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?

R. Tartler Automatic OS Kernel TCB Reduction by Leveraging Configurability (HOTDEP’12)
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

Configuration

Implementation

R. Tartler  Automatic OS Kernel TCB Reduction by Leveraging Configurability (HOTDEP’12)
A Closer Look at the State of the Art

Single Configuration provided by the Distributor

Complex Implementation of Variability

KCONFIG

#define-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

automatically derive

CONFIG_X86=y
CONFIG_SCSI=n
[...]

Tailored Configuration
Approach at a glance

Specific Scenario

automatically derive

CONFIG_X86=y
CONFIG_SCSI=n
[...]

Tailored Configuration

Identify in Source Code automatically derive ftrace observe debug symbols

ϕ

Holistic Variability Model establish employ SAT checker
Approach at a glance

Specific Scenario

downward arrow

debg symbols

Tailored Configuration

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

Identify in Source Code

automatically derive

CONFIG_X86=y
CONFIG_SCSI=n
[...]

R. Tartler  Automatic OS Kernel TCB Reduction by Leveraging Configurability (HOTDEP’12) 6-11
Approach at a glance

Specific Scenario

Identify in Source Code

Tailored Configuration

automatically derive

CONFIG_X86=y
CONFIG_SCSI=n
[...]

\( \varphi \)

Holistic Variability Model

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

Specific Scenario

Identify in Source Code

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

automatically derive

CONFIG_X86=y
CONFIG_SCSI=n
[...]

Tailored Configuration

employ SAT checker

Holistic Variability Model

establish

R. Tartler Automatic OS Kernel TCB Reduction by Leveraging Configurability (HOTDEP’12)
Approach at a glance

Identify in Source Code

Compiled Linux Kernel

Specific Scenario

Tailored Configuration

Automatically

CONFIG_X86=y
CONFIG_SCSI=n

ϕ

Holistic Variability Model

employ SAT checker

establish

identify in

Source Code

\[\ldots 5000 \text{ more} \]

\[0x8043566 \quad \text{ker/sched.c:80}\]

\[0x80452d8 \quad \text{drivers/scsi.c:4302}\]
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 \texttt{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:

![Graph showing replies per second over requests per second]

Debian tailored
### 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

→ 10 percent less functions with known vulnerabilities (using a semi-automated process to scan published CVE issues)
Reduction of enabled features

Kernel Shipped by Debian
- Loaded Kernel Modules: 29
- \texttt{KCONFIG} options set to \texttt{y}: 1,093
- \texttt{KCONFIG} options set to \texttt{m}: 2,299
- Functions with CVE entries: 179

Intermediary kernel used for tracing
- Loaded Kernel Modules: 0
- \texttt{KCONFIG} options set to \texttt{y}: 3,298
- \texttt{KCONFIG} options set to \texttt{m}: 0

Resulting application-tailored kernel
- Loaded Kernel Modules: 0
- \texttt{KCONFIG} options set to \texttt{y}: 379
- \texttt{KCONFIG} options set to \texttt{m}: 0
- Functions with CVE entries: 162

→ 10 percent less functions with known vulnerabilities
(using a semi-automated process to scan published CVE issues)
## Code Reduction

<table>
<thead>
<tr>
<th>Stage</th>
<th>Loaded Code</th>
<th>Total Loadable Code</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Kernel Shipped by Debian</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Loaded Code</td>
<td>5,465,602 Bytes</td>
<td>42,188,538 Bytes</td>
</tr>
<tr>
<td>Total Loadable Code</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Intermediary kernel used for tracing</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Loaded Code</td>
<td>36,341,888 Bytes</td>
<td>36,341,888 Bytes</td>
</tr>
<tr>
<td>Total Loadable Code</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Resulting application-tailored kernel</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Loaded Code</td>
<td>3,990,153 Bytes</td>
<td>3,990,153 Bytes</td>
</tr>
<tr>
<td>Total Loadable Code</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

90% less executable code
### Code Reduction

<table>
<thead>
<tr>
<th>Kernel Shipped by Debian</th>
<th>Loaded Code</th>
<th>5,465,602 Bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Total Loadable Code</td>
<td>42,188,538 Bytes</td>
</tr>
<tr>
<td>Intermediary kernel used for tracing</td>
<td>Loaded Code</td>
<td>36,341,888 Bytes</td>
</tr>
<tr>
<td></td>
<td>Total Loadable Code</td>
<td>36,341,888 Bytes</td>
</tr>
<tr>
<td>Resulting application-tailored kernel</td>
<td>Loaded Code</td>
<td>3,990,153 Bytes</td>
</tr>
<tr>
<td></td>
<td>Total Loadable Code</td>
<td>3,990,153 Bytes</td>
</tr>
</tbody>
</table>

> 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 be guided with whitelists and blacklists

Going further:
- More use-cases, also on real hardware
- Necessity of a 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 be 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?