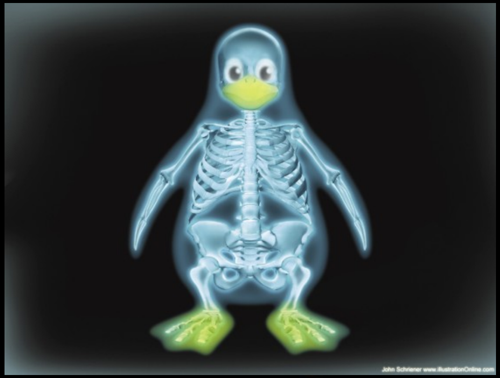
Exploiting the Linux Dynamic Loader with LD_PRELOAD

David Kaplan

david@2of1.org

DC9723 – June 2011

The Executable and linking format (ELF)



John Schriener www.IllustrationOnline.com

linkers

loaders

libraries

Linkers

combine compiled code fragments into single memory-loadable executable

```
$ ld obj1.o obj2.o –o linked.o
```

symbol resolution

program components reference each other through symbols (ELF .symtab)

Relocation

adjustment of code/data sections

(also performed by the loader)

Loaders

copy code and data into memory

memory allocation/mapping

relocation

Also performed by the linker

execve()

Libraries

collections of reusable compiled code

statically-linked

dynamically-linked (shared)*

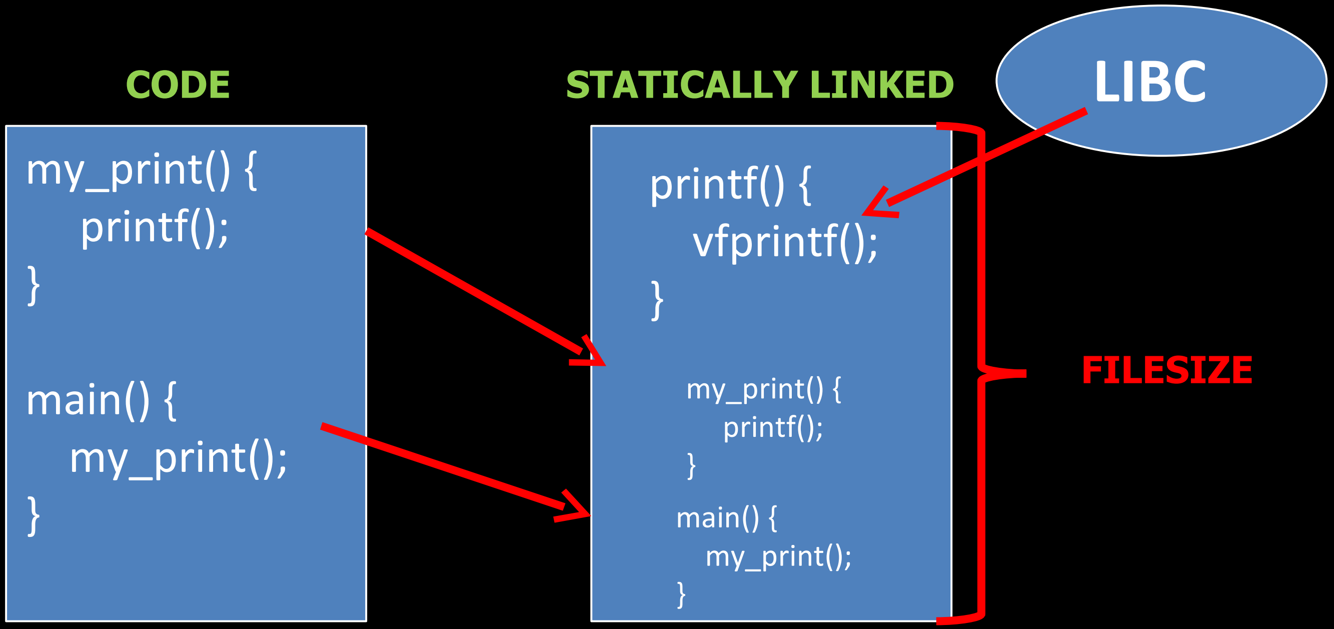
*historically: a shared library was something else entirely

Statically-linked libraries

code copied into final binary

be aware of: cyclic dependencies, multiple symbol definitions

```
$ld obj1.o obj2.o /usr/lib/libname.a
```



```
my_print() {
    printf();
}

main() {
    my_print();
}
```

```
printf() {
    vfprintf();
}

my_print() {
    printf();
}
main() {
    my_print();
}
```

Dynamically-linked libraries

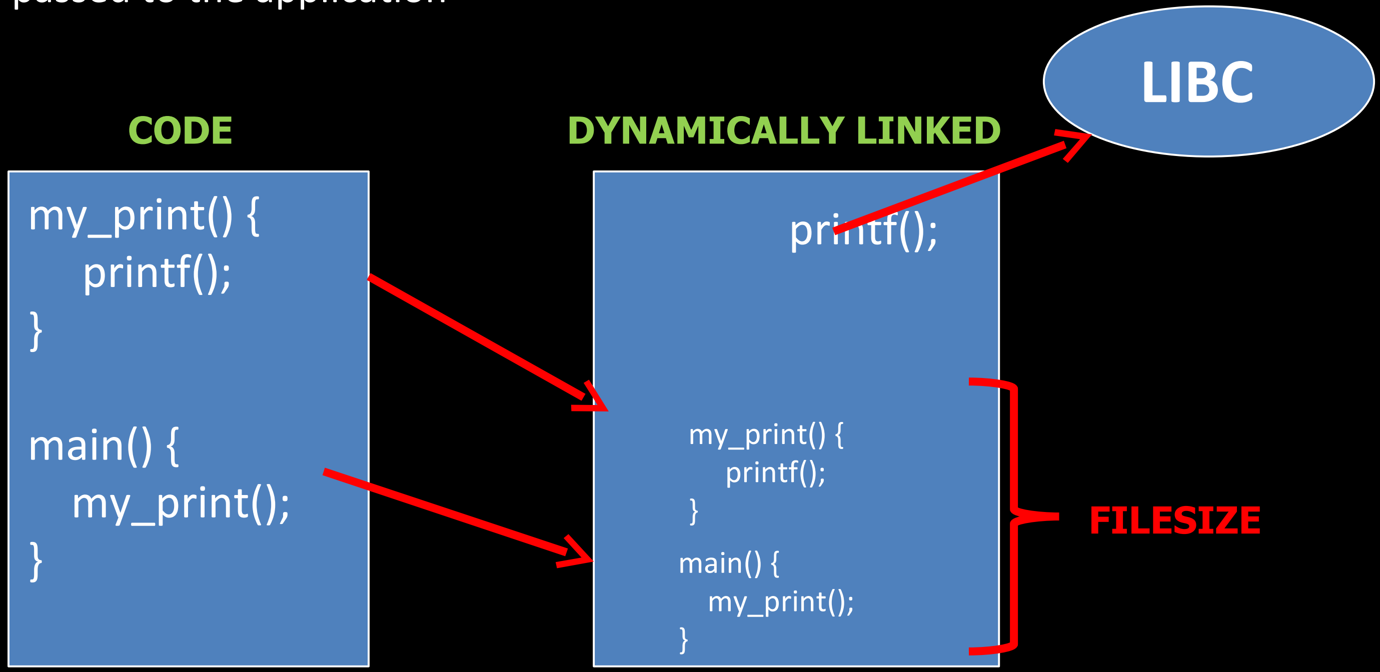
dynamic loader (ld.so) resolves symbols at exectime

can be called from within the application at runtime

By linking ld and calling dlopen(), etc.

cess:

- execve() loads executable code into memory
- control is passed to the dynamic linker (ld.so) which maps shared objects to program   address space (resolve
- control is then passed to the application



```
my_print() {
    printf();
}

main() {
    my_print();
}
```

```
printf();

my_print() {
    printf();
}
main() {
    my_print();
}
```

So what is LD_PRELOAD?

environment var queried by dynamic linker on exec

allows dynamic linker to prioritize linking defined shared libs

```
$ LD_PRELOAD=“./mylib.so” ./myexec
```

Attack enablers

OS ‘features’

weak system *security*

good coding *practices*

goto general_rule;

general_rule:

good_for_devs == good_for_hackers;

Attack advantages

easy, effective on unprotected systems

code interception

code injection

program flow manipulation

debugging using wrapper functions

Attack disadvantages

can be ***protected*** *against*

requires ***access*** *to executable*

requires relevant ***privileges***

works on ***used, imported symbols***

Example 1 – Hello World

```
$ nm -D hello
```

Undefined symbol

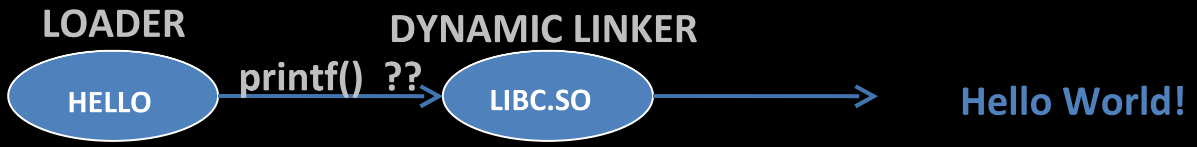
```
w __gmon_start__
U __libc_start_main
U printf
```

```
$ nm -D make_goodbye.so
000000000000069c T printf
                 U stdout
                 U vfprintf
```

Symbol exists in .text

Example 1 – Hello World – cont.

NORMAL SYMBOL RESOLUTION:



LOADER

DYNAMIC LINKER

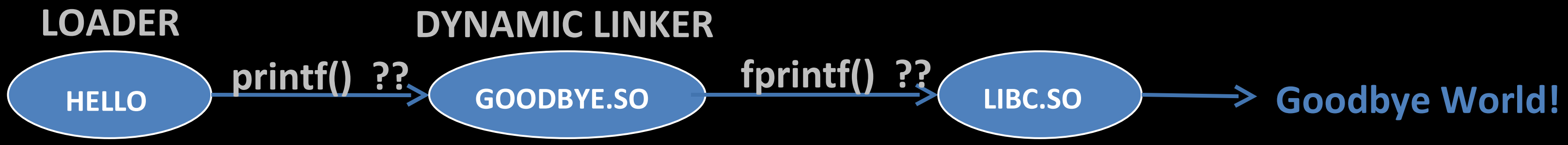
HELLO

printf() ??

LIBC.SO

Hello World!

REDIRECTED SYMBOL RESOLUTION:



LOADER

DYNAMIC LINKER

HELLO

printf() ??

GOODBYE.SO

fprintf() ??

LIBC.SO

Goodbye World!

*in practice it works slightly differently – this is just a conceptual explanation

Example 2 – OpenSSH MITM

dynamically links openssl

checks public key against known_hosts with BN_cmp()

BN_cmp() must pass (== 0) for iterations 3 and 5

Example 3 – OpenSSH password logger

catch write() w/ string literal "'s password"

log read()s until '\n'

Example 4 – Extending ‘cat’ functionalit

intercept __snprintf_check() to add to usage()

wrap getopt_long() to catch new command line option

catch write(), vfork() and launch browser for each link

./preloader

tool that does *some* of the work for you

provides reusable library of function sigs

reduces repetitive tasks

(sorry about the code quality!)

http://www.github.com/2of1/preloader

Further reading

Reverse Engineering with LD_PRELOAD *(Itzik Kotler)*

http://securityvulns.com/articles/reveng/

Linkers and Loaders *(Sandeep Grover)*

http://www.linuxjournal.com/article/6463

Dynamic Linker *(Wikipedia)*

http://en.wikipedia.org/wiki/Dynamic_linker

man ***ld.so***

Final thoughts

ur enemy and know yourself and you can fight a thousand battles without

Sun Wu Tzu, The Art of War

“There is no right and wrong. There’s only fun and boring”

The Plague, Hackers 1995