Visual Analysis of Control Coupling for Executables

Arne Wichmann, Sibylle Schupp
Technische Universität Hamburg-Harburg, Hamburg, Germany
{arne.wichmann, schupp}@tuhh.de

Program comprehension of stripped executables is hard because neither modules and function names, nor any other structural information are available. We introduce an algorithm that, using morphological operations, highlights fan-in, fan-out, and module coupling in the adjacency matrix of the control flow graph and thus allows initial orientation at function level.

This paper introduces the structures of interest and our algorithm, and analyzes the `yaboot` bootloader.

1 Control Coupling Analysis of Stripped Executables

A general problem of analyzing stripped executables is that very little information is available. Specifically, a disassembler can only help to regroup the code to functions, but on its own cannot generate useful function names, or recreate modules of the code. A first orientation in the code needs a method that is independent of debug information, execution traces, or known libraries and operating systems.

In this paper we present an algorithm that reconstructs information about functions of interest, modules, and their control coupling and thereby gives an overview of the executable. An exploitable property for such an algorithm is that linkers keep the functions and object files in the sequence they were given during compilation and thereby encode such information into the program's addresses and its control flow graph (compare [2]). Our algorithm mines the control flow graph for fan-in, fan-out, and reference clusters between functions using morphological operations on the graph's adjacency matrix and presents them in a comprehensive plot, so that the analyst can get a rough overview of the executable.

While alternative visual coupling analyses exist, they are usually applied to structures extracted from high-level code [1] or use trace information [7].

Table 1: Interpretations of Visual Structures

Structure	Interpretation
Diagonal	Local Control Flow
Box/Triangle on Diagonal	Module
Box/Triangle not on Diag.	Module Coupling
Global Vertical	Fan-In/Library
Vertical near Diag.	Helper Function
Global Horizontal	Fan-Out/Dispatch
Horizontal near Diag.	Dispatch Function

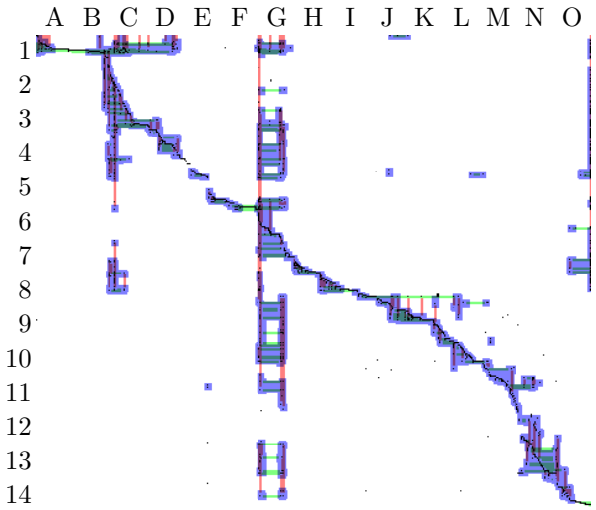


Figure 1: Control Flow Plot for `yaboot`/fc15

2 Visual Analysis of `yaboot`

Figure 1 shows an example of the `yaboot` bootloader for PowerPC from the Fedora Core 15 Linux distribution with rough row (A..O) and column (1..14) notations for orientation. Each horizontal row in the plot represents one of the about 350 functions, each column tracks control accesses in 128 byte steps. We describe the structures of interest and their interpretation:

A1..D1 Big green horizontal line: A setup/dispatch module with large fan-out.

B2..C8 Several red vertical lines: A library module with high fan-in.

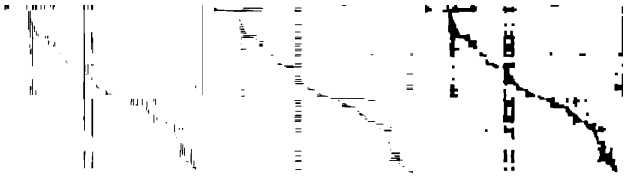
G1..G14 Long red vertical lines: Basic library functions with high fan-in.

J9..N14 Different shapes near diagonal, little coupling in J1..N8 and A9..I14: External library with good separation.

Manual inspection using information from debug symbols confirms the interpretation: A1 is a module that contains an interactive shell, B2 is a wrapper module to the openfirmware (BIOS like), G1 is the C library and J9 is the ext2fs library.

3 Control Flow Plots and Visual Structures

The goal is to create an algorithm that produces a control flow plot, like the one shown in the example above. A control flow plot is based on a binary adja-



(a) H (red) (b) V (green) (c) B (blue)

Figure 2: Separated Results for `yaboot/fc15`

cency matrix of control accesses (similar to [6]), where both axes represent numerically ordered program addresses. To represent function sizes, outgoing accesses are grouped per function (horizontal) and incoming accesses (vertical) use bins in the size of an average basic block.

In such a plot different key visual structures (see Table 1) can be mined and highlighted that provide semantics for an analyst. The diagonal line is caused by intraprocedural control flow, like jumps and next instructions, and its steepness represents the functions sizes. Modules can be identified by their intra-module coupling, which shows up as boxes or triangles next to the diagonal line. Coupling between modules shows up as boxes or triangles disconnected from the diagonal. Setup functions with high fan-out will produce a scattered horizontal line, helper and library functions with high fan-in will leave a similar but vertical line.

The scope of functions and modules can be estimated by their access lines' length as well as their relative closeness to the diagonal. Longer lines correlate to broader access across modules and shorter lines near the diagonal identify intramodular functions.

4 Morphological Highlighting of Structures

We provide a simple algorithm to highlight the structures in the image using basic image processing operations like erosion, dilation, opening (erode + dilate), and closing (dilate + erode) from mathematical morphology. An erosion operation leaves only pixels in the image, where a structuring element fully fitted in the original image. A dilation adds pixels wherever a structuring element aligned with a pixel from the original image has a pixel.

A custom script in IDA Pro [3] extracts the control flow graphs as plots, which are then processed using the octave image package [5].
Listing 1 shows the code used to highlight the structures in the plot. The separated results (see Figures 2a to 2c) are composed in an additional step to form the result image (see Figure 1). The line detection in both horizontal and vertical direction first uses *closing* on the image with a long line as structuring element to create initial lines, then *erodes* with a shorter element to keep only long lines. The line is finally broadened by a *dilation* with an orthogonal structure.

```
H = imdilate(imerode(imclose(J,ones(hlen,1)),
        ones(hlen/3,1)),ones(1,vlen/10));
V = imdilate(imerode(imclose(J,ones(1,vlen)),
        ones(1,vlen/3)),ones(hlen/10,1));
B = H + V + J;
s = ones(blen/3,blen/3);
B = imclose(imdilate(imclose(B,s)-B,s),s);
```

Listing 1: Line and Block Emphasis Algorithm

To create the boxes and triangles in the image, the original image is combined with the detected lines and then *closed* using a block structuring element to form a blurred version. Single dots and lines are removed by subtracting the combined image, and the resulting gaps are removed in one *dilation* step. The remaining structures in the image are again *closed* to produce the final area information.

5 Summary and Future Work

The algorithm works independently of the architecture and therefore supports all the processor modules in IDA Pro. The prototype implementation was run on a testsuite [4] of several embedded executables as well as executables from the CPU2006 benchmark. Manual checks of the results are consistent with debug as well as reverse-engineered information. The next steps are to integrate the implementation in a reverse engineering workflow and to expand the evaluation in both quality and quantity.

References

[1] Johannes Bohnet and Jürgen Döllner. "Visual Exploration of Function Call Graphs for Feature Location in Complex Software Systems". In: *Proceedings of the 2006 ACM Symposium on Software Visualization*. SoftVis '06. Brighton, United Kingdom: ACM, 2006, pp. 95–104.

[2] Peter J. Denning. "The Locality Principle". In: *Communications of the ACM* 48.7 (July 2005), pp. 19–24.

[3] *IDA Pro: Interactive Disassembler*. URL: `http://www.hex-rays.com/products/ida/index.shtml`.

[4] *Küstennebel*. URL: `http://www.tuhh.de/sts/research/projects/kuestennebel.html`.

[5] *Octave-forge: Image*. URL: `http://octave.sourceforge.net/image/`.

[6] Reese T. Prosser. "Applications of Boolean Matrices to the Analysis of Flow Diagrams". In: *Papers Presented at the December 1-3, 1959, Eastern Joint IRE-AIEE-ACM Computer Conference*. IRE-AIEE-ACM '59 (Eastern). Boston, Massachusetts: ACM, 1959, pp. 133–138.

[7] Daniel Quist and Lorie M. Liebrock. "Reversing Compiled Executables for Malware Analysis via Visualization". In: *Information Visualization* 10.2 (2011), pp. 117–126.