x86 Architecture

The program code



Static variables,  
Strings, etc



Data in use



Registers e.g.

The Accumulator

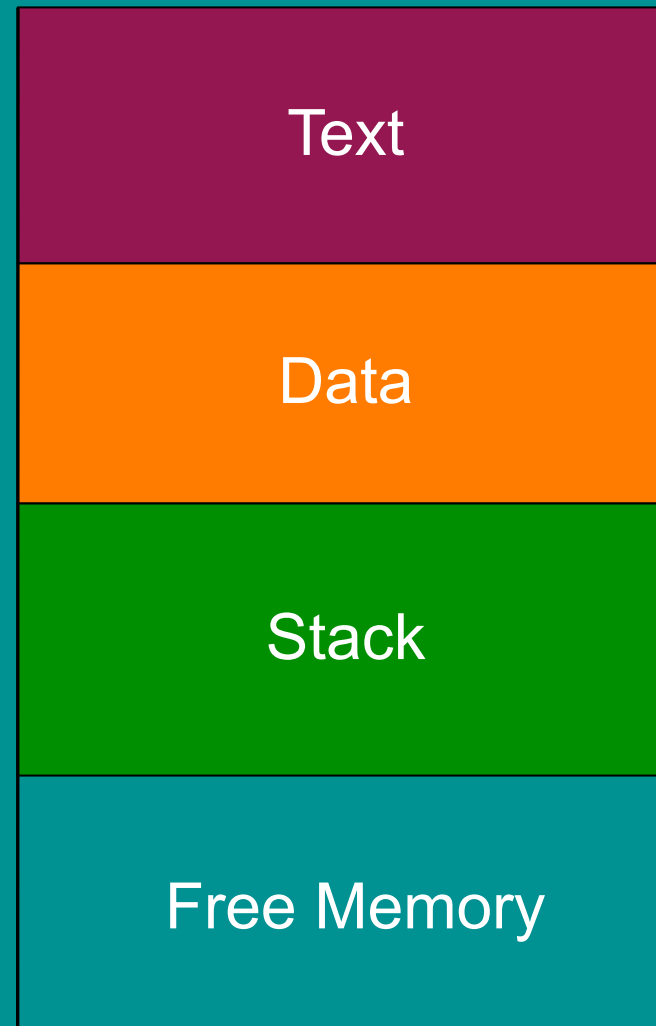
EAX

Instruction point

EIP

Stack point

ESP



CPU - main thread, module ntdll

```

77970082 . E8 85760B00 CALL ntdll.77A2770C
77970087 . 85C0 TEST EAX,EAX
77970089 . 75 04 JNZ SHORT ntdll.7797008F
7797008B . 5B POP EBX
7797008C . C2 1000 RETN 10
7797008F > FF33 PUSH DWORD PTR DS:[EBX]
77970091 . 53 PUSH EBX
77970092 . 68 A0009777 PUSH ntdll.779700A0
77970097 . FF7424 18 PUSH DWORD PTR SS:[ESP+18]
7797009B . E8 A96C0300 CALL ntdll.RtlUnwind
779700A0 > 5B POP EBX
779700A1 . 50 PUSH EAX
779700A2 . 6A 00 PUSH 0
779700A4 . 6A 00 PUSH 0
779700A6 . E8 BDF70000 CALL ntdll.ZwCallbackReturn
779700AB . 8BF8 MOV EDI,EAX
779700AD > 57 PUSH EDI
779700AE . E8 026E0300 CALL ntdll.RtlRaiseStatus
779700B3 . EB F8 JMP SHORT ntdll.779700AD
779700B5 . 8D49 00 LEA ECX,DWORD PTR DS:[ECX]
779700B8 $ 64:8B00 000000 MOV ECX,DWORD PTR FS:[0]
779700BF . BA 70009777 MOV EDX,ntdll.77970070
779700C4 . 8D4424 10 LEA EAX,DWORD PTR SS:[ESP+10]
779700C8 . 894C24 10 MOV DWORD PTR SS:[ESP+10],ECX
779700CC . 895424 14 MOV DWORD PTR SS:[ESP+14],EDX
779700D0 . 64:A3 00000000 MOV DWORD PTR FS:[0],EAX
779700D6 . 83C4 04 ADD ESP,4
779700D9 . 5A POP EDX
779700DA . 64:A1 30000000 MOV EAX,DWORD PTR FS:[30]
779700E0 . 8B40 2C MOV EAX,DWORD PTR DS:[EAX+2C]
779700E3 . FF1490 CALL DWORD PTR DS:[EAX+EDX*4]
779700E6 . 50 PUSH EAX
779700E7 . 6A 00 PUSH 0
779700E9 . 6A 00 PUSH 0
779700EB . E8 78F70000 CALL ntdll.ZwCallbackReturn
779700F0 . 8BF0 MOV ESI,EAX
779700F2 > 56 PUSH ESI
779700F3 . E8 BD6D0300 CALL ntdll.RtlRaiseStatus
779700F8 . EB F8 JMP SHORT ntdll.779700F2
779700FA . C2 0C00 RETN 0C
779700FD . 8D49 00 LEA ECX,DWORD PTR DS:[ECX]
77970100 $ FC CLD
77970101 . 8B4C24 04 MOV ECX,DWORD PTR SS:[ESP+4]
77970105 . 8B1C24 MOV EBX,DWORD PTR SS:[ESP]
77970108 . 51 PUSH ECX
77970109 . 53 PUSH EBX
    
```

```

_eax_value
pExoptRec
ReturnAddr = nt
pRegistrationFr
RtlUnwind
    
```

Registers (FPU)	
EAX	C0000034
ECX	00000000
EDX	00000000
EBX	00000001
ESP	0018F9B0
EBP	0018F9E8
ESI	00525034
EDI	0000000C
EIP	7797F9CD ntdll.7797F9CD
C 0	ES 002B 32bit 0(FFFFFFFF)
P 0	CS 0023 32bit 0(FFFFFFFF)
A 1	SS 002B 32bit 0(FFFFFFFF)
Z 0	DS 002B 32bit 0(FFFFFFFF)
S 0	FS 0053 32bit 7EFDD000(FFF)
T 0	GS 002B 32bit 0(FFFFFFFF)
D 0	
O 0	LastErr ERROR_ACCESS_DENIED (00000005)
EFL	00000212 (NO,NB,NE,A,NS,PO,GE,G)
ST0	empty 0.0
ST1	empty 0.0
ST2	empty 0.0
ST3	empty 0.0
ST4	empty 0.0
ST5	empty 0.0
ST6	empty 0.0
ST7	empty 0.0

Address	Hex dump	ASCII
00405000	00 00 00 00 00 00 00 00	.....
00405008	00 00 00 00 00 00 00 00	.....

K Call stack of main thread

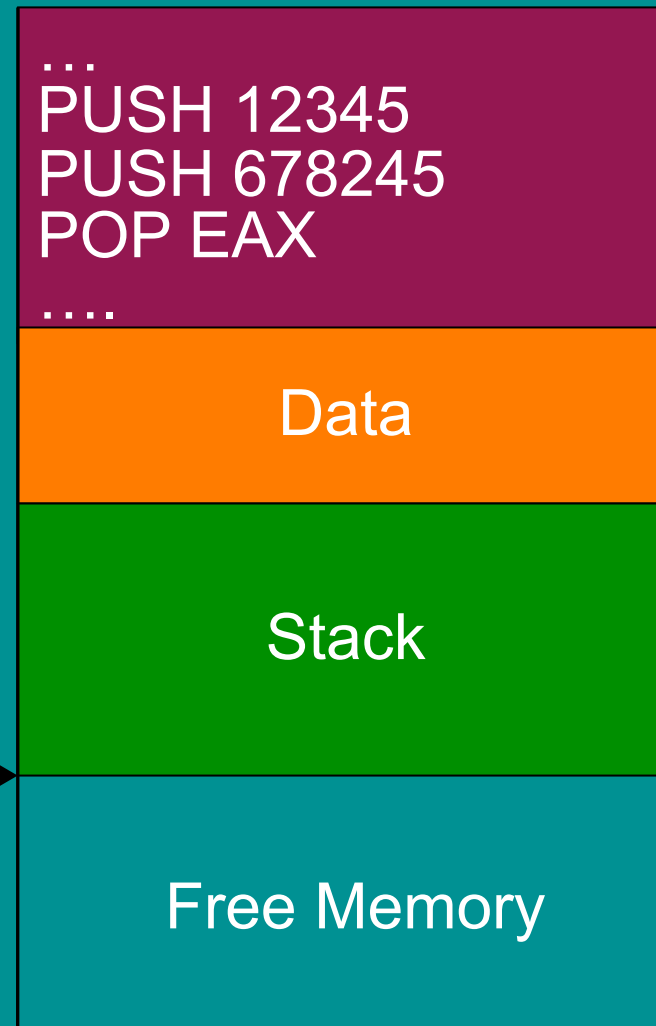
Address	Stack	Procedure / arguments	Called from
0018F9B0	779C34C0	ntdll.ZwOpenKey	ntdll.779C34BB
0018F9EC	779C343B	ntdll.779C3444	ntdll.LdrOpenImageFileOptionsKey
0018FA00	779E70AB	ntdll.LdrOpenImageFileOptionsKey	ntdll.779E70A6
0018FA04	00525034	Arg1 = 00525034	
0018FA08	00000000	Arg2 = 00000000	
0018FA0C	0018FA34	Arg3 = 0018FA34	
0018FA18	779AC160	ntdll.LdrQueryImageFileExecutionOptions	ntdll.LdrQueryImageFileExecutionOptions
0018FA3C	779D07D0	ntdll.LdrQueryImageFileExecutionOptions	ntdll.779D07D8
0018FA40	00525034	Arg1 = 00525034	
0018FA44	7798400C	Arg2 = 7798400C	
0018FA48	00000004	Arg3 = 00000004	
0018FA4C	0018FAF8	Arg4 = 0018FAF8	
0018FA50	00000004	Arg5 = 00000004	
0018FA54	00000000	Arg6 = 00000000	
0018FB38	779B37EE	? ntdll.7799DC25	ntdll.779B37E9

# The Stack

The stack part of the memory is mostly “Last In, First Out”.

We can only write and read to the top of the stack.

EAX:
EIP: 7797F9CD
ESP: 0018F9B0

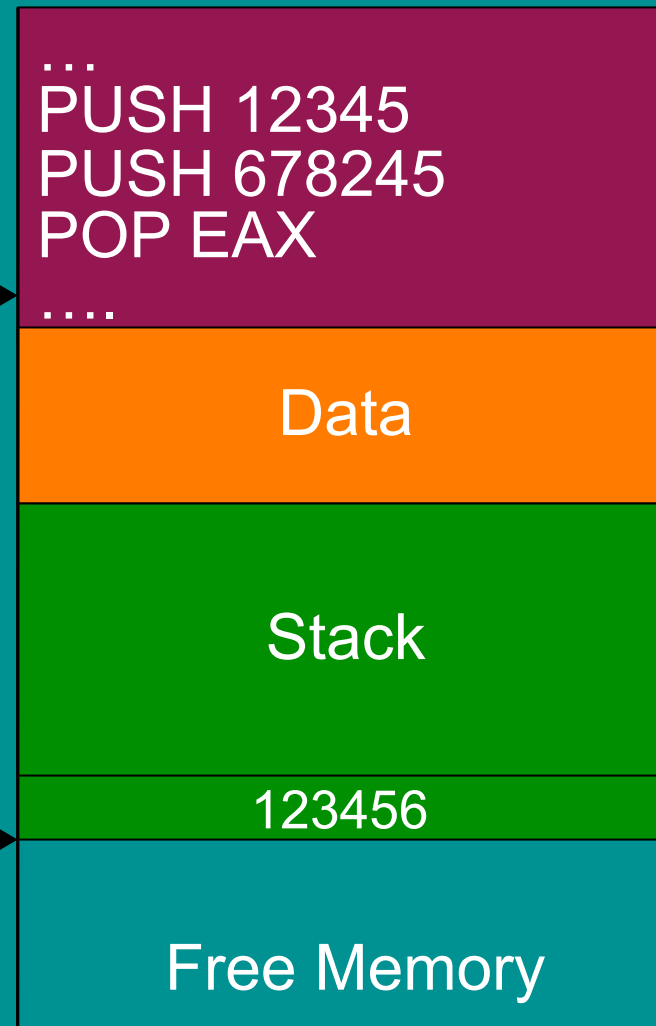


# The Stack

You write to the stack with push


You read and remove an item from the stack with pop

EAX: 678245  
EIP: 7797F9CF  
ESP: 0018F9B1



# Function calls

```
void main () {  
    function (1,2);  
}
```



```
PUSH <2>  
PUSH <1>  
CALL <function>
```

- Arguments 1 & 2 are passed on the stack.
- The CALL instruction runs a function

# Function Calls

```
PUSH <arg2>
```

```
PUSH <arg1>
```

```
CALL <function>
```

`CALL` writes the instruction point (EIP) onto the stack and then sets the EIP to to equal the code for the function.

Later a return instruction restores the old EIP and the program continues



# Buffer Overflows

- The instruction pointer controls which code executes,
- The instruction pointer is stored on the stack,
- I can write to the stack ... 😊

# Buffers

1. Functions called with "Hello World" ...  
`function (user input);`
2. Arg and EIP written to stack ...
3. Function runs  
`function (char *str) {`
4. Buffer allocated  
`char buffer[16];`
5. String copied  
`strcpy(str,buffer);`  
`}`

	Hello World	Old EIP	Hello World	Stack
--	-------------	---------	-------------	-------

# Buffer Overflow

If user input is more than 16 bytes

1. Runs as before
2. But the string flows over the end of the buffer
3. EIP corrupted, segmentation fault

```
...  
function (user input);  
...  
  
function (char *str) {  
    char buffer[16];  
    strcpy(str,buffer);  
}
```



# Once more, with malice

1. Runs as before
2. Attack send a very long message, ending with the address of some code that gives him a shell.
  - The attackers code could also be part of the message
3. The attackers value is copied over the old EIP
4. When the function returns the attacks code is run

	Hello WorldXX	7797F9	Hello WorldX X7797F9	Stack
--	---------------	--------	-------------------------	-------

# Metasploit

- Metasploit is a framework for testing buffer overflow attacks.
- <http://metasploit.com/>

# Over Writing Other Values

Attacking the instruction pointer (EIP) is the most powerful technique. However, any memory value can be attacked:

- Over write arguments on the stack
  - e.g. change the parameters to a chmod call
- Overflows on the heap
  - e.g. rewrite a password in memory

# Defenses

- Stack canaries:
  - values placed on the stack, which are later tested.
  - if the stack is over written then the value test will fail.
- Randomisation
  - Layout of the memory is randomised.
  - This makes it very hard for the attack to find the memory to overwrite or code to jump to.

For more information see the Secure Programming Module

# Recommend Paper:

- “Smashing the Stack for Fun and Profit”  
Elias Levy (Aleph One)

A simple introduction to buffer overflows from the mid 90s.

Standard defences now stop the attacks in this paper, but it gives an excellent introduction.

# Conclusion

Buffer overflows are the result of poor memory management in languages like C – even the best programmers sometimes make mistakes.

Buffer overflow attacks exploit these to overwrite memory values.

This often lets an attack execute arbitrary code.