Automatic OS Kernel TCB Reduction by Leveraging Compile-Time Configurability

Reinhard Tartler, Anil Kurmus, Andreas Ruprecht, Bernhard Heinloth, Valentin Rothberg, Daniela Dorneanu, Rüdiger Kapitza, Wolfgang Schröder-Preikschat, Daniel Lohmann

System Software Group Friedrich-Alexander University Erlangen-Nuremberg
IBM Research - Zurich
TU Braunschweig

October 7, 2012

supported by DFG and TClouds

System Software is Configurable

- Especially Linux is incredibly configurable

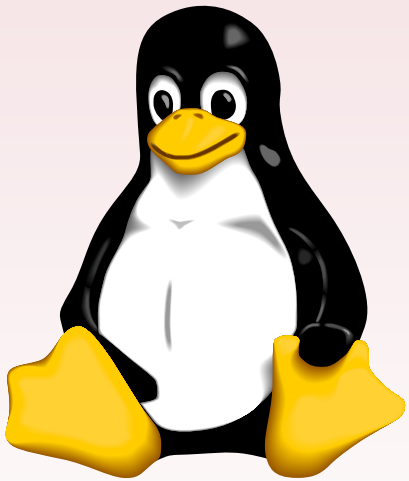
System Software is Configurable

- Especially Linux is incredibly configurable
- Complexity increases considerably

System Software is Configurable

- Especially Linux is incredibly configurable
- Complexity increases considerably
- How to find the best configuration?

System Software is Configurable



Linux v3.2 contains:

11.800 features

1.700 build-system files

18.300 source-code files

89.000 `#ifdef` blocks

How much functionality does Linux come with?

- Configuring Linux is a hard task
  - The Linux kernel contains ample functionality:
    - 24 Architectures (x86, arm, powerpc, ...)
    - Drivers
    - (Default) scheduling strategies
    - Optional security features
    - You name it!
  - Hard to see what your application exactly requires

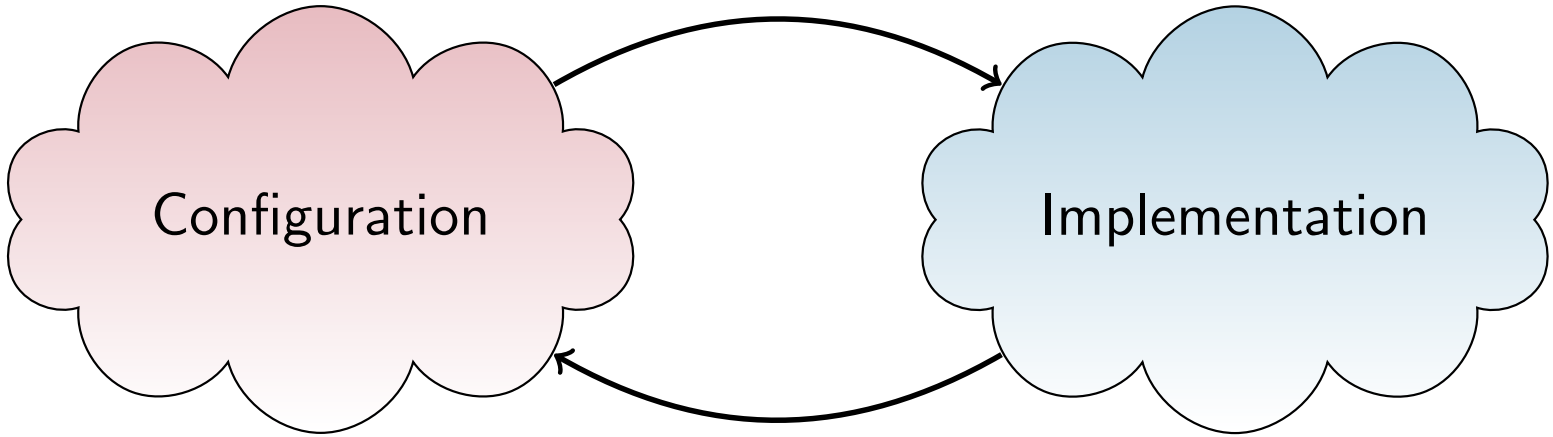
How much functionality does Linux come with?

- Configuring Linux is a hard task
  - The Linux kernel contains ample functionality:
    - 24 Architectures (x86, arm, powerpc, ...)
    - Drivers
    - (Default) scheduling strategies
    - Optional security features
    - You name it!
  - Hard to see what your application exactly requires
- Configuration options, may appear or disappear with every new Linux kernel release

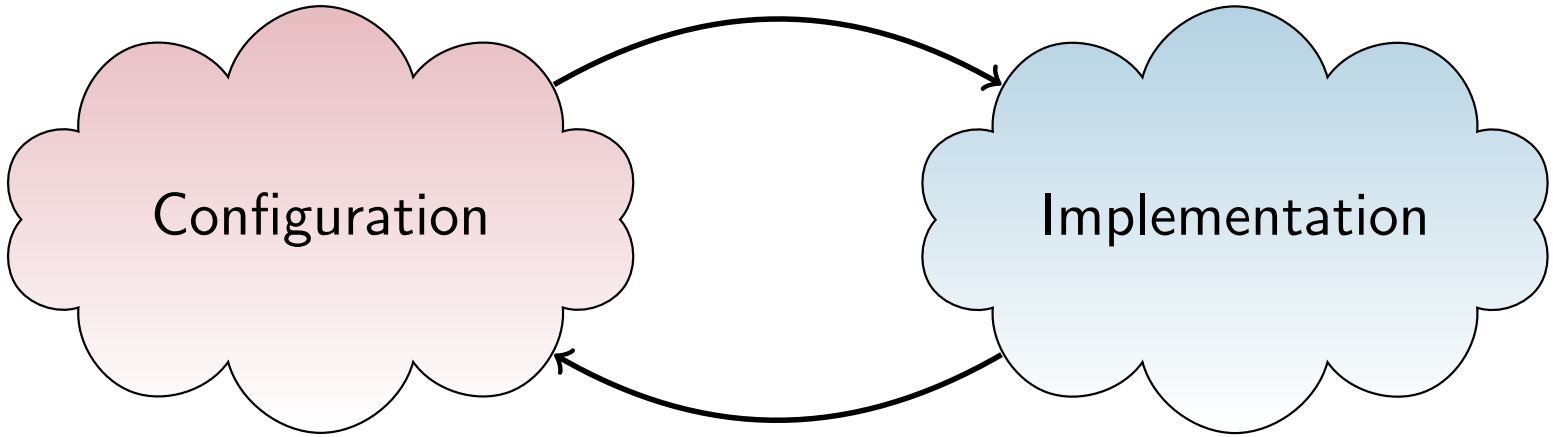
How much functionality does Linux come with?

- Configuring Linux is a hard task
  - The Linux kernel contains ample functionality:
    - 24 Architectures (x86, arm, powerpc, ...)
    - Drivers
    - (Default) scheduling strategies
    - Optional security features
    - You name it!
  - Hard to see what your application exactly requires
- Configuration options, may appear or disappear with every new Linux kernel release
- Distribution kernels mostly do work
  - Distros invest an enormous amount of manpower to keep up
  - Make little assumptions on hardware and use case

A Closer Look at the State of the Art

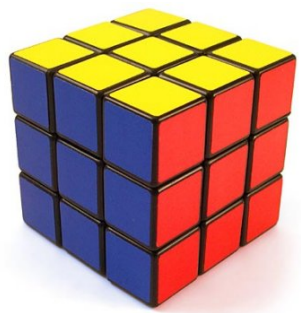


A Closer Look at the State of the Art



Single Configuration provided by the Distributor

Complex Implementation of Variability

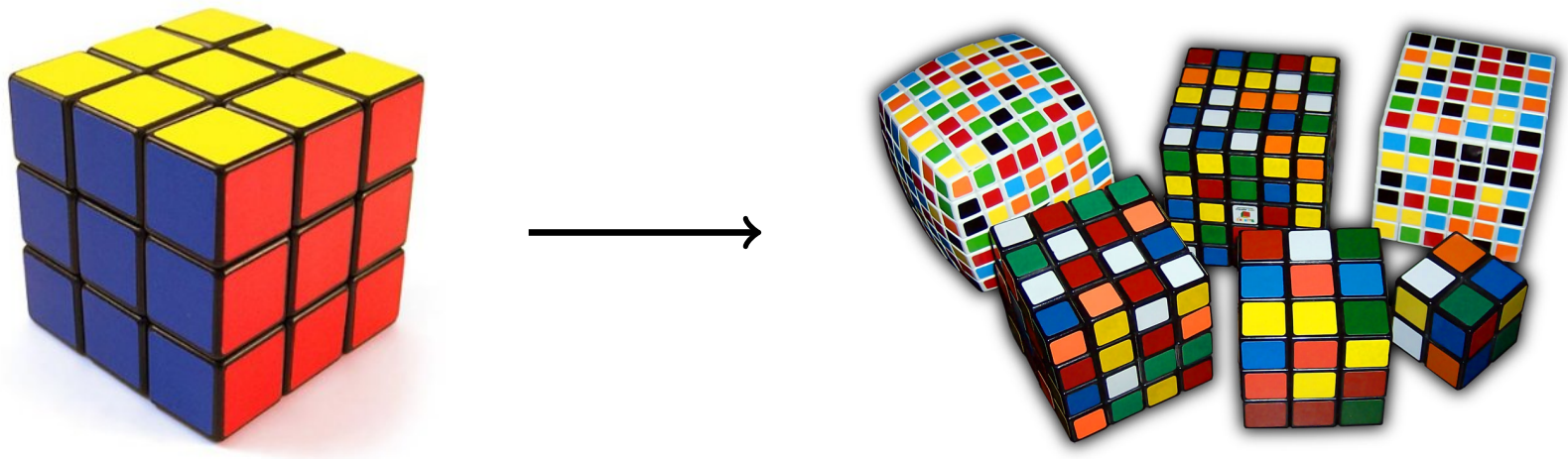


KCONFIG

`#ifdef`-Hell
Convoluted `make` files
etc.

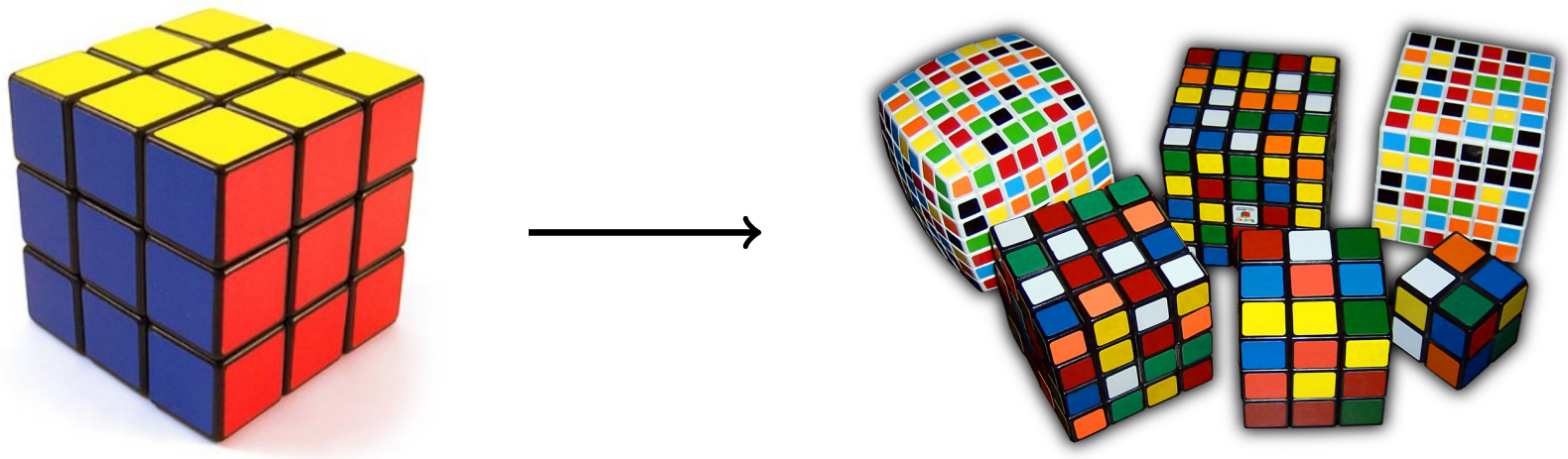
The Problem

- Without a given use-case, the distribution kernel has to include all available functionality
- As side-effect, this maximizes the attack surface!
- Each use-case needs its specific, ideal configuration



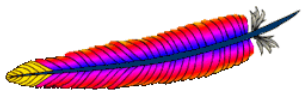
Idea: Automated Tailoring of the Linux Configuration

- Without a given use-case, the distribution kernel has to include all available functionality
- As side-effect, this maximizes the attack surface!
- Each use-case needs its specific, ideal configuration



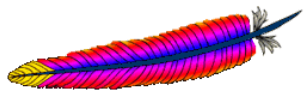
→ Automatically derive an ideal configuration for a given use case.

Approach at a glance



Specific
Scenario

Approach at a glance

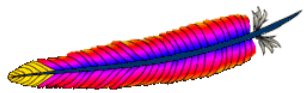


Specific Scenario

CONFIG_X86=y
CONFIG_SCSI=n
[. . .]

Tailored Configuration

Approach at a glance



Specific Scenario

automatically derive

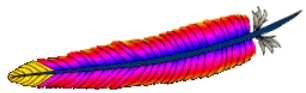
CONFIG_X86=y
CONFIG_SCSI=n
[. . .]

Tailored Configuration

observe

FTRACE

Approach at a glance



Specific Scenario

automatically derive

CONFIG_X86=y
CONFIG_SCSI=n
[...]

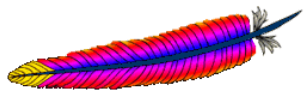
Tailored Configuration

debug symbols

```
0x8043566 — kern/sched.c:80
0x80452d8 — drivers/scsi.c:4302
[...5000 more locations]
```

Identify in Source Code

Approach at a glance



Specific Scenario

automatically derive

CONFIG_X86=y
CONFIG_SCSI=n
[...]

Tailored Configuration

0x8043566 — kern/sched.c:80
0x80452d8 — drivers/scsi.c:4302
[...5000 more locations]

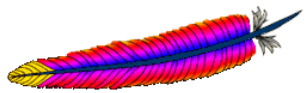
Identify in Source Code

establish

φ

Holistic Variability Model

Approach at a glance



Specific Scenario

automatically derive

CONFIG_X86=y
CONFIG_SCSI=n
[...]

Tailored Configuration

employ SAT checker

φ

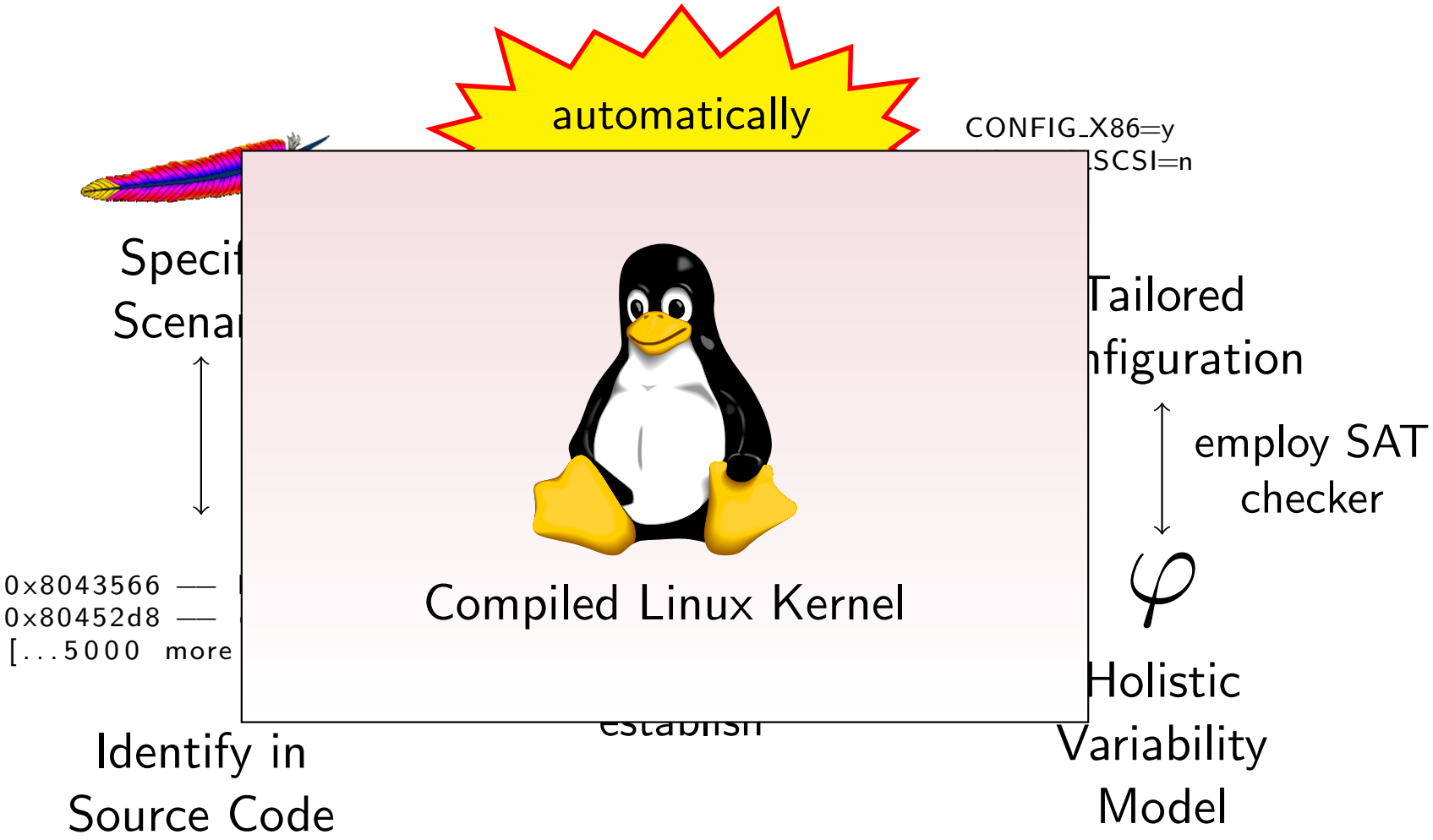
```
0x8043566 — kern/sched.c:80
0x80452d8 — drivers/scsi.c:4302
[...5000 more locations]
```

Identify in Source Code

establish

Holistic Variability Model

Approach at a glance



Challenges

- Trace the required functionality from the application
  - Traces need to be sufficiently complete

→ Best results with *fairly homogeneous services*

Challenges

- Trace the required functionality from the application
  - Traces need to be sufficiently complete

  → Best results with *fairly homogeneous services*

- Kernel with `ftrace` and debugging information available
  - We have improved our implementation by now

Challenges

- Trace the required functionality from the application
  - Traces need to be sufficiently complete

  → Best results with *fairly homogeneous services*

- Kernel with `ftrace` and debugging information available
  - We have improved our implementation by now

- The holistic variability model must include constraints from:
  - KCONFIG feature dependencies among each other
  - MAKE files (→ what KCONFIG feature compiles this file)
  - complicated CPP expressions and nested `#ifdef` blocks

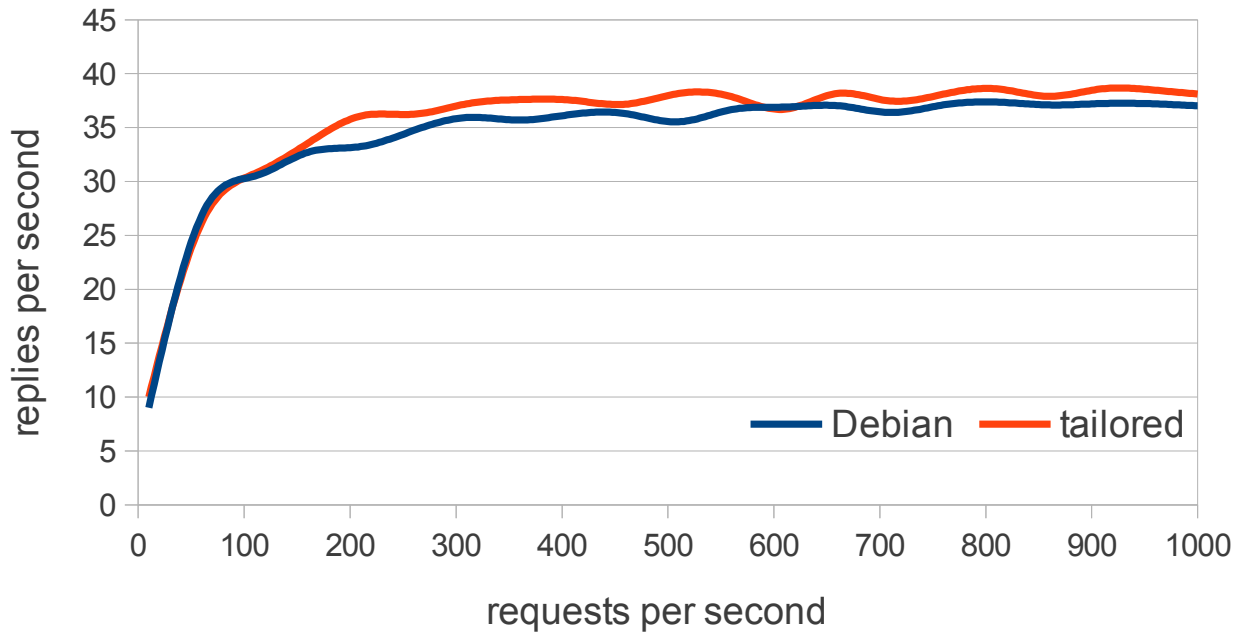
  → solved in previous work by the UNDERTAKER tool

Evaluation

- Standard LAMP (Linux, MySQL, Apache, PHP) System
  - Standard setup with DOKUWIKI and PHPBB3
  - Google Skipfish to systematically trigger all functionality
- Trace contains 5.377 unique kernel functions
- Transformation into a configuration takes 69 seconds

Evaluation

- Standard LAMP (Linux, MySQL, Apache, PHP) System
  - Standard setup with DOKUWIKI and PHPBB3
  - Google Skipfish to systematically trigger all functionality
- Trace contains 5.377 unique kernel functions
- Transformation into a configuration takes 69 seconds
- No performance impact observable:



Reduction of enabled features

Kernel Shipped by Debian	
Loaded Kernel Modules	29
KCONFIG options set to `y`	1,093
KCONFIG options set to `m`	2,299
Functions with CVE entries	179
Intermediary kernel used for tracing	
Loaded Kernel Modules	0
KCONFIG options set to `y`	3,298
KCONFIG options set to `m`	0
Resulting application-tailored kernel	
Loaded Kernel Modules	0
KCONFIG options set to `y`	379
KCONFIG options set to `m`	0
Functions with CVE entries	162

Reduction of enabled features

Kernel Shipped by Debian	
Loaded Kernel Modules	29
KCONFIG options set to `y`	1,093
KCONFIG options set to `m`	2,299
Functions with CVE entries	179
Intermediary kernel used for tracing	
Loaded Kernel Modules	0
KCONFIG options set to `y`	3,298
KCONFIG options set to `m`	0
Resulting application-tailored kernel	
Loaded Kernel Modules	0
KCONFIG options set to `y`	379
KCONFIG options set to `m`	0
Functions with CVE entries	162

$\rightarrow$ 10 percent less functions with known vulnerabilities
(using a semi-automated process to scan published CVE issues)

Code Reduction

Kernel Shipped by Debian	
Loaded Code	5,465,602 Bytes
Total Loadable Code	42,188,538 Bytes
Intermediary kernel used for tracing	
Loaded Code	36,341,888 Bytes
Total Loadable Code	36,341,888 Bytes
Resulting application-tailored kernel	
Loaded Code	3,990,153 Bytes
Total Loadable Code	3,990,153 Bytes

Code Reduction

Kernel Shipped by Debian	
Loaded Code	5,465,602 Bytes
Total Loadable Code	42,188,538 Bytes
Intermediary kernel used for tracing	
Loaded Code	36,341,888 Bytes
Total Loadable Code	36,341,888 Bytes
Resulting application-tailored kernel	
Loaded Code	3,990,153 Bytes
Total Loadable Code	3,990,153 Bytes

$> 90\%$ less executable code

Results and Future Work

- TCB is significantly smaller
- Easy to use: process is fully automated after tracing
- If necessary, the tailoring can guided with whitelists and blacklists
- Going further:
  - More use-cases, also on *real hardware*
  - Necessity of dedicated kernel for tracing?
  - What config option shall be configured as module and what statically?
  - Better metrics for measuring the attack surface improvements

Results and Future Work

- TCB is significantly smaller
- Easy to use: process is fully automated after tracing
- If necessary, the tailoring can guided with whitelists and blacklists
- Going further:
  - More use-cases, also on *real hardware*
  - Necessity of dedicated kernel for tracing?
  - What config option shall be configured as module and what statically?
  - Better metrics for measuring the attack surface improvements

Questions?