TU WIEN Informatics

Abfangen und Manipulieren von System-/Funktionsaufrufen in Linux-Systemen

BACHELORARBEIT

zur Erlangung des akademischen Grades

Bachelor of Science

im Rahmen des Studiums

Software & Information Engineering

eingereicht von

Lorenz Stechauner
Matrikelnummer 12119052

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Peter Puschner
Mitwirkung: Univ.Ass. Dipl.-Ing. Florian Mihola, BSc

Wien, 1. Juni 2025

Lorenz Stechauner	Peter Puschner

TU WIEN Informatics

Intercepting and Manipulating System/Function Calls in Linux Systems

BACHELOR'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Bachelor of Science

in

Software & Information Engineering

by

Lorenz Stechauner
Registration Number 12119052

to the Faculty of Informatics

at the TU Wien

Advisor: Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Peter Puschner
Assistance: Univ.Ass. Dipl.-Ing. Florian Mihola, BSc

Vienna, June 1, 2025

Lorenz Stechauner	Peter Puschner

Erklärung zur Verfassung der Arbeit

Lorenz Stechauner

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.
Ich erkläre weiters, dass ich mich generativer KI-Tools lediglich als Hilfsmittel bedient habe und in der vorliegenden Arbeit mein gestalterischer Einfluss überwiegt. Im Anhang „Übersicht verwendeter Hilfsmittel“ habe ich alle generativen KI-Tools gelistet, die verwendet wurden, und angegeben, wo und wie sie verwendet wurden. Für Textpassagen, die ohne substantielle Änderungen übernommen wurden, haben ich jeweils die von mir formulierten Eingaben (Prompts) und die verwendete IT-Anwendung mit ihrem Produktnamen und Versionsnummer/Datum angegeben.

Wien, 1. Juni 2025

Lorenz Stechauner

Kurzfassung

Lorem Ipsum.

Abstract

Lorem Ipsum.

Contents

CHAPTER 1

Introduction

Lorem Ipsum.

1.1 TODO: Why intercept?

Lorem Ipsum.

1.2 TODO: Why are current solutions not enough?

Lorem Ipsum.

1.3 TODO: Linux/C/ELF call structure

Lorem Ipsum.

1.4 TODO: System Calls vs. Function Calls

Lorem Ipsum.

CHAPTER 2

Intercepting Function Calls

Lorem Ipsum.

2.1 Identified Methods for Intercepting Function and System Calls

Lorem Ipsum.

2.1.1 Preloading using `LD_PRELOAD`

To execute binary files on Linux systems, a dynamic linker is needed at runtime. (Unless the binaries were statically linked at compile-time.) Usually, `ld.so` and `ld-linux.so` are used as dynamic linkers. They find and load the shared objects (shared libraries) needed by a program, prepare the program and finally run it. [4]

As the overwhelming majority of programs are dynamically linked, most function calls to other libraries (like to the C standard library) reference a shared object, which has to be loaded by the linker at runtime. Therefore, it would be possible to "hijack" (or intercept) these function calls, when the linker would allow loading other functions instead of the proper ones.

Luckily, `ld.so` allows this so-called "preloading". See the ENVIRONMENT section in the ld.so(8) Linux manual page [4]:

> **`LD_PRELOAD`** A list of additional, user-specified, ELF shared objects to be loaded before all others. This feature can be used to selectively override functions in other shared objects. [...]

This means, by setting the environment variable `LD_PRELOAD`, it is possible to override specific functions. The listings 2.1 and 2.2 try to illustrate this.

```c
#include <stdlib.h>
#include <dlfcn.h>
#include <errno.h>

void *malloc(size_t size) {
    // before call to malloc
    void *(*_malloc)(size_t);
    if ((_malloc = dlsym(RTLD_NEXT, "malloc")) == NULL) {
        errno = ENOSYS;
        return NULL;
    }
    void *ret = _malloc(size);
    // after call to malloc
    return ret;
}
```

Listing 2.1: Contents of `preload.c`.

```
# ./main is already compiled and ready
gcc -shared -fPIC -o preload.so preload.c
LD_PRELOAD="$(pwd)/preload.so" ./main
```

Listing 2.2: Compile `preload.so` and run a program with `LD_PRELOAD`.

The function `dlsym` is used to retrieve the original address of the `malloc` function. `RTLD_NEXT` indicates to find the next occurrence of `malloc` in the search order after the current object. [1]

Using this method, it is possible to override, and therefore wrap, any function as long as the targeted binary was not statically linked. Although, one has to be aware that not only function calls inside the targeted binary, but also calls inside other libraries (e.g., to `malloc`) are redirected to the overriding function.

2.1.2 Wrapper Functions in `gcc`

From the OPTIONS section in the ld(1) Linux manual page [3]:

> **`-wrap=`*`symbol`** Use a wrapper function for *`symbol`*. Any undefined reference to *`symbol`* will be resolved to `__wrap_`*`symbol`*. Any undefined reference to `__real_`*`symbol`* will be resolved to *`symbol`*.

> This can be used to provide a wrapper for a system function. The wrapper function should be called `__wrap_`*`symbol`*. If it wishes to call the system function, it should call `__real_`*`symbol`*. [...]

From the OPTIONS section in the gcc(1) Linux manual page [2]:

> **`-Wl,`*`option`*** Pass *`option`* as an option to the linker. If *`option`* contains commas, it is split into multiple options at the commas. You can use this syntax to pass an argument to the option. For example, `-Wl,-Map,output.map` passes `-Map output.map` to the linker. When using the GNU linker, you can also get the same effect with `-Wl,-Map=output.map`. [...]

2.1.3 Kernel Module

Lorem Ipsum.

2.1.4 Emulation

Lorem Ipsum.

2.1.5 Modifying the Kernel

Lorem Ipsum.

2.1.6 Conclusion

Lorem Ipsum.

2.2 Combining Preloading and Wrapper Functions

Lorem Ipsum.

2.3 Retrieving Function Argument Values

Lorem Ipsum.

2.4 Determining Function Call Location

Lorem Ipsum.

2.5 Example

Lorem Ipsum.

2.6 Analyzing Intercepted Function Calls

Lorem Ipsum.

2.7 Parsing Intercepted Function Calls in Python

Lorem Ipsum.

2.8 Automated Testing on Intercepted Function Calls

Lorem Ipsum.

CHAPTER 3

Manipulating Function Calls

Lorem Ipsum.

Unix-Sockets, TCP-Sockets, ...

3.1 Defining a Protocol

Lorem Ipsum.

3.2 Parsing Responses

Lorem Ipsum.

3.3 Creating a Socket Server in Python

Lorem Ipsum.

3.4 Automated Testing using Function Call Manipulation

Lorem Ipsum.

CHAPTER 4

Related Work

Lorem Ipsum.

What other solutions are available? What are the differences? What are the characteristics?

CHAPTER 5

Conclusion

Lorem Ipsum.

Perhaps do some study/“research” on performance (CPU/memory/...).

Overview of Gen. AI Tools Used

Enter your text here.

List of Figures

List of Tables

List of Algorithms

Bibliography

[1] *dlsym(3) – Library Functions Manual – Linux manual pages.*

[2] *GCC(1) – GNU – Linux manual pages.*

[3] *ld(1) – GNU Development Tools – Linux manual pages.*

[4] *ld.so(8) – System Manager's Manual – Linux manual pages.*