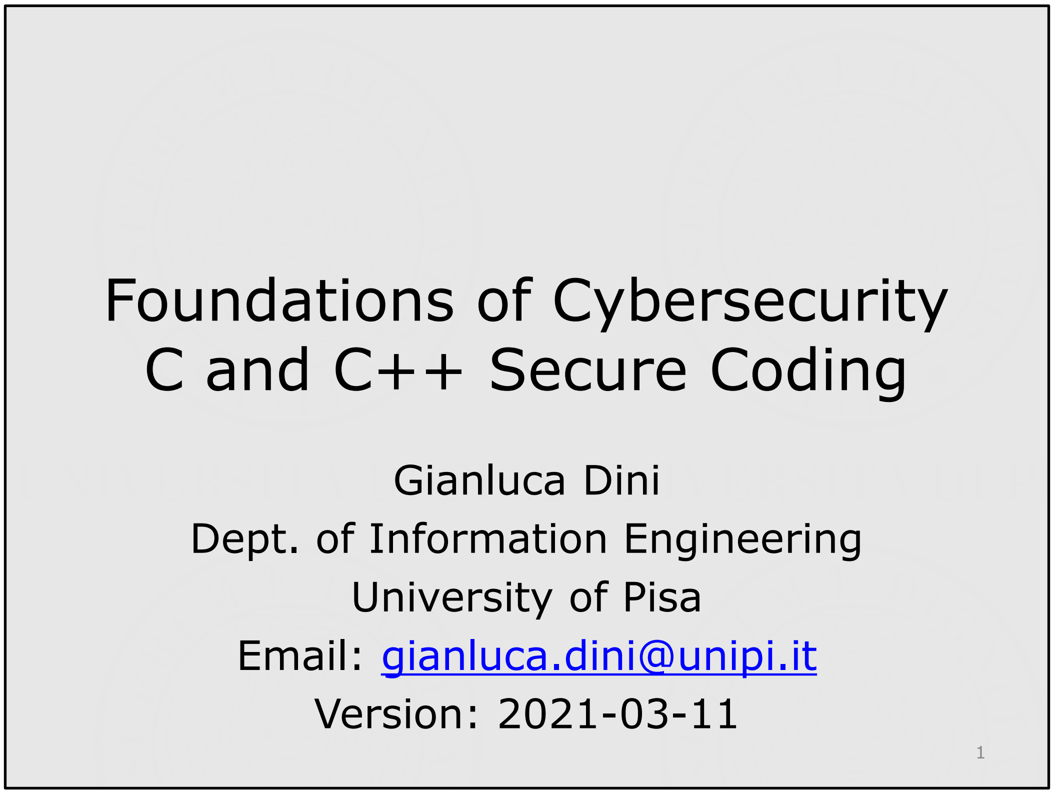
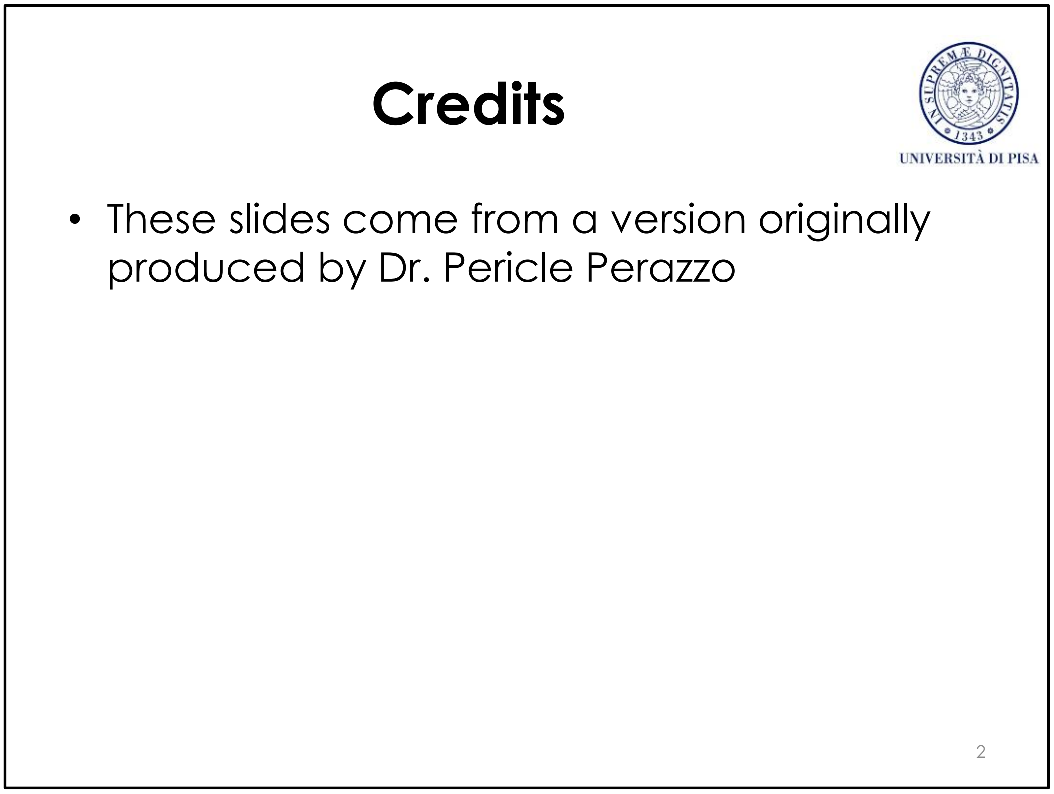


Foundations of Cybersecurity C and C++ Secure Coding

Gianluca Dini
Dept. of Information Engineering
University of Pisa
Email: gianluca.dini@unipi.it

Version: 2021-03-11

1



Credits

UNIVERSITÀ DI PISA

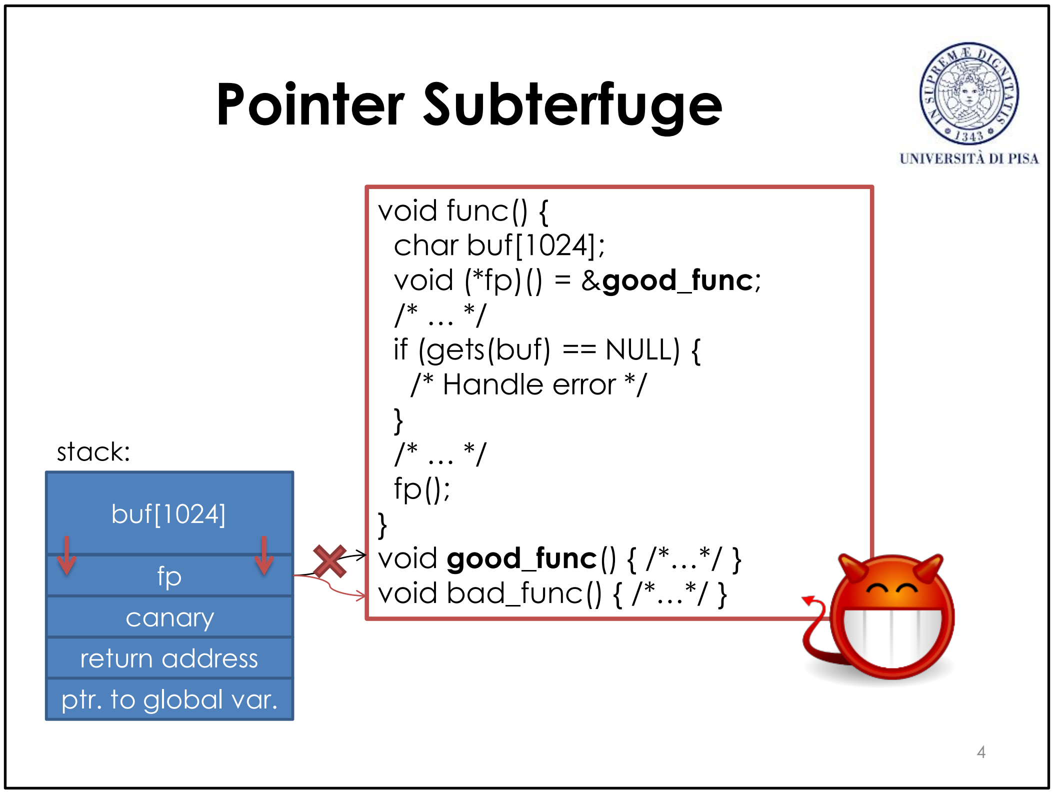
- These slides come from a version originally produced by Dr. Pericle Perazzo

2

C and C++ Secure Coding

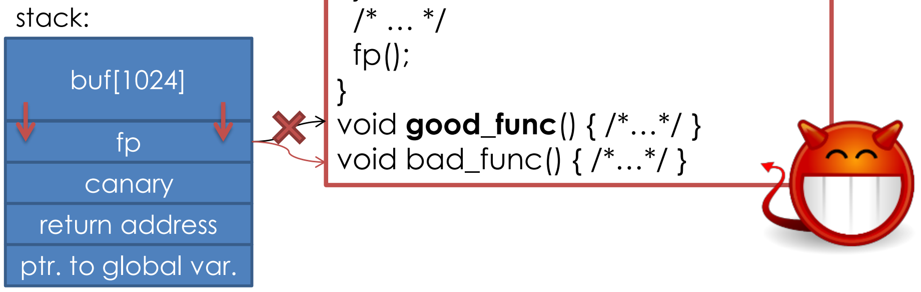
POINTER SUBTERFUGE

3



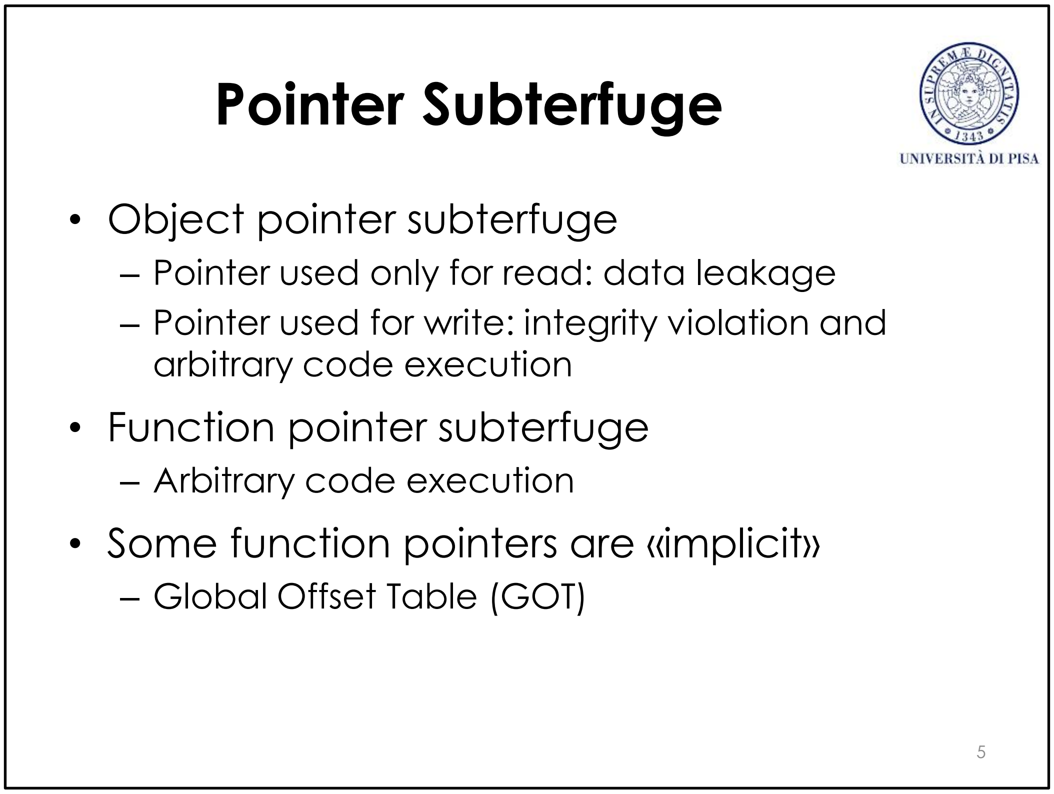
Pointer Subterfuge

```
void func() {
  char buf[1024];
  void (*fp)() = &good_func;
  /* … */
  if (gets(buf) == NULL) {
    /* Handle error */
  }
  /* … */
  fp();
}
void good_func() { /*…*/ }
void bad_func() { /*…*/ }
```



4

This example program is of course vulnerable to buffer overflow due to the gets() function. Due to this, the attacker is able to change the execution path by overwriting the "fp" variable to point to a different function than good_func(). This attack is an example of *pointer subterfuge*. A pointer subterfuge refers to a malicious change of a pointer before the victim process uses it. A pointer subterfuge attack may be allowed by many kinds of vulnerabilities, for example buffer overflows, errors in dynamic memory management, or format string vulnerabilities.

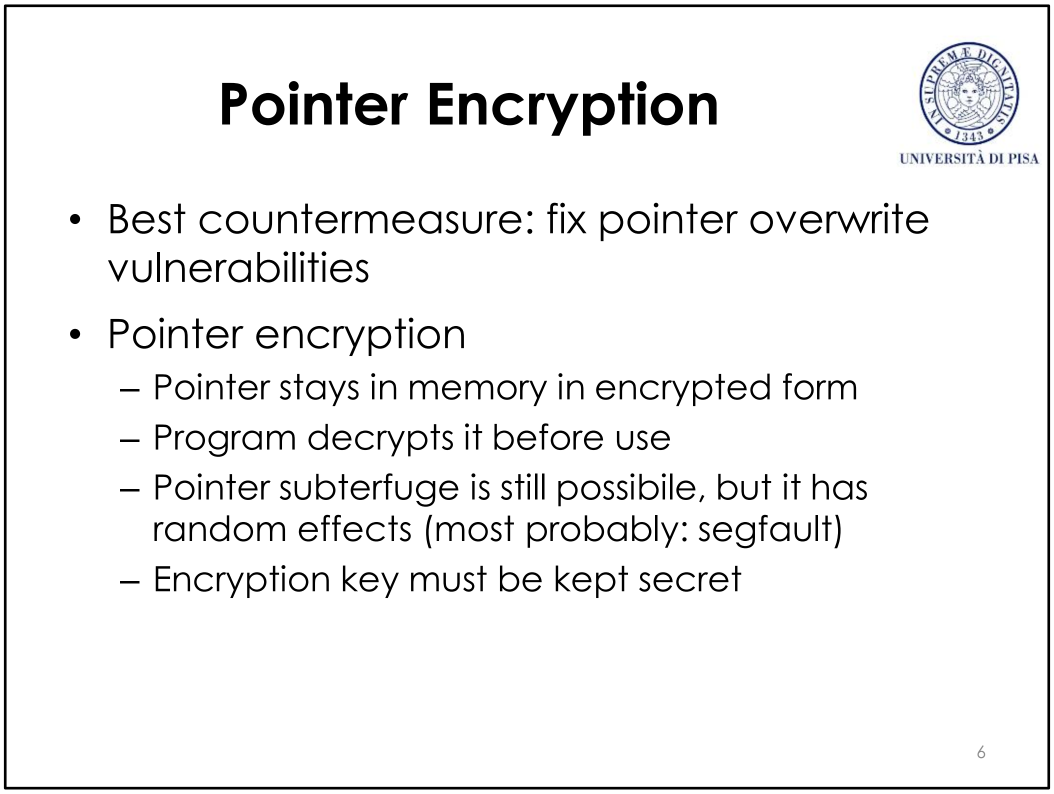


Pointer Subterfuge

- Object pointer subterfuge
  - Pointer used only for read: data leakage
  - Pointer used for write: integrity violation and arbitrary code execution
- Function pointer subterfuge
  - Arbitrary code execution
- Some function pointers are «implicit»
  - Global Offset Table (GOT)

5

In C and C++ there are two main types of pointers: *object pointers* and *function pointers*. An *object pointer subterfuge* can read or write on arbitrary memory locations, depending if the pointer is used to read or write the object. Writing on arbitrary memory location can in turn lead to arbitrary code execution. A function pointer subterfuge (like the above example) can also lead to arbitrary code execution. Note that even if a program's code does not explicitly use function pointers, some of them can still be defined and used to implement various mechanism. For example, the Global Offset Table (GOT) is a table of function pointers used to transfer the control to library functions in Linux and Windows systems. All these «implicit» function pointers are susceptible to pointer subterfuge.



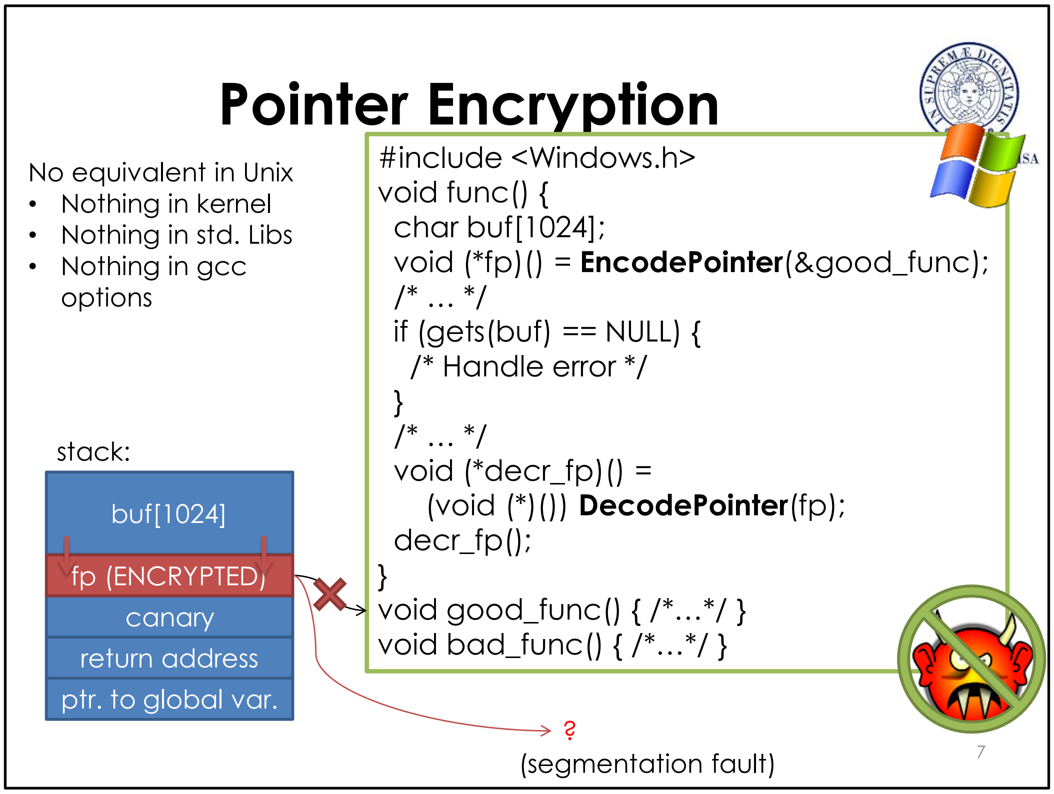
Pointer Encryption

UNIVERSITÀ DI PISA

- Best countermeasure: fix pointer overwrite vulnerabilities
- Pointer encryption
  - Pointer stays in memory in encrypted form
  - Program decrypts it before use
  - Pointer subterfuge is still possibile, but it has random effects (most probably: segfault)
  - Encryption key must be kept secret

6

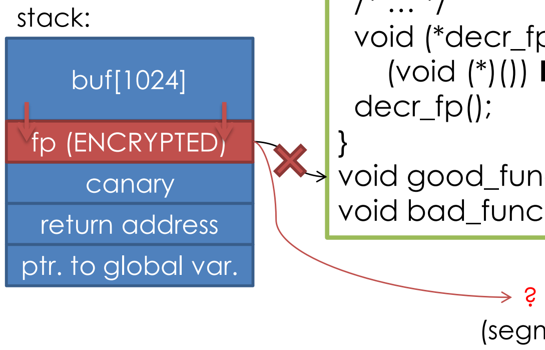
The best countermeasure against pointer subterfuge is of course to fix the vulnerabilities that allow pointer overwriting. As a further mitigation strategy, it is possible to store the pointers in memory in an *encrypted* form, and decrypt them just before using them. This does not avoid pointer overwriting, but the attacker would need to break the encryption to redirect the pointer to a *specific* address. Without breaking the encryption, the attacker could only redirect the pointer to a random address, resulting most probably in a segmentation fault, which is safer than arbitrary code execution. Of course, the encryption key should be kept secret from the attacker.



Pointer Encryption

No equivalent in Unix
- Nothing in kernel
- Nothing in std. Libs
- Nothing in gcc options

```
#include <Windows.h>
void func() {
  char buf[1024];
  void (*fp)() = EncodePointer(&good_func);
  /* … */
  if (gets(buf) == NULL) {
    /* Handle error */
  }
  /* … */
  void (*decr_fp)() =
      (void (*)()) DecodePointer(fp);
  decr_fp();
}
void good_func() { /*…*/ }
void bad_func() { /*…*/ }
```



Microsoft Windows provides two functions to perform pointer encryption: EncodePointer() and DecodePointer(). The encryption key is generated automatically by the kernel, and it is different process by process. To the time of writing (2020), we are not aware of a *nix kernel nor a standard library nor even an option of the gcc compiler that provide pointer encryption, so such a functionality must be developed by the programmer on *nix platforms.