Charles University in Prague

Faculty of Mathematics and Physics

BACHELOR THESIS

Daniel Lovásko

Multiplatform CTF implementation and use in UNIX-like operating systems

Department of Applied Mathematics

Supervisor of the bachelor thesis:	Mgr. Petr Baudiš
Study programme:	Informatics
Specialization:	Programming

Prague 2015

Dedication.

Název práce: Název práce

Autor: Jméno a příjmení autora

Katedra: Název katedry či ústavu, kde byla práce oficiálně zadána

Vedoucí bakalářské práce: Jméno a příjmení s tituly, pracoviště

Abstrakt:

Klíčová slova:

Title:

Author: Jméno a příjmení autora

Department: Název katedry či ústavu, kde byla práce oficiálně zadána

Supervisor: Jméno a příjmení s tituly, pracoviště

Abstract:

Keywords:

Contents

Introduction

1. Compact C Type Format

1.1 History and motivation

1.1.1 Contemporary usage

1.2 Format description

1.2.1 ELF section

1.2.2 Header

1.2.3 Labels and uniquification

1.2.4 Function objects

1.2.5 Data objects

1.2.6 Types

1.2.7 Variable data

Integer number

Floating point number

Enumeration

Structure and union

Function

1.2.8 Strings

2. libctf

2.1 API

2.2 Implementation

2.3 Infrastructure

2.3.1 Version control

2.3.2 Build system

2.3.3 Documentation

2.4 Licence

2.5 Performance

2.6 Memory usage

2.6.1 Memory leak prevention

2.7 Correctness

2.7.1 Unit tests

2.7.2 Code coverage

3. Toolset

While `libctf` is a suitable tool for a low-level access and manipulation of the CTF data set, praxis shows certain tasks to be reoccurring and generalizable. Such procedures are implemented as stand-alone binaries performing set objective. With the UNIX philosophy in mind [referencia!], each program is focusing on a single, discrete task.

3.1 ctfdump

The most basic of these tools is the `ctfdump` program. Its sole purpose is to print the CTF data set to the standard output. The program supports many options to further adjust the printed information:

An important feature of the program is the ability to output the data set in the JSON, XML and HTML formats [referencia!] by employing the `libxo` provided by Juniper Networks that used in many base utilities. This enables developers to structurally handle the output and process it without using complicated regular expressions.

3.2 ctfquery

When a programmers needs to access a specific declaration of a `struct` or to solve a `typedef` chain, they can examine the source code of the application. In certain cases the source is not available or there is no guarantee that the source will map with full accuracy to the binary. `ctfquery` tries to solve this issues by providing a way to query the binary for type definitions. Second usage of the utility is to obtain the type associated with a symbol stored in the symbol table.

3.3 ctfstats

3.4 ctfconvert

To overcome the lack of compiler support with respect to generation of the CTF data, `ctfconvert` was created. The source format, DWARF, is a debugging standard provided by all major compilers - such as GCC or LLVM/Clang - and is consumed by the `ctfconvert` to produce the CTF output.

Pouzitie pri buildovani systemu

3.5 ctfmerge

As the name suggests, `ctfmerge` performs the merging of two CTF data sets using the uniquification algorithm described in the section 1.3.

3.6 ctfdiff

3.7 ctfmemusage

4. Pretty-printing kernel data structures

Memory correctness plays the key role in any algorithm. Being able to swiftly and efficiently examine arbitrarily complex data structures is a mandatory trait of every modern toolchain. The GNU Debugger `gdb`, Modular Debugger `mdb` and the LLVM Debugger `lldb` fulfil the expectations with respect to userland processes. Unfortunately, the kernel debugger DDB on FreeBSD (and other systems using it) has been shipped without such functionality from its inception many decades ago. We aim to solve this issue by creating a mechanism to pretty-print data structures contained in the currently loaded kernel image.

4.1 Approach comparison

Currently, DDB offers two techniques to access the memory which are affected by many limitations, either lack of modularity or excessive straightforwardness.

4.1.1 Low-level examination

The `examine` command [referencia!] in conjecture with its options such as `x` or `f` is used to examine optional number of memory blocks. The fact that the user must specify the encoding of the memory is a .

This feature of DDB is undoubtedly important in scenarios when a specific bit or byte needs to be analysed, but falls short in examining complex structures, as it literally forces user to use pen and paper to map the output values to the data structure.

4.1.2 Hard-coded structures

The `show` command [referencia!] improves upon the simplicity of th `examine` command by providing support for pretty-printing a restricted set of data structures. Such structures are:

- bio
- mutex

The problem with this approach lies in its *code change linearity*: every time a data structures member changes, the corresponding code in DDB needs to adapt. The same applies in situations when a new data structure is introduced - a new pretty-printing code needs to be written.

4.1.3 Universal CTF solution

A

4.2 Kernel-space adaptation

4.3 Type lookup

4.4 Pretty printing

4.5 Data structure speculation

4.6 Views

4.7 Corrupted structures

5. Type-aware kernel virtual memory access

5.1 Problem

5.2 CTF inspection

5.3 Solution

5.4 Usage in base utilities

5.4.1 Obsoleting kernel wrapper structures

5.4.2 procstat

6. Potential applications

6.1 Compilers

6.2 System-wide availability

6.3 Automatic binding generator

6.4 Core dump examination

6.5 C++ classes

Conclusion

Bibliography

List of Tables

List of Abbreviations

Attachments