Quarkslab

GreHack

New is *still not* always better.

ANSSI

# Quokka

## A Fast and Accurate Binary Exporter

Alexis Challande, Robin David, Guénaël Renault

# Biography

## whoami

- Alexis Challande
- Security Engineer at Quarkslab
- 🎓 Doctor since Oct'22

## Quarkslab

- **QLab**: about 50 engineers
- Offensive and Defensive Research
- Looking for **interns**

https://blog.quarkslab.com/internship-offers-for-the-2022-2023-season.html

# Introduction

# (My) Context

## Research Problem

How to automate the search for **security patches** in binary code?

# (My) Context

## Research Problem

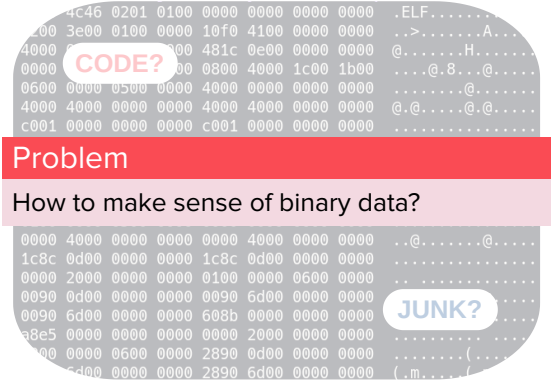
How to automate the search for **security patches** in binary code?

We need solution to improve static analysis of disassembled binaries

# Binary Analysis


Analyzing binaries is the **only** solution when the source code is **unavailable**

# Binary Analysis




**Problem**

How to make sense of binary data?

Analyzing binaries is the **only** solution when the source code is **unavailable**

# Disassemblers

*Translate binary blob into meaningful data*

Disassembling is **hard**[1] as compilation is a one step process

- Distinguish code from data
- Uncover the control flow
- Explore references
- Find function boundaries
- ...

[1] Meng, Xiaozhu, and Barton P. Miller. "Binary code is not easy." Proceedings of the 25th International Symposium on Software Testing and Analysis. 2016.

# Disassemblers

*Translate binary blob into meaningful data*

Disassembling is **hard**[1] as compilation is a one step process

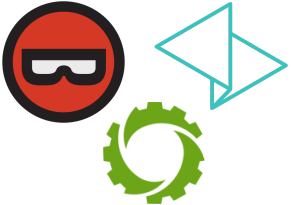
- Distinguish code from data
- Uncover the control flow
- Explore referenc
- Find function bou
- ...

Reversers use specialized **tools**

[1] Meng, Xiaozhu, and Barton P. Miller. "Binary code is not easy." Proceedings of the 25th International Symposium on Software Testing and Analysis. 2016.

# Disassemblers

## Commercial Tools



- Professionnal support
- Expensive licenses

Example: IDA, Binary Ninja, JEB

## Open Source Tools

- Free to use
- Extensible

Example: Ghidra, radare2, Rizin

# IDA Pro

*The disassembler from Hex-Rays*

## IDA in a nutshell

- Created in 1990 *thirty years old*
- Supports 60 processors/architectures
- Very **expensive**

There are many competitors but ...
**IDA** is still the best disassembler in town

# Automating Analysis

*Also the name of my team at Quarkslab*

IDA *and other disassemblers* offer an API to script their usage

## IDC

- Pseudo-C scripting language
- ✗ Deprecated

## C++ SDK

- Full SDK for IDA functions
- ✗ Poor documentation
- ✓ **Best for performance sensitives plugins**

## IDAPython

*Started as a plugin*

- Python bindings to the C++ API
- ✓ Easy to prototype
- ✓ **Best for a quick scripting usage**

# Scripting & Limits

Scripting enables to automate many **repetitive tasks** within a single binary

but

requires to run within **IDA**

# Scripting & Limits

Scripting enables to automate many **repetitive tasks** within a single binary

but

requires to run within **IDA**

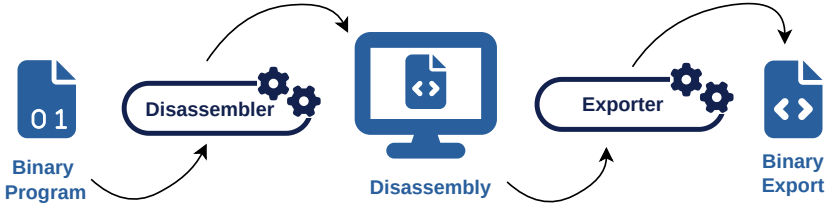
How to **programatically** manipulate the disassembly?

# Binary Exporters

# Binary Exporters

## Definition

A **binary export** is a standalone file containing data from the disassembled binary



# Why using exports?

| | With IDA | With an export |
|---|---|---|
| Query the disassembly | ✓ | ✓ |
| Analyze once, use anywhere | ✗ | ✓ |
| Side by side analysis | ✗ | ✓ |
| Reusable scripts | ✗ | ✓ |
| Online interactivity | ✓ | ✗ |

# State of the Art

| Disassembler | Exporter | |
|---|---|---|
| IDA | BinExport | Exporter from Zynamics |
| | Ghidra-IDA | Export a project from `IDA` to Ghidra |
| | McSema | Exporter for McSema lifter |
| Ghidra | BinExport | *BinExport port for Ghidra* |
| | Ghidra | Built-in exporter from Ghidra |

# State of the Art

| Disassembler | Exporter | |
|---|---|---|
| IDA | BinExport | Exporter from Zynamics |
| | Ghidra IDA | Export a project from IDA to Ghidra |
| Ghidra | | |

**Limits**

- Missing bindings
- Limited exported data
- Not compact

# Disk Usage

*Why compactness matters?*

## Use Case

We had a binary dataset composed of **2200** libraries of OpenSSL

- IDA disassembly took about 6 hours
- And the generation of the **IDB** for each binary took **25 GB** of storage *from 1.5 GB*

But this was only an excerpt of the dataset...

What happens when we analyze the whole dataset? *15 GB*

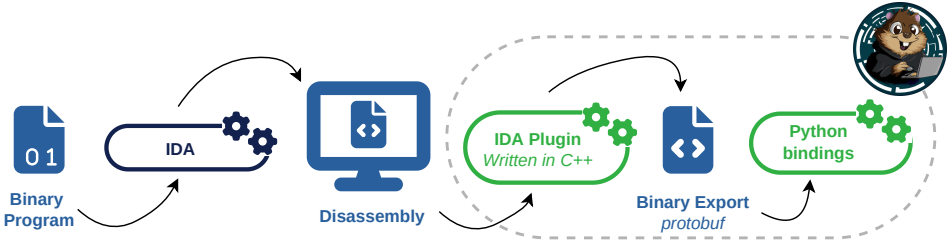
# Quokka

**Quokka** is a **fast** and **accurate** binary exporter

- Exhaustive
  *exports many data*
- Efficient
  *export time is negligeable*
- Compact
  *disk usage reduced to minimum*

Open Source and available on
`https://github.com/quarkslab/quokka`

# Quokka Overview



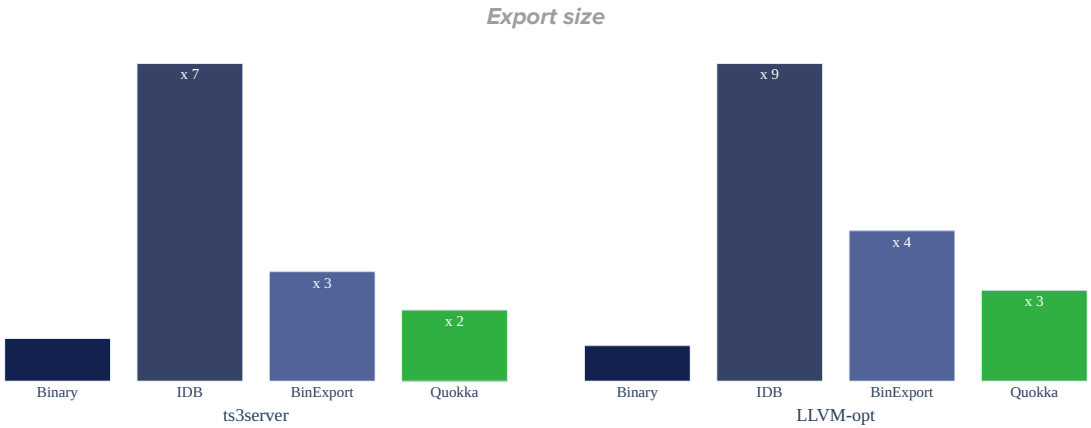

**Quokka** exports a binary disassembly to a Protobuf file and expose an API to manipulate the export

# Exported Items

**Quokka** aims to export **everything**

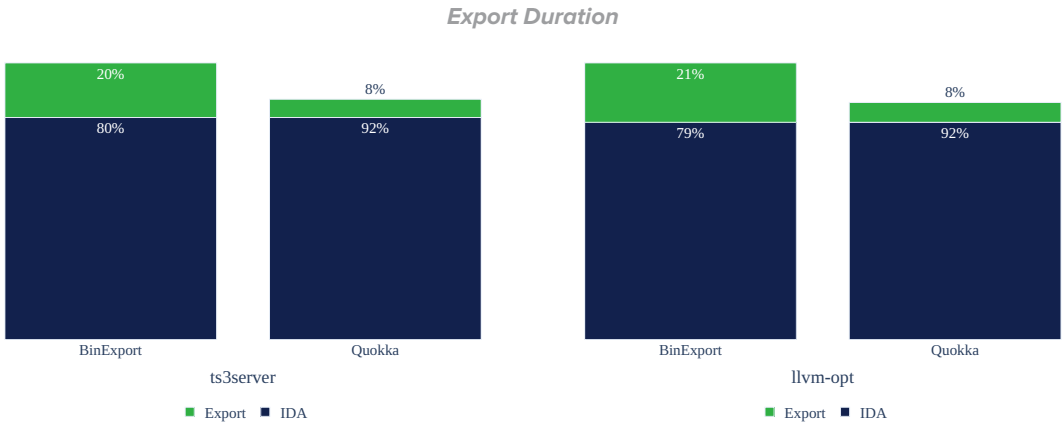
- Program Layout *Segments, Code Layout, ...*
- Data *Address, Type, Size, ...*
- Graphs *CG, CFG*
- Code *Functions, Instructions, Basic Blocks*
- Analysis *Comments, Structures, Enums*
- ...

# Performances




**Quokka** is **compact**

# Performances




**Quokka** is **fast**

# Exporting A Binary

*Requires a working IDA installation*

Using the command line:

```
$ idat64 -OQuokkaAuto:true -A qb-crackme
```

Using the Python bindings

```
import quokka
prog = quokka.Program.from_binary("qb-crackme")
```

Exporting is a **one step** process and the **only** one requiring a disassembler

# Manipulating the exported file

*No disassembler required anymore*

```
import quokka

program = quokka.Program("qb-crackme.quokka", "qb-crackme")
for func in program:
    print(f"Function {func.name} at 0x{func.start:x}")
    for block_start in func.graph.nodes:
        block = func.get_block(block_start)
        print(f"\tBlock at 0x{block_start:x} with {len(block)}
        ↪  instructions")
```

# Demo 1: Feature Extraction

# Introduction

## Context

New **shiny** neural network model to perform *<insert task name >*[2]

Problem

How to extract features from every binary in your dataset?

[2] vulnerable code identification, binary diffing, ...

# Strategy

**Without Quokka**:

Write custom IDA scripts ...
with their API ...
and maybe start to cry

# Strategy

**Without Quokka**:

Write custom IDA scripts ...
with their API ...
and maybe start to cry

**With Quokka**:

Write a simple Python script using Quokka's API!

# Example

## How ML Is Solving the Binary Function Similarity Problem[2]

- Compare the implementation of multiple approaches
- Implement the **feature extraction** using IDA scripts[4]

Let's walk through one of their plugin and how to replicate it with **Quokka**

[3] Andrea Marcelli, Mariano Graziano, Xabier Ugarte-Pedrero, Yanick Fratantonio, Mohamad Mansouri, Davide Balzarotti. How Machine Learning Is Solving the Binary Function Similarity Problem. USENIX Security '22.

[4] `https://github.com/Cisco-Talos/binary_function_similarity/blob/main/IDA_scripts/IDA_acfg_features/IDA_acfg_features.py`

# Demo 2: Bionic

# Introduction

## Context

Bionic is the libc implementation of Android *by Google*
There are some **key** differences between the two

One is notably the `/etc/passwd` file is embedded within the binary

Task:

How to recover the user UID mapping from the binary?[5]

[5] Idea from Robin David

# Strategy

1. Identify a function using the user table
2. Find the **data reference** to the table
3. Identify table boundaries
4. Read one entry
5. Repeat

# Strategy

1. **Identify a function using the user table**
   Solution: **getpwuid**

```
The getpwuid() function returns a pointer to a struc-
ture containing the broken-out fields of the record
in the password database that matches the user ID
uid.
```

# Strategy

1. Identify a function using the user table
   Solution: **getpwuid**

2. **Find the data reference to the table**
   Solution: **found**

```
EXPORT getpwuid
getpwuid
; __unwind {
PUSH            {R4,R5,R7,LR}
MOV             R4, R0
MRC             R0, TPIDRURO
LDR             R1, =(_ZL11android_ids - 0x1CFEC) ; android_ids
MOVW            R2, #0x26B8
LDR             R0, [R0,#4]
ADD             R1, PC  ; android_ids
LDR.W           R0, [R0,#0x684]
ADDS            R5, R0, R2
MOVS            R0, #0
```

# Strategy

1. Identify a function using the user table
   Solution: **getpwuid**

2. Find the data reference to the table
   Solution: **found**

3. **Identify table boundaries**
   Solution: Use an heuristic

# Strategy

1. Identify a function using the user table
   Solution: **getpwuid**

2. Find the data reference to the table
   Solution: **found**

3. Identify table boundaries
   Solution: Use an heuristic

4. **Read one entry**
   Solution: Read a **string** for the username and a **DWORD** for the UID

```
struct android_id_info {
    const char name[22];
    unsigned aid;
};
```

# Strategy

1. Identify a function using the user table
   Solution: **getpwuid**

2. Find the data reference to the table
   Solution: **found**

3. Identify table boundaries
   Solution: Use an heuristic

4. Read one entry
   Solution: Read a **string** for the username and a **DWORD** for the UID

5. **Repeat**

# Conclusion

# Going further

*What to do next?*

## First steps:

- Export types information
- References handling improvments
- Meaningful instructions and operands export
- Primitives to write back
- ...

# Going further

*What to do next?*

## First steps:

- Export types information
- References handling improvments
- Meaningful instructions and operands export
- Primitives to write back
- ...

## To the moon™:

- Support for other disassemblers
- Export decompiled information
- Integration with other tools
- ...

# Help is welcome!

# Final words

- **Quokka** is a Binary Exporter
- Works as an IDA plugin *to generate the export*
- With Python Bindings

`https://github.com/quarkslab/quokka`

`https://quarkslab.github.io/quokka`

@DarkaMaul

Quarkslab