Moka

Research Report

Jonathan Miodownik & Jacob Torrey

26 May 2016

The views, opinions, and/or findings contained in this presentation are those of the authors and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government

Jonathan Miodownik

ais
devastating capability, revolutionary advantage

- Security Researcher at Assured Information Security

- Interests
  - Computation/Language Theory
  - Transmission mediums
- Long, bounded walks on the beach

Overview

- History of Moka
- Objective
- Experiment
- Results
- Summary

If at any point you have a question, please feel free to ask.

Crema

- **Research effort preceding Moka**
  - Demonstrated feasibility of use and security benefits of sub-TC programming languages
  - Calculated the effect of restricted computation on formal methods

153 Brooks Road, Rome, NY | 315.336.3306 | http://ainfosec.com

Moka

- **Quantify the effectiveness of applying LangSec principles to real life code**
  - Large data set
  - Builds off work done with Crema
- **Quantify:**
  - code termination (sub turing complete (**sub-TC**))
  - non-halting code (absolutely turing complete (**TC**))

Why do we care?

- **Turing Complete Execution**
  - Code will accept arbitrary inputs
  - These inputs control execution behavior
  - An input crafted by an attacker, can lead the system into an untrustworthy state. (The land of Weird Machines)

Why do we care? II

devastating capability, revolutionary advantage

- **Sub-TC Execution:**
  - Input languages are restricted and well-defined
  - Program execution is determinate and can be proven safe (from a LangSec perspective.)
  - Cannot be applied to all applications.

    Eg. Input handling

A Case for LangSec

- **LangSec oriented coding is not only feasible to use in day to day coding, it is downright foolish NOT to program with it in mind!**
  - How much utility does it have in large programs
  - Difficulty of implementation
  - Identification of refactorable areas
- **Think:** We should be pushing for the 5th stage of acceptance, as per Dan Geer*

http://spw15.langsec.org/geer.langsec.21v15.txt

Tools and Data

- LLVM/Clang-3.8
- Linux Kernel 4
- Python 2.7

LLVM

- **LLVM**
  - Modular and abstracted open-source compilation tool-chain
  - Compiles to immediate representation (IR) for advanced optimization and static analysis/symbolic execution
  - Highly extensible passes, allowing development of custom interpretation and modification of compilation units

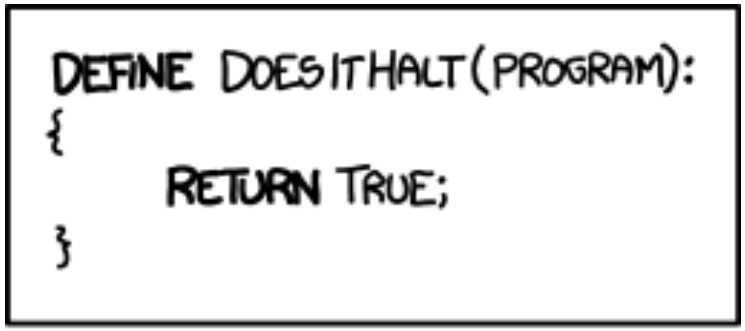
Goals

- **Create a series of diagnostic tools to gather relevant data***
  - Input language
  - SLOC (Source Lines of Code)
  - Minimum machine power classification

  * the halting problem is a harsh mistress. Some manual analysis required!

Goals II

- **Use LLVM passes to modify code**
  - Is the code able to be expressed in a sub-TC after modifications?



THE BIG PICTURE SOLUTION
TO THE HALTING PROBLEM

Technical Approach

- **Loops**
  - Loops with a defined bound for their induction variable and no other internal variable modifiers terminate
  - What about more ambiguous bounds?

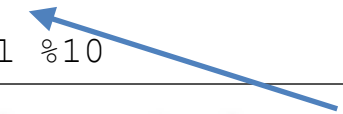
- **Recursion***
  - If we can unwind with LLVM then it counts as sub-TC, otherwise manually analyze or count as TC to be safe

* Not really a factor here, as we'll see later…

Pass Example


```
for (i = 0 ; i < 10 ; i++)
```

```
; <label>:3                                     ; preds = %7, %0
  %4 = load i32, i32* %i, align 4
  %5 = icmp slt i32 %4, 10
  br i1 %5, label %6, label %10
```



induction variable is defined and known to have an upper bound.

Internal induction variable modifications are simplified with llvm alias analysis

What about something a little more complicated?

Pass Example II


```
int buff[5] = {0,1,2,3,4};
for (i = 0 ; i < buff[2] ; i++)
```

```
; <label>:3                                          ; preds = %9, %0
  %4 = load i32, i32* %i, align 4
  %5 = getelementptr inbounds [5 x i32], [5 x i32]* %buff, i64 0, i64 2
  %6 = load i32, i32* %5, align 8
  %7 = icmp slt i32 %4, %6
  br i1 %7, label %8, label %12
```

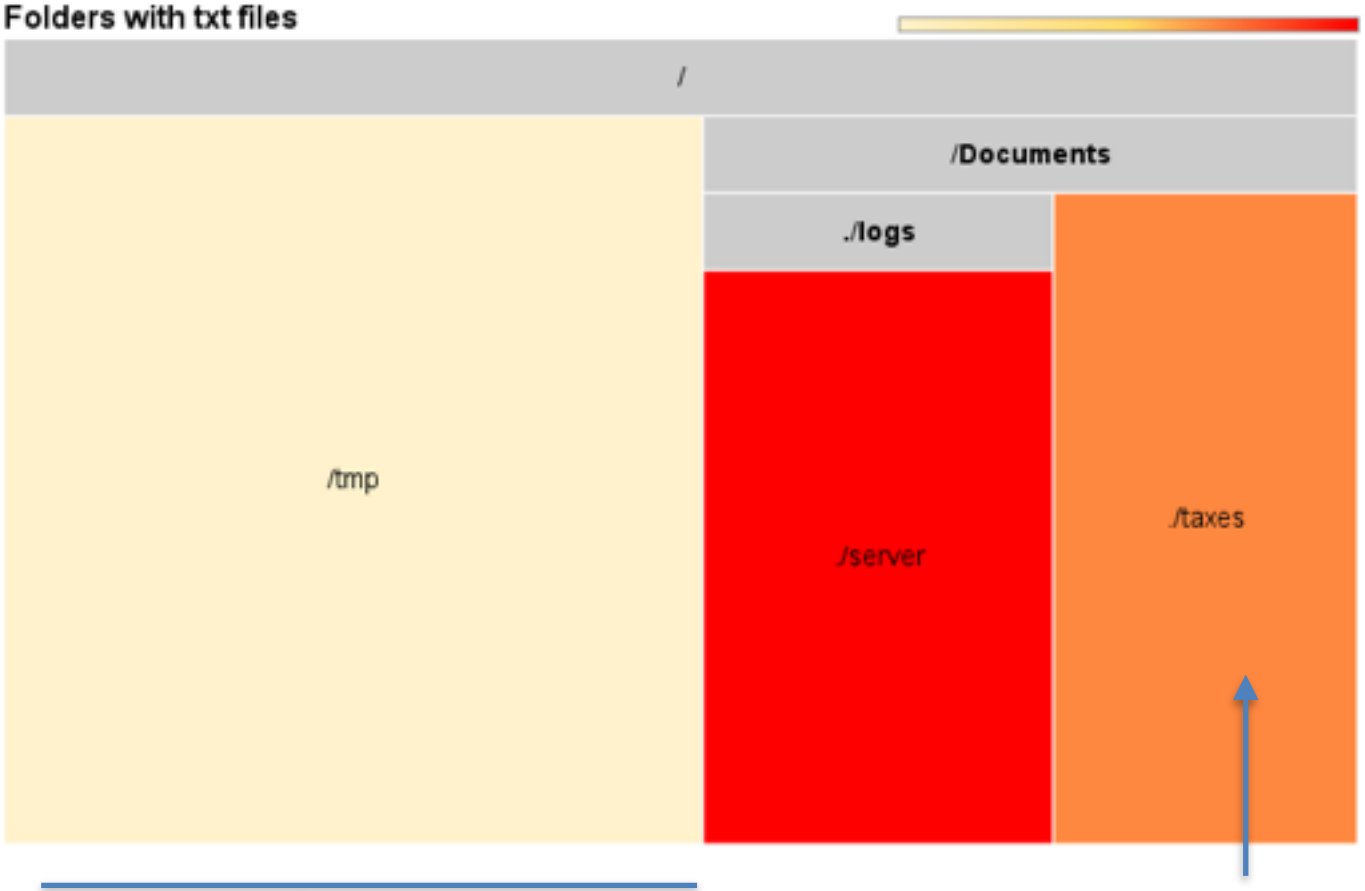
No Argument Promotion!

After promotion pass:

```
%4 = load i32, i32* %i, align 4
%5 = getelementptr inbounds [5 x i32], [5 x i32]* %buff, i64 0, i64 2
%6 = load i32, i32 * %4, align 8
; <label>:3                                          ; preds = %9, %0
  %7 = load i32, i32* %5, align 8
  %8 = icmp slt i32 %4, %6
  br i1 %8, label %9, label %12
```

Argument promoted!

Graphs are Hard!

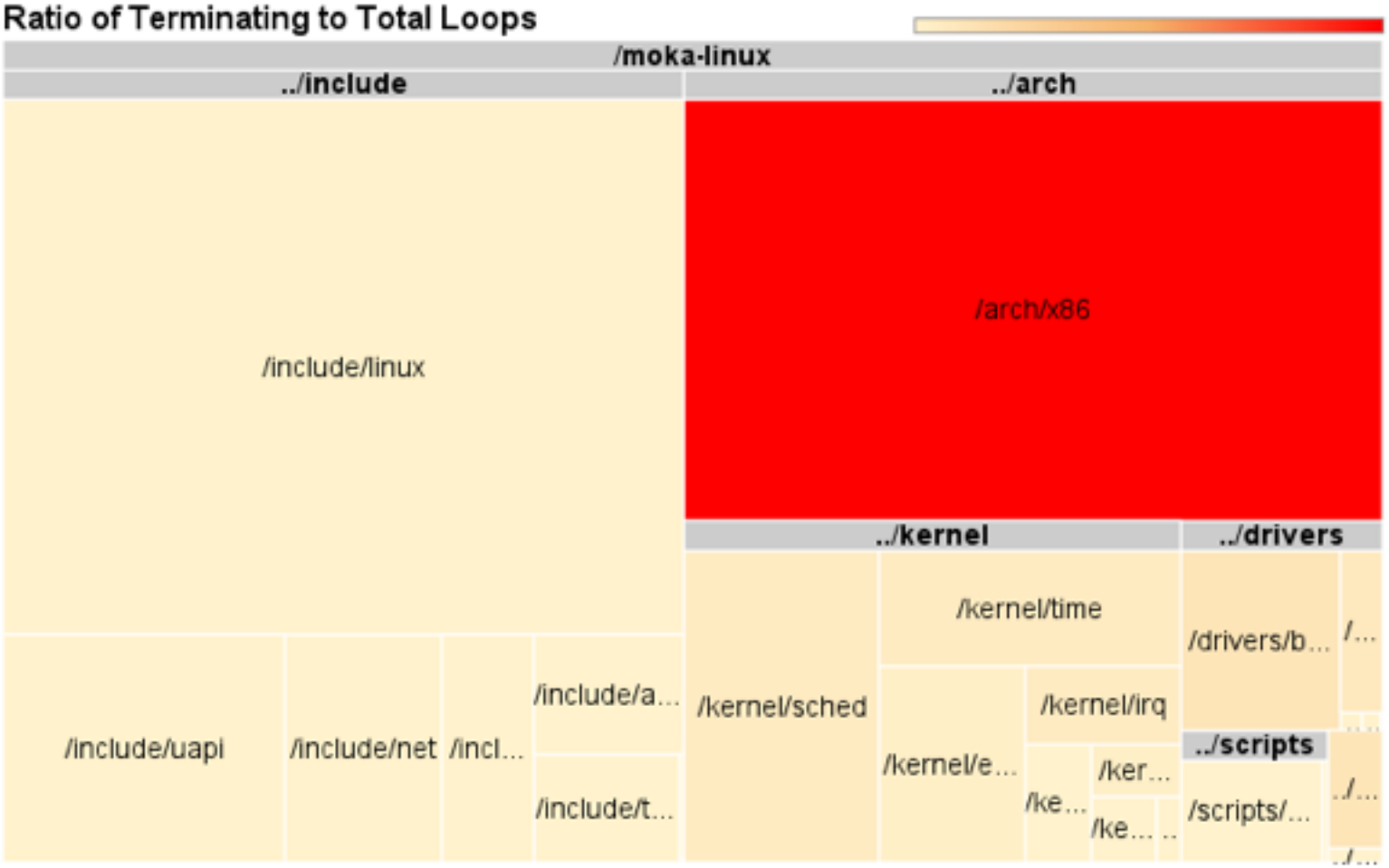



```
/
  /tmp
    foo.json
    passwords.pdf
    … (no txt)
  /Documents
    /taxes
      SSID.jpg
      expenses.txt
    /logs
      /server
        23986129.txt
        10298182.txt
        … (lots more txt)
```

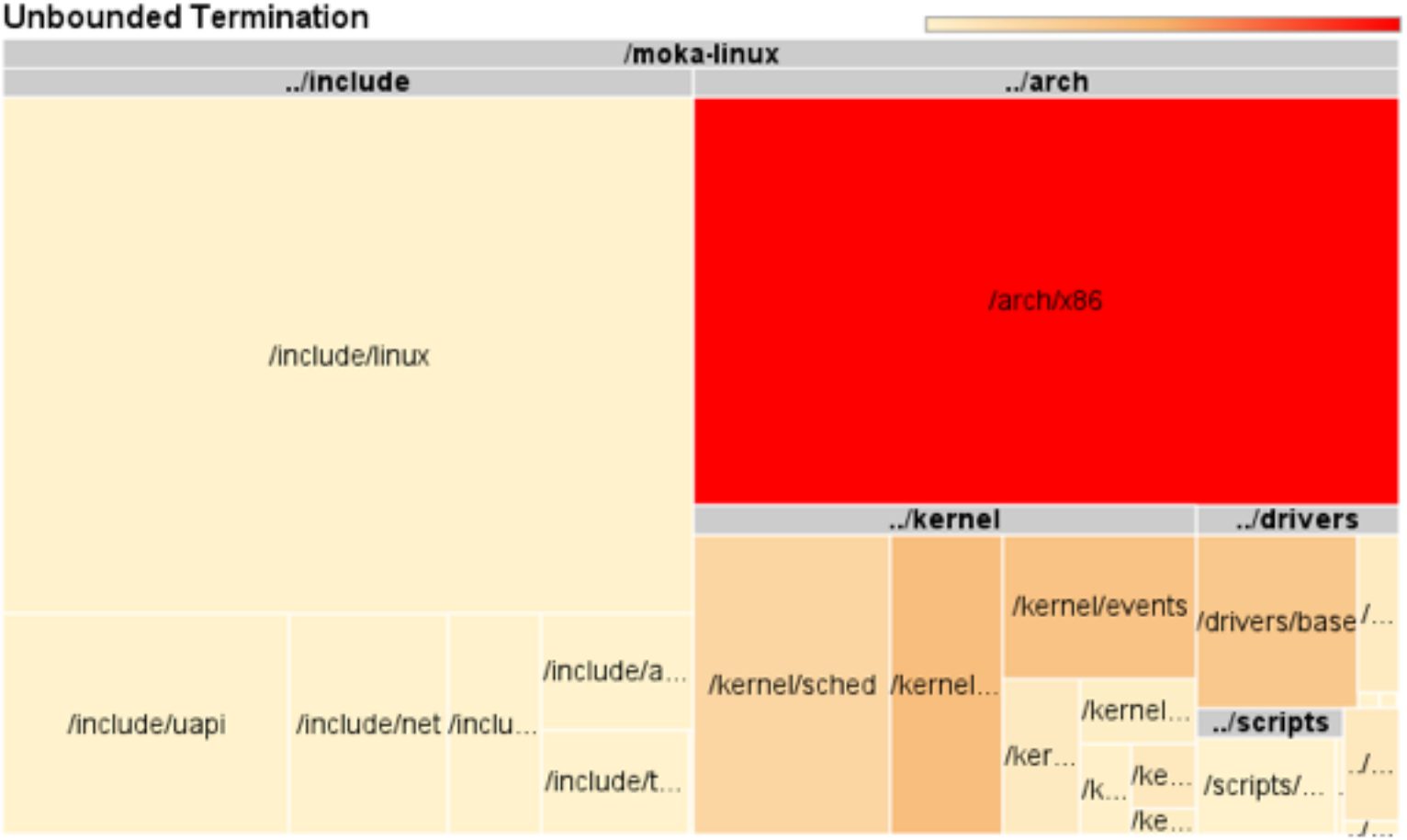
Area of Rectangle will ALWAYS be Source Lines Of Code (SLOC)

Saturation is concentration with respect to graph title

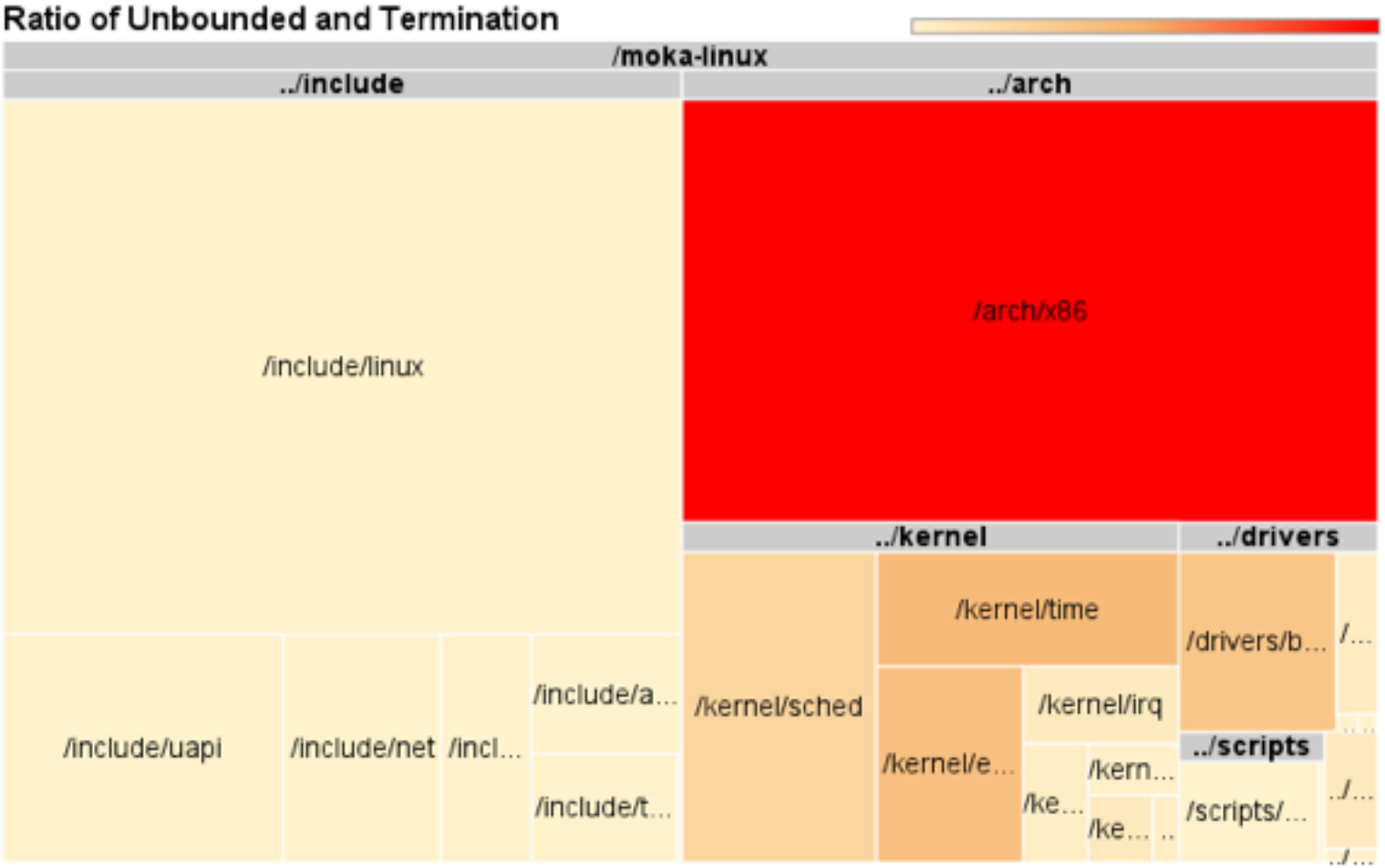
Sub-TC


Ratio of Terminating to Total Loops
/moka-linux
../include
../arch
/include/linux
/arch/x86
../kernel
../drivers
/kernel/time
/drivers/b...
/...
/include/uapi
/include/net
/incl...
/include/a...
/include/t...
/kernel/sched
/kernel/e...
/kernel/irq
/ke...
/ker...
/ke...
../scripts
/scripts/...
../...
../...

Unknown (So Far)



Sub-TC Potential



Results So Far

SLOC	354337
% ANSI C	~96%
Number of Loops	2473
Terminating Loops	305
Undecided (need evaluation)	2168
Functions w/ Undecided Loops Vs Total Functions (%)	23%

Summary / Follow-up

- **Much of the linux kernel already can be run sub-TC**
- **It is possible to determine if loops can be run sub-TC or modify them to run with lower computational power**
- **Look at a different domain (Eg. Application Servers)**
  - More recursive calls
  - Less refinement than kernel

Acknowledgements

- DARPA and Tim Fraser for sponsoring this research effort

- Sergey Bratus for technical input and support of all LangSec pursuits

Questions/Discussion

devastating capability, revolutionary advantage

Twitter

@jmiodownik

Email

miodownikj@ainfosec.com