JOHNS HOPKINS
APPLIED PHYSICS LABORATORY

A Code Pirate’s Cutlass:

Recovering Software Architecture from Embedded Binaries

evm
@evm_sec

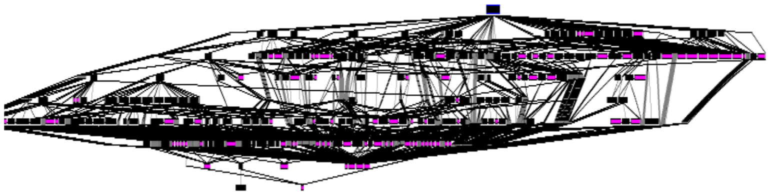
Motivation

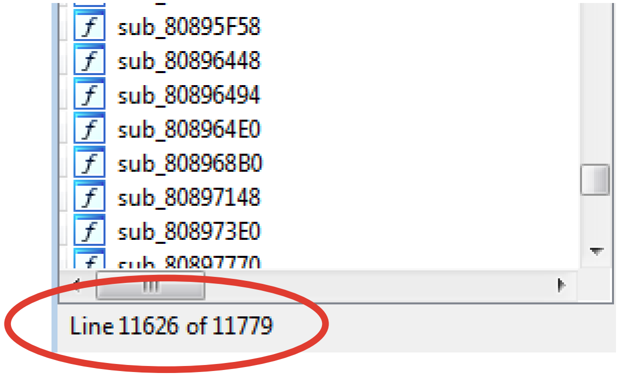
- Much of infosec is built on top of reverse engineering (RE)

- RE is manually intensive and requires multi-domain expertise, particularly for embedded systems

- Embedded systems
  - Combine OS, libraries, and application code into a single program space
  - Binary is fully linked with no symbols (usually)

- Previous research in RE has focused on
  - Code-to-code translation: Binary -> Intermediate Language -> High Level Language
  - Function-level matching





© 2018 The Johns Hopkins University Applied Physics Laboratory LLC. All Rights Reserved.

Towards Automated RE

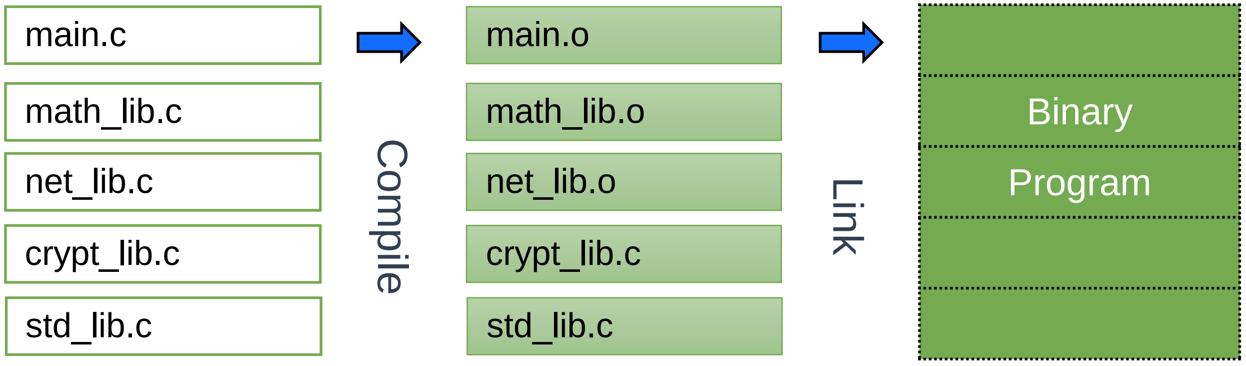
- Objects / Libraries
- Subroutines / Functions
- Statements / Constructs
- Assembly / Opcodes

- Reverse engineers operate on at least 4 levels
- Usually when a new project gets started we are spinning our wheels a bit at the bottom in order to label enough functions to start to make sense of the bigger picture
- For ML/DL approaches – we are going to need methods to chunk up a large binary – and give a sense of context for each function

© 2018 The Johns Hopkins University Applied Physics Laboratory LLC. All Rights Reserved.

The CodeCut Problem

- Assumptions:
  - Embedded developers organize code into multiple source files
  - Source files are compiled into object files
  - Linker produces final binary that is a linear concatenation of object files
  - No intentional obfuscation

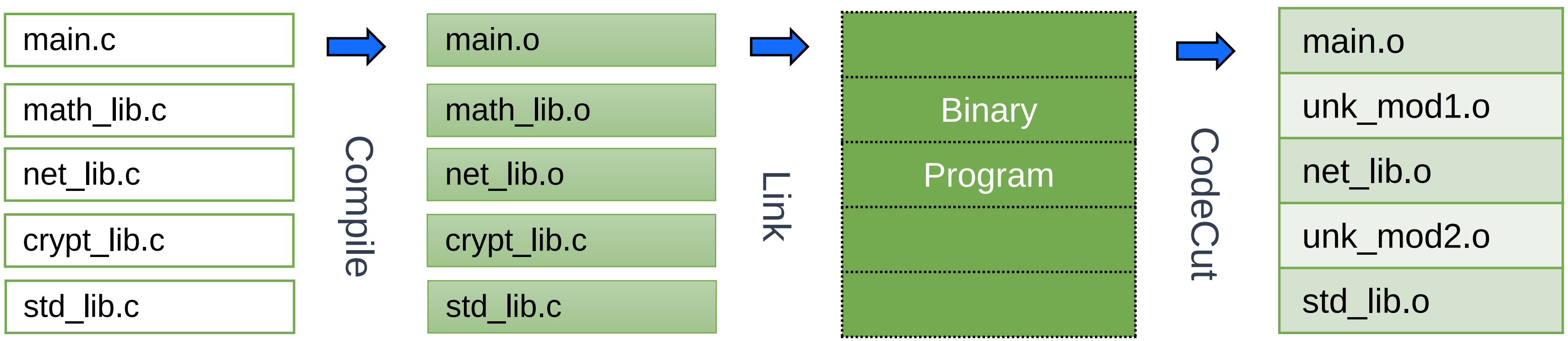


© 2018 The Johns Hopkins University Applied Physics Laboratory LLC. All Rights Reserved.

The CodeCut Problem

- Problem Statement: Given only call graph information for a large binary, recover the boundaries of the original object files

- Notes:
  - Essentially architecture independent (as long as a call graph can be generated through disassembly)
  - Inherent ambiguity: CodeCut algorithms might locate multiple functional clusters within an original source file - or combine two files because they are highly related



© 2018 The Johns Hopkins University Applied Physics Laboratory LLC. All Rights Reserved.

Local Function Affinity Concept

```
#include <stdio.h>
int helper_1() {
    return helper_2()/100;
}
int helper_2() {
    …
}
int more_complex() {
    …
    while (helper_1() < 100) {
        foo = helper_2() % 20;
    }
    …
}
void main_functionality() {
    more_complex();
    …
    while (helper_2() > 1000) {
        foo = helper_1();
        bar = more_complex();
    }
}
```

© 2018 The Johns Hopkins University Applied Physics Laboratory LLC. All Rights Reserved.

Local Function Affinity Concept

```
#include <stdio.h>
int helper_1() {
      return helper_2() / 100;
}
int helper_2() {
      …
}
int more_complex() {
      …
      while (helper_1() < 100) {
            foo = helper_2() % 20;
      }
      …
}
void main_functionality() {
      more_complex();
      …
      while (helper_2() > 1000) {
            foo = helper_1();
            bar = more_complex();
      }
}
```

- If we eliminate external calls…

© 2018 The Johns Hopkins University Applied Physics Laboratory LLC. All Rights Reserved.

Local Function Affinity Concept

~~#include <stdio.h>~~

```
int helper_1() {
      return helper_2() / 100;
}
int helper_2() {
      …
}
int more_complex() {
      …
      while (helper_1() < 100) {
            foo = helper_2() % 20;
      }
      …
}
void main_functionality() {
      more_complex();
      …
      while (helper_2() > 1000) {
            foo = helper_1();
            bar = more_complex();
      }
}
```

- If we eliminate external calls…
- Directionality of calls at the beginning of the module is in the positive direction

© 2018 The Johns Hopkins University Applied Physics Laboratory LLC. All Rights Reserved.

Local Function Affinity Concept

~~#include <stdio.h>~~

```
int helper_1() {
    return helper_2() / 100;
}
int helper_2() {
    …
}
int more_complex() {
    …
    while (helper_1() < 100) {
        foo = helper_2() % 20;
    }
    …
}
void main_functionality() {
    more_complex();
    …
    while (helper_2() > 1000) {
        foo = helper_1();
        bar = more_complex();
    }
}
```

- If we eliminate external calls…
- Directionality of calls at the beginning of the module is in the positive direction
- Directionality of calls generally switch to the negative direction towards the end of the module

© 2018 The Johns Hopkins University Applied Physics Laboratory LLC. All Rights Reserved.

Local Function Affinity Concept

```
#include <stdio.h>
int helper_1() {
    return helper_2() / 100;
}
int helper_2() {
    …
}
int more_complex() {
    …
    while (helper_1() < 100) {
        foo = helper_2() % 20;
    }
    …
}
void main_functionality() {
    more_complex();
    …
    while (helper_2() > 1000) {
        foo = helper_1();
        bar = more_complex();
    }
}
```

- If we eliminate external calls…
- Directionality of calls at the beginning of the module is in the positive direction
- Directionality of calls generally switch to the negative direction towards the end of the module
- We can detect edges by finding the switch from negative back to positive

© 2018 The Johns Hopkins University Applied Physics Laboratory LLC. All Rights Reserved.

Local Function Affinity Concept

```
#include <stdio.h>
int helper_1() {
    return helper_2() / 100;
}
int helper_2() {
    …
}
int more_complex() {
    …
    while (helper_1() < 100) {
        foo = helper_2() % 20;
    }
    …
}
void main_functionality() {
    more_complex();
    …
    while (helper_2() > 1000) {
        foo = helper_1();
        bar = more_complex();
    }
}
```

- If we eliminate external calls…
- Directionality of calls at the beginning of the module is in the positive direction
- Directionality of calls generally switch to the negative direction towards the end of the module
- We can detect edges by finding the switch from negative back to positive

© 2018 The Johns Hopkins University Applied Physics Laboratory LLC. All Rights Reserved.

Local Function Affinity Definition

$$Affinity(f) = \frac{\sum_{x \in references(f),} sign(x - f) * Log(|x - f|)}{|references(f)|}$$

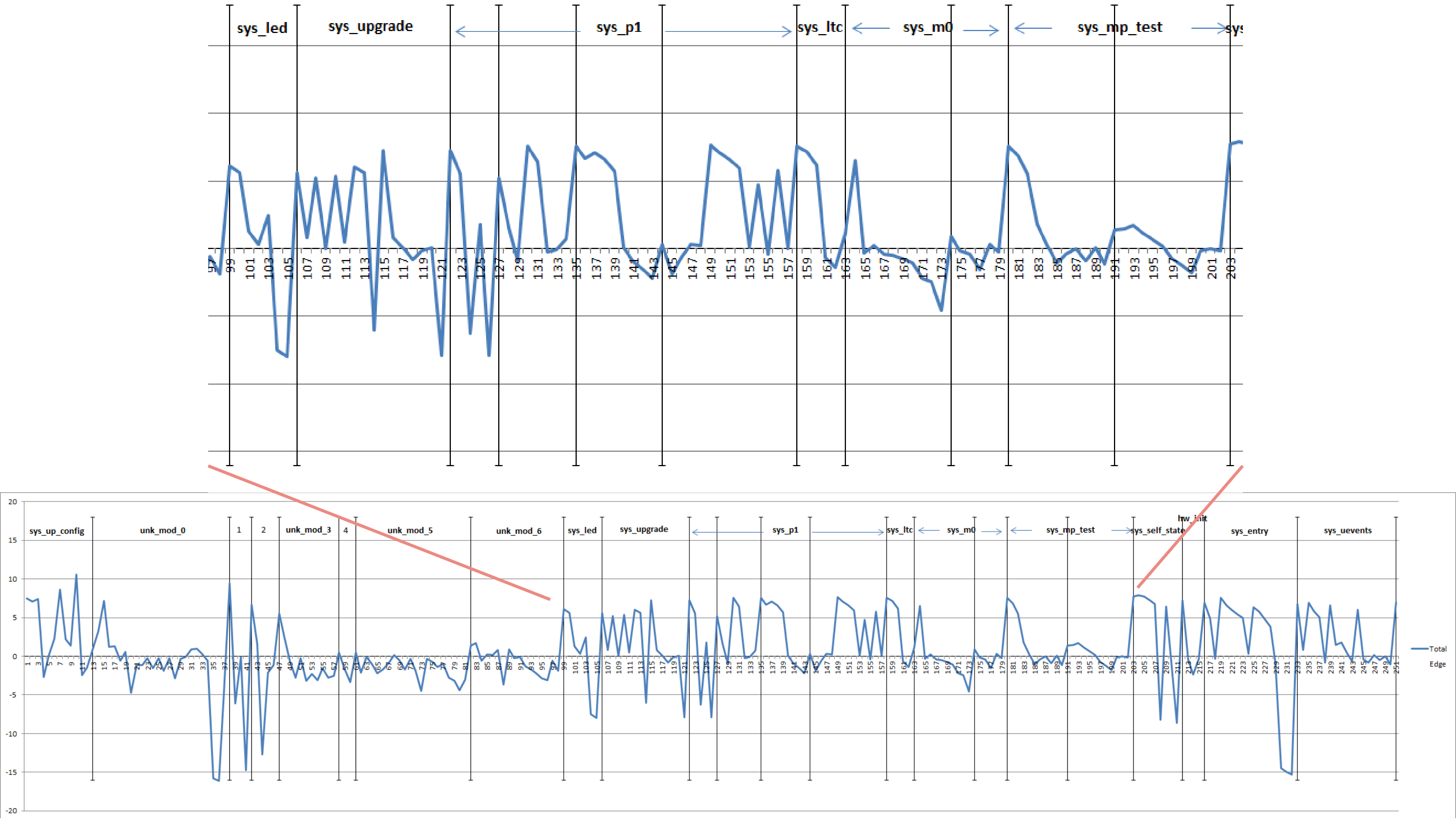
Where *references(f)* is defined as the set of functions that call f or are called by f for which the distance from f to the function is below a chosen threshold. Multiple references are counted.

- Using fixed threshold of 4K*
- Edge Detection*:
  - General negative trend
  - Change to positive value ($\Delta > 2$)
  - Treat calls to / calls from as separate scores – for functions without one of the scores, interpolate from last score

* room for improvement!

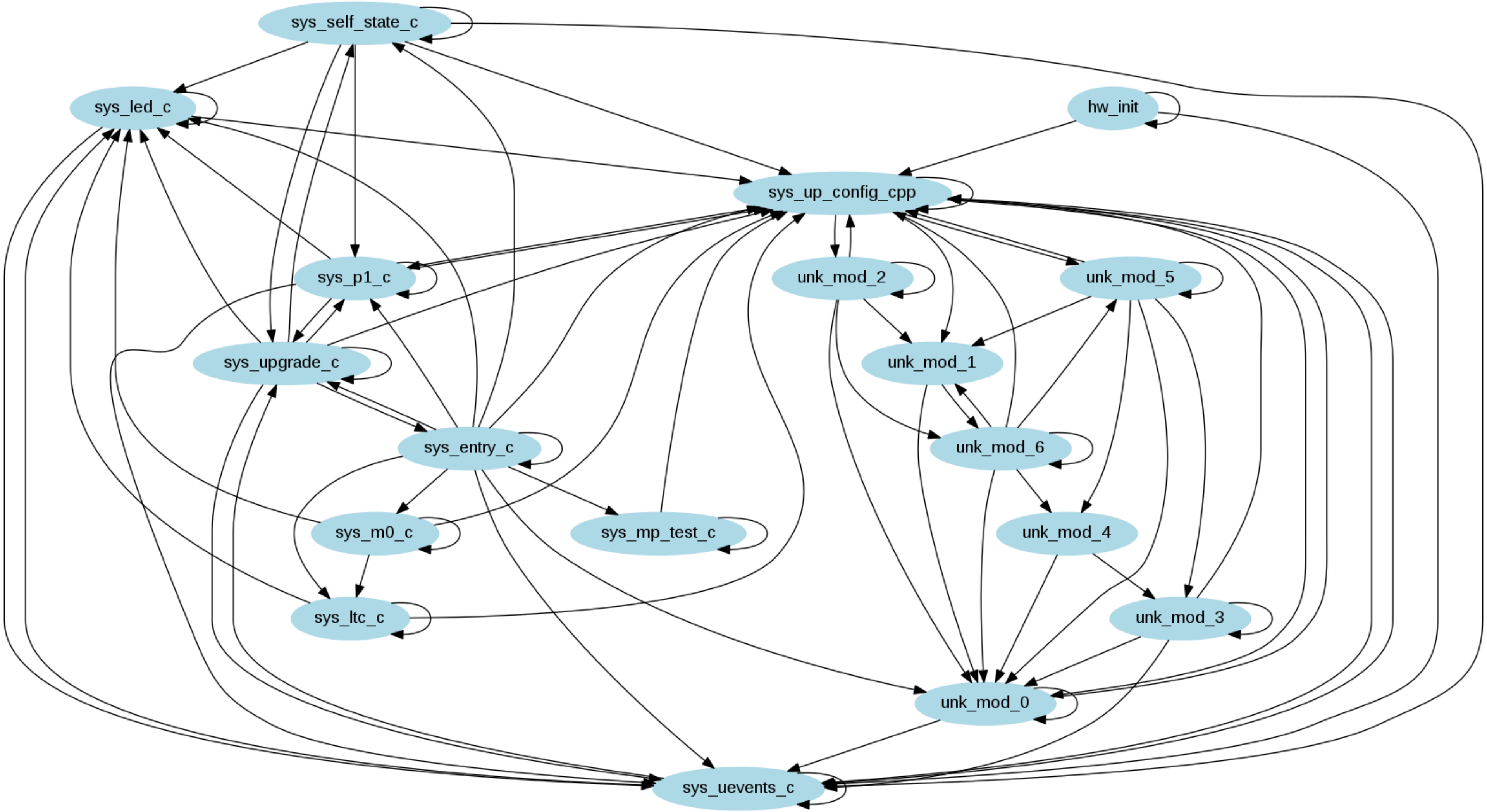
© 2018 The Johns Hopkins University Applied Physics Laboratory LLC. All Rights Reserved.

Call Directionality Metric



© 2018 The Johns Hopkins University Applied Physics Laboratory LLC. All Rights Reserved.

Module-to-Module Call Graph (Auto-Generated)



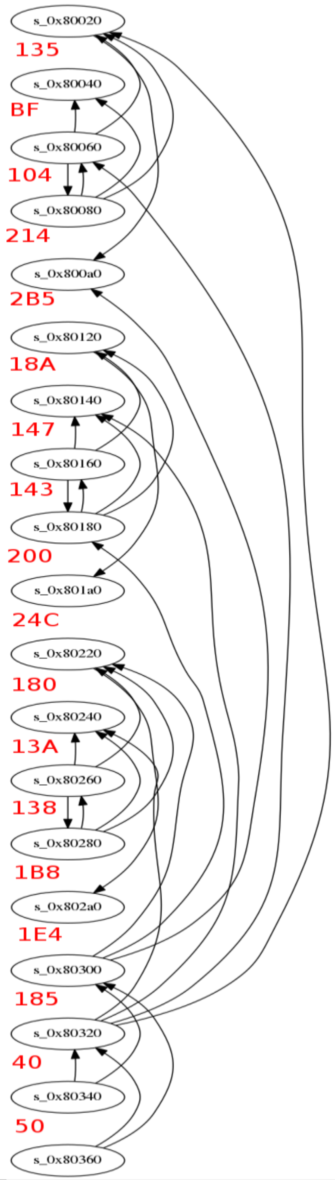
© 2018 The Johns Hopkins University Applied Physics Laboratory LLC. All Rights Reserved.

LFA Results to Date

	Match / Gap / Underlap (%)		
• Gnuchess (x86)	76.1	3.2	20.7
• PX4 Firmware/NuttX (ARM)	82.2	13.6	4.2
• GoodFET 41 Firmware (msp430)	76.1	0	23.9
• Tmote Sky Firmware/Contiki (msp430)	93.3	0	6.7
• NXP Httpd Demo/FreeRTOS (ARM)	86.7	1.4	11.9

© 2018 The Johns Hopkins University Applied Physics Laboratory LLC. All Rights Reserved.

A Maximum Cut Graph Algorithm



- $Weight(C) = \dfrac{\sum_{E \in crossings(C),} |E|}{|crossings(C)|}$

where $crossings(C)$ is defined as the set of edges (calls) that "cross" the cut address

- Algorithm:
  - For every possible cut C, calculate *Weight*(C) and choose C with maximum weight
  - Remove edges that cross C from graph
  - Divide graph into two subgraphs
  - Recursively evaluate subgraphs, stop when modules are below a chosen threshold

© 2018 The Johns Hopkins University Applied Physics Laboratory LLC. All Rights Reserved.

Show Me The Code!

CodeCut is available at:

http://github.com/jhuapl/CodeCut

(LFA only for now)

Contact Info:

@evm_sec

evm.ftw@gmail.com

© 2018 The Johns Hopkins University Applied Physics Laboratory LLC. All Rights Reserved.

JOHNS HOPKINS
APPLIED PHYSICS LABORATORY

JOHNS HOPKINS
APPLIED PHYSICS LABORATORY

A Code Pirate's Cutlass:

Recovering Software Architecture from Embedded Binaries

evm
@evm_sec