

CyberSecurity: Principle and Practice

*BSc Degree in Computer Science
2022-2023*

Lesson 14: Intro to PWNING

Prof. Mauro Conti

Department of Mathematics
University of Padua
conti@math.unipd.it
<http://www.math.unipd.it/~conti/>

Teaching Assistants

Tommaso Bianchi
tommaso.bianchi@phd.unipd.it.
Pier Paolo Tricomi
pierpaolo.tricomi@phd.unipd.it



UNIVERSITÀ
DEGLI STUDI
DI PADOVA



SPRITZ
SECURITY & PRIVACY
RESEARCH GROUP



DIPARTIMENTO
MATEMATICA

All information presented here has the only purpose of teaching how reverse engineering works.

Use your mad skillz only in CTFs or other situations in which you are legally allowed to do so.

Do not hack the new Playstation. Or maybe do, but be prepared to get legal troubles 😊

PWN?

The **term** was created accidentally by the misspelling of "own" in video game design due to the keyboard proximity of "O" and "P."

It implies domination or humiliation of a rival, used primarily in the Internet-based video game culture to taunt an opponent who has just been soundly defeated (e.g., "You just got **pwned!**").

Here, we want to dominate the victim program/process, exploiting **memory corruption!**

WHAT'S MEMORY CORRUPTION?

Modifying a process' memory in a way the programmer (or compiler) didn't intend.

If we control the memory, we control the process.

MEMORY CORRUPTION IN THE WILD

- **Malware**
 - Morris (1988!), CodeRed, Blaster, Sasser, Conficker, ...
 - More recently, StuxNet and WannaCry
- **Can be used to attack remote services and user applications**
 - Exposed to untrusted data
- **Unlocking devices**
 - Android roots, iOS jailbreaks, gaming consoles

MEMORY CORRUPTION VULNERABILITIES

- Buffer overflows
 - data written to a **buffer** corrupts data values in memory addresses adjacent to **buffer**
- Out-of-bounds accesses
 - Exceeding the **bounds** of the array to **accessing memory** that is not assigned to the array
- Format strings
 - Submitted data of an input string is evaluated as a command by the application
- Dangling pointers (e.g., use-after-free)
 - Referencing memory after it has been freed can cause a program to crash
- Type confusion
 - code that doesn't verify the type of object that is passed to it, and uses it blindly without type-checking

... and many more

L. Szekeres, M. Payer, T. Wei, D. Song. “SoK: *Eternal War in Memory*”
(2013 IEEE Symposium on Security and Privacy)

MEMORY CORRUPTION ATTACKS

Two main subclasses:

- *Non-Control-Data Attacks* manipulate the application's state and data
- *Control-Flow Attacks* manipulate the execution flow

EXPLOITATION

Finding a vulnerability is just the first step.

Uncontrolled memory corruption typically results in a crash (e.g., SIGSEGV).

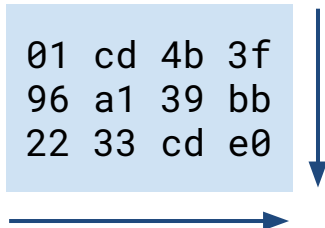
We need to convey the vulnerability into whatever we want to do: this process is called ***exploitation***.

WHAT'S MEMORY?

Memory is a flat sequence of bytes. That's it.

Each byte is identified by an *address*.

Via *memory protection*, areas of memory can be marked as readable, writable, executable.



01	cd	4b	3f
96	a1	39	bb
22	33	cd	e0

INTERPRETATIONS OF MEMORY

Types do not exist in memory. They are just abstractions that define how a certain range of bytes is interpreted.

Example: integers (and pointers) are little-endian on x86.

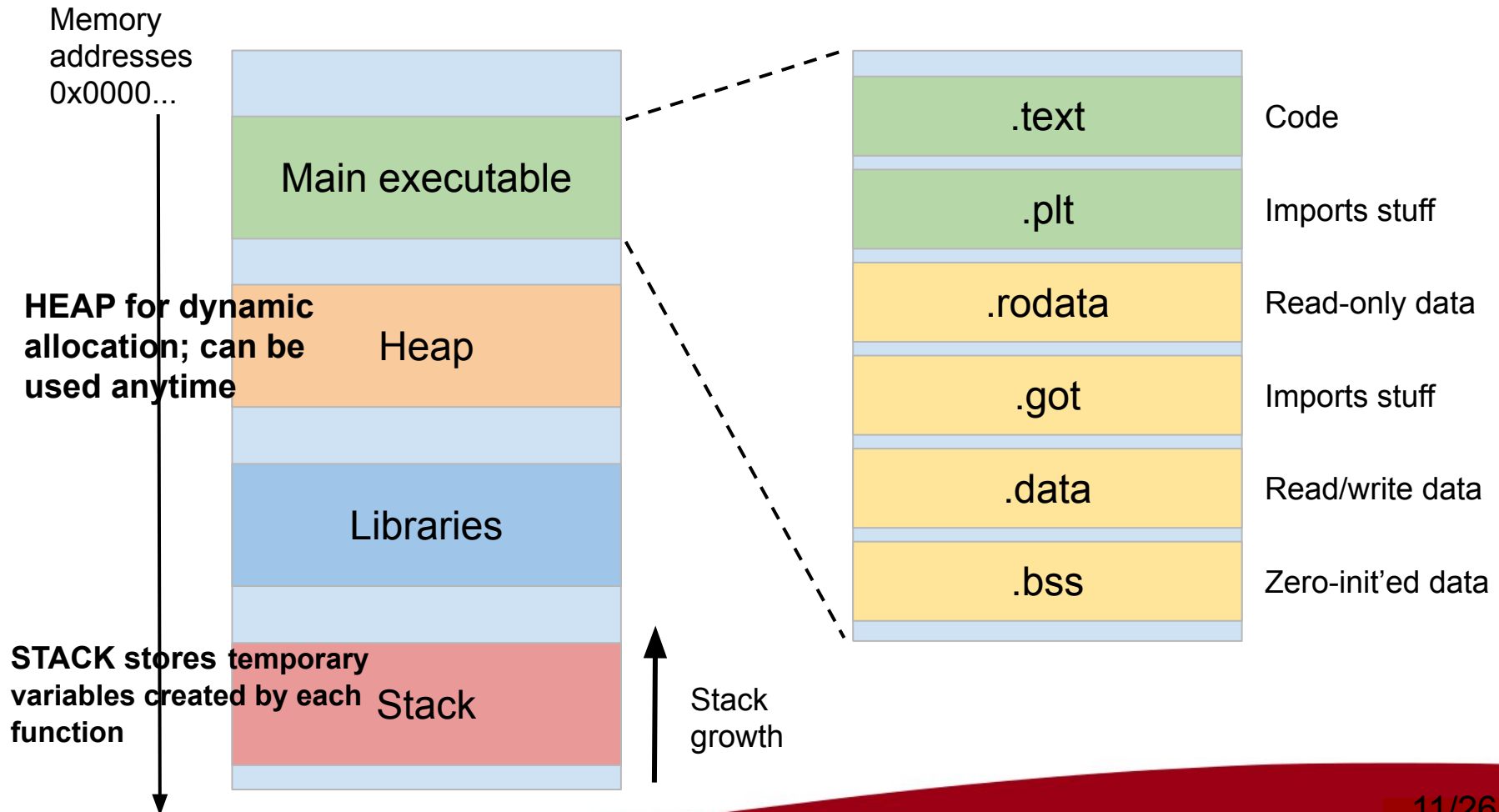
78 56 34 12 \leftrightarrow 0x12345678

Little Endian Byte Order:
The least significant byte (the "little end") of the data is placed at the byte with the lowest address.



Example: C arrays are a contiguous sequence of elements.

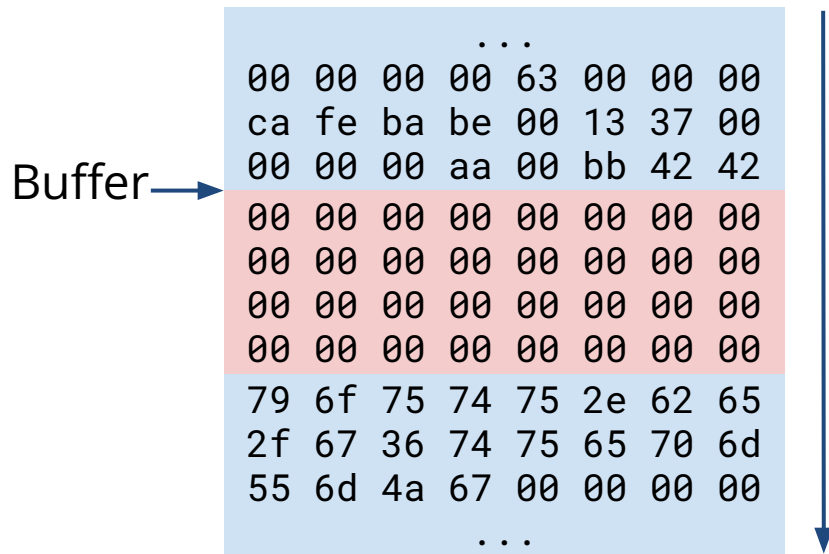
a (Linux) process' memory



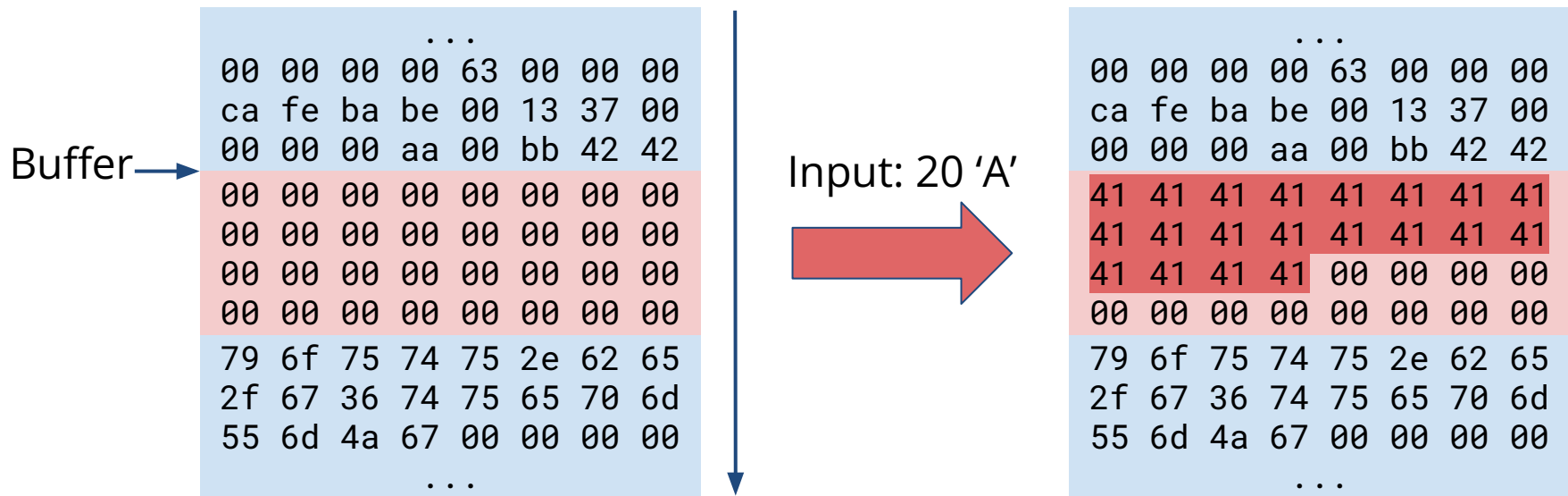
Some languages (such as C/C++) do not check array bounds.

If the programmer doesn't perform those checks, s/he might write data beyond the buffer's boundaries.

This program copies the user's input to a fixed size 32-byte buffer.



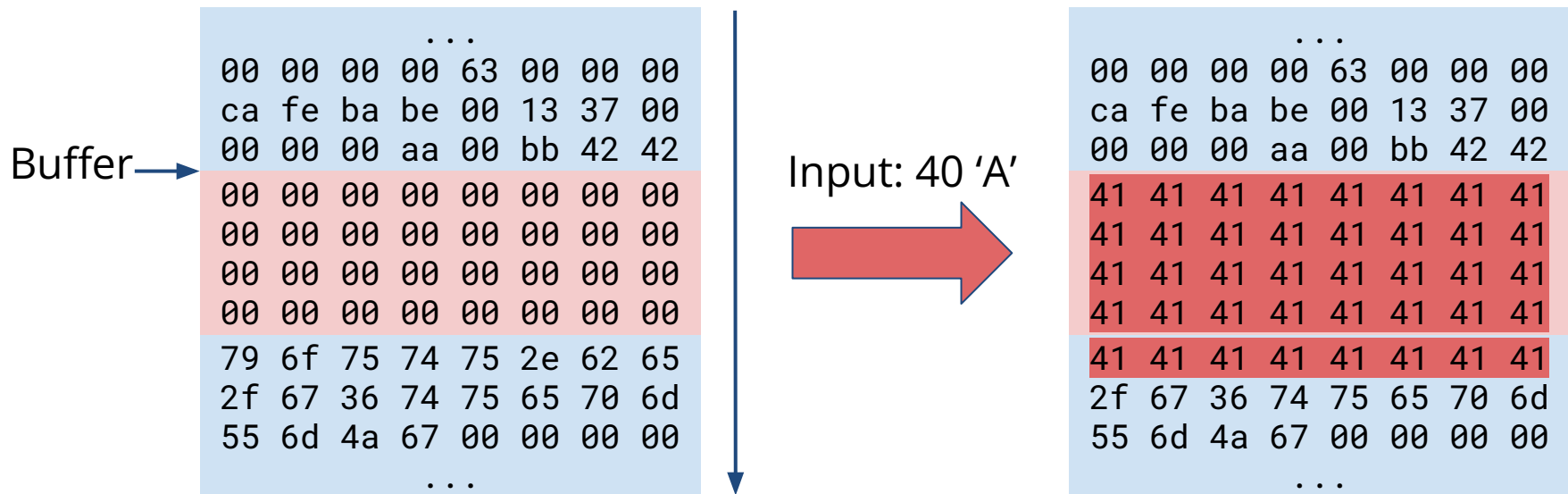
This program copies the user's input to a fixed size 32-byte buffer.



Buffer Overflows



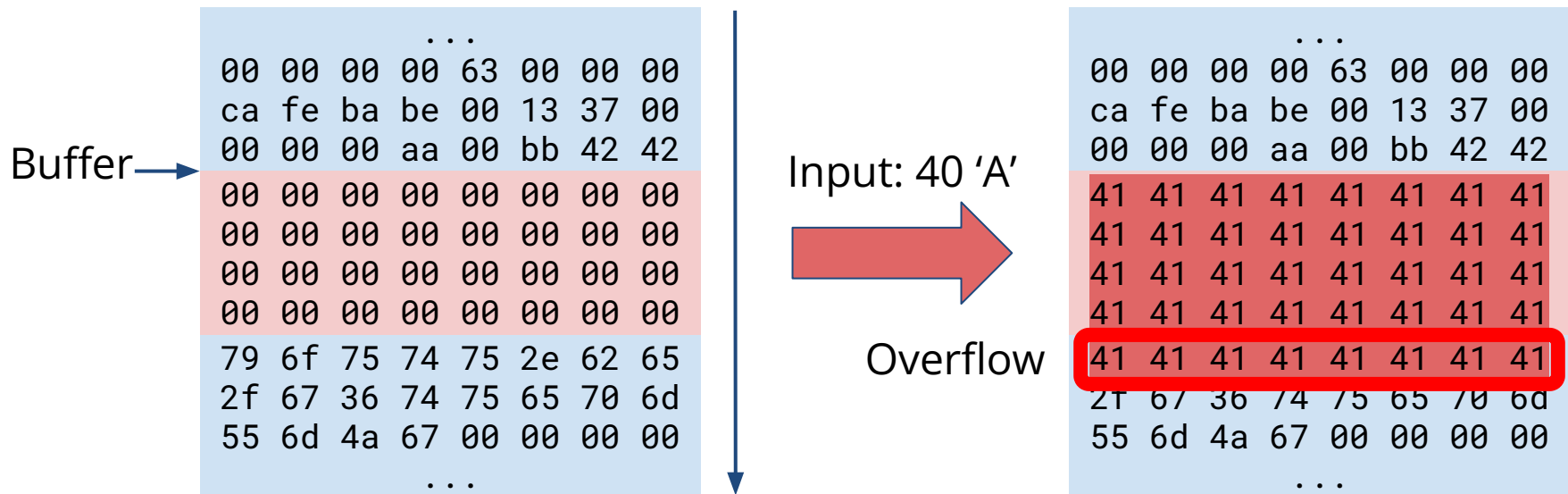
This program copies the user's input to a fixed size 32-byte buffer.



Buffer Overflows



This program copies the user's input to a fixed size 32-byte buffer.



AUTH OVERFLOW

Inspired from Jon Erickson's *"Hacking: The Art of Exploitation"*

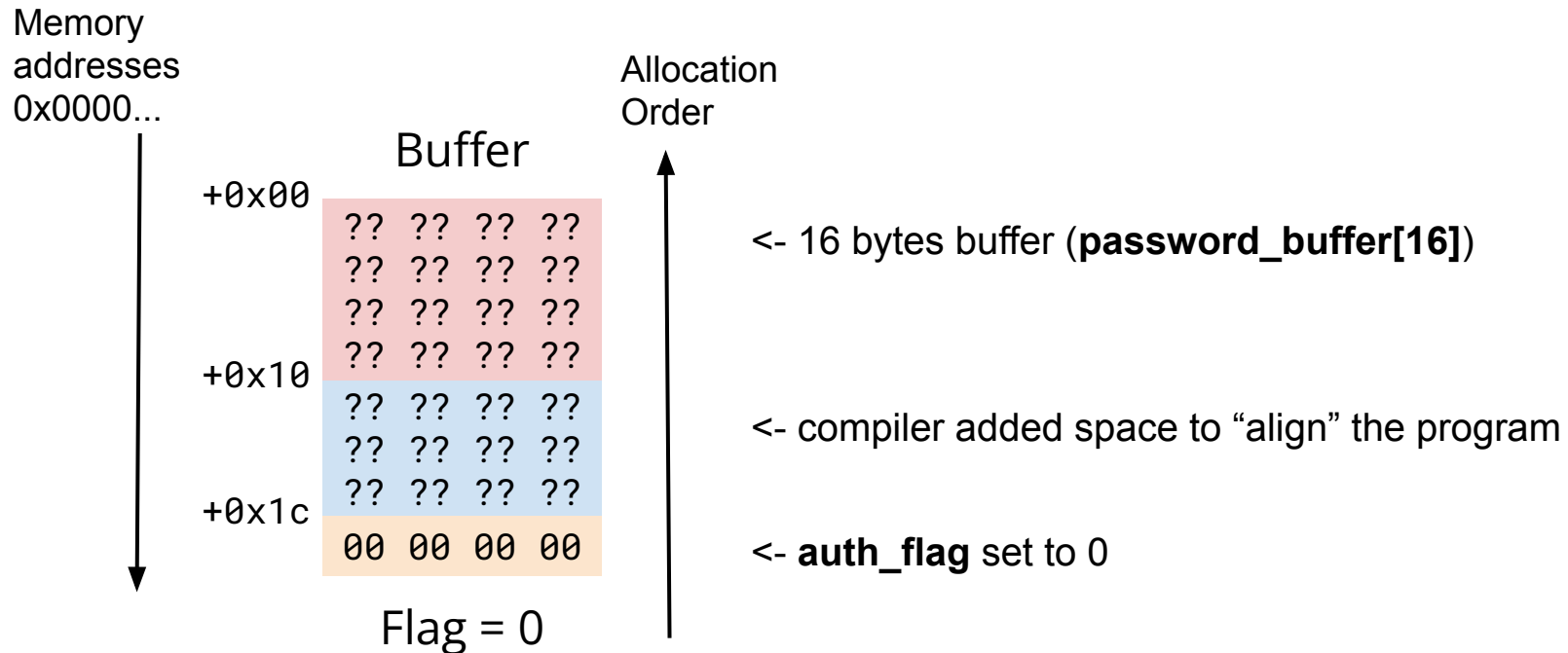
```
int check_authentication() {  
    int auth_flag = 0;  
    char password_buffer[16];  
    printf("Enter password");  
    scanf("%s", password_buffer);  
    /* password_buffer ok? => auth_flag = 1 */  
    return auth_flag;  
}
```

AUTH OVERFLOW

Inspired from Jon Erickson's *"Hacking: The Art of Exploitation"*

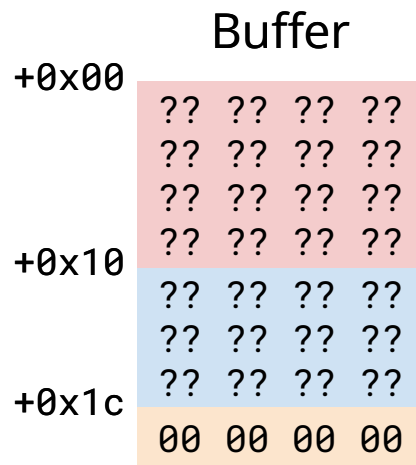
```
int check_authentication() {  
    int auth_flag = 0;  
    char password_buffer[16];  
    printf("Enter password");  
    scanf("%s", password_buffer);  
    /* password_buffer ok? => auth_flag = 1 */  
    return auth_flag;  
}
```

AUTH OVERFLOW



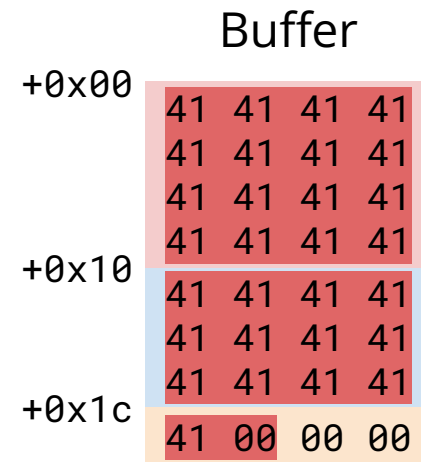
Note for alignment: Code that's executed on WORD or DWORD boundaries executes faster because the processor fetches whole (D)words. So if your instructions aren't aligned then there is a stall when loading. If a (D)word is fetched and contains just a “piece” of instruction, it is still interpreted as an instruction, and might cause errors or crashes

AUTH OVERFLOW



Flag = 0

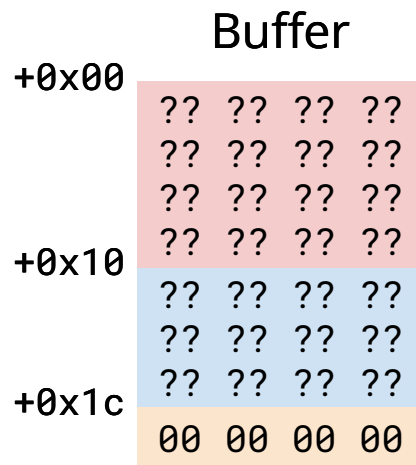
Input: 29 'A'



Flag = 65

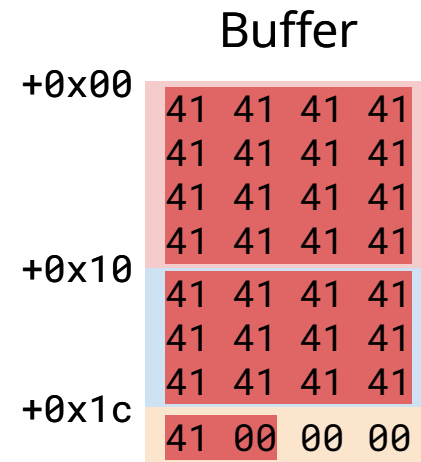
(41 00 Little Endian
=> Hex 00 41
=> 65 base 10)

AUTH OVERFLOW



Flag = 0

Input: 29 'A'



Flag = 65

check_authentication will now return 65.

AUTH OVERFLOW - THE CHECK

```
if (check_authentication())  
    /* access granted */
```

Returns 65



In C, anything $\neq 0$ is true.

The check will pass and grant us access. Profit!

Ida/radare2 & gdb/peda-gdb if reverse is involved

<https://github.com/longld/peda>

pwntool (python library):

<http://docs.pwntools.com/en/stable/>

Very good pwntool tutorials:

<https://github.com/Gallopsled/pwntools-tutorial>

DEMO TIME!

- 1) A password is required to access. Do we really need it?
- 2) How's the josh?
- 3) Comparing strings in assembly is an hard task for java n00bs.

Questions? Feedback? Suggestions?



UNIVERSITÀ
DEGLI STUDI
DI PADOVA

