CyberSecurity: Principle and Practice

BSc Degree in Computer Science
2020-2021
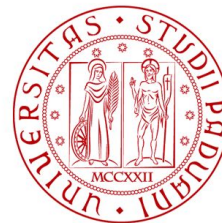
Lesson 16: PLT & GOT

Prof. Mauro Conti
Department of Mathematics
University of Padua
conti@math.unipd.it
http://www.math.unipd.it/~conti/

Teaching Assistants
Luca Pajola
pajola@math.unipd.it.
Pier Paolo Tricomi
pierpaolo.tricomi@phd.unipd.it

UNIVERSITÀ
DEGLI STUDI
DI PADOVA

SPRITZ
SECURITY & PRIVACY
RESEARCH GROUP

DIPARTIMENTO
MATEMATICA

1

Disclaimer

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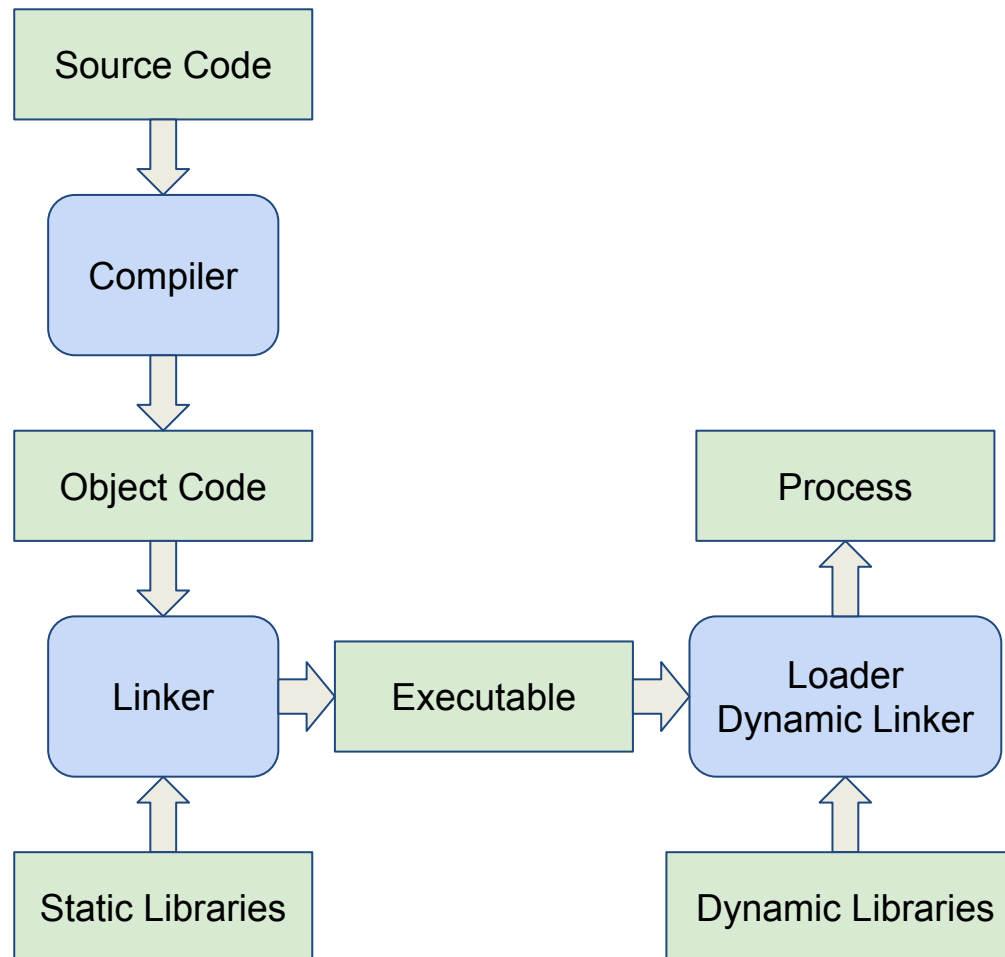
Do not hack the new Playstation. Or maybe do, but be prepared to get legal troubles 😊

ABUSING ELF DYNAMIC LINKING

With some vulnerabilities, you have the opportunity to write **arbitrary data** to **(almost) arbitrary memory addresses** (e.g., out-of-bounds array accesses).

One way to exploit this is by **abusing the internals of ELF dynamic linking**.

A PROGRAM'S LIFECYCLE
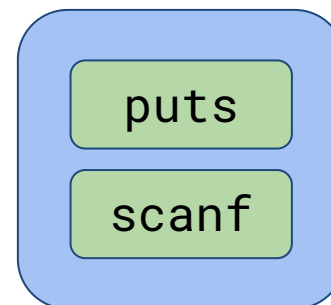
Global Offset Table (GOT)

Program imports (GOT)

puts = ?
scanf = ?
decode_image = ?
draw_pixels = ?

GOT enables compiled code (such as ELF) to run correctly
- **independently from actual memory addresses where loaded at runtime**

libc.so

puts

scanf

External Libraries

libgraphics.so
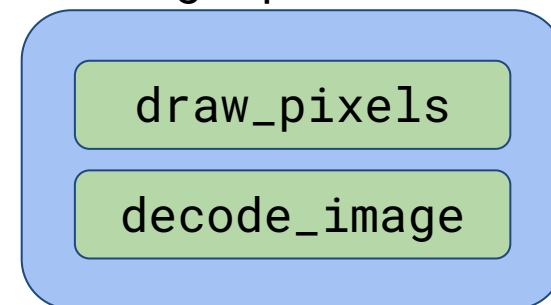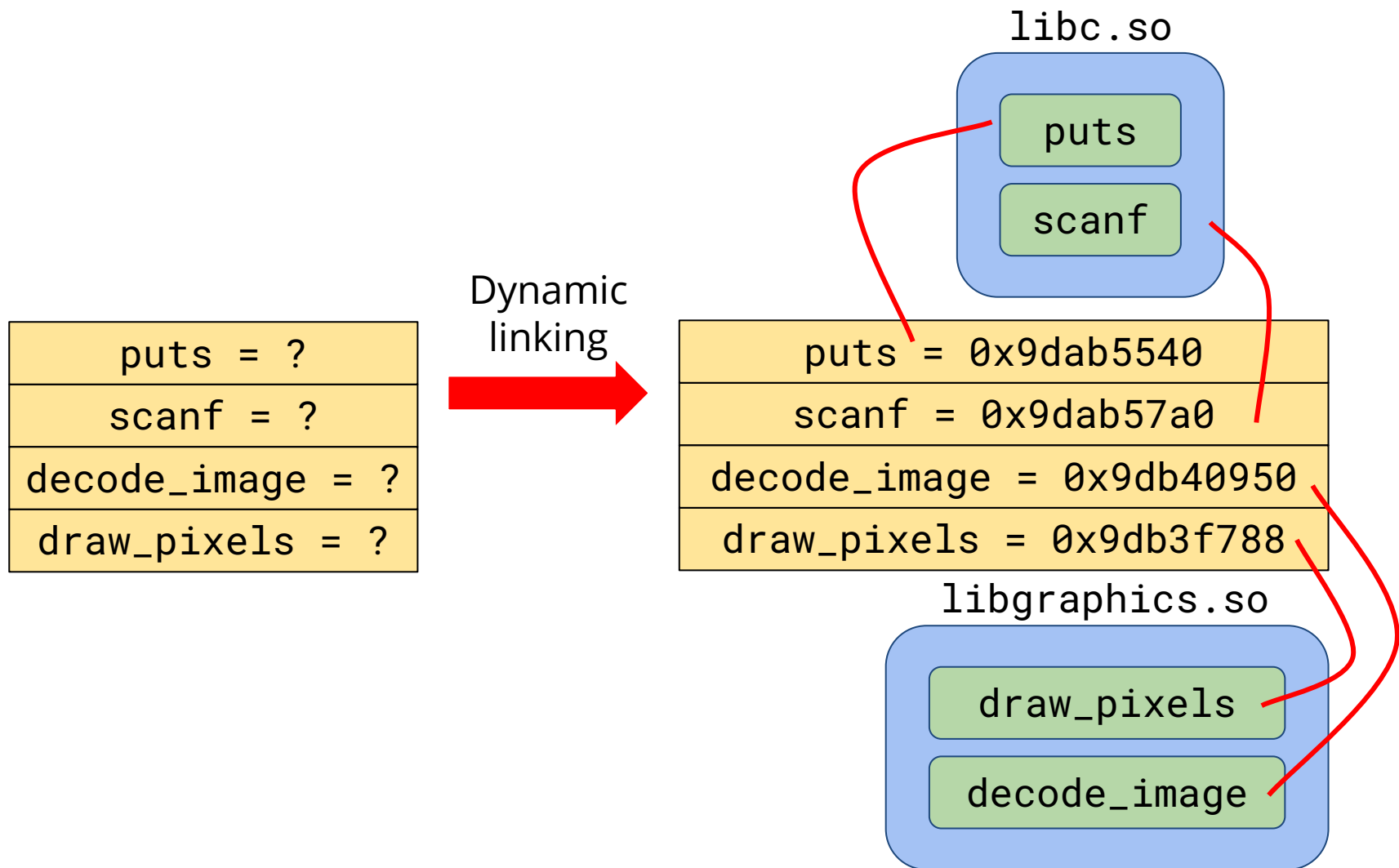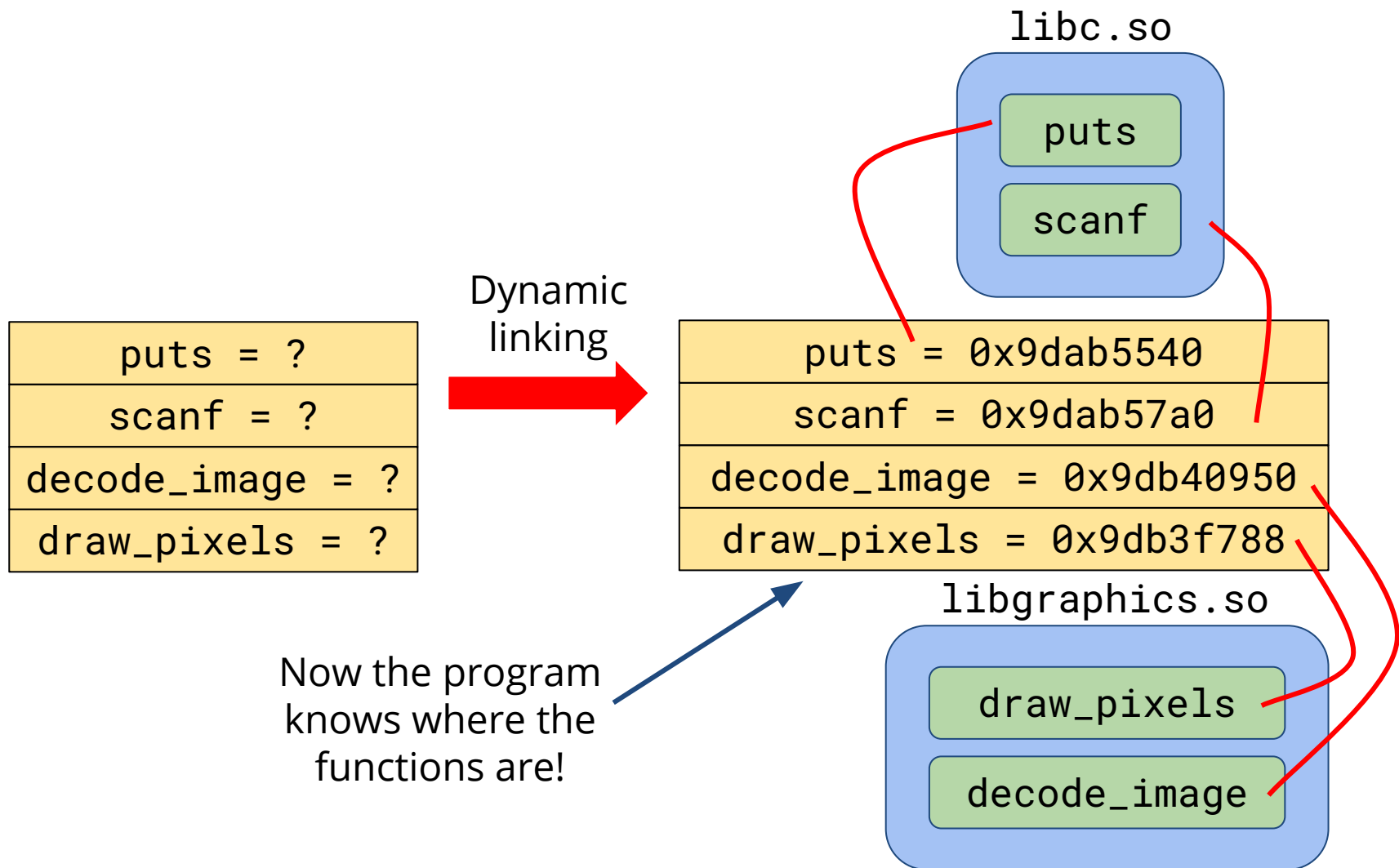
draw_pixels

decode_image

Global Offset Table (GOT)

libc.so

puts

scanf

Dynamic
linking

puts = ?	
scanf = ?	
decode_image = ?	
draw_pixels = ?	

puts = 0x9dab5540	
scanf = 0x9dab57a0	
decode_image = 0x9db40950	
draw_pixels = 0x9db3f788	

libgraphics.so

draw_pixels

decode_image

libc.so

puts

scanf

Dynamic
linking

| puts = ? |
| scanf = ? |
| decode_image = ? |
| draw_pixels = ? |

| puts = 0x9dab5540 |
| scanf = 0x9dab57a0 |
| decode_image = 0x9db40950 |
| draw_pixels = 0x9db3f788 |

libgraphics.so

draw_pixels

decode_image

Now the program
knows where the
functions are!

```
.got.plt:0000000000202000 ; =========================================================
.got.plt:0000000000202000
.got.plt:0000000000202000 ; Segment type: Pure data
.got.plt:0000000000202000 ; Segment permissions: Read/Write
.got.plt:0000000000202000 ; Segment alignment 'qword' can not be represented in assembly
.got.plt:0000000000202000 _got_plt          segment para public 'DATA' use64
.got.plt:0000000000202000                   assume cs:_got_plt
.got.plt:0000000000202000                   ;org 202000h
.got.plt:0000000000202000 _GLOBAL_OFFSET_TABLE_ dq offset _DYNAMIC
.got.plt:0000000000202008 qword_202008      dq 0                      ; DATA XREF: sub_8A0↑r
.got.plt:0000000000202010 qword_202010      dq 0                      ; DATA XREF: sub_8A0+6↑r
.got.plt:0000000000202018 off_202018        dq offset recv            ; DATA XREF: _recv↑r
.got.plt:0000000000202020 off_202020        dq offset _Z10uuid_parsePKcPh
.got.plt:0000000000202020                                             ; DATA XREF: uuid_parse(ch
.got.plt:0000000000202020                                             ; uuid_parse(char const*,u
.got.plt:0000000000202028 off_202028        dq offset write           ; DATA XREF: _write↑r
.got.plt:0000000000202030 off_202030        dq offset strlen          ; DATA XREF: _strlen↑r
.got.plt:0000000000202038 off_202038        dq offset __stack_chk_fail
.got.plt:0000000000202038                                             ; DATA XREF: ___stack_chk_
.got.pl                    f_202040        dq offset htons           ;
.got.pl                    f_202048        dq offset memset          ;
.got.pl                    f_202050        dq offset close           ;
.got.pl                    f_202058        dq offset memcpy          ;
.got.pl                    f_202060        dq offset inet_aton       ;                      n↑r
.got.plt:0000000000202068 off_202068        dq offset perror          ; DATA XREF: _perror↑r
.got.plt:0000000000202070 off_202070        dq offset strtoul         ; DATA XREF: _strtoul↑r
.got.plt:0000000000202078 off_202078        dq offset connect         ; DATA XREF: _connect↑r
.got.plt:0000000000202080 off_202080        dq offset isxdigit        ; DATA XREF: _isxdigit↑r
.got.plt:0000000000202088 off_202088        dq offset socket          ; DATA XREF: _socket↑r
.got.plt:0000000000202088 _got_plt          ends
```

Offset in the GOT to call write

Actual write function address

To reach the GOT table, we have another indirection, PLT

- Collection of *trampolines* (one for each import)

  - Program calls PLT entry

  - PLT entry `jmps` through the GOT

- Why this extra indirection?

  - Lazy linking

  - Non-PIE (Position-independent- executable)

    dynamically-linked programs

More: https://www.technovelty.org/linux/plt-and-got-the-key-to-code-sharing-and-dynamic-libraries.html

Procedure Linkage Table (PLT)


```
.plt:00000000000008D0
.plt:00000000000008D0 ; =============== S U B R O U T I N E ==============
.plt:00000000000008D0
.plt:00000000000008D0 ; Attributes: thunk
.plt:00000000000008D0
.plt:00000000000008D0 ; ssize_t write(int fd, const void *buf, size_t n)
.plt:00000000000008D0 _write          proc near               ; CODE XREF
.plt:00000000000008D0                                         ; AddUnit+1
.plt:00000000000008D0                 jmp     cs:off_202028
.plt:00000000000008D0 _write          endp
.plt:00000000000008D0
.plt:00000000000008D6 ; -----------------------------------------------------
.plt:00000000000008D6                 push    2
.plt:000                              jmp     sub_8A0
.plt:0000
.plt:0000
.plt:0000                      ======== S U B R O U T I N E ==============
.plt:0000
.plt:0000                      es: thunk
.plt:0000
.plt:00000000000008E0 ; size_t strlen(const char *s)
.plt:00000000000008E0 _strlen         proc near               ; CODE XREF
.plt:00000000000008E0                 jmp     cs:off_202030
.plt:00000000000008E0 _strlen         endp
.plt:00000000000008E0
.plt:00000000000008E6 ; -----------------------------------------------------
.plt:00000000000008E6                 push    3
.plt:00000000000008EB                 jmp     sub_8A0
```

Jump to corresponding GOT entry

1. **Overwrite a GOT entry** via some memory corruption

2. Make the program call the corresponding PLT function

    ○ Will be dispatched through the GOT

3. ?!?

4. Profit, got PC control :)

FUNCTION REUSE EXAMPLE

Let's assume we had a chance for GOT hijacking before this code:

```c
char buf[100];
scanf("%99s", buf);
puts(buf);
```

FUNCTION REUSE EXAMPLE

Let's assume we had a chance for GOT hijacking before this code:

```
char buf[100];
scanf("%99s", buf);
puts(buf);
```

What happens if we overwrite the **puts** GOT entry with the address of **system**?

FUNCTION REUSE EXAMPLE

Let's assume we had a chance for GOT
hijacking before this code:

```
char buf[100];
scanf("%99s", buf);
puts(buf);
```

What happens if we overwrite the `puts` GOT
entry with the address of `system`?

```
char buf[100];
scanf("%99s", buf);
system(buf);
```

Argument to system()
is attacker-controlled!

Lazy Linking

- Problem: finding symbols is slow

  - We're resolving all symbols **at startup**

  - Slow startup times => user is unhappy!

- Observation: most symbols aren't actually used

  - (in a specific execution)

- Solution: **lazy linking**

  - Delay symbol resolution until actually used

- What does this mean to an attacker?

  - Need to **corrupt a GOT entry for a function** that's **already been called at least once** before

  - Otherwise, the malicious entry will be overwritten by the linker when it lazily resolves the symbol

RELOCATION **R**EAD-**O**NLY (RELRO)

- **Full RELRO**: whole GOT is read-only

  + makes the **whole GOT read-only to avoid hijacking**

  - incompatible with lazy linking

- **Partial RELRO**: part of the GOT is readonly (part managing global variables---not our business here), but still functions have problems.

  + **compatible with lazy linking**

  - hijacking still possible

Exercises

1) Can you spawn a shell and get the flag?

2) If you mess some bytes around, you might print the flag :)

3) This is a position-independent binary which gives you a module address, and a trivial write-what-where. Can you spawn a shell?