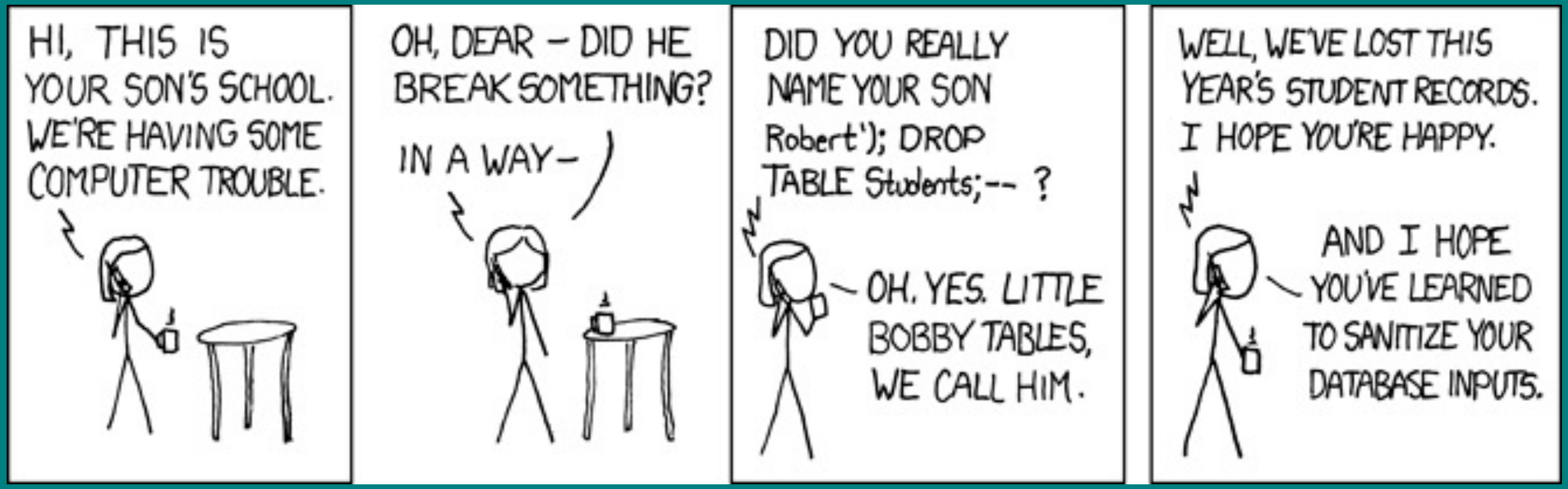
Binary analysis and application security

David Oswald and Eike Ritter
Small changes for ‘21 by Ian Batten
Introduction to Computer Security,
Based on a course by Tom Chothia

Data can be Code

- Lots of the attacks we have seen trick a program into accept data that is really code, e.g.,
  - SQL injection
  - XSS
  - Buffer overflow (next lecture)
- This is a very common way to attack systems.

“Little Bobby Tables”



Code is Data

- In this lecture we are going to do the opposite.
- Executable code can be written and edited, just like an other document.
- Ultimately, an attacker/analyst can do ***anything*** they want with a program.

Introduction

- Compiled code is really just data…
  … which can be edited and inspected.
- By examining low-level code, protections can be removed and the function of programs altered.
- Good protection tends to slow down this process, not stop it.

This lecture

- Java Byte code:
  - High level overview
  - Inspecting the byte code
  - Decompiling back to Java
- x86 assembly:
  - High level overview
  - Inspecting and altering binaries in Ghidra

Learning Objectives

- I ***don't*** want you to memorise assembly, or Java byte code commands.
- I ***do*** want you to have a understanding of how machine code works, and is compiled.
- I ***do*** want you to know what buffer overflow attacks are and how they work.
- I ***do*** want you to understand that an attacker can view and edit assembly.

Reasons For Reverse Engineering

- Analyse malware
- Debug memory errors
- Analyse legacy code
- Security audit

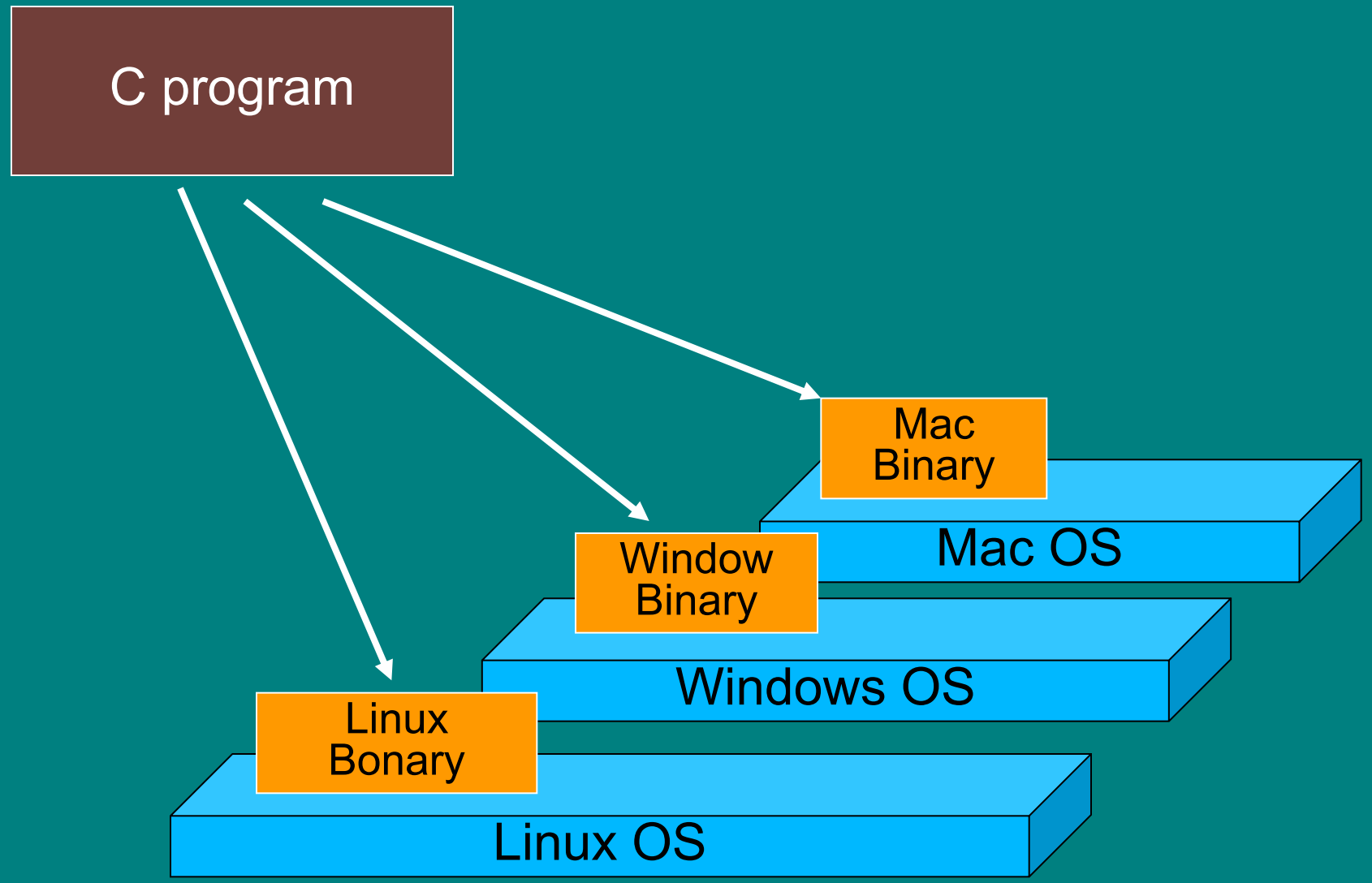
Live-Demo

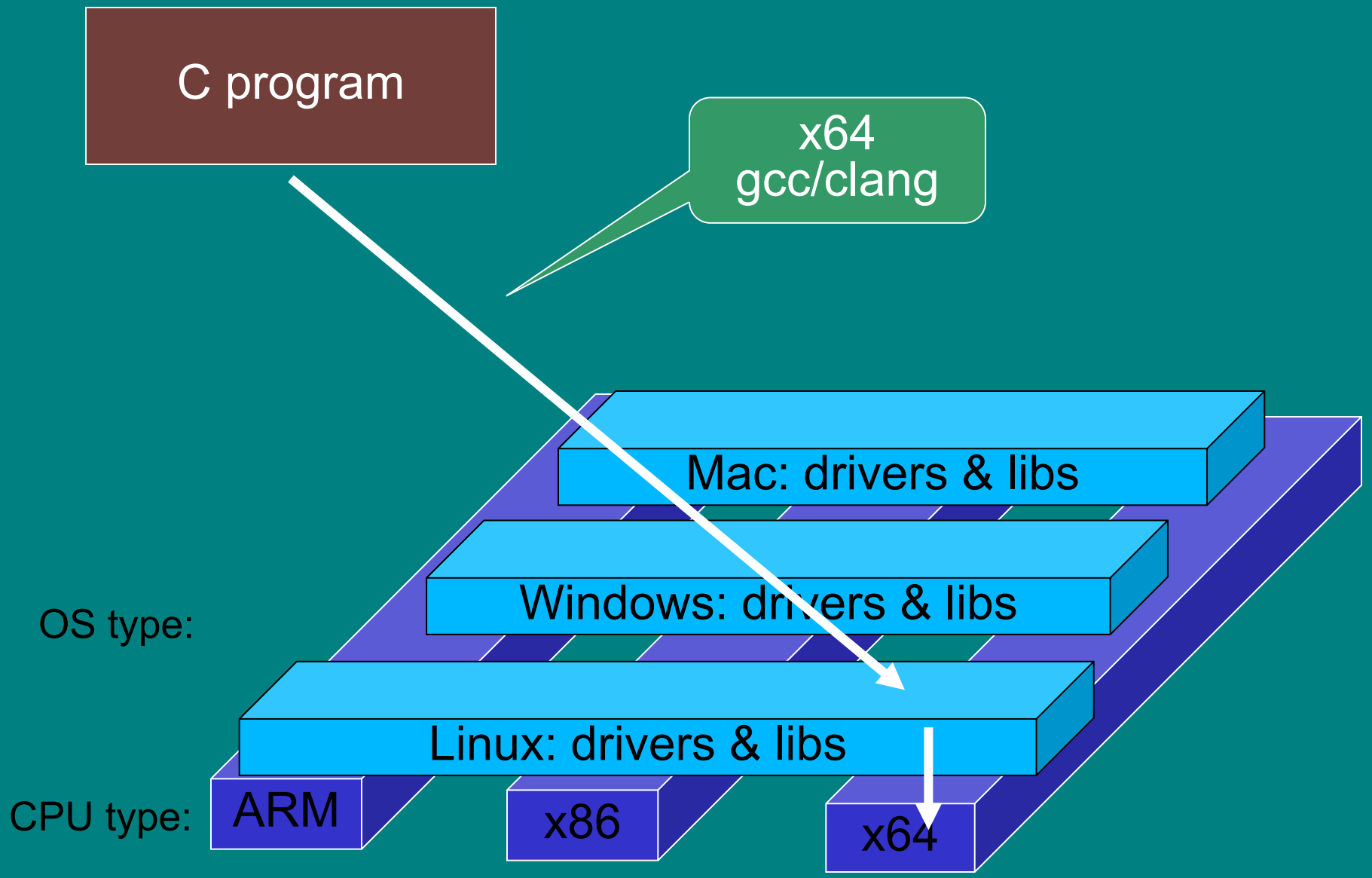
"Anything that can go wrong, will go wrong"

A password checker in Java

Binaries

- Binaries are written in assembly
- Much lower level than Java byte code
- Assembly compiled for one type of machine won’t run on another
- But the same techniques apply


C program
Mac
Binary
Mac OS
Window
Binary
Windows OS
Linux
Bonary
Linux OS


C program
x64
gcc/clang
Mac: drivers & libs
Windows: drivers & libs
Linux: drivers & libs
OS type:
CPU type:
ARM
x86
x64

IDA pro and Ghidra

- IDA pro is an Interactive DisAssembler.
- It helps a human understand binaries.
- This is the standard tool for malware binary analysis, security analysis of firmware and reverse engineering.
- There is are free & demo versions: http://www.hex-rays.com/
- NSA released (open-source) Ghidra – very powerful as well, decide for yourself

Some x86 Commands

PUSH: add to top of stack

POP: read and remove from top of stack

CALL: execute a function

JMP: jump to some code (like writing to EIP)

RET, RETN, RETF: end a function and restart calling code.

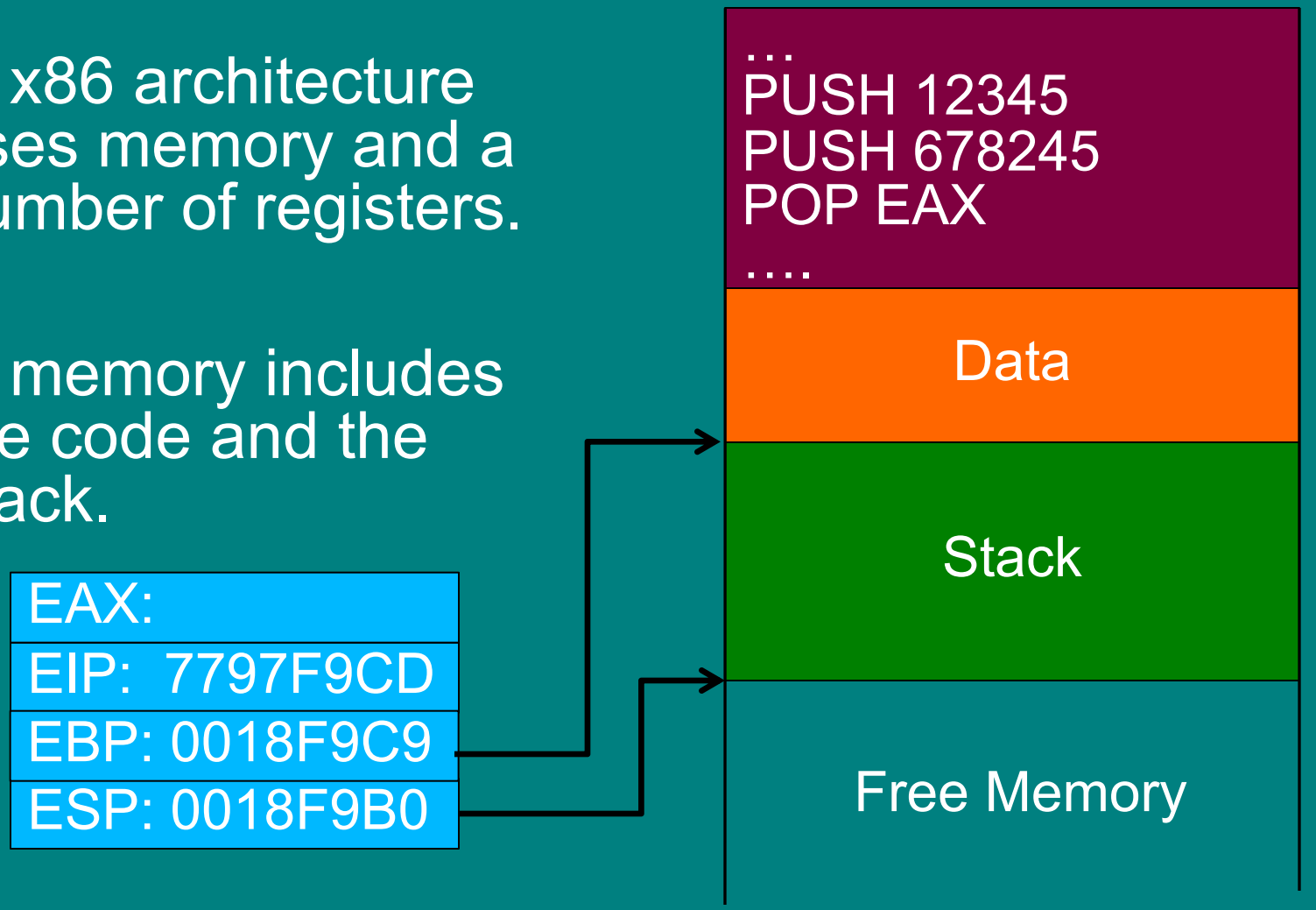
MOV: move value between registers

MOV r1,r2 = PUSH r2
POP r1

x86

The x86 architecture uses memory and a number of registers.

The memory includes the code and the stack.



Common Pattern 1

Data is moved to a register, operation is called, result stored in memory location or register.

```
mov     eax, [esp+1Ch]
add     [esp+18h], eax
```

- Value at [esp+1Ch] is moved to register eax,
- It is added to the value at [esp+18h]
- The result is stored at [esp+18h]

Flags

After an arithmetic operation flags are set.

- ZF: Zero flag
  - Set to 1 if result is 0
- SF: Sign flag
  - Set to 1 if result is negative
- OF: Overflow flag:
  - Set to 1 if operation overflowed.

Compare and Test

Compare and tests will set these flags, with no other effect.

- CMP a b
  - calculates a-b then sets flags
- TEST a b
  - does a bitwise “and”: a /\ b then sets flags

Jump Commands

- Jump if equal, Jump if zero
  - JE,JZ address
  - Jumps to address if ZF = 1
- Jump if not equal, Jump if not zero
  - JNE,JNZ address
  - Jumps to address if ZF =/= 0
- Jump if less than
  - JL address
  - Jump to address if SF=1 and OF=/=1

Common Pattern 2

Data is compared using “cmp” or “test”, then a jump is made based on the result.

```
cmp     dword ptr [esp+1Ch], 3
jle     short loc_80483DF
```

- Value [esp+1Ch] – 3 is calculated (not stored)
- If it is less than or equal to zero, the program jumps to location “loc_80483DF”
- Otherwise it continues to the next command.

Common Pattern 3

- Data is loaded onto the stack
- Function is called that uses these values,
- The result will be pointed to by eax

```
mov     [esp+4], eax    ; s2
mov     dword ptr [esp], offset s1 ; "exit"
call    _strncmp
```

- Value in eax is moved to [esp+4]
- “exit” is put on top of the stack
- String compare is called on these.
- The result will be returned in the eax register

Live-Demo

"Anything that can go wrong, will go wrong"

Analysing and patching password checkers in C with Ghidra

Common Techniques

- Look for strings
- Identify key tests and check the values in the register using a debugger
- Swap JEQ and JNEQ etc.
- Jump over the instructions that perform checks (replace with NOP)

Defenses

- Dynamically construct the code
  - Attacker can run code
- Encrypt the binary
  - Your program must include the key in plain text, so the attacker can find it
- Obfuscate the code, e.g. mix data and code, so it's not clear which is which
  - Can slow down attacks by months or years! (e.g. Skype)

Defense

- Require online activation:
  Activation can be completely disabled, users don’t like this.
- Require online content, e.g. WoW, BlueRay
- Require a hardware dongle of some sort
- Hardware-based protection, i.e. store and *run* part of the code in tamper-resistant hardware.

Summary

- Machine code can be inspected and edited.
- Many tools exist to inspect, debug and decompile code.
- Most software protection can be removed.
- But slowing this down by months or years can save a business.