A Quantitative Analysis of Variability Warnings in Linux

Elvis Flesborg

September 22nd 2015 – ITU – Copenhagen

About Presentation

- About project
- Background
- Methodology
- Results
- Threats to validity

About project

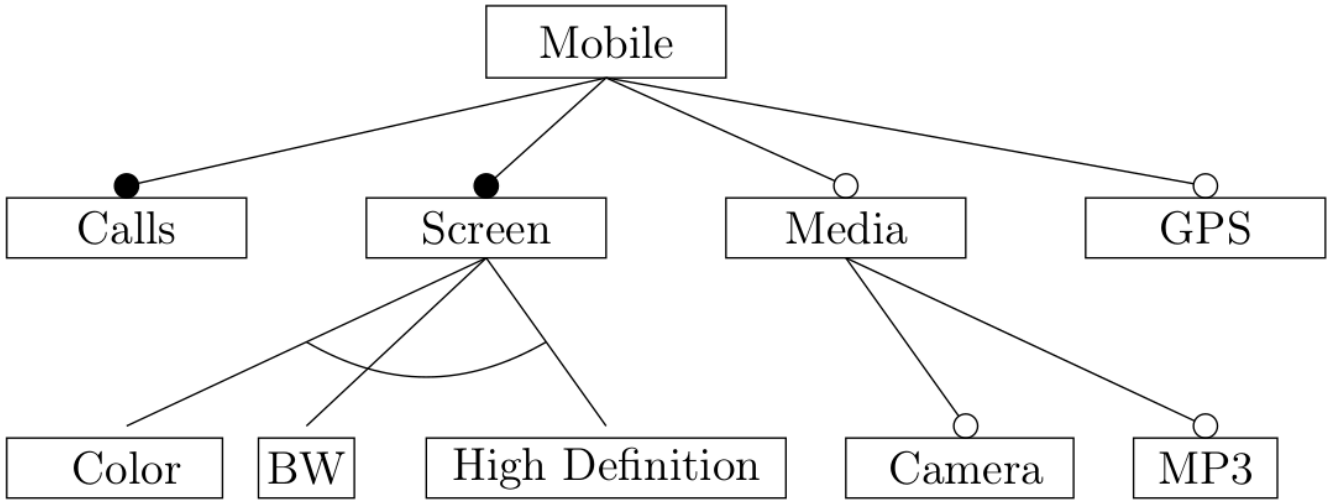
- 42 bugs paper (Brabrand, Abal et al.)
  - *Variability bugs* were qualitatively analyzed
  - Bias in selection
  - Nothing can be said quantitatively about the bugs
- Objective: Quantify the bugs
  - 2 different versions of Linux
    - Stable
    - In-development
  - Data on warning types
  - Data on warning locations (mainly subsystems)

Background

- Variability
- Feature models
- Linux
- Warning types

Background - Variability

- aka Software Product Lines
- One code base – many products
  - No need to create many different programs
- Example: Mobile phone



Media depends on **High Definition**

Background - Feature Models

- A way to represent the different variants
  - Features
  - Dependencies and restrictions
- Many other products use this
  - Busybox, Drupal, Eclipse…
- Often wide and shallow
- The language in Linux is called *Kconfig*
  - Is also used in Busybox amongst others

Background - Linux

- Large open source project
  - 19 million LOC
- Large degree of variability
  - 10,000 to 14,000 features
  - 2^(10,000) different variants
- Easily accessible
  - Free online
  - Lots of documentation

Background – Linux variability

- #ifdef
- Configurators
  - Allyesconfig
  - Allnoconfig
  - Randconfig
  - More …

```
1  int main(void) {
2      int var;
3      #ifdef CONFIG_A
4      var = 1;
5      #endif
6      #ifdef CONFIG_B
7      var = 2;
8      #endif
9      return var+100;
10 }
```

```
1  int main(void) {
2      int var;
3      var = 1;
4      return var+100;
5  }
```

Background – Warning types

- Wrong data
  - Stop compilation
  - Runtime bugs
- Code pollution
  - Confuse programmers
  - Bad for space limitation
- Bad code practice
  - Confuse programmers
- Irrelevant

Background – Warning types

- Array-bounds
- Cpp
- Deprecated-declarations
- frame-larger-than=N
- Implicit-function-declaration
- Int-to-pointer-cast
- Maybe-uninitialized

Background – Warning types

- Overflow
- Pointer-to-int-cast
- Return-type
- Uninitialized
- Unused-function
- Unused-variable
- Unused-label

Methodology – Experiment setup

- 1) Create random variant
- 2) Check for warnings
- 3) Analyze the data

Methodology – 1) Random variant

- Linux' built-in `randconfig` configurator
- Fast
  - 1 second
- Easy
  - `make randconfig`
- Sound
  - Only valid configurations
- (Not quite representative)

Methodoly – 2) Check for warnings

- *Gcc* testing is done when compiling
  - So all we have to do is compile the Linux variant
- *Gcc* outputs warnings
  - Collect the standard error output
- Search through the output for warnings
  - Easy to differentiate
  - Unique naming convention [-Wunused-function]

Methodology – 3) Analyze

- Analyzing the data
- Answer the research questions

Methodology – Research Questions

- **RQ1**: What warnings are the most common in the stable Linux kernel?
- **RQ2**: Where do most warnings occur?
- **RQ3**: Are there any differences between an in-development version of Linux and a stable version?

Methodology – info on experiment

- Was compiled at a HPC at ITU
  - And a spare laptop
- 42,000 variants were tested
  - 21,000 were stable version
  - 21,000 were in-development version
- On average 4.5 warnings per experiment
- Max was 111 warnings

Results

- Warnings in stable
- Locations in stable
- Stable vs. in-development

Results – Warnings in stable version

Warning	Percentage	Category
unused-function	59.%	Code pollution
maybe-uninitialized	45.%	Wrong data
unused-variable	29.%	Code pollution
cpp	24.%	Irrelevant
uninitialized	19.%	Wrong data
ERROR	17.%	Irrelevant
pointer-to-int-cast	17.%	Wrong data
frame-larger-than=	14.%	Irrelevant
array-bounds	11.%	Wrong data
return-type	7.7%	Bad Code Practice
int-to-pointer-cast	7.6%	Wrong data
overflow	6.5%	Wrong data
unused-label	5.4%	Code Pollution
deprecated-declarations	5.4%	Wrong data
implicit-function-declaration	5.6%	Bad code practice

- Observation: Unused function/variable is in the top 3
- Observation: Uninitialized is the top of the Wrong Data group

- Conclusion: Code Pollution in top, then uninitialized warnings

Results – Locations in stable version

Subsystem	Percentage
drivers/	64.%
include/	40.%
crypto/	17.%
fs/	14.%
samples/	12.%
net/	10.%
arch/	9.2%
arch/x86/	9.2%
lib/	9.1%
mm/	7.9%
kernel/	5.9%
sound/	3.8%
scripts/	1.6%
usr/	.076%
block/	.75%
security/	.0%

- Observation: The *drivers*/ and *include*/ subsystems are at the top
- Observation: *security*/ has zero

- Conclusion: *drivers*/ has most, *security*/ has least.

Results – Stable vs. in-development

Warning	%
unused-function	59.%
unused-variable	29.%
ERROR	17.%
frame-larger-than=	14.%
int-to-pointer-cast	7.6%
implicit-function-declaration	5.6%

(a) Stable version

Warning	%
unused-function	62.%
unused-variable	51.%
ERROR	38.%
int-to-pointer-cast	25.%
implicit-function-declaration	23.%
frame-larger-than=	7.8%

(b) In-development version

Observation: 4 of the warning types occur more in the in-development version
Observation: 1 of the warning types occur more in the stable version

Conclusion: The in-development version has more warnings

Results – Stable vs. in-development

Subsystem	Percentage
arch/	9.2%
arch/x86/	9.2%
mm/	7.9%
kernel/	5.9%
sound/	3.8%
scripts/	1.6%

(a) Stable version

Subsystem	Percentage
scripts/	25.%
arch/	14.%
arch/x86/	14.%
mm/	13.%
kernel/	3.0%
sound/	1.5%

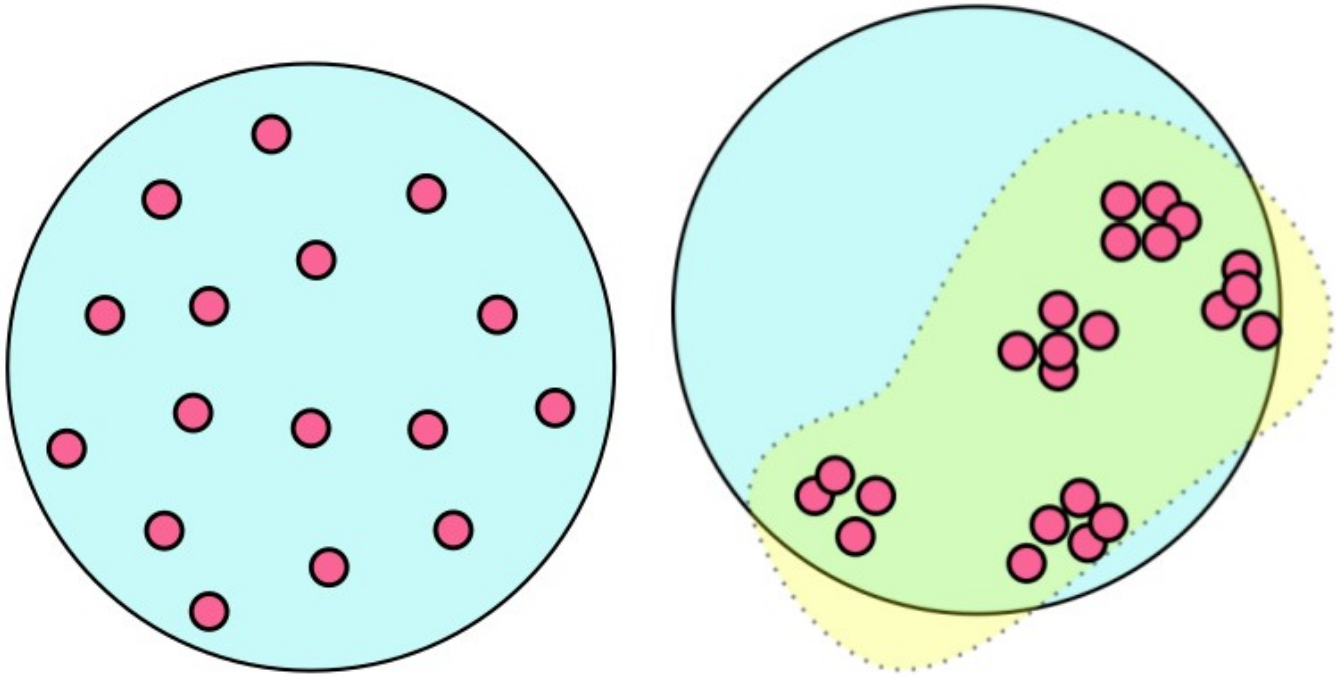
(b) In-development version

Observation: 2 of the larger subsystems contain more warnings in the in-dev
Observation: 2 of the larger subsystems contain more warnings in the stable

Conclusion: The change is not as large as with warning types

Threats to Validity - Internal

- 2 Gcc versions
  - Newer may have found more warnings
- Representativeness



(a) A representative sample (b) An unrepresentative sample

Threats to Validity - External

- Only one architecture
  - Cross-compilation can be done
- Not enough in-development versions
  - Only 9 different versions. More would even out.
- Not compiling with firmware

The end