Configurability Bugs in Linux

The 10000 Feature Challenge

Reinhard Tartler, Julio Sincero, Wolfgang Schröder-Preikschat, Daniel Lohmann
Friedrich–Alexander University Erlangen–Nuremberg
{tartler,sincero,wosch,lohmann}@cs.fau.de

Reinhard and Julio are PhD students.

If accepted, we will demo our tool in action, finding real defects and bugs in an up-to-date Linux snapshot.

1. INTRODUCTION AND MOTIVATION

Most system software can be configured at compile time to tailor it with respect to a broad range of supported hardware architectures and application domains; the Linux kernel, for example, provides more than 10000 configurable features, growing rapidly.
From the maintenance point of view, compile-time configurability imposes big challenges. The selectable features (and their constraints!), which are presented to the user and the configurability that is *actually* implemented in the code, have to be kept in sync, which, if performed manually, is a tedious and error-prone task. In the case of Linux, this has led to numerous defects in the source code, many of which are actual bugs. In our work, we target at:

1. checking for configurability-related implementation defects under the consideration of symbolic *and* logic integrity.
2. calling for attention on the maintenance problems caused by the increasing configurability of system software.
3. providing a practical and scalable tool chain that has detected **1615** configurability-related bugs and inconsistencies from the latest version of Linux; meanwhile **205** of our bug fixes have already been confirmed by Linux kernel developers.

2. ANALYZING CONFIGURATION CONSISTENCY

Linux maintains its configuration options in a dedicated tool and language called *Kconfig*, in which the user can graphically customize the kernel. In this tool, kernel developers declare constraints and dependencies to ensure a consistent configuration. However, as our results show, this consistency does not necessarily apply to the implementation of conditionally-compiled code fragments using the C Preprocessor (CPP). We translate both *Kconfig* and CPP into propositional formulas so that we can use a SAT solver to automatically find *configuration defects* in Linux. Our tooling, the *undertaker*, thereby finds three types of defects:

Code-Code defects are blocks that cannot be selected or unselected because of a contradiction in their presence condition, such as in the following example:

```
#ifdef CONFIG_MMU
... // some pages of source code
#ifndef CONFIG_MMU
... // this block is dead
```

Subsystem	#ifdefs	Code-Code	Missing	Kconfig
arch/	31792	18	917	18
drivers/	30966	37	2450	10
fs/	2938	4	52	0
sound/	3042	2	102	0
other subsystems	11857	5	222	3

Table 1: configuration defects per subsystem for Linux version 2.6.33.

Kconfig defects are blocks for which there is no valid *Kconfig* selection that causes the block to be selected or unselected during compilation.

Missing defects are defects that result from identifiers that cannot be referenced from the source code to Kconfig or vice versa, such as in the following patch:

```
diff --git a/kernel/smp.c b/kernel/smp.c
--- a/kernel/smp.c
+++ b/kernel/smp.c

 -#ifdef CONFIG_CPU_HOTPLUG
 +#ifdef CONFIG_HOTPLUG_CPU
```

In our evaluation experiment, we had two undergraduate students analyze the defect reports produced by our tooling and have them propose patches to kernel maintainers. A total of **132** patches have been submitted, from which **71** patches have been thankfully accepted by the Linux kernel maintainers so far.

3. CONCLUSIONS AND FUTHER RESEARCH QUESTIONS

With our tooling, we are now in a position to answer: How accurate is our approach and how relevant are our submissions? What impact do configuration defects have on the Linux kernel development? And has the distributed nature of the Linux kernel development a positive or negative impact on configuration defects?
From the feedback that we got from kernel developers, our experiment to produce and submit patches has proven to be very effective. From here, we look for a workflow that mitigates configuration defects. We believe that if our tooling gets integrated into the Linux build scripts and is used by developers during development and integration of changes, such configuration induced bugs can be detected and, at best, avoided in very early stages of kernel development.
Our work is not limited to the Linux kernel, but can also be applied to other operating systems, like eCos, from which configuration constraints can be extracted into propositional formulas.