Streamlining Firmware Analysis with Inter-Image Call Graphs and Decompilation

Robin David <rdavid@quarkslab.com>

Quarkslab

WHOAMI

Whoami:

- Security Research @ Quarkslab *(since 2017)*
- Trainer
- R&D Lead / Manager

Research Topics:

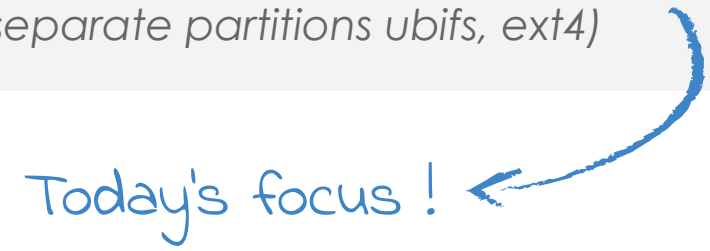
- Obfuscation / Deobfuscation
- Fuzzing / Symbolic Execution
- Graph based ML
- Firmware Analysis

RE//verse

Firmware Analysis

What ?

- "Flat" firmware: self-contained firmware within a memory segment *(low-level firmwares)*
- "Structured" firmware: made of a kernel and a filesystem *(usually using separate partitions ubifs, ext4)*



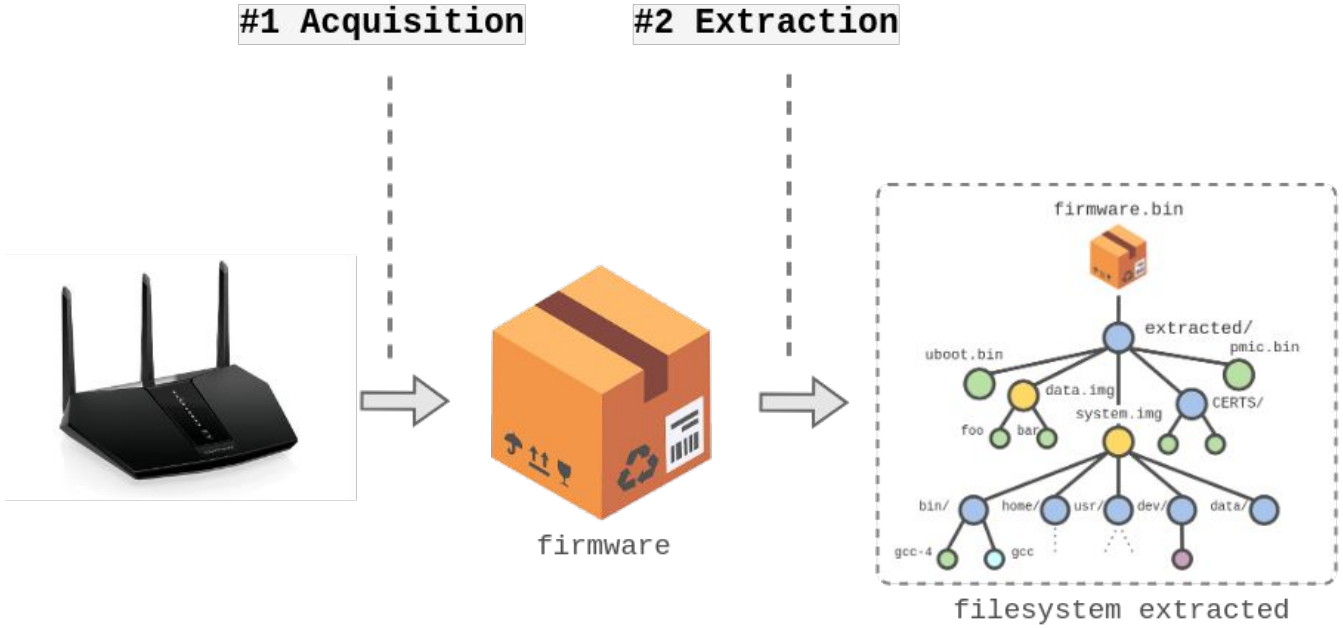
Why ?

⇒ For an **efficient** firmware recon

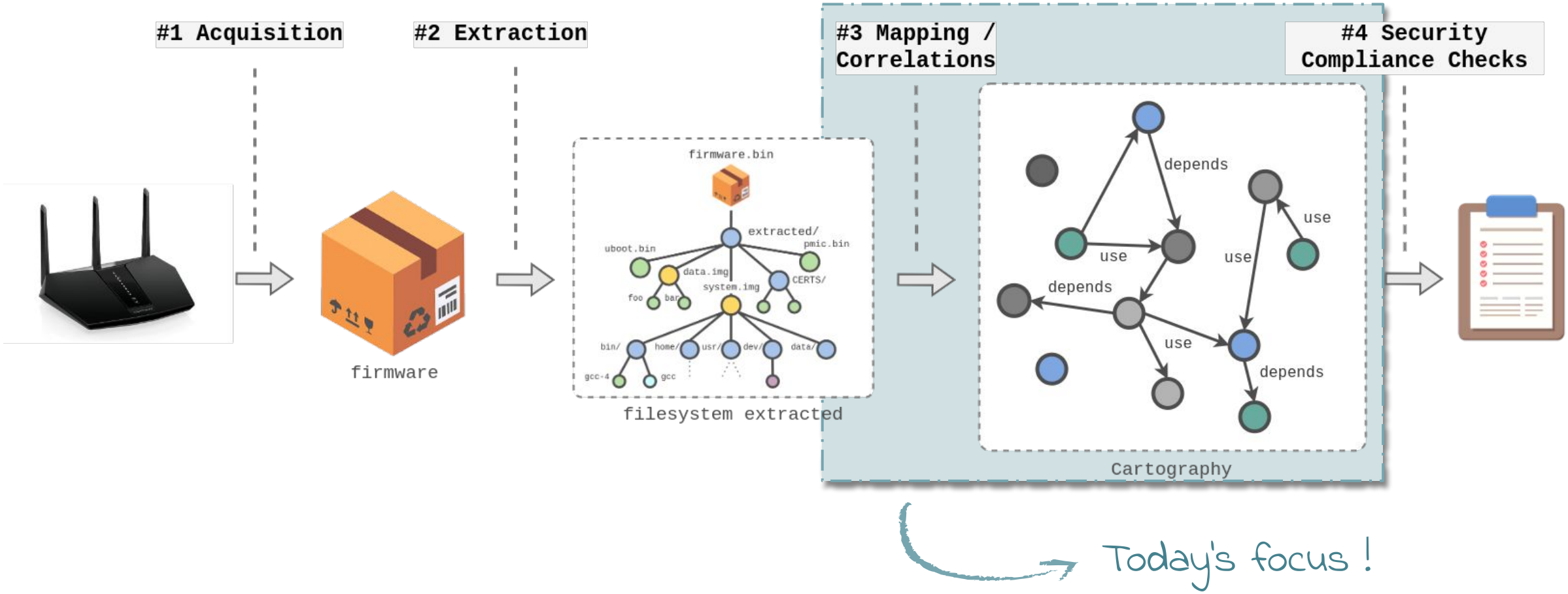
Use-cases:

- Vulnerabilities research
- Artifacts finding *(private keys, build files, symbols)*
- Compliance checks *(patch, fw sigs..)*
- Bill-of-Materials extraction

Problematics



Problematics



We extracted 1000 executables and files

Now what ??

⇒ **Issue**: finding the needle in the "haystack of binaries" *(for vuln, patch analysis etc..)*

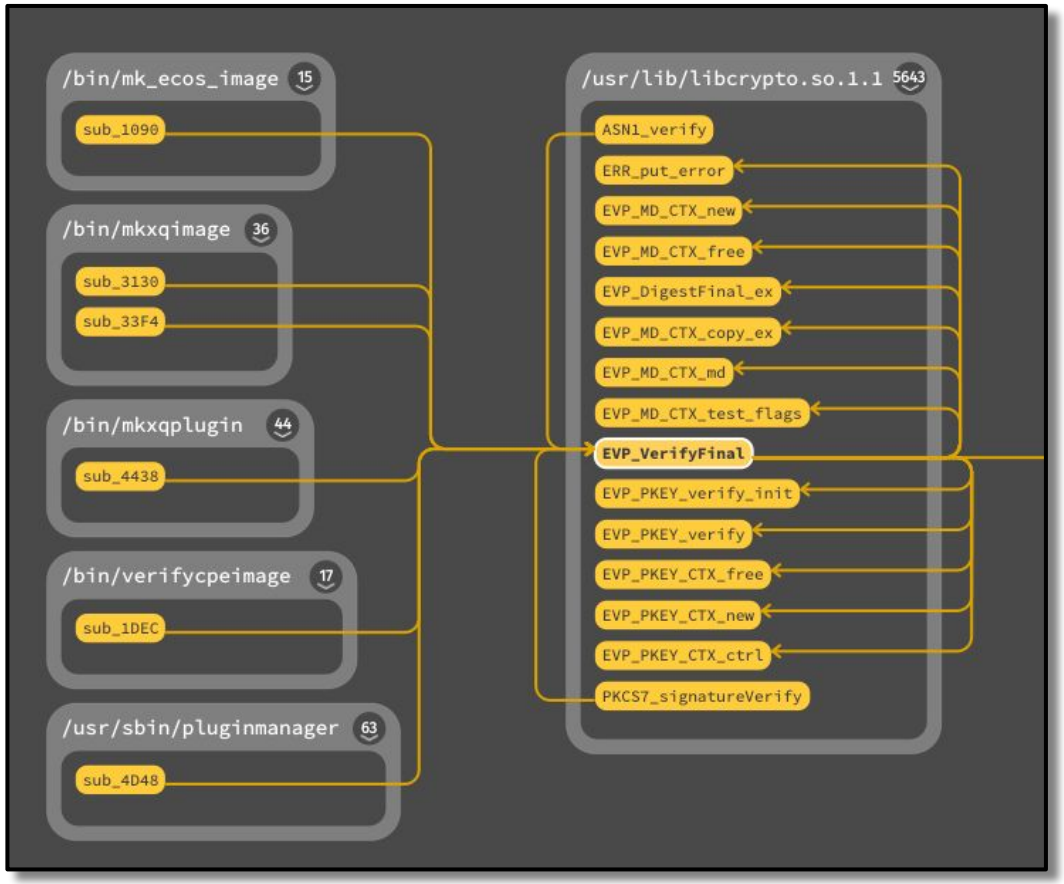
⇒ **Goal**: Mapping dependencies between executables *(and also files)* as a **graph**

⇒ **How**: Leveraging existing tools and our binary analysis tooling *(no rocket science but efficient and automated tooling)*

Graph Visualization For Vuln Research

Reversing Use-Cases:

- Which binaries are calling this exported library function ? *(with which parameters)*
- What programs:
  - opens this file ?
  - do interprocess comms ?
  - interact with a service ?

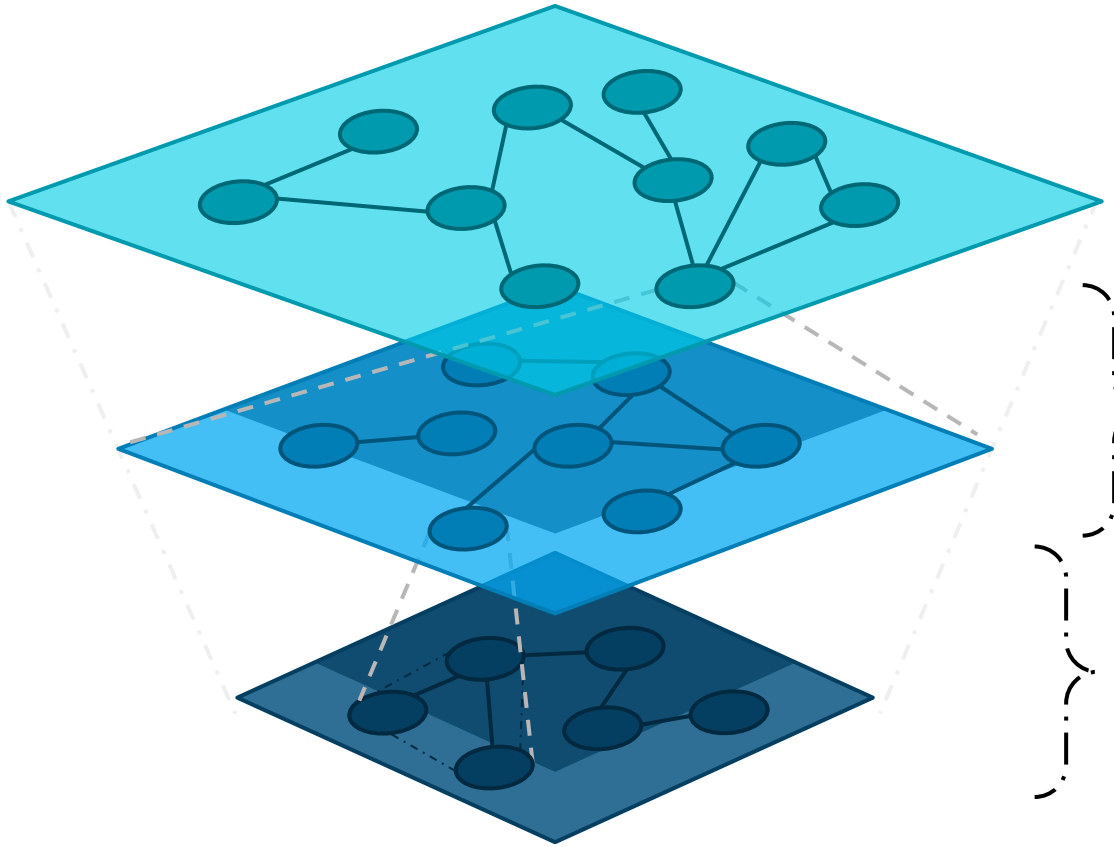


e.g: *"Who is using `EVP_Verifyfinal` ?"*

useful to check firmware signature mechanisms ...

Contribution: Fractal Graph Representation Approach

“Nested style graph”



Level 1: Firmware Call Graph *(all binaries)*

Level 2: Program Call Graph with Decompilation

Level 3: Program in a disassembler: CG and CFGs

RE//verse

Inter-Image Call Graph

(Level 1)

Inter-Image Call Graph

Definition

An inter-image call graph is a directed graph representing **call dependencies** between functions regardless of theirs executable location.
(nodes are functions, and edges call from one function to the other).

Computation requirements:

- We assume a filesystem is his "running state" *(where all partitions are mounted)*
- Requires taking in account fs peculiarities *(symbolic links etc..)*
- On "Unix-like" system, it somewhat requires doing what ld.so does.

⇒ No disassemblers enables computing IMG-CG and resolving cross-references across executables and libraries.

IMG-CG Algorithm

Resolving phase, what could go wrong ?

- Collision *(multiple libraries exposing the same symbol)*
- Symbol imported but exposed by no libraries *(e.g: httpd modules using logging function of httpd)*

⇒ Calls on PLT are directly bound on the target function in the library.

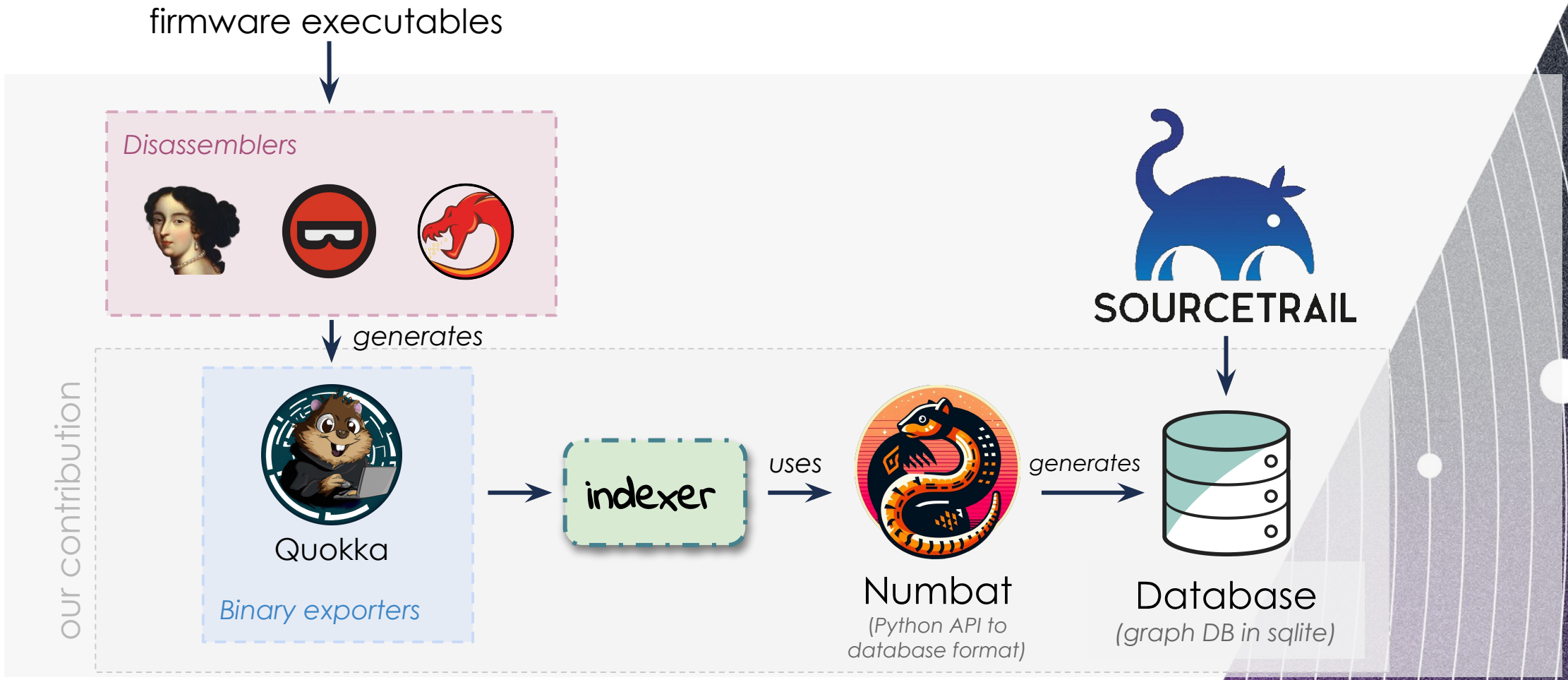
Algorithm *(in pseudo code)*

```
bins = []
G = DiGraph()

# create nodes
for bin in iter(filesystem):
    exe = disassemble(bin)
    bins.append(exe)
    G.add_nodes(calc_cg(exe))

# add all edges
for exe in bins:
    for frm, to in calc_cg(exe):
        if is_imported(to):
            real_to = resolve(to, bins)
            G.add_edge((frm, real_to))
        else:
            G.add_edge((frm, to))
```

Implementation *(Software Stack)*



⇒ *The whole process is fully automated!*

Sourcetrail *(in a Nutshell)*

Purpose: Source code Explorer

Bio:

- open-source !
- developed by CoatiSoftware
- **but abandoned**

Indexers:

- C / C++
- Java
- Python

⇒ Database format is open-source !

(but strongly source code oriented)

https://drive.google.com/file/d/1SI_ncKxl4h984BnWdddNNX4Y9RWXBsPb/view?usp=sharing

Sourcetrail ⇔ Numbat = <3

Main Idea: Database format is "somewhat" **open**.

⇓

So anyone can create **arbitrary databases** and make any data to fit into!*1

⇓

Numbat provides a Python API to do it !*2

Goal: Bringing back reverse-engineering tooling into source-based tools.

*1 former colleague Romain Thomas idea

*2 implemented by Sami Babigeon

Demo

navigating a firmware

(small Netgear RAX30 router)

Use-Case: libcurl

Fetching an URL

	Command line	Using the library in C

secure	`$ curl URL`	`curl_easy_setopt(p1, CURLOPT_URL, url);`
		`curl_easy_setopt(p1, CURLOPT_SSL_VERIFYHOST, 1);`
		`curl_easy_setopt(p1, CURLOPT_SSL_VERIFYPEER, 1);`

insecure	`$ curl --insecure URL`	`curl_easy_setopt(p1, CURLOPT_URL, url);`
		`curl_easy_setopt(p1, CURLOPT_SSL_VERIFYHOST, 0);`
		`curl_easy_setopt(p1, CURLOPT_SSL_VERIFYPEER, 0);`

*As given by curl.h: CURL_SSL_VERIFYHOST = **81** CURL_SSL_VERIFYPEER = **64***

Decompiled Executable Visualization

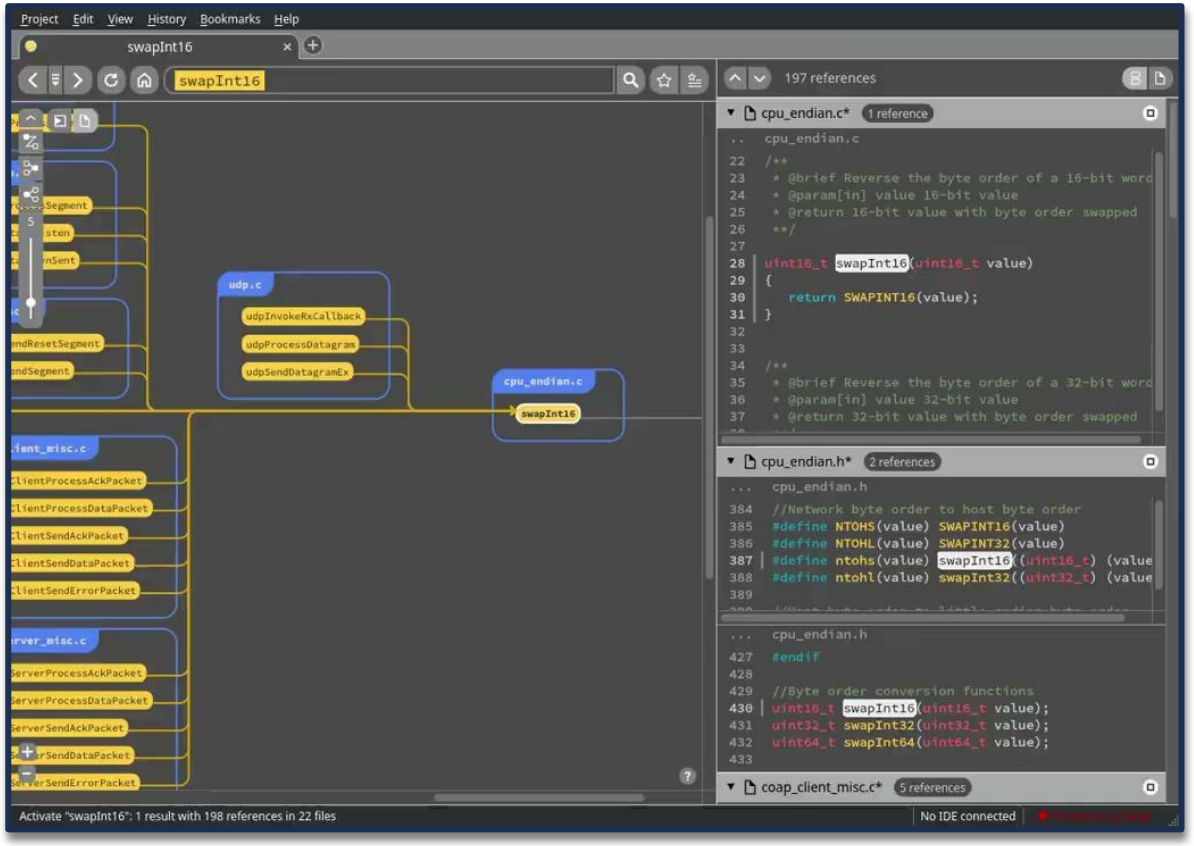
(Level 2)

Problematic

In most disassemblers:

- either navigate calls in a graph view
- either navigate decompiled code in text view *(and follow cross-refs one by one)*

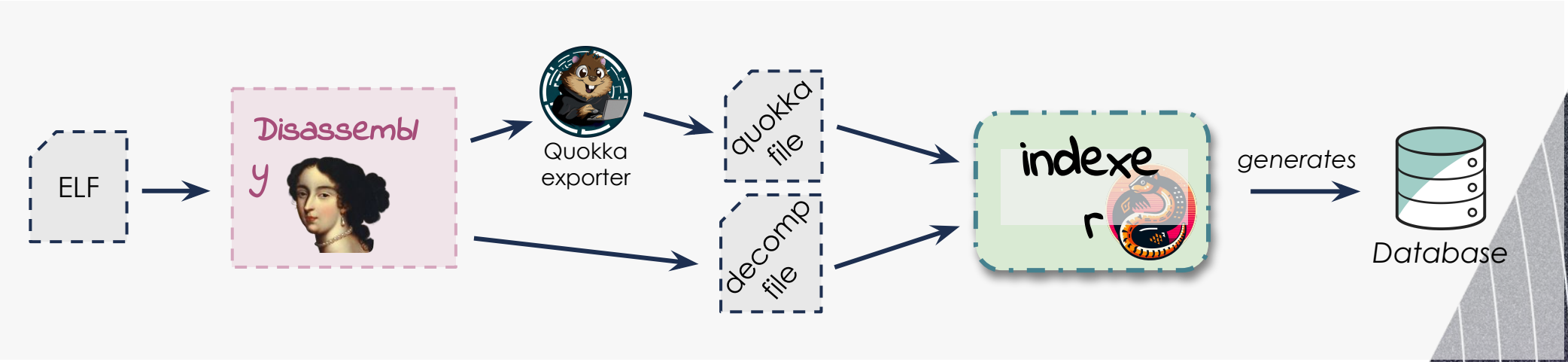
⇒ hard to have quick peek of all usages



Can we travel decompiled code like standard source code in Sourcetrail ? *(is it achievable with Numbat?)*

Can't use C indexer as decompiled code is not pure "compilable C".

(Retro)fitting Decompiled Code in Sourcetrail



- Xrefs requires tokenizing the whole source code
  - IDA API not of any help *(⇒ more of a hack at the moment)*
- Decompiled code on whole the IMG-CG do not scale
  (too massive for sourcetrail's schema model in sqlite)

Demo
(navigating decompiled)

Program in a Disassembler

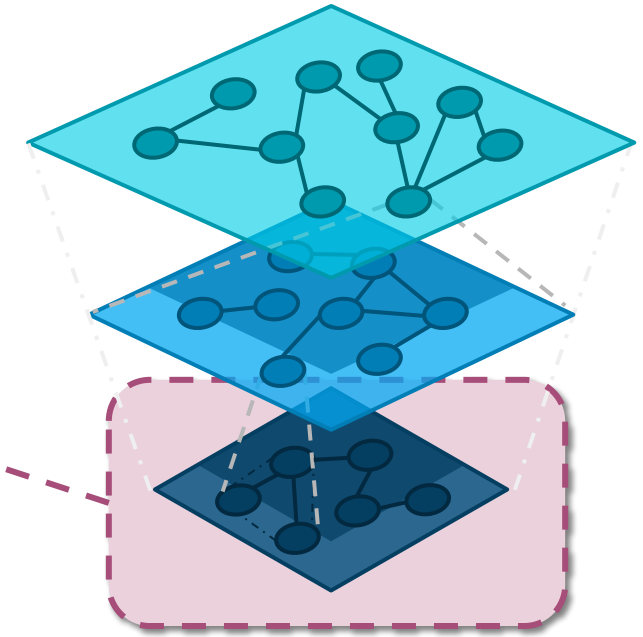
(Level 3)

Your Favorite Disassembler: [HERE]

Deepest analysis level

Call Graph at decompiled level is **helpful**, but will never fulfill what can be done within a disassembler.

- Handling all cases where recovering functions is difficult
- Manipulating data *(and data-xrefs)*
- Scripting API
- everything that can be done within **your disassembler**



⇒ ***Link every representations*** *(by tweaking sourcetrail UI)*

Demo

Beyond Structured Firmwares

(standard filesystem..)

Applying Similar Methodology

Can apply **same approach** to any context where multiple binaries are **somewhat linked** with each other.

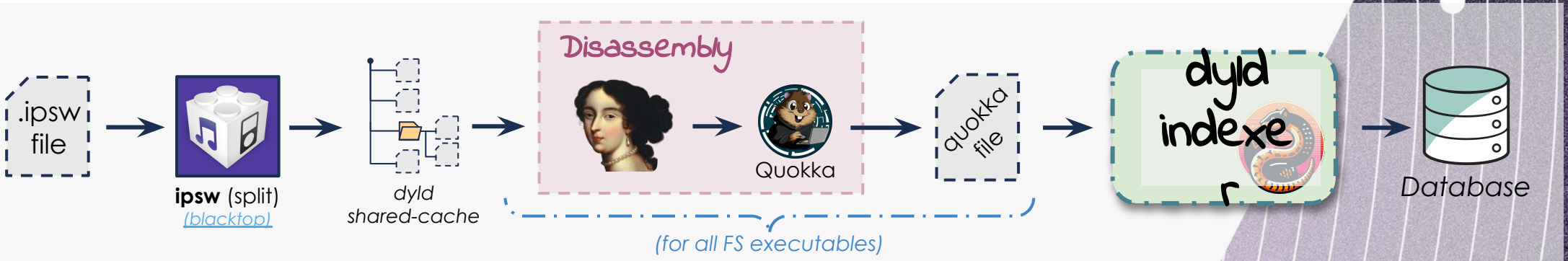
dyld shared cache
(on Apple devices)

Dyld Shared Cache Primer

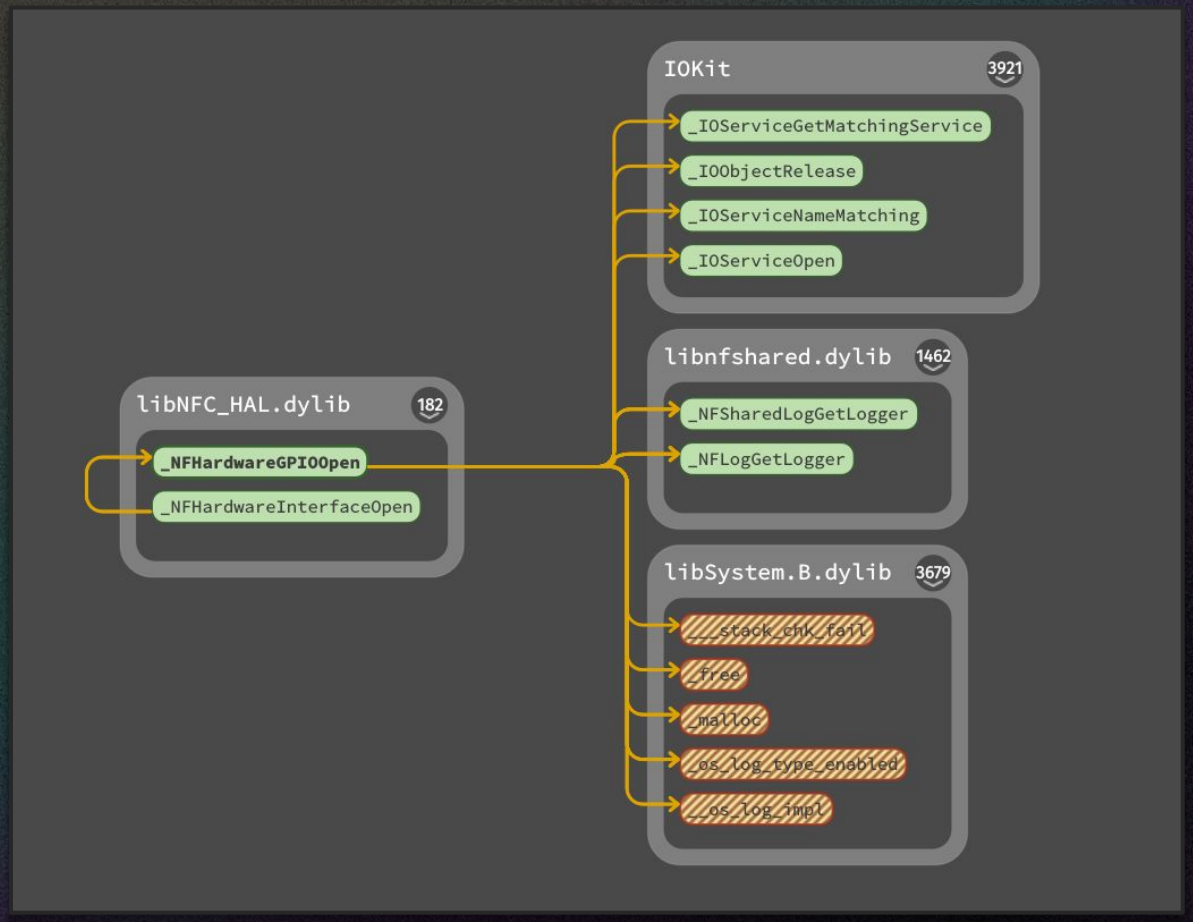
Set of **dynamic libraries** pre-linked together AOT into **a single file**. Then shared and used by running applications.

Mapping problematics:

- stub libraries *(not included in cache e.g private framework)*
- function re-exports *(LC_REEXPORT_DYLIB)*
- symbol coalescing
- stub-island jump mechanism



Example



Dyld size: **3.4 Gb**
Executables: **2505**
Nodes: **13.1** millions
Edges: **62.2** millions

⇒ Reaching DB limits

Conclusions & Takeaways

Conclusion & Takeaways

What:

- **IMG-CG** representation
  (inter-image call graph)
- nested graph analysis methodology
  (Firmware ↦ one binary ↦ disassembler)
- Used existing **of-the-shelf** tools to do it:

( +  +  + ) = 🚀

⇓

Trying to **bridge** reversing into **source code** audit tools.

Using another disassembler is doable

Why:

- better data-visualization
  "having the big picture"
- expanding reversing to multiple binaries all at once

⇒ Gaining RE efficiency

Getting Further

Current indexers *(IMG-CG, decomp, dyld)*:

⇒ still in a PoC state *(under integration in Pyrrha, shall be published soon™ (June maybe..))*

⇒ could go way further ! *(what if we also incorporate data and types ?)*

New use-cases:
(thanks to Numbat & Sourcetrail)

- binaries ⇔ kernel modules *(syscalls)*
- Inter-process communications *(dbus, Binder, XPC…)*
- links between ARM security-levels *(e.g: kernel and TAs interactions)*
- Permissions modeling *(SELinux ..)*
- Your use case..

Anything that can be modeled as relationships or a graph !

RE//verse

Thank you

Me:

rdavid@quarkslab.com

@RobinDavid

robin-david-17304685

Quarkslab:

contact@quarkslab.com

@quarkslab

www.quarkslab.com

RE//connect

RE//think

RE//solve

RE//verse

Some Statistics

⇒ On Netgear RAX 30 router in version 1.0.7.78

Infos

Size	161 Mb
#files	1746
#Executables	111
#Libraries	318
#kernel modules	136

565

	Time			Size	
	Mean	Total *(1 cpu)*	Total *(8 cpu)*	Mean	Total
Disassembly *(.i64)*	25s	4h10m	29min18s	1.8 Mb	1.1 Gb
Quokka	0.77s	438s	68s	460 Kb	255 Mb
Binexport	0.85s	481s	72s	661 Kb	366 Mb
Decompilation	19s	3h03	37m50s	387 Kb	214 Mb
Indexing	1.9s	18m30s	3m28s	1.5 Mb	831 Mb
	Total:	7h38m	1h9m	**Total*:**	1.27 Gb

Favor on-demand creation rather than systematic DB creation

*(without the .i64 and only Quokkas)