

Towards producing a ground-truth dataset of safety bugs in the Linux Kernel

Michel Maes-Bermejo¹, Gregorio Robles², Jesus M. Gonzalez-Barahona²,
Micael Gallego¹, Daniel Izquierdo-Cortazar³ and Paul Sherwood⁴

¹Department of Computer Science, Universidad Rey Juan Carlos, Spain

²Department of Telematic and Computational Systems Engineering Universidad Rey Juan Carlos, Spain

³Bitergia S.L., Madrid, Spain

⁴Codethink Ltd.

Abstract

The automotive industry is interested in certifying free/open source components with the ISO 26262 standard, which is a risk-based standard that addresses the damage that may be caused to the electronic and electrical (E/E) systems of vehicles in the event of failures. We hypothesize that the techniques, processes, and controls developed by widely used open source projects, such as the Linux Kernel, are successful in catching and avoiding bugs that are considered to be critical in safety standards. We want to test our hypothesis by analyzing historic records of bugs found in production releases of the Linux Kernel. Therefore, we first aim to create a ground-truth dataset of safety bugs in the Linux Kernel and characterize them. So, we have conducted a preliminary study to identify, manually locate and extract the most relevant information from the last 100 Bug-Fixing Commits (BFC) in the most recent commits of the Linux Kernel repository. We have found, among others, that out of the 100 BFCs, 80 have a link to the Bug-Introduction Commit (BIC); most errors are in the “drivers” module, and we were able to confirm 9 regressions. The next step in this research is to classify the bugs to detect and correctly classify those that are safety-related. As the classification is difficult to perform and requires specialized knowledge of the Linux Kernel development, we plan to get help from Linux Kernel developers to validate our identification and to manually classify the bugs in one of the defined categories.

Keywords

linux kernel, safety bugs, bug analyzing

1. Introduction

There is a strong industrial interest in certifying free/open source components (such as the Linux Kernel) with common automobile standards, specifically ISO 26262. ISO 26262 is intended to be applied to safety-related systems that include one or more electrical and/or electronic (E/E) systems and that are installed in series production passenger cars with a maximum gross vehicle mass up to 3,500 kg. [2]. The objective of ISO 26262 is to address the damage that may

BENEVOL'23: Belgium-Netherlands Software Evolution Workshop, 27-28 November, 2023, Nijmegen, NE


✉ michel.maes@urjc.es (M. Maes-Bermejo); gregorio.robles@urjc.es (G. Robles); jesus.gonzalez.barahona@urjc.es (J. M. Gonzalez-Barahona); micael.gallego@urjc.es (M. Gallego); dizquierdo@bitergia.com (D. Izquierdo-Cortazar); paul.sherwood@codethink.co.uk (P. Sherwood)

🆔 0000-0002-8138-9702 (M. Maes-Bermejo); 0000-0002-1442-6761 (G. Robles); 0000-0001-9682-460X

(J. M. Gonzalez-Barahona); 0000-0002-2875-7342 (M. Gallego); 0000-0002-0633-4146 (D. Izquierdo-Cortazar)



© 2023 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

 CEUR Workshop Proceedings (CEUR-WS.org)

be caused to the electronic and electrical (E/E) systems of vehicles in the event of failures. The ISO 26262 standard is risk-based, that is, it evaluates possible damage qualitatively and defines security measures to reduce, control or avoid system failures.

Our hypothesis is that *the techniques, processes and controls developed by widely used open source projects, such as the Linux Kernel, are successful in catching and avoiding the kinds of bugs that are considered to be critical in those safety standards*. Furthermore, bugs that do get into production, i.e., those that need to be fixed after release, in these projects seem not to be the kind of issues that we might expect that methods and principles in a safety standard detect, either.

In this research, we want to test our hypothesis by analyzing historic records of bugs found in production releases of the Linux Kernel. With this in mind, we perform (and present) a preliminary study to see if we are able to detect the bugs in the Linux Kernel and identify those critical to safety standards.

2. Related work

The study of Linux Kernel bugs and their classification has been widely discussed in the literature. Tan et al. [8] provide a study on the characteristics of bugs in open source software, where one of the test subjects is the Linux Kernel, from which a total of 300 bug-fixing commits are analyzed. They offer a categorization in three dimensions: root causes, impacts and components. Among their conclusions, the authors find that (1) semantic bugs are the dominant root cause; these bugs increase as the software evolves, while memory-related bugs decrease; (2) the Linux Kernel has many concurrency bugs compared with other non-OS subjects of the study; and (3) security bugs increase over time and most of them are semantic bugs. Applying the knowledge gained from the manual analysis of these 300 bug-fixing commits, they have been able to classify, using machine learning techniques, more than 100,000 bugs.

Xiao et al. [9, 10] analyze 5,741 Linux Kernel bugs obtained using Bugzilla, focusing on bugs reported as “CODE_FIX” and with a “CLOSED” status, during the period 2002-2016. Authors report that only 76% of the reports are actually a bug report, and distinguish between two types of bugs: Bohrbug (BOH), a bug that can be easily isolated and reproduced, and Mandelbug (MAN), the complementary antonym of Bohrbug. The authors focus on bugs categorized as Mandelbug, distinguishing between Aging Related bug (ARB); a type of bug that can cause increased failure rate and/or performance degradation, and Non-Aging related Mandelbug (NAM); a complementary antonym of ARB. Among their results we find that 55% of the bugs they have detected are BOHs, while 36% are MANs. They also found that more than half of the bugs are regressions, being these predominant in BOHs. In addition, they find that the proportion of regression bugs increases over time.

Also, several studies [4, 6, 5] reveal that it is possible to detect clone-related bugs automatically in the Linux Kernel.

3. Method

We will use commits from the official Linux Kernel repository² as a data source. For a preliminary study, we have selected the last 1,000 commits from the repository (September 2023) with the goal of manually locating the last 100 published Bug-Fixing Commits (BFC).

The method used for the selection and extraction of information from the BFCs is as follows:

- **Commit selection.** As a first inclusion criterion, we consider those commits that unequivocally indicated that it was a bug fix in the comment of the commit or the bug report (if available). The second inclusion criterion is that the change was made in the source code (in a functionality), excluding cases such as variable renaming or comments. We have also excluded those commits where the comment began with “Merge ...”, since they include changes from several commits already present in the analyzed branch.
- **Commit info extraction.** For each commit, we extract the following information: the message of the commit, a link to the commit, a link to the mailing list or Bugzilla report (if it exists), a link to the Bug-Introduction Commit (BIC) (if it exists), the location (based on the folder where the changes were made), and if it is a regression bug (i.e., it has been verified that either the commit author reports it as such or if it reverses a change of the BIC).

In this preliminary study, we are interested in checking if there is a reference to the bug report in the commit (either in mailing lists or in Bugzilla), in how many BFCs there is a link to the BIC, in what components the BFCs are located and how many of them are regression bugs that we can identify.

4. Preliminary results

We were able to identify the first 100 BFCs after checking the last 203 commits.

In the following, we will show the preliminary results obtained from these 100 BFCs:

- **How many BFCs have a link to the mailing list or Bugzilla?**

Half of the BFCs (49) have an associated mailing list. We found that mailing lists are clearly identified, following the formats *Link* :< URL > or *Closes* :< URL >. There are cases where there may be more than one mailing list associated with a BFC. However, only 5 BFCs have a reference to the report to Bugzilla. This leads us to believe, in contrast to previous literature, that data sources such as Bugzilla are not complete.

- **How many BFCs have a link to the BIC?**

The numbers are surprisingly high (80 BFCs have a link to the BIC). This is because Linux Kernel developers follow a format in which they report the BIC in the following format *Fixes* :< hash > (< comment >). This identification is done mainly with Git Bisect¹,

²<https://github.com/torvalds/linux>

¹<https://git-scm.com/docs/git-bisect>

which is documented in the Linux Kernel documentation² as a method to report the BIC of a bug.

- **In what components are the BFCs located?**

The distribution of BFCs by component can be found in Table 1. The “drivers” module is where we found almost half of the errors (48), followed by FileSystem (24) and Architecture (7). This is in line with previous studies [7, 1], which show that these are the modules where most bugs are fixed.

Table 1
Distribution of BFCs by component

Component	Directory	Count
DRIVER	drivers/	48
FILESYSTEM	fs/	24
ARCH	arch/	7
NETWORK	net/	6
KERNEL	kernel/	5
INCLUDE	include/	4
SCRIPTS	scripts/	2
BLOCK	block/	1
IO	io_uring/	1
MEMORY	mm/	1
SECURITY	security/	1

- **How many bugs have we detected that are due to regression in the code?**

We have been able to confirm (through developer confirmation or reversion of a change) 9 regressions. However, this number could be higher (as reported in [9, 10]) because the developers may have missed that it is a regression.

5. Next steps

The next step in our research would be to be able to classify the bugs to detect (and correctly classify) those that are safety-related. As a result we would obtain a “golden set” of manually validated bugs from the Linux Kernel, which can be used as a benchmark for automatic classification algorithms, be these algorithms based on machine-learning techniques or others.

The categories of bugs that are most relevant according to ISO 26262 are related to “freedom from interference” or “FFI”, initially bugs associated with:

- **Timing and execution.** Following ISO-26262-6 Annex D (informative) [3] (*Freedom from interference between software elements*), with respect to timing constraints, the

²<https://docs.kernel.org/admin-guide/bug-bisect.html>

effects of faults such as blocking of execution, deadlocks, livelocks, incorrect allocation of execution time or incorrect synchronization between software elements can be considered for the software elements executed in each software partition. We expect that bugs falling into this category will probably mention some of the following key words/phrases: interfere(nce), blocking, race (or race condition), deadlock, livelock, time, execution time, sync, synchronise, synchronize, synchronisation, synchronization, sequence, arrival, rate, cyclic, trigger(ed), schedule, scheduling, allocate or allocation.

- **Memory.** Following ISO 26262-6 Annex D.2.3 (*Memory*) [3], with respect to memory, the effects of faults such as those listed below can be considered for software elements executed in each software partition: corruption of content, inconsistent data (e.g. due to update during data fetch), stack overflow or underflow or read or write access to memory allocated to another software element. We expect that bugs falling into this category will probably mention some of the following key words/phrases: corrupt(ion), memory, stack, overflow, underflow, read, write, access, allocate, allocation, free, leak, parity, error-correct(ion), static or dynamic.
- **Exchange of information.** Following ISO-26262-6 Annex D [3], with respect to the exchange of information, the causes for faults or effects of faults such as those listed below can be considered for each sender or each receiver: repetition of information, loss of information, delay of information, insertion of information, masquerade or incorrect addressing of information, incorrect sequence of information, corruption of information, asymmetric information sent from a sender to multiple receivers, information from a sender received by only a subset of the receivers or blocking access to a communication channel. We expect that bugs falling into this category will probably mention some of the following key words/phrases: information, data, flow, i/o, io, bus, slot, arbitrate, arbitration, exchange, communication, communicate, protocol, send(er), receive(r), signal, message, loss, delay, insert, sequence, corrupt(ion)or asymmetric.

This classification can be complex, as it may not depend directly on whether the words/phrases we are considering appear. That is why we are going to consider the help of Linux Kernel developers. We expect to have the support of several Linux Kernel developers, 1) to validate the identification of the 100 bugs found with our naive method, and 2) to manually classify the bugs (if possible) in one of the three defined categories. We expect that the developers can help us from their experience to better classify this type of errors.

References

- [1] Chou, A., Yang, J., Chelf, B., Hallem, S., Engler, D.: An empirical study of operating systems errors. In: Proceedings of the eighteenth ACM symposium on Operating systems principles, pp. 73–88 (2001)
- [2] Road vehicles – Functional safety. Standard, International Organization for Standardization, Geneva, CH (2018)
- [3] Road vehicles – Functional safety – Part 6: Product development at the software level. Standard, International Organization for Standardization, Geneva, CH (2018)

- [4] Jiang, L., Su, Z., Chiu, E.: Context-based detection of clone-related bugs. In: Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering, pp. 55–64 (2007)
- [5] Li, J., Ernst, M.D.: Cbcd: Cloned buggy code detector. In: 2012 34th International Conference on Software Engineering (ICSE), pp. 310–320. IEEE (2012)
- [6] Li, Z., Lu, S., Myagmar, S., Zhou, Y.: Cp-miner: Finding copy-paste and related bugs in large-scale software code. *IEEE Transactions on software Engineering* **32**(3), 176–192 (2006)
- [7] Palix, N., Thomas, G., Saha, S., Calvès, C., Lawall, J., Muller, G.: Faults in linux: Ten years later. In: Proceedings of the sixteenth international conference on Architectural support for programming languages and operating systems, pp. 305–318 (2011)
- [8] Tan, L., Liu, C., Li, Z., Wang, X., Zhou, Y., Zhai, C.: Bug characteristics in open source software. *Empirical software engineering* **19**, 1665–1705 (2014)
- [9] Xiao, G., Zheng, Z., Yin, B., Trivedi, K.S., Du, X., Cai, K.: Experience report: fault triggers in linux operating system: from evolution perspective. In: 2017 IEEE 28Th international symposium on software reliability engineering (ISSRE), pp. 101–111. IEEE (2017)
- [10] Xiao, G., Zheng, Z., Yin, B., Trivedi, K.S., Du, X., Cai, K.Y.: An empirical study of fault triggers in the linux operating system: An evolutionary perspective. *IEEE Transactions on Reliability* **68**(4), 1356–1383 (2019)