FoxDec User Manual

VIRGINIA TECH

Open Universiteit

FoxDec 0.4

Decompilation based on Formal Methods

prof. Binoy Ravindran PI
dr. Freek Verbeek co-PI
Joshua Bockenek PhD
Daniel Spaniol PhD
freek@vt.edu

Supported by the DARPA project FALCON: Formal Analysis of Legacy COde domaiNs

User Manual

June 9, 2023

FoxDec is a tool actively developped at Virginia Tech (US) and the Open University of the Netherlands. Its aim is to lift binaries to a higher level of abstraction, in such a way that formal guarantees can be provided that the lifted representation is sound with respect to the original binary. This document provides a user manual, further information on implementation and limitations, as well as references for further reading.

Remark: *FoxDec is evolving quickly, and new features and capabilities are actively being developped. Do not hesitate to contact us for questions, remarks and suggestions.*

1 FoxDec Story

One of the key goals for the VT FALCON project is trustworthy, provably sound x86-64 binary lifting. The first step of any approach to binary lifting is disassembly. A fundamental challenge herein is to resolve indirect branches. Instead of leveraging heuristics, a provably sound approach requires indirections to be resolved based on binary-level invariants that provide, e.g., information on pointer values stored in stateparts, and bounds on indices into jump tables. We have thus developped FoxDec, a tool that integrates disassembly, indirection resolving, and invariant generation into one tool. Key characteristic is that FoxDec enables formal verification: the invariants can be exported to the Isabelle/HOL theorem prover. In Isabelle/HOL, proof scripts automatically show that all generated invariants are sound.

When applying FoxDec to a binary, information is generated on disassembled instructions, reconstructed control flow, function boundaries, and invariants. All this information is programmatically available through an interface, but it also stored in graphical representations. Most notably, an annotated call graph is generated. This call graph privdes an overview of all reached functions, as well as assumptions made that were required for verification of elementary memory-related sanity properties. These properties include return address integrity (no function should overwrite its own return address) as well as adherence to a calling convention.

We have applied this approach to Linux Foundation and Intel's Xen Hypervisor and to Ubuntu binaries that do file accesses, networking, databases, and a text editing with a UI. Moreover, we have applied FoxDec to MacOs libraries from Mozilla Firefox that include cryptographic and security-related libraries and AV codecs. Running times vary from seconds to about 15 minutes for the vi editor, spanning over 426.258 instructions and 3139 functions.

2 User Manual with Example

2.1 Download, Build & Installation

Up-to-date information on where to download FoxDec, and instructions for building and installation, can be found at:

`https://ssrg-vt.github.io/FoxDec/#build`

We assume that the following is run on an x86-64 machine. On ARM MacBooks with Rosetta, one can use Docker with the `FROM -platform=linux/amd64 ubuntu:20.04` to emulate x86.

2.2 Running FoxDec to create Hoare Graph

Compile. As running example, we will consider the `wc` command. For sake of explanation, we consider a small and simple implementation instead of taking the binary as available in a standard Linux or Mac distribution[1]. We assume the source ocde is in a file `wc.c`. First, we compile the example. Go to the directory for the running example `./binary/`. There, we compile the file `wc.c` to an executable `wc`.

Compile the running example

```
cd ./binary
rm wc wc.entry
gcc wc.c -o wc
cd ..
```

Run FoxDec. The command-line usage for FoxDec is:

`./foxdec.sh $NAME`

`$NAME` The filename of the binary without path

Run FoxDec

```
./foxdec.sh wc
  ↪ Function pointer 16fc introduced at [11b8]
```

[1]The source code of the `wc` example can be found here:
`https://www.gnu.org/software/cflow/manual/html_node/Source-of-wc-command.html`

The output indicates that FoxDec has finished, but the result may be incomplete. FoxDec needs as manual input some *function entries*, i.e., instruction addresses of the instructions where functions start. After a run, it provides an overview of *dangling function pointers*, i.e., addresses that it *guesses* to be function entries and that have not been explored yet. Manual analysis shows that the reported dangling function pointer is indeed a valid function entry. We thus add it to `./binary/wc.entry` and rerun FoxDec.

Run FoxDec Again

```
echo '0x16fc' >> ./binary/wc.entry
./foxdec.sh wc
```

We no longer see any message reporting dangling function pointers. As such, we can now observe the results.

2.3 Observing the generated Hoare Graph

Observe Metrics.

A series of metrics have been generated to provide quantative output on precision, coverage, running time, etc.

Observe Metrics

```
less ./artifacts/wc.metrics.txt
less ./artifacts/wc.metrics.json
```

Observe Hoare Graph.

At this point, FoxDec will have generated output concerning the *control flow* of the program, the *function boundaries*, it will have generated *invariants* and *disassembled instructions*, etc. All of this information is stored in a `.json` file. For sake of convenience, the information is also outputted in a humanly readable format. We also specifically provide an overview of all resolved indirections.

Observe Hoare Graph

```
less ./artifacts/wc.json.txt
less ./artifacts/wc.json
less ./artifacts/wc.indirections
```

Observe Call Graph with Function Contexts.

The call graph has been generated in a `.dot` file, and needs to be visualized to a `.pdf` using Graphviz.

Observe Call Graph

```
dot -Tpdf -o ./artifacts/wc.calls.pdf ./artifacts/wc.calls.dot
```

Opening the `.pdf` file, the call graph has been generated, as well as the *function contexts*. These provide information on pointers initially provided to the functions, as well as there mutual relations (aliassing/separation).