DOI: 10.15514/ISPRAS-2023-35(3)-16

Analyzing Hot Bugs in the Linux Kernel by Clustering Fixing Commit Messages

S.M. Staroletov, ORCID: 0000-0001-5183-9736 <serg_soft@mail.ru>
N.A. Starovoytov, ORCID: 0009-0007-0242-0198 <nikstarall@gmail.com>
N.A. Golovnev, ORCID: 0009-0008-0258-4560 <kolya.golovnev@mail.ru>

Polzunov Altai State Technical University
46, prospect Lenina, Barnaul, Altai region, 656038, Russia.

Abstract. In system software environments, a vast amount of information circulates, making it crucial to utilize this information in order to enhance the operation of such systems. One such system is the Linux kernel, which not only boasts a completely open-source nature, but also provides a comprehensive history through its git repository. Here, every logical code change is accompanied by a message written by the developer in natural language. Within this expansive repository, our focus lies on error correction messages from fixing commits, as analyzing their text can help identify the most common types of errors. Building upon our previous works, this paper proposes the utilization of data analysis methods for this purpose. To achieve our objective, we explore various techniques for processing repository messages and employing automated methods to pinpoint the prevalent bugs within them. By calculating distances between vectorizations of bug fixing messages and grouping them into clusters, we can effectively categorize and isolate the most frequently occurring errors. Our approach is applied to multiple prominent parts within the Linux kernel, allowing for comprehensive results and insights into what is going on with bugs in different subsystems. As a result, we show a summary of bug fixes in such parts of the Linux kernel as kernel, sched, mm, net, irq, x86 and arm64.

Keywords: bugs; Linux; clustering; fixing commits; kernel.

For citation: Staroletov S.M., Starovoytov N.A., Golovnev N.A. Analyzing hot bugs in the Linux kernel by clustering fixing commit messages. Trudy ISP RAN/Proc. ISP RAS, vol 35, issue. 3, 2023., pp. 215-242. DOI: 10.15514/ISPRAS-2023-35(3)-16

Анализ актуальных ошибок в ядре Linux путем кластеризации сообщений об исправлениях в git-репозитории

С.М. Старолетов, ORCID: 0000-0001-5183-9736 <serg_soft@mail.ru>
Н.А. Старовойтов, ORCID: 0009-0007-0242-0198 <nikstarall@gmail.com>
Н.А. Головнев, ORCID: 0009-0008-0258-4560 <kolya.golovnev@mail.ru>

АлтГТУ им. И.И. Ползунова
Россия, 656038, Алтайский край, г. Барнаул, пр. Ленина, 46.

Аннотация. В средах системного программного обеспечения циркулирует огромное количество информации, поэтому крайне важно использовать эту информацию для улучшения их работы. Одной из таких систем является ядро Linux, которое не только поставляется с полностью открытым исходным кодом, но и предоставляет исчерпывающую историю о разработке в своем git-репозитории. Здесь каждое логическое изменение кода сопровождается сообщением, написанным разработчиком на естественном языке. Обрабатывая данные репозитория, мы сосредотачиваемся на коммитах с сообщениями об исправлении ошибок, поскольку анализ их текста может помочь выявить наиболее распространенные типы ошибок. Основываясь на наших предыдущих работах, в этой статье мы предлагаем использовать методы анализа данных. Для достижения наших целей мы предлагаем

различные методы обработки сообщений в git-репозиториях и используем автоматизированные методы для выявления распространенных ошибок в них. Вычисляя расстояния между сообщениями об исправлении ошибок, превращая их в вектора и группируя в кластеры, мы далее можем эффективно классифицировать и выявлять наиболее часто возникающие ошибки. Наш подход применяется к нескольким важным частям ядра Linux, что позволяет понять, что происходит с ошибками в различных его подсистемах. В результате мы показываем сводку исправлений ошибок в таких частях ядра Linux, как kernel, sched, mm, net, irq, x86 и Arm64.

Ключевые слова: ошибки; Linux; кластеризация; исправляющие коммиты; ядро.

Для цитирования: Старолетов С.М., Старовойтов Н.А., Головнев Н.А. Анализ актуальных ошибок в ядре Linux путем кластеризации сообщений об исправлениях в git-репозитории. Труды ИСП РАН, том 35, вып. 3, 2023 г., стр. 215-242 (на английском языке). DOI: 10.15514/ISPRAS–2023–35(3)–16.

1. Introduction

In today's software development landscape, closed systems are no longer able to compete with open ones due to the involvement of highly skilled users who can not only test the software but also understand its code and suggest changes. The git version control system and its associated services are based on a fork and pull request approach, allowing users to easily propose changes and administrators to accept them after reviewing diffs. Git is designed for distributed work and encourages local changes, with each commit being accompanied by a comment about what was done. Originally created by Linus Torvalds for coordinating the development of the Linux kernel, git has become a super successful project.

While system program code development direction is not that popular among most modern software developers mainly due to the scarcity of qualified engineers, there is a large amount of data circulating in system software environments that can be analyzed using popular data analysis methods. This paper proposes to analyze commit messages in the development of the Linux kernel using automated methods. With a large number of commit messages available, common patterns can be automatically identified from the big data in natural language. The focus is on identifying and correcting the most typical errors in system software.

The Linux OS, based on an open modular kernel concept by Linus Torvalds, is constantly evolving with contributions from a large number of developers, both individuals and representatives of leading companies in the industry. All changes are made by committing them to developers' gits, and some eventually become available in the mainstream kernel at Torvalds GitHub. Such commits are usually verified by leading developers using the pull-request mechanism. Therefore, commit data analysis can provide insights into the evolution of the kernel.

The objective of this work is to automatically analyze commits in the Linux kernel repository to identify the most representative bugs. The paper discusses and explores data analysis methods for Linux commit messages.

The present paper is an extension of the report presented at SYRCoSE Software Engineering Colloquium 2023 in Penza [1].

2. Related work

In the pioneering work by Chou, Yang, Chelf and Engler [2] as well as ten years later by Palix, Thomas, Saha, Calves, Lawall and Muller [3], static analyzers were used to automatically check for potential errors in the Linux kernel code based on a given configuration over different kernels. Classes of errors were defined as predefined messages of a static analyzer, and graphs of the evolution of errors over time and for different subsystems were presented. Specifically, drivers have been found to be 3-7 times more error prone than other components.

Mutilin, Novikov and Khoroshilov made an analysis of typical errors in the drivers of the Linux operating system [4]. Here the concept of a typical error is introduced. According to the researchers, it is specific to a large number of drivers (for example, resource leaks, incorrect use of locks), while

a non-typical error is domain-specific for a particular driver. The authors manually analyzed the changes during the transition from one kernel version to another and compiled tables of the distribution of errors by classes. It was also found that drivers make 85% of all errors in the kernel. The paper by Novikov [5] continues this work, summarizes various statistics on changes in the kernel and concludes that about 40% of changes between stable versions of the kernel are fixes of typical errors. Since more versions were analyzed and the code evolved, the author had to supplement the previous created classes. Such manual analysis is more difficult, but the authors note that it is more careful.

Lus and Arpaci-Dusseaus manually analyzed 5079 patches related to file systems made over 8 years [6]. Classes of bugs, the so-called bug patterns, are identified and graphs of their evolution are given, as a result, a dataset of 1800 bugs is compiled.

The empirical work by Tan, Liu, Li, Wang, Zhou and Zhai [7] is devoted to a broad study of bugs in open-source software, including the Linux kernel. As their results for Linux, bugs are simply assigned to one of the subsystems (core, driver, network, FS, arch, other), while several open-source components are analyzed using message text from BugZilla with its vectorization and further automatic classification.

The work by Xiao, Zheng, Yin, Trivedi, Du and Cai [8] is devoted to the study of 5741 Linux kernel bug reports, which were analyzed according to the description, comments and attached files from the Linux kernel bug tracker [9]. Bugs are classified into fast-reproducible (Bohrbug), difficult-to-reproduce (Mandelbug) or context-dependent, and are also defined categories from which the bug context depends, that is, errors with memory, not freed resources, etc. At the same time, the authors built a network based on the Linux call graph, with the help of which they track the impact of the functions affected in bug reports by counting various metrics.

The researchers Melo, Flesborg, Brabrand and Wasowski present the results of compiling 42,060 kernels with all warnings enabled [10]. As a result of the analysis of 400,000 warnings, they classified by type and distribution by kernel subsystems and identified drivers as the most vulnerable portion of the kernel.

The work by Hoang, Lawall, Tian, Oentaryo and Lo [11] presents the PatchNet network, created as a result of automatic analysis of patches for the kernel, in order to predict whether a given patch will be accepted in the mainline kernel or not. For evaluation, the texts of the commit messages and the vector representation of the changes from the diff of the commit are used, which are then used to build a convolution neural network.

The research by Tian, Lawall and Lo [12] is separately devoted to determining whether a patch to the kernel is a bug fix or not. The authors note that simple analysis based on commit messages does not always lead to correct results and propose a model that uses two classification algorithms: Learning from Positive and Unlabeled Examples and Support Vector Machine. It also uses features extracted from the commit diff.

In the study by Acher, Martin, Pereira, Blouin, Khelladi and Jezequel [13], the authors provide infrastructure, classify and analyze Linux kernel-specific errors associated with errors in configuration files, as a rule, these are errors with dependencies. 95,854 Linux kernel builds were produced on random configurations, and of these, about 6% ended with errors, and which are discussed in the work. It is noted that the number of errors has decreased with previous findings, apparently due to testing processes with randomized configurations.

Summarizing the above on Linux bug analysis, it can be seen that (1) bugs in drivers are the most common; (2) different methods are used for classification, this is static analysis, build logs and patch analysis; (3) a lot of huge manual work has been done but the results may now be considered no longer relevant (the code is constantly changing). However, automatic classification by analyzing commits in git repositories has not been applied yet.

3. Preliminaries

To solve the problem of analyzing commit messages, we need to work with the Git repository at the program level. This means that from a program in one of the programming languages using some API, we need to get a list of commits, filter commits (by date, for example), iterate through them, obtain changes and the text of the commit message. Probably, the most famous C-library for this is *libgit2*. For JVM programs, the *JGit* library is popular. One can also use *EGit* to work with remote repositories, including access to pull requests data, but this is not necessary for the current project because we are working with the mainline kernel with changes already accepted. To be able to work with a git repository from code in Python, the *GitPython* library can be a good choice.

In order to obtain a set of fixing commits from a set of interesting commits, the easiest way to check the commit message to find a list of some signal words, but it is better to use some sort of AI-based commit classifier like the one discussed in the work [12].

The next step is finding the similar fixing commit messages in order to reveal the most common bug fixes. To compare commit messages, it is necessary to work out *fuzzy string matching* algorithms. Note that fuzzy string comparison is popular in bioinformatics.

Of practical interest are efficiently calculated string similarity measures, such as the Levenshtein distance. Formally, the Levenshtein distance $L(s_1, s_2)$ [14] between strings s_1 and s_2 can be calculated according to the following formulas:

$$\begin{aligned} L(s_1, s_2) := \quad & \forall\, i \in (0..|s_1|): d_{i,0} := i + 1; \\ & \forall\, j \in (0..|s_2|): d_{0,j} := j + 1; \\ & \forall\, i \in (1..|s_1|): \\ & \quad (\forall\, j \in (1..|s_2|): cost := (s_1[i-1] = s_2[j-1])\ ?\ 0 : 1 \\ & \quad d_{i,j} := \min(\min(d_{i-1,j} + 1, d_{i,j-1}), d_{i-1,j-1} + cost); \\ & \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad d_{|s1|,|s2|}. \end{aligned} \tag{1}$$

With it, to find the closest string to the existing ones, in the simplest implementation, one needs to calculate the distances between them using formula (1) and choose the minimum one. This method does not require preliminary preparation of strings and is susceptible to slight changes in them.

Our previous papers [15, 16] demonstrated that the use of the Levenshtein distance can provide a good understanding what is going on with the fixes in major Linux kernel parts. However, such an analysis for big repositories takes a lot of resources (we need to compare $O(\text{fixing commit count}^2)$ string comparisons) and its accuracy is very difficult to verify.

Therefore, in this paper, we would like to apply another simple method known from its use in search engines (the use of "bag of words" to convert a phrase into a vector + calculation of cosine similarity between vectors to further determine the minimal distance).

To calculate the distance between commits messages using the cosine similarity approach, it is required to represent the commit message string as a vector (from the features as the words of the message). Here we denote $w_{i,j}$ as the sign of the presence of the word j in the string i, while n specifies the number of words in the dictionary of unique words. Then we are able to calculate the cosine similarity between the vectors:

$$D(s_1, s_2) \coloneqq D((w_{1,1}, w_{1,2}, ..., w_{1,n}), (w_{2,1}, w_{2,2}, ..., w_{2,n})) = \frac{\sum_{i=1}^{n} w_{1,i} \cdot w_{2,i}}{\sqrt{\sum_{i=1}^{n} w_{1,i}^2 \cdot \sum_{i=1}^{n} w_{2,i}^2}} \tag{2}$$

In this case, permutations of words in a string will not change anything.

If we use dictionaries that give the stem word form for each word (without cases, endings, etc.), we can get rid of the problems of counting the same words in different components of vectors. The process is called lemmatization [17] or lemma normalization. In the simplest case, the Catvar

dictionary can be used [18]. Here, for each word from the commit message (column 1), its normalized form can be obtained (column 2), for example, here is a dictionary fragment for the words “fix”:

```
fix     fix $N$
fix     fix $V+0$
fixed   fix $V+ed$
fixed   fix $V+en$
fixes   fix $N+s$
fixes   fix $N+s$
fixes   fix $V+s$
fixing  fix $V+ing$
```

In more advanced cases, the StanfordCoreNLP API [19, 20] can be used. Since not only the stem, but also the part of speech is known for each word, when converting phrases into vectors, it is advisable to filter them, cutting off articles and frequently used words.

For the purposes of searching for strings with “strong components” or relevant words/tokens (i.e., to reduce the weights of frequently occurring words in a string), the tf-idf approach [21] can be applied. It does frequency counting, and with this, the vector components (features) instead of word appearance (1 or 0) will contain tf-idf weights. If we denote m_w as the number of occurrences of the word w into a commit message $m \in M$, and n_w as the total number of words in the document, and $|M|$ as the total number of messages, then:

$$tfidf(w,m,M) := tf(w,m) \times idf(w,M) = \frac{n_w}{\sum\limits_k n_k} \times \log \frac{|M|}{|\{m_i \in M \mid w \in d_i\}|} \quad (3)$$

If we are able to vectorize commit messages, then it makes sense to try to cluster them automatically. Clustering or cluster analysis involves the vectorization of given objects, calculating the distances between them according to a certain metric and dividing objects into clusters or groups of nearby objects. Vectorization involves the selection of key entities of objects and their presentation as a set of vectors of the same dimension. The clustering algorithm is a function $X \to Y$ that assigns a cluster identifier $y \in Y$ to any object $x \in X$. Some popular clustering algorithms are K-means, DBSCAN, and hierarchical clustering. The K-means algorithm iteratively minimizes the total square deviation of cluster points from the centers of these clusters (a classical approach presented in [22]). The density-based spatial clustering of applications with noise (DBSCAN) algorithm groups points in a high-density area into one cluster, while marking lonely points as noise [23]. With hierarchical clustering, a tree (dendrogram) is built, from leaves to root. Initially, each object is contained in its own cluster. Next, an iterative process of merging the two nearest clusters takes place until all clusters are combined into one, or the required number of clusters is found [24].

4. On the implementation

To work with git repositories, we utilize the Python git library, which enables us to iterate over commits and insert conditions to process them in the code. Initially, we began our solution as a JDK program since we had prior experience working with the JGit library. Subsequently, we implemented a prototype for analyzing a Thunderbolt repository (as described in the work-in-progress article [1]) by overriding classes for processing commit messages using the minimum Levenshtein distance, vectorization, and searching for minimum distances between phrases. Later on, we developed methods for clustering vectors from phrases. Currently, we use the Python language since it facilitates clustering methods with scipy packages.

The solution scheme is illustrated in *Fig. 1*. In this solution, we acquire all the commits of a given repository, filter them according to the dates of interest, and extract only those explicitly indicating a bug fix.

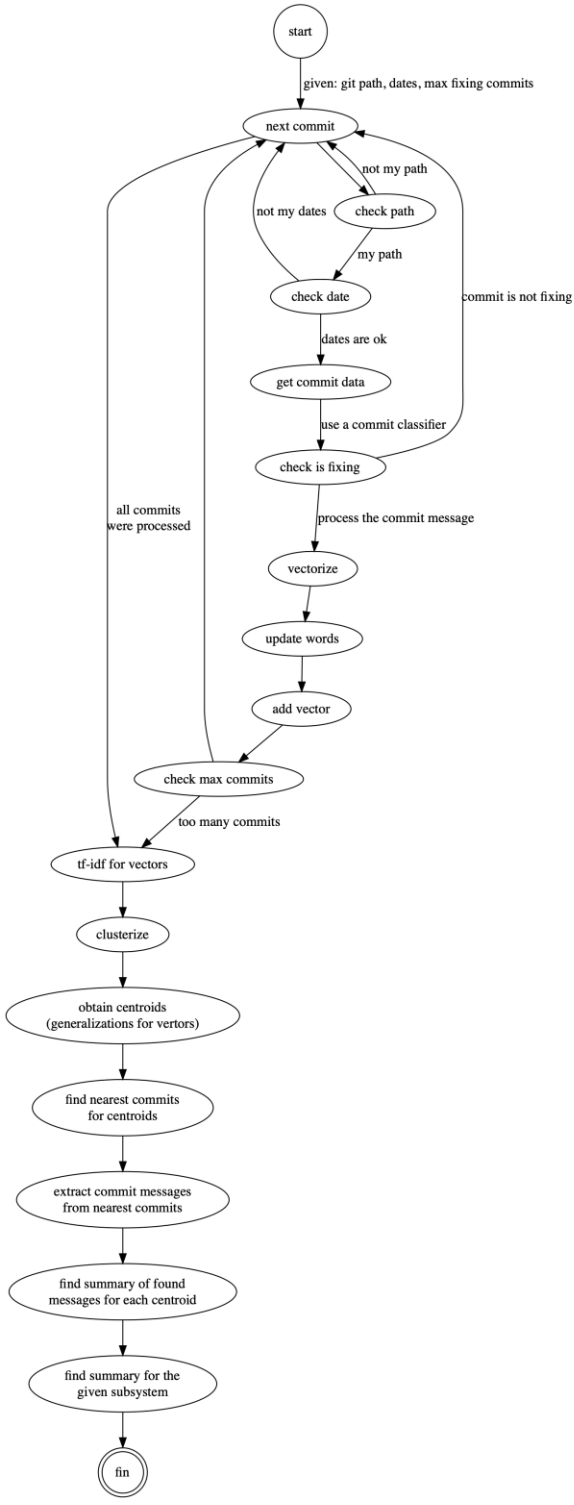

Fig. 1. A diagram of our solution

For this purpose, we employ our own implementation of the method discussed in the paper [12] using pre-trained classifiers based on information about the changes in the commit. Next, we work with vectorized and lemmatized fixing commit messages. We further form flat clusters from hierarchical clustering with a given threshold (which is calculated using some heuristics and can subsequently be improved by expert evaluation of the resulting clusters). We store the resulting vectors sorted by distance from the centroid, marking the most important words first. To obtain meaningful results, we need to find the nearest commits for each of the discovered vectors using the commit text. Afterward, through manual analysis, we can determine the most general message about the fix and an example of such a fix for the error in the code.

5. Findings

In this section, we provide the results of our analysis for the major subsystems of the Linux kernel. We analyzed some selected subsystems of the Linux kernel by examining the corresponding parts of the path in the main git repository. Due to limited computing resources, we were only able to process a maximum of 10,000 fixing commits for each subsystem. We present our findings in the form of a list of vectors, each representing a grouping (blurring) of the most frequent messages. For each vector, we provide a list of its components, sorted by importance, as well as git messages from the closest commits to that vector. They we analyze these messages in order to describe each vector in natural language. Finally, we provide a generalization of the fixes found for each analyzed subsystem.

5.1 Kernel (/linux/kernel)

Vector #1: [cpu, period, grace, rcu, event, callback, commit, probe, function, state, check, structure, rcu_node, stall, file]

```
Fix day-one dyntick-idle stall-warning bug
rcu: Suppress more involved false-positive preempted-task splats
rcu: Accelerate grace period if last non-dynticked CPU
rcu/segcblist: Prevent useless GP start if no CBs to accelerate
Go dyntick-idle more quickly if CPU has serviced current grace period
```

These fixes address the very important RCU subsystem in Linux, which provides ways to non-blockingly synchronize concurrent entities [25, 26]. However, there are problems in the form of potential unfinished waiting or inefficiency in its implementation, since processors in modern computing systems can go into energy-efficient hibernation, which leads to bugs in the RCU implementation for the tasks running on them (problems with the waiting period or grace period).

Vector #2: [buffer, kernel, ring, warning, doc, function, page, parameter, iterator, read, type, member, resource, trace, tracepoint]

```
ring-buffer: Fix kernel-doc
ring-buffer: Always reset iterator to reader page
resource/docs: Fix new kernel-doc warnings
seccomp: fix kernel-doc function name warning
rcu: Fix a kernel-doc warnings for "count"
```

These fixes refer to corrections to code documentation, which are done in the form of comments embedded in the code. The kernel-doc tool collects these comments and checks their completeness [27]. The comments here refer to the ring buffer, which can be used to implement efficient network applications [28].

Vector #3: [module, panic, patch, build, state, cpu, error, message, new, unloaded, code, function, kernel, notifier, list, load_module]

```
module: Ensure a module's state is set accordingly during module coming cleanup
code
livepatch: Fix subtle race with coming and going modules
debug: track and print last unloaded module in the oops trace
```

```
[PATCH] Kprobes: Reference count the modules when probed on it
debug: show being-loaded/being-unloaded indicator for modules
```

These fixes concern errors when loading and unloading kernel modules, more precisely during their live loading, when the already loaded code is replaced in a running system [29]. This is possible using the function tracing approach.

Vector #4: [tracer, function, graph, ret_stack, task, tracing, option, new, callback, code, ftrace, return, add, boot, buffer]

```
tracing/function-graph-tracer: drop the kernel_text_address check
function-graph: allow unregistering twice
tracing: Move mmio tracer config up with the other tracers
function-graph: move initialization of new tasks up in fork
tracing: Add ftrace events for graph tracer
```

These fixes concern the actual implementation of tracing [30] and working with the function call graph.

Vector #5: [timer, base, cpu, target, clk, idle, code, interval, posix, task, tick, jiffy, case, race, trace]

```
posix-timers: Fix full dynticks CPUs kick on timer rescheduling
timers: Use proper base migration in add_timer_on()
timer/trace: Improve timer tracing
posix-cpu-timers: Unbreak timer rearming
hrtimer: Preserve timer state in remove_hrtimer()
```

These fixes are aimed at fixing the kernel code for implementing timers according to the POSIX standard (see for example a discussion on its userspace interface [31]), namely errors during recharging (when changing the response time of already set timers), which entails working with related processes that may be located on temporarily retired processors.

Vector #6: [lock, ftrace, error, kernel, rlock, trace, deadlock, incompatible, type, possible, comparison, lockdep, info, irq, timekeeping]

```
timekeeping: Avoid possible deadlock from clock_was_set_delayed
sched/core: Make dl_b->lock IRQ safe
timekeeping: Fix HRTICK related deadlock from ntp lock changes
cpu/hotplug: Drop the device lock on error
pid: fix lockdep deadlock warning due to ucount_lock
```

These fixes are related to the internal kernel elapsed time measurement subsystem [32, 33] and associated incorrect locking in the implementation.

Vector #7: [console, printk, boot, message, list, line, early, srcu, add, time, tracepoint, use, patch, problem, console_lock]

```
console: prevent registered consoles from dumping old kernel message over again
[PATCH] CON_CONSDEV bit not set correctly on last console
printk: don't prefer unsuited consoles on registration
Revert "printk: Block console kthreads when direct printing will be required"
console: allow to retain boot console via boot option keep_bootcon
```

This series of fixes is devoted to kernel diagnostic messages and their output via tty consoles.

Vector #8: [lockdep, lock, patch, time, code, cross, performance, release, run, second, boot, case, kernel, bug, counter]

```
lockdep: spin_lock_nest_lock(), checkpatch fixes
lockdep, bug: Exclude TAINT_FIRMWARE_WORKAROUND from disabling lockdep
lockdep: more robust lockdep_map init sequence
locking/lockdep: Add a boot parameter allowing unwind in cross-release and disable
it by default
tracing: use raw spinlocks for trace_vprintk
```

These fixes are related to the work of the lockdep deadlock prevention tool [34] in the kernel and the work of the checkpatch tool [35] to check the formal requirements of the patches associated with it.

Vector #9: [kernel, inline, bpf, event, type, btf, pid, trace, number, buffer, foo, lock, rip, code, cat]

```
bpf: prevent decl_tag from being referenced in func_proto
tracing: Free buffers when a used dynamic event is removed
coredump: fix crash when umh is disabled
tracing: Fix memory leak in eprobe_register()
tracing: Check return value of __create_val_fields() before using its result
```

The fixes are related to BPF integration into the kernel and tracing (described in [36] and discussed in [37]), which were detected by the Syzkaller tool [38]. Since the tool reports contain listings with the same keywords (register dump, call stack), they were detected as similar vectors.

Vector #10: [error, return, code, function, value, failure, case, cgroup, file, negative, ret, add, userspace, bpf, caller]

```
cred: add missing return error code when set_cred_ucounts() failed
rcutorture: Fix error return code in rcu_perf_init()
bpf: Fix error return code in map_lookup_and_delete_elem()
ftrace: Deal with error return code of the ftrace_process_locs() function
genirq/timings: Fix error return code in irq_timings_test_irqs()
```

This series of fixes included fixes for the “fix error return code” error in various parts of the kernel, including the BPF and RCU torture functions [39].

In general, based on this key subsystem of the Linux kernel, we can conclude that most of the problems found and corrected were associated with incorrect operation of multiprocessor concurrent systems due to incorrect processing of all scenarios in the control flow, which involve accurate processing in conditions of variability of resources such as processors, pages memory, etc. That is, the handling of unexpected situations was not carried out completely correctly. The key kernel components mentioned were the RCU subsystem, swap management, timing, BPF and tracing. The code identified problems were related with the correct processing of return codes.

5.2 Drivers (/linux/drivers)

Fixes for kernel drivers are distinguished by the presence of a large number of identical changes (“serial patches”). Essentially, some change to the API is made and then the code for a large number of drivers that depend on that API should be changed. Such changes can be described in the form of so-called semantic patches and applied to a given set of files [40] and also attempted to be generalized from a set of source code files [41]. The next 10 vectors found describe exactly such changes, all of them are repeating.

Vector #1: [remove, return, function, null, check, unused, error, value, staging, is_err, macro, pointer, test, debug, definition]

```
mfd: pm8008: Fix return value check in pm8008_probe()
misc/pvpanic: fix return value check in pvpanic_pci_probe()
drm/i915/selftests: Fix return value check in live_breadcrumbs_smoketest()
n64cart: fix return value check in n64cart_probe()
net: sparx5: fix return value check in sparx5_create_targets()
```

Vector #2: [error, code, return, negative, function, case, success, net, scsi, ethernet, drm, mtk_eth_soc, path, add, einval]

```
RDMA/srpt: Fix error return code in srpt_cm_req_recv()
HID: pidff: fix error return code in hid_pidff_init()
mmc: usdhi6rol0: fix error return code in usdhi6_probe()
mtd: mtd_oobtest: fix error return code in mtd_oobtest_init()
net: sparx5: fix error return code in sparx5_register_notifier_blocks()
```

Vector #3: [dev_err, error, redundant, remove, message, print, devm_ioremap_resource, avoid, function, platform_get_irq, drivers, unnecessary, line, coccicheck, follow]

```
can: ctucanfd: Remove redundant dev_err call
fbdev: imxfb: Remove redundant dev_err() call
crypto: aspeed - Remove redundant dev_err call
mailbox: arm_mhu_db: Remove redundant dev_err call in mhu_db_probe()
soc/tegra: cbb: Remove redundant dev_err call
```

Vector #4: [array, member, flexible, element, replace, length, zero, struct, help, kernel, code, helper, use]

```
scsi: smartpqi: Replace one-element array with flexible-array member
staging: r8188eu: Replace zero-length array with flexible-array member
staging: rtl8723bs: Replace zero-length array with flexible-array member
scsi: megaraid_sas: Replace one-element array with flexible-array member in
MR_PD_CFG_SEQ_NUM_SYNC
scsi: megaraid_sas: Replace one-element array with flexible-array member in
MR_FW_RAID_MAP
```

Vector #5: [irq, interrupt, code, dt, core, hierarchical, resource, setup, static, use, platform_get_irq, allocation, cause, chaining, domain]

```
ata: pata_pxa: Use platform_get_irq() to get the interrupt
can: ti_hecc: ti_hecc_probe(): use platform_get_irq() to get the interrupt
serial: 8250_bcm7271: Use platform_get_irq() to get the interrupt
net: pxa168_eth: Use platform_get_irq() to get the interrupt
i2c: riic: Use platform_get_irq() to get the interrupt
```

Vector #6: [pm_runtime_resume_and_get, error, pm, counter, usage, order, runtime, use, decrement, add, medium, commit, usage_count, deal, dev]

```
media: i2c: ov9734: use pm_runtime_resume_and_get()
media: i2c: ov5675: use pm_runtime_resume_and_get()
media: i2c: ov5647: use pm_runtime_resume_and_get()
media: i2c: hi556: use pm_runtime_resume_and_get()
media: i2c: dw9807-vcm: use pm_runtime_resume_and_get()
```

Vector #7: [dev_err_probe, error, code, probe, use, check, dev_err, path, helper, print, debugfs, defer, reason, switch, replace]

```
usb: usb251xb: Switch to use dev_err_probe() helper
drm/panel: simple: Use dev_err_probe() to simplify code
backlight: ktd253: Switch to use dev_err_probe() helper
USB: PHY: JZ4770: Switch to use dev_err_probe() helper
usb: phy: generic: Switch to use dev_err_probe() helper
```

Vector #8: [line, blank, comment, checkpatch, declaration, issue, style, patch, staging, net, block, add, error, parenthesis]

```
net: c101: add blank line after declarations
net: hdlc_cisco: add blank line after declaration
net: hdlc: add blank line after declarations
net: sealevel: add blank line after declarations
net: pci200syn: add blank line after declarations
```

Vector #9: [return, error, i2c, remove, message, callback, make, value, void, core, device, preparation, patch, result, cleanup]

```
iio:light:tsl2583: Remove duplicated error reporting in .remove()
iio:light:isl29028: Remove duplicated error reporting in .remove()
iio:accel:mc3230: Remove duplicated error reporting in .remove()
iio:light:opt3001: Remove duplicated error reporting in .remove()
iio:light:jsa1212: Remove duplicated error reporting in .remove()
```

Vector #10: [dev_err_probe, use, error, code, benefit, deal, debugfs, defer, devices_deferred, file, function, help, helper, issue, make]

```
iio: st_lsm9ds0: Make use of the helper function dev_err_probe()
iio: st_sensors: Make use of the helper function dev_err_probe()
drm/panel: y030xx067a: Make use of the helper function dev_err_probe()
drm/panel: xpp055c272: Make use of the helper function dev_err_probe()
drm/panel: sofef00: Make use of the helper function dev_err_probe()
```

5.3 Memory management (/linux/mm)

Vector #1: [page, swap, entry, pte, mm, dirty, cache, pmd, patch, error, code, check, bit, fault, lock]

```
mm: fix data corruption caused by lazyfree page
mm, swap: Fix a race in free_swap_and_cache()
mm: filemap: coding style cleanup for filemap_map_pmd()
mm: invalidate hwpoison page cache page in fault path
mm, swap: fix swapoff with KSM pages
```

The most common error fixes in the memory subsystem are errors related to paging, including disabling swap for KSM pages [42], accessing one page from two processes at the same time, attempting to free the same page at a time; use a hardware-poisoned page remain in the page cache, leading to data corruption, as well as problems with proper handling of shared pages when using swapoff.

Vector #2: [slab, object, kmemleak, size, patch, kfence, slub, partial, add, debug, mm, kernel, allocation, list, lock]

```
SLUB: ensure that the number of objects per slab stays low for high orders
slub: Fix calculation of cpu slabs
slub: When allocating a new slab also prep the first object
slab, slub: remove size disparity on debug kernel
kfence: add sysfs interface to disable kfence for selected slabs.
```

These fixes are devoted to correcting errors in the implementation of the SLUB allocation subsystem. Such allocators are discussed in the presentation [43]. Some fixes were initiated after using the kmemleak tool [44]. The identified fixes relate to working with slab objects that the allocator operates on. The fix also concerns the implementation of the kfence tool [45], allowing users to selectively disable kfence for certain slabs to improve performance.

Vector #3: [page, thp, migration, error, mm, memory, check, fault, cow, hugetlb, process, issue, reference, anonymous, swapcache]

```
mm/memory-failure.c: transfer page count from head page to tail page after split
thp
mm: Wait for THP migrations to complete during NUMA hinting faults
mm: optimize do_wp_page() for exclusive pages in the swapcache
mm/huge_memory: streamline COW logic in do_huge_pmd_wp_page()
mm/rmap: fix missing swap_free() in try_to_unmap() after arch_unmap_one() failed
```

These fixes are devoted to the code for operating with transparent huge page (THP) [46], namely, counting the number of pages when a huge page is divided into several small ones, errors while waiting for the completion of migrations of such pages, incorrect copy-on-write behavior, as well as incorrect handling of errors when making unmap.

Vector #4: [damon, user, monitoring, patch, interface, space, context, region, scheme, mm, address, sysfs, file, support, debugfs]

```
mm/damon/core: allow non-exclusive DAMON start/stop
mm/damon/core: add a new callback for watermarks checks
mm/damon/core: add a callback for scheme target regions check
mm/damon/core: add a function for damon_operations registration checks
```

```
mm/damon/vaddr: register a damon_operations for fixed virtual address ranges
monitoring
```

The listed fixes concern expanding the functionality of the DAMON subsystem [47]. This subsystem provides a means of monitoring data in memory, and with its use, system developers can better understand data circulation and implement optimizations.

Vector #5: [compaction, scanner, patch, zone, page, free, migration, mm, pageblock, pfn, order, check, migrate, success, allocation]

```
mm, compaction: make whole_zone flag ignore cached scanner positions
mm, compaction: more robust check for scanners meeting
mm, compaction: more focused lru and pcplists draining
mm: compaction: reset cached scanner pfn's before reading them
mm: compaction: detect when scanners meet in isolate_freepages
```

These fixes are dedicated specifically to the compaction feature [48]. Compaction is a process that tries to reduce fragmentation in memory by grouping pages around. The fixes affect the processes of scanning such places using scanners that classify candidate pages as LRU (Least Recently Used) and PCP (Per-CPU Page). This is important because if two scanners meet, it means that the compaction process has reached a dead end and needs to be restarted. These fixes make the process more focused, reducing the amount of unnecessary work done during compaction. Also a fix is dealing with cached scanner page frame numbers (PFN), which are not being reset before being read.

Vector #6: [folio, convert, page, use, compound_head, mm, filemap, function, patch, byte, deactivate_page, head, caller, conversion, migrate]

```
migrate: convert unmap_and_move() to use folios
mm/memory: add vm_normal_folio()
mm/hugetlb: convert dissolve_free_huge_page() to folios
mm/damon: Convert damon_pa_young() to use a folio
mm/hugetlb: convert move_hugetlb_state() to folios
```

The fixes concern the functionality of functions for use folios instead of pages. Folios are a new data structure introduced in the kernel that represent a contiguous range of pages [49]. The use folios instead of pages can improve the efficiency of memory migration, moving huge pages or freeing huge pages.

Vector #7: [page, soft, offline, free, mm, compound, patch, hugetlb, hwpoison, tail, change, check, order, buddy, flag]

```
mm: check __PG_HWPOISON separately from PAGE_FLAGS_CHECK_AT_*
mm, hugetlb, soft_offline: save compound page order before page migration
mm/page_alloc: move pages to tail in move_to_free_list()
mm: verify compound order when freeing a page
mm,hwpoison: cleanup unused PageHuge() check
```

Current fixes are dedicated to improving the efficiency of page allocation by reducing fragmentation and improving cache locality, as well ensuring that the correct number of pages is freed to avoid issues with page splitting.

Vector #8: [memblock, memory, mm, static, make, add, list, array, free, region, reserved, allocator, bootmem, debug, expose]

```
mm: free memblock.memory in free_all_bootmem
mm/memblock: make memblock_remove_range() static
revert "mm/memblock: add missing include <linux/bootmem.h>”
mm/memblock: define memblock_physmem_add()
memblock: also dump physmem list within __memblock_dump_all
```

These fixes are dedicated to operate with memory blocks. These functions add a new range of physical memory to the memory block allocator, allowing for more efficient memory management.

In the fixes something is done to provide more information about the system memory layout and aid in debugging.

Vector #9: [doc, kernel, mm, function, parameter, warning, description, markup, error, add, shmem, slab, vmalloc, cleanup]

```
mm/mempolicy.c: parameter doc uniformization
mm/page_vma_mapped.c: add colon to fix kernel-doc markups error for check_pte
[PATCH] more kernel-doc cleanups, additions
mm: fix fatal kernel-doc error
mm: fix kernel-doc markups
```

This series of fixes is dedicated to documenting this subsystem using the kernel-doc annotations.

Vector #10: [kasan, tag, mode, based, memory, hardware, report, fault, kernel, boot, feature, patch, qemu]

```
kasan: simplify quarantine_put call site
mm/mm_init.c: report kasan-tag information stored in page->flags
kasan, kmemleak: reset tags when scanning block
kasan: add kasan mode messages when kasan init
kasan, arm64: move initialization message
```

These fixes are devoted to special tags when scanning a memory block in both the Kernel Address Sanitizer (KASAN) [50, 51] and Kernel Memory Leak (kmemleak) subsystems. This is done to provide more detailed information about memory usage and aid in debugging.

To summarize all these fixes, it can be noted that modern changes in the Linux memory subsystem relate to virtual memory, large memory blocks and various tools for memory monitoring and debugging. Working with memory is so complex that without additional tools that are provided by the kernel, debugging and optimization are not possible.

5.4 Scheduling (/linux/kernel/sched)

Vector #1: [deadline, dl_bw, task, total_bw, bw, dl_nr_running, dl_rq, sched, runtime, dl, parameter, sched_deadline, bandwidth, problem, value]

```
sched/deadline: Fix a bug in dl_overflow()
sched/debug: Add deadline scheduler bandwidth ratio to /proc/sched_debug
sched/deadline: Fix switching to -deadline
sched/deadline: Show leftover runtime and abs deadline in /proc/*/sched
sched/deadline: Fix migration of SCHED_DEADLINE tasks
```

A large number of fixes are dedicated to the Deadline Scheduler subsystem [52]. Namely, when tasks with a deadline exceeding the maximum value were not being correctly handled; when it is needed to add the ability to display the leftover runtime and absolute deadline for SCHED_DEADLINE tasks in the /proc/*/sched file, and also when tasks were not being correctly migrated between CPU cores.

Vector #2: [numa, node, fault, memory, rate, sched, task, mm, pte config_sched_debug, scan, access, commit, local]

```
sched/numa: Count pages on active node as local
mm: sched: numa: Delay PTE scanning until a task is scheduled on a new node
Revert "mm: sched: numa: Delay PTE scanning until a task is scheduled on a new
node"
mm: numa: Rate limit setting of pte_numa if node is saturated
sched/numa: Disable sched_numa_balancing on UMA systems
```

The following fixes are dedicated to scheduling in NUMA (Non-Uniform Memory Access) systems to improve performance and reduce memory latency. In particular, pages on the currently active node were not being counted as local; delays can be appear in scanning of page table entries (PTEs)

for NUMA balancing until a task is scheduled on a new node; pages on a saturated NUMA node could cause excessive memory usage; the scheduler NUMA balancing feature on Uniform Memory Access (UMA) systems, which do not have non-uniform memory access characteristics should be disabled.

Vector #3: [cpu, idle, load, task, active, online, domain, utilization, fair, balancing, nohz, numa, core, ilb, balance]

```
sched: Improve load balancing in the presence of idle CPUs
sched: Prevent raising SCHED_SOFTIRQ when CPU is !active
sched: Allow migrating kthreads into online but inactive CPUs
sched: Fix cpu_active_mask/cpu_online_mask race
sched/fair: Trigger the update of blocked load on newly idle cpu
```

The current fixes are devoted to the work of the scheduler when a CPU is idle. For example,to better handle situations where there are idle CPUs in the system; prevent the scheduler from raising the SCHED_SOFTIRQ interrupt on CPUs that are not currently active; allow kernel threads (kthreads) to be migrated to CPUs that are online but currently inactive; ensure that the scheduler updates the blocked load on CPUs that become idle. This can improve performance by making better use of available CPU resources.

Vector #4: [macro, sched, kernel, debug, check, commit, default, fair, field, magic, new, number, rename, runtime, schedstat_val]

```
sched: Move SCHED_LOAD_SHIFT macros to kernel/sched/sched.h
sched/debug: Fix /proc/sched_debug regression
sched/debug: Rename 'schedstat_val()' -> 'schedstat_val_or_zero()'
sched: Fix kernel-doc warnings in kernel/sched/fair.c
sched: Move wait.c into kernel/sched/
```

These fixes are about reorganizing the code.

Vector #5: [load, cpu, task, group, cfs_rq, entity, weight, cgroup, vruntime, fair, time, sched, cpuset, real, share]

```
sched/fair: Fix unfairness caused by missing load decay
sched/fair: Fix incorrect task group ->load_avg
sched: Avoid scale real weight down to zero
sched/fair: Remove task and group entity load when they are dead
sched: Aggregate total task_group load
```

The presented fixes usually relate to errors in the implementation of Completely Fair Scheduler (CFS) [53] and to task group load, which could lead to unfairness in the distribution of CPU resources. The design of CFS leads to the concept of schedulable entities, where tasks are managed by the scheduler as a whole. The fixes address the following situations: when the scheduler was not properly decaying the load of tasks over time, when it was calculating the load average of task groups incorrectly, when it was not properly removing the load of dead tasks and task groups, as well us there is some fix improves the way the scheduler aggregates the total load of task groups.

Vector #6: [sched, kernel, declaration, error, function, core, asm, implicit, paravirt, cputime, include, print_cfs_rq, werror, change, commit]

```
sched/headers: Prepare header dependency changes, move the <asm/paravirt.h>
include to kernel/sched/sched.h
sched/s390: Fix compile error in sched/core.c
sched/debug: Move print_cfs_rq() declaration to kernel/sched/sched.h
sched/core: Fix compilation error when cgroup not selected
sched/debug: Move the print_rt_rq() and print_dl_rq() declarations to…
```

These fixes are also about reorganizing the code.

Vector #7: [cpu, kernel, sched_domain, time, commit, flag, foundation, infradead, link, linux]

```
sched/isolcpus: Fix "isolcpus=" boot parameter handling when !CONFIG_CPUMASK_OFFSTACK
sched: Fix the broken sched_rr_get_interval()
sched/cputime: Resync steal time when guest & host lose sync
sched/nohz: Fix overflow error in scheduler_tick_max_deferment()
sched: Fix init NOHZ_IDLE flag
```

Current fixes match with scheduler domains [54] from different scheduling subsystems, including a bug that was not returning the correct time quantum for round-robin scheduling; a bug with incorrect CPU isolation; and an issue that steal time of a virtual CPU in a guest operating system could become out of sync with the host operating system.

Vector #8: [rq, task, cpu, curr, deadline, test, enqueue_task_dl, run, dl, hotplug, lock, sched, stress, offline, wa]

```
sched/deadline: Fix bad accounting of nr_running
sched/rt: Fix task stack corruption under __ARCH_WANT_INTERRUPTS_ON_CTXSW
sched/deadline: Add missing update_rq_clock() in dl_task_timer()
sched: Add missing rcu protection to wake_up_all_idle_cpus
sched/deadline: Fix the intention to re-evalute tick dependency for offline CPU
```

The next series of fixes concerns the work of deadline and real-time schedulers leading to incorrect scheduling decisions and system crashes: a counter for tasks and a clock value in a run queue were not properly updated; the task stack could become corrupted when interrupts were enabled during a context switch; a function was not properly protected by RCU.

Vector #9: [rt, task, cpu, current, run, user, tick, deadline, priority, pull, value, issue, sched, signal, time]

```
sched/rt: Avoid updating RT entry timeout twice within one tick period
sched/rt: Kick RT bandwidth timer immediately on start up
sched/rt: Do not pull from current CPU if only one CPU to pull
sched/rt: Fix RT utilization tracking during policy change
sched,cgroup: Fix up task_groups list
```

These fixes relate to RT (real-time) type schedulers for control groups [55, 56]. The latter concept defines a set of tasks, and all their future children, as a hierarchical group with specialized behavior. In details, the fixes address an issue where the real-time scheduler was updating the timeout value for a task twice within a single tick period; correct an issue where the real-time scheduler was delaying the start of the bandwidth timer; improve the behavior when the real-time scheduler was unnecessarily pulling tasks from the current CPU even when there was only one other CPU available, and fix the situation where the task_groups list was not properly updated when tasks were moved between cgroups, which lead to unnecessary overhead and degraded performance.

Vector #10: [update_rq_clock, add, core, sched, address, bug, clock, rq, update, double, effort, minimize, post_init_entity_util_avg, way]

```
sched/core: Add missing update_rq_clock() call in set_user_nice()
sched/core: Add missing update_rq_clock() in post_init_entity_util_avg()
sched/core: Add missing update_rq_clock() in detach_task_cfs_rq()
sched/core: Fix double update_rq_clock) calls in attach_task()/detach_task()
```

These fixes solve the same problem: lack or excess of function calls to update time data in the run queue per cpu structure of the scheduler.

To summarize, the fixes in the scheduling subsystem are mainly devoted to improving deadline, fair and realtime scheduling for groups, correct work with idle cpu state, calculation of deadlines and updating time counters, and work in NUMA systems.

5.5 Network (/linux/net)

Vector #1: [error, return, code, null, rate, function, check, value, case, net, use, icmp, mac80211, pointer, netfilter]

```
Bluetooth: fix error return code in rfcomm_add_listener()
sctp: fix error return code in __sctp_connect()
ieee802154: fix error return code in ieee802154_add_iface()
ah6: fix error return code in ah6_input()
netfilter: nf_conntrack: fix error return code
```

A large number of fixes concern returning error codes from functions. Such fixes were made for many parts of the network subsystem.

Vector #2: [net, inline, fb, mm, kasan, fc, kernel, include, socket, common, core, arch, linux, ipv6, x86]

```
net: sched: fix race condition in qdisc_graft()
ipv6: Fix KASAN: slab-out-of-bounds Read in fib6_nh_flush_exceptions
net: igmp: respect RCU rules in ip_mc_source() and ip_mc_msfilter()
tcp: cdg: allow tcp_cdg_release() to be called multiple times
devlink: Fix use-after-free when destroying health reporters
```

These fixes are related to typical errors in the code in this case for processing network things: a race condition that could occur when multiple processes attempt to modify the same network queueing discipline; a memory access error in the IPv6 forwarding information base (FIB6) code; a violation of RCU synchronization rules in the Internet Group Management Protocol (IGMP) code; an issue with the TCP Congestion Detection and Avoidance (CDG) algorithm, which could cause crashes or other errors if its release function was called multiple times; and a memory management error in the devlink subsystem, which could cause a "use-after-free" error if a health reporter object was destroyed while still in use.

Vector #3: [tcp, variable, unused, error, udp, warning, remove, function, ipv4, patch, net, compile, ipv6, mac80211, packet]

```
net: ipv4: avoid unused variable warning for sysctl
[IPV4]: Fix "ipOutNoRoutes" counter error for TCP and UDP
mac80211: fix warning: unused variable invoke_tx_handlers
ipv4: ipconfig: avoid unused ic_proto_used symbol
[TCP]: TCP highspeed build error
```

These fixes exclusively concern warnings and errors when building code for implementing things related to TCP.

Vector #4: [rfkill, net, error, whitespace, state, input, issue, change, core, pointer, case, hardware, software, class, device]

```
net/rfkill/rfkill-input.c needs <linux/sched.h>
[NET] RFKILL: Fix whitespace errors.
rfkill-gpio: include linux/mod_devicetable.h
rfkill: copy the name into the rfkill struct
rfkill: allow to get the software rfkill state
```

These fixes encapsulate various fixes associated with the RFKILL subsystem [57]. RFKILL can be used to disable wireless communication on a device. This can include devices such as Wi-Fi and Bluetooth radios. The RFKILL subsystem in the kernel provides a unified interface for controlling these devices. rfkill-gpio is a driver for RFKILL devices that are controlled by GPIO pins [58]. It allows the kernel to control the state of the RFKILL device by toggling the GPIO pin.

Vector #5: [vlan, bridge, device, packet, add, error, hardware, port, driver, network, filtering, header, notification, address, state]

```
bridge: vlan: Prevent possible use-after-free
rtnetlink: catch -EOPNOTSUPP errors from ndo_bridge_getlink
```

```
[VLAN]: Fix hardware rx csum errors
bridge: add vlan filtering change for new bridged device
vlan: Enable software emulation for vlan accleration.
```

This series of fixes is dedicated to VLAN and bridge devices. In particular, an error occurred when a user tries to get information about a bridge device that does not support the requested operation. Another fix addresses an issue with hardware rx checksums in VLAN devices. Hardware rx checksums are used to verify the integrity of network packets, but some devices were reporting incorrect checksums. Some fix adds support for VLAN filtering on new bridged devices. VLAN filtering is used to separate network traffic into different virtual LANs. And a fix enables software-based VLAN acceleration, ensuring that these systems can benefit from improved network performance. VLAN acceleration is used to improve network performance by offloading some of the processing required for VLAN filtering to hardware, but not all systems have this hardware capability.

Vector #6: [command, hci, error, typo, request, event, nfc, support, code, patch, net, add, framework, hci_req_add send]

```
Bluetooth: Fix HCI request framework
Bluetooth: HCI request error handling
Bluetooth: Reorganize set_connectable HCI command sending
Bluetooth: Add support for custom event terminated commands
NFC: Implement HCI driver or internal error management
```

These fixes are dedicated to Bluetooth and HCI framework [59]. Previously, we already realized that fixes for bluetooth are often made in the Linux kernel [15, 16]. The HCI request framework is responsible for sending commands and receiving responses between the host and the Bluetooth controller. This fix corrects the implementation of the framework, ensuring that commands are sent and received correctly.

Vector #7: [static, address, br, edr, discovery, controller, function, make, le, mode, command, patch, type, dual, setting]

```
Bluetooth: Introduce controller setting information for static address
Bluetooth: Support static address when BR/EDR has been disabled
Bluetooth: Fix issue with switching BR/EDR back on when disabled
Bluetooth: Check capabilities in BR/EDR and LE-Only discovery
Bluetooth: Fix advertising data flags with disabled BR/EDR
```

This fixes are devoted to a feature in the Bluetooth subsystem of the Linux kernel that allows users to set a static Bluetooth address for their device's Bluetooth controller. Previously, the Bluetooth address was generated randomly each time the device was restarted, which could cause issues with some Bluetooth devices that relied on a consistent address. With this fix, users can now set a static address that will remain the same across reboots. Then, the static Bluetooth address set in the previous fix is still used even if the device's Bluetooth controller has been disabled for BR/EDR (Basic Rate/Enhanced Data Rate) communication. Previously, if BR/EDR was disabled, the Bluetooth subsystem would generate a new random address each time the controller was enabled again.

Vector #8: [net, refcount, add, netpoll, tracker, device, struct, error, netns, socket, core, leak, dev_put, help, kernel]

```
net: add net device refcount tracker to struct pneigh_entry
netpoll: add net device refcount tracker to struct netpoll
net: add net device refcount tracker to struct netdev_rx_queue
net: add net device refcount tracker to struct netdev_adjacent
net: bridge: add net device refcount tracker
```

The mentioned fixes in the Linux kernel are related to adding a net device refcount tracker to different structures within the network subsystem. The reference count tracker helps to keep track

of the number of references to this structure, which is useful for ensuring accurate and reliable management of the network devices.

Vector #9: [unlock, error, path, mac80211, function, return, double, wifi, case, commit, deflink, lock, mlme, net, netfilter]

```
ceph: unlock on error in ceph_osdc_start_request()
mac80211: unlock on error path in ieee80211_ibss_join()
wifi: mac80211: unlock on error in ieee80211_can_powered_addr_change()
Bluetooth: delete a stray unlock
tipc: unlock in error path
```

These fixes check that any locks held are released before returning from a specified function even where an error is appeared. For example, in the Transparent Inter-Process Communication (TIPC) protocol implementation, this fix ensures that if an error occurs during specific operations, any locks held during the operation are released. This can prevent potential resource leaks.

Vector #10: [port, devlink, switch, cpu, dsa, user, case, driver, net, link, number, upstream, mtu, sw1p4, attribute]

```
net: dsa: give preference to local CPU ports
net: dsa: Fix type was not set for devlink port
devlink: append split port number to the port name
net: dsa: calculate the largest_mtu across all ports in the tree
netfilter: nf_ct_h323: fix bug in rtcp natting
```

These fixes are devoted to issues with network ports and improve the network performance on systems using Distributed Switch Architecture (DSA) [60]. DSA is a framework in Linux kernel that allows network switches to be controlled by the kernel. For example, with a fix, the kernel gives preference to the local CPU (Central Processing Unit) ports, which reduces latency and improves packet processing efficiency. Another fix addresses a bug related to the Maximum Transmission Unit (MTU) calculation in the Distributed Switch Architecture. And there is a fix to resolve a bug in the nf_ct_h323 module, which is responsible for Network Address Translation (NAT) handling for the H.323 protocol in netfilter (Linux network packet filtering subsystem). The bug specifically relates to Real-Time Control Protocol (RTCP) packets and their translation during NAT.

5.6 IRQ (/linux/kernel/irq)

Vector #1: [function, cleanup, kernel documentation parameter comment genirq irq patch recent update add addition commit core]

```
genirq: Fix handle_bad_irq kerneldoc comment
[PATCH] more kernel-doc cleanups, additions
[PATCH] genirq: cleanup: no_irq_type cleanups
Update __irq_domain_alloc_fwnode() function documentation
cpumask: Cleanup more uses of CPU_MASK and NODE_MASK
```

A large series of fixes in the IRQ subsystem is devoted to documentation, comments and code organization.

Vector #2: [pointer, null, function, check, alias, bug, crash, debugfs, genirq, handle, boot, compilation, dereference, driver, irqdomain]

```
genirq/irqdomain: Check pointer in irq_domain_alloc_irqs_hierarchy()
genirq/debugfs: Remove redundant NULL pointer check
genirq: Fix null pointer reference in irq_set_affinity_hint()
sparseirq: work around __weak alias bug
irqdomain: Fix debugfs formatting
```

Another series of fixes is devoted to errors in working with pointers.

Vector #3: [affinity, node, irq, interrupt, domain, genirq, software, case, device, driver, irqdomain, setup, allocation, bad, default]

```
genirq: Respect NUMA node affinity in setup_irq_irq affinity()
genirq/affinity: Fix node generation from cpumask
genirq: Move initial affinity setup to irq_startup()
irqdomain: Allow software nodes for IRQ domain creation
genirq/irqdomain: Add an irq_create_mapping_affinity() function
```

The main domain fixes in the IRQ subsystem relate to working with IRQ affinity [61]. That specifies which target CPUs are permitted for a given IRQ source. For example, a fix ensures that interrupts are assigned to the correct NUMA node based on the affinity settings, improving performance and reducing latency. Another fix corrects an issue where the node generation from a cpumask was not working correctly, leading to incorrect affinity settings. The next fix improves the initialization of interrupt affinities by moving it to the irq_startup() function, ensuring that affinities are set correctly from the beginning. Some other fix allows the creation of software nodes for interrupt domains, improving flexibility and allowing for better management of interrupts. Finally, a fix adds a new function to create interrupt mappings with specific affinity settings, allowing for more fine-grained control over interrupt allocation.

Vector #4: [gpio, error, drivers, ko, export, irq, add, function, chip, commit, export_symbol_gpl, following, generic, genirq]

```
irq: Add EXPORT_SYMBOL_GPL to function of irq generic-chip
device property: export irqchip_fwnode_ops
genirq: Add missing irq_to_desc export for CONFIG_SPARSE_IRQ=n
irq: Export handle_fasteoi_irq
```

The following fixes concern exporting some IRQ functionality so that it is available to other parts of the kernel, enabling them to interact with the irq chip functionality. For example, a fix adds the missing export statement for irq_to_desc, ensuring that it is accessible to other kernel components even when CONFIG_SPARSE_IRQ is disabled. Another fix exports the handle_fasteoi_irq symbol from the irq subsystem. handle_fasteoi_irq is a function that handles FastEOI (Fast End Of Interrupt) interrupts, which is a type of interrupt management mechanism [61].

Vector #5: [state, flag, interrupt, disabled, genirq, hardware, irq_data, add, callback, force, suspend, access, avoid, certain, change]

```
genirq/PM: Properly pretend disabled state when force resuming interrupts
genirq: Add irq disabled flag to irq_data state
genirq: Reflect IRQ_MOVE_PCNTXT in irq_data state
```

The latest series of big fixes relates to IRQ state management. A fix addresses a problem related to resuming interrupts after they have been disabled. In certain scenarios, interrupts may be forcibly resumed even if they were disabled. However, the interrupt controller may still maintain some internal state indicating that the interrupts are disabled. This fix ensures that the proper disabled state is correctly reflected when interrupts are forcefully resumed. In a next fix, an additional flag for the irq_data state is added to indicate whether the interrupts are disabled. This enables efficient checking of the disabled state, as well as simplifying the interrupt handling code. Another fix enhances the implementation of the IRQ_MOVE_PCNTXT feature in the genirq subsystem. IRQ_MOVE_PCNTXT is used to move an interrupt context from one CPU to another.

As a result, we can say that the main fixes for IRQ in recent years have been devoted to code refactoring, improving work with IRQ affinity, providing some functions for export to other subsystems, and improving IRQ state management to support disabling interrupts and interrupt context transfer to other processors.

5.7 x86 (/linux/arch/x86)

Vector #1: [xen, x86, microcode, error, arch, function, kernel, declaration, build, implicit, cpu, acpi, include, page, guest]

```
arch/x86/xen/suspend.c: include xen/xen.h
```

```
x86, xen: fix hardirq.h merge fallout
xen/tracing: fix compile errors when tracing is disabled.
xen/trace: Fix compile error when CONFIG_XEN_PRIVILEGED_GUEST is not set
xen: Move xen_setup_callback_vector() definition to include/xen/hvm.h
```

A large number of fixes for x86 are devoted to various aspects of implementing support for the Xen hypervisor [62] in the Linux kernel.

Vector #2: [iommu, tlb, flush, amd, x86, gart, add, aperture, device, kvm, vpid, patch, code, function, l2]

```
AMD IOMMU: add stats counter for domain tlb flushes
AMD IOMMU: add stats counter for single iommu domain tlb flushes
amd-iommu: add function to flush tlb for all devices
x86, AMD IOMMU: add detect code for AMD IOMMU hardware
x86, AMD IOMMU: add dma_ops initialization function
```

Another large number of fixes are devoted to the implementation of AMD IOMMU and working with TLB. IOMMU (Input-Output Memory Management Unit) is a hardware component that allows for virtualization of devices and memory management. TLB (Translation Lookaside Buffer, see the discussion in the tutorial [63]) is a cache used by the processor to store recently accessed memory translations. The TLB is used to speed up memory access. For example, a fix adds a statistics counter to track the number of times the IOMMU domain TLB is flushed. This improves performance monitoring and helps identify potential issues. Another fix adds detection code for AMD IOMMU hardware, allowing the kernel to properly identify and utilize the hardware.

Vector #3: [x86, vector, kernel, linux, redhat, boot, apic, intel, interrupt, code, cpu]

```
x86/entry/64: Clear registers for exceptions/interrupts, to reduce speculation
attack surface
x86/entry/64/compat: Clear registers for compat syscalls, to reduce speculation
attack surface
x86/traps: Fix up general protection faults caused by UMIP
x86, kasan, ftrace: Put APIC interrupt handlers into .irqentry.text
x86/kconfig/32: Rename CONFIG_VM86 and default it to 'n'
```

The set of updates focuses entirely on fixing known vulnerabilities. The first fix targets a security vulnerability called "speculation attack," which allows attackers to exploit modern processors' speculative execution feature to access sensitive data. This fix clears registers used during exceptions and interrupts to prevent unauthorized access. The second fix is similar, but it specifically addresses compatibility syscalls in 64-bit mode. These syscalls are used to run 32-bit applications on a 64-bit system, and the fix clears registers used during these syscalls to prevent exploitation. The third fix addresses a bug where UMIP was causing GP faults in certain situations, which could be exploited by attackers. UMIP is a security feature that prevents certain instructions from being executed in user mode. The fourth fix moves the APIC interrupt handlers to a more secure location in memory to prevent exploitation. Finally, the last fix renames a kernel configuration option and sets its default value to "no" to prevent attackers from exploiting the VM86 feature, which allows 16-bit applications to run in virtual 8086 mode. This feature is no longer necessary in modern systems and can be exploited by attackers to gain access to sensitive information.

Vector #4: [page, table, error, x86, code, k8, powernow, kvm, cpumask, cleanup, fault, struct, guest, address, bit]

```
KVM: SVM: Limit PFERR_NESTED_GUEST_PAGE error_code check to L1 guest
x86/espfix/xen: Fix allocation of pages for paravirt page tables
kvm: svm: Add support for additional SVM NPF error codes
x86, vmi: put a missing paravirt_release_pmd in pgd_dtor
mm: add pt_mm to struct page
```

The next set of fixes focuses on correcting vulnerabilities in the KVM, Xen hypervisor, AMD Secure Virtual Machine, and Virtual Machine Interface subsystems. These fixes aim to prevent malicious

guest operating systems from crashing the host system and exploiting vulnerabilities to gain access to sensitive information. The updates include limiting error code checks for PFERR_NESTED_GUEST_PAGE to only the first level guest, correcting page allocation for paravirtualized page tables, adding support for additional error codes related to Nested Page Faults, and adding a new field to track memory management context in the struct page data structure.

Vector #5: [mmu, page, tdp, kvm, shadow, vcpu, sp, x86, guest, code, change, check, l1, use, table]

```
KVM: x86/mmu: Pivot on "TDP MMU enabled" to check if active MMU is TDP MMU
KVM: x86/mmu: Protect marking SPs unsync when using TDP MMU with spinlock
KVM: MMU: move mmu pages calculated out of mmu lock
KVM: x86: nSVM: fix switch to guest mmu
KVM: MMU: move the relevant mmu code to mmu.c
```

The KVM hypervisor also undergo changes to improve its performance and prevent data corruption. The changes include checking if TDP (two dimensional paging) MMU is enabled instead of checking the CPU model to ensure that the correct MMU is used for virtual machines running on different CPU models. The addition of spinlock protection when marking shadow page tables as unsynchronized in a TDP MMU environment prevents race conditions. Moving the calculation of MMU pages outside of the MMU lock reduces contention on the lock. Finally, a bug in the nested SVM code is corrected to ensure that the hypervisor switches to the guest MMU when running a nested virtual machine. Some links and an approach to formally verify such things are presented in the paper [64].

Vector #6: [reboot, x86, dell, pci, quirk, add, platform, acpi, kernel, power, show_bug, ce4100, id]

```
ACPI, x86: fix Dell M6600 ACPI reboot regression via DMI
x86/reboot: Add Zotac ZBOX CI327 nano PCI reboot quirk
x86/ce4100: Fix reboot by forcing the reboot method to be KBD
x86: Add Dell OptiPlex 760 reboot quirk
x86/reboot: Limit Dell Optiplex 990 quirk to early BIOS versions
```

These fixes address specific issues related to reboot functionality on certain computer systems by adding or limiting certain quirks in the kernel.

Vector #7: [bank, error, mce, cpu, x86, check, hardware, machine, number, value, amd, interrupt, type, register, disable]

```
x86/MCE: Determine MCA banks' init state properly
x86/mce: Avoid reading every machine check bank register twice.
x86/mce/AMD, EDAC/mce_amd: Enumerate Reserved SMCA bank type
x86/MCE: Initialize mce.bank in the case of a fatal error in mce_no_way_out()
x86/MCE/AMD: Define a function to get SMCA bank type
```

These fixes have been made to correct a bug in the Machine Check Architecture (MCA) code, optimize it by avoiding unnecessary reads of machine check bank registers, add support for enumerating reserved System Management Control Address (SMCA) bank types in the MCA code for AMD processors, and initialize the MCA bank variable in case of a fatal error in the MCA code. MCA is an internal architecture subsystem which detects and captures errors occurring within the microprocessor's logic [65].

Vector #8: [pci, bridge, window, e820, host, lenovo, region, space, x86, acpi, bar, address, device, resource, bus]

```
x86/PCI: Revert "x86/PCI: Clip only host bridge windows for E820 regions"
x86/PCI: Add kernel cmdline options to use/ignore E820 reserved regions
x86/PCI: Mark ATI SBx00 HPET BAR as IORESOURCE_PCI_FIXED
x86/PCI: Use host bridge _CRS info on Foxconn K8M890-8237A
x86/PCI: Ignore E820 reservations for bridge windows on newer systems
```

These fixes aim to improve the functionality of PCI devices in computer systems, in order to improve PCI device enumeration and resource allocation and ensure efficient use of available memory

resources (the so-called PCI quirks [66]). They include reverting a previous change that caused issues with certain devices accessing memory regions, adding kernel command line options for users to specify the use of reserved memory regions, marking the HPET BAR on ATI SBx00 chipsets as a fixed PCI resource, updating the PCI code for Foxconn K8M890-8237A systems, and modifying the PCI code to ignore reserved memory regions for bridge windows on newer systems.

Vector #9: [x86, style, problem, i8259, function, cleanup, impact, include, arch, asm, shutdown, code, error, file, irqinit_32]

```
x86: i8259.c fix style problems
x86: module_64.c fix style problems
x86: irq_32.c fix style problems
x86: time_32.c fix style problems
x86: irqinit_32.c fix style problems
```

Here are code style issues in x86-related components.

Vector #10: [bit, 32, 64, ptrace, x86, kernel, value, argument, code, syscall, firmware, mixed, mode, sign, signal]

```
x86_64: make ptrace always sign-extend orig_ax to 64 bits
x86/mpx: Fix 32-bit address space calculation
x86: ptrace: set TS_COMPAT when 32-bit ptrace sets orig_eax>=0
x86/efi: Truncate 64-bit values when calling 32-bit OutputString()
efi/libstub: Distinguish between native/mixed not 32/64 bit
```

The fixes for x86_64 and x86/efi are important to ensure proper handling of 32-bit applications running on a 64-bit system and to distinguish between native and mixed mode in the EFI libstub code accurately.

To sum up, fixes for this historical part of the kernel are devoted to code reorganization, addressing known types of vulnerabilities, better support for virtualization, playing around with quirks for specific hardware, as well as providing support for legacy systems and subsystems.

5.8 ARM64 (/linux/arch/arm64)

It turned out that a huge number of fixes for the ARM platform are devoted primarily to additions to device trees [67, 68]. There are a huge number of hardware configurations for the platform, and such configurations are described through a device tree in order to be accessible later programmatically by loading the appropriate drivers. Our software classified such fixes and the first 6 classes found describe support for different types of devices in the device tree.

Vector #1: [add, node, arm64, dts, renesas, support, device, thermal, tpu, patch, soc, zone, fd, hdmi, r8a77995]

```
arm64: dts: renesas: r8a77995: add thermal device support
arm64: dts: renesas: r8a774c0: Add thermal support
arm64: dts: renesas: r8a779a0: Add TPU device node
arm64: dts: renesas: r8a77990: add thermal device support
arm64: dts: renesas: r8a77965: Add SATA controller node
```

Vector #2: [clock, node, arm64, dts, add, pcie, controller, qcom, sdm845, ipa, phy, support, soc, dt, device]

```
arm64: dts: exynosautov9: add fsys0/1 clock DT nodes
arm64: dts: qcom: sdm845: add apr nodes
arm64: dts: rockchip: add the PCIe PHY for RK3399
arm64: dts: sdm845: Add lpasscc node
arm64: dts: qcom: sm6350: add IPA node
```

Vector #3: [dts, arm64, add, device, node, r8a7795, r8a7796, renesas, enable, salvator, hihope, support, usb3, board, patch]

```
arm64: dts: r8a7795: add usb2_phy device nodes
```

```
arm64: dts: r8a7795: Add USB-DMAC device nodes
arm64: dts: r8a7796: Add USB-DMAC device nodes
arm64: dts: renesas: r8a7795: add USB3.0 peripheral device node
arm64: dts: renesas: r8a7796: add USB3.0 peripheral device node
```

Vector #4: [meson, gxbb, arm64, dts, add, node, gxl, pin, usb, cec, enable, regulator, controller, amlogic, dt]

```
ARM64: dts: meson-gxbb: Add Meson GXBB PWM Controller nodes
ARM64: dts: meson-gxbb: Add CEC pins nodes
ARM64: dts: meson-gxbb-p20x: Enable USB Nodes
ARM64: dts: meson-gxbb-odroidc2: Enable USB Nodes
ARM64: dts: meson-gxbb-nexbox-a95x: Enable USB Nodes
```

Vector #5: [regulator, pwm, supply, usb, add, arm64, dts, vddcpu, device, power, dummy, gpu, boot, change, board, meson]

```
arm64: dts: meson-g12b: Fix the pwm regulator supply properties
arm64: dts: meson-sm1: Fix the pwm regulator supply properties
arm64: tegra: Fix Jetson Nano GPU regulator
arm64: dts: meson: odroid-c2: Add missing regulator linked to P5V0 regulator
arm64: dts: rockchip: Correct regulator for USB host on Odroid-Go2
```

Vector #6: [node, table, opp, soc, arm64, dts, bus, add, cpu_thermal, doe, property, build, cpu, following, qup]

```
arm64: dts: imx8mq: Move the opp table out of bus node
arm64: dts: ls1046a: Move cpu_thermal out of bus node
arm64: dts: ls1043a: Move cpu_thermal out of bus node
arm64: dts: ls1012a: Move cpu_thermal out of bus node
arm64: dts: qcom: sm8250: Move qup-opp-table out of soc node
```

Vector #7: [tcsr, mutex, address, space, mmio, device, arm64, dts, qcom, syscon, allow, check, dedicated, dt, halt]

```
arm64: dts: qcom: sdm845: switch TCSR mutex to MMIO
arm64: dts: qcom: msm8998: switch TCSR mutex to MMIO
arm64: dts: qcom: sm8150: switch TCSR mutex to MMIO
arm64: dts: qcom: qcs404: switch TCSR mutex to MMIO
arm64: dts: qcom: sc7180: switch TCSR mutex to MMIO
```

TCSR (Thread Context Save and Restore) is a feature in Qualcomm processors that provides a mechanism for saving and restoring the state of a thread during context switching. These fixes modify the implementation to use MMIO (Memory-Mapped Input/Output), which enables direct access to the TCSR register through memory addresses.

Vector #8: [vdso, arm64, offsets, kernel, support, arch, makefile, rule, enable, mremap, page, vma, bti, handling, orphan]

```
arm64: fix vdso-offsets.h dependency
arm64/vdso: Support mremap() for vDSO
arm64: vdso: Enable vDSO compat support
arm64: vdso: Map the vDSO text with guarded pages when built for BTI
arm64: vdso: enable orphan handling for VDSO
```

These fixes are devoted to the virtual dynamic shared object (vDSO) [69] implementation on arm64. vDSO provides a fast and efficient interface for certain system calls, allowing user-space programs to directly access kernel functionality without needing to transition to kernel mode. Enabling vDSO compatibility support ensures that both newer and older versions of user-space programs can make use of the vDSO. The mremap() system call is used to move memory mappings to a new address range or change their size. By enabling mremap() support for the vDSO, it allows for more efficient management of the vDSO memory mapping. A listed fix enhances the security of the vDSO on arm64 when built with support for Branch Target Identification (BTI). BTI is a security feature that

protects against certain control-flow attacks. Orphan handling refers to the ability to handle processes that have become detached or disassociated from their parent process. Enabling orphan handling for the vDSO ensures that it behaves correctly in scenarios where a process using the vDSO is orphaned.

Vector #9: [asm, arm64, function, arch, error, declaration, implicit, kernel, insn, mark, rutland, include, linux, bug]

```
arm64: fix missing asm/io.h include in kernel/smp_spin_table.c
arm64: fix missing asm/alternative.h include in kernel/module.c
arm64: fix missing asm/pgtable-hwdef.h include in asm/processor.h
arm64: fix missing linux/bug.h include in asm/arch_timer.h
arm64: kaslr: Use standard early random function
```

These fixes concern the reorganization of included files with headers for the ARM64 platform.

Vector #10: [sve, sme, state, flag, arm64, register, host, load, storage, vcpu, access, kvm, mode, context, el1]

```
arm64/sme: Don't flush SVE register state when allocating SME storage
arm64/signal: Clean up SVE/SME feature checking inconsistency
KVM: arm64: Move SVE state mapping at HYP to finalize-time
KVM: arm64: Trap host SVE accesses when the FPSIMD state is dirty
KVM: arm64: Always start with clearing SVE flag on load
```

Several fixes have been made to improve the handling and synchronization of the Scalable Vector Extension (SVE) state [70]. The first fix prevents unnecessary flushing of the SVE register state during Secure Memory Encryption (SME) storage allocation. The next fix resolves an inconsistency in feature checking for SVE and SME during signal processing, ensuring correct behavior. Moving the mapping of SVE state in Kernel-based Virtual Machine (KVM) to finalize-time ensures proper handling and synchronization. Host SVE accesses are now trapped in KVM when the Floating-Point SIMD (FPSIMD) state is dirty, preventing corruption. Lastly, consistently clearing the SVE flag during load operations in KVM guarantees correct initialization and handling of SVE features during virtualization, avoiding potential issues related to its state.

Thus, for ARM64, the defining fixes are the reorganization of the device tree set, reorganization of the code, synchronization of access to the state of internal registers and operations with virtual dynamic shared objects.

6. Conclusion

In this study, we utilized natural language messages from the Linux kernel git repository to identify the most common types of errors. Our approach involved clustering fixing commit messages to gain insight into the prevalent error classes in specific domains, such as memory management and scheduling subsystems. Unlike previous studies, we included links to specific commits, providing examples of bug reports for each generalized vector identified. This differed from our earlier approach, which relied on matching similar messages and working with frequently occurring messages but not similar generalization vectors.

Our analysis yielded satisfactory results, presenting the main vectors (in the form of keywords) and example messages for each class of fix. We also conducted a manual summary analysis for each class, revealing typical fixes for each subsystem and the automated tools used to detect them. We referenced these tools in our article and concluded that developers must study them when working with the corresponding subsystem.

It is important to note that the analysis conducted in this study is relatively straightforward and can be applied to any repository, with the ability to evaluate changes retrospectively. By selecting specific dates and parts of the repository, it is possible to identify errors and gain valuable insights into their prevalence. However, there is always room for improvement in any analysis, and our current implementation could benefit from enhancements such as refining the detection of fixing

commits, improving the commit coupling thresholds for hierarchical clustering, better normalization and removal of stop words, and developing methods for automatically describing each class of errors found. These improvements are crucial for ensuring the accuracy and effectiveness of our analysis, and we plan to address them in future research.

Ultimately, our goal is to provide system developers with a comprehensive understanding of the most common types of errors in various subsystems, along with the tools and techniques needed to address them effectively.

Acknowledgement

The work on this article was carried out at personal expense and in their own time by the authors.

We would like to express our gratitude to A.K. Petrenko and A. Kamkin for organizing the SYRCoSE colloquium, as well as A. Martyshkin and A. Vorontsov for conducting excursions around Penza and Nikolsk.

References

[1]. Starovoytov N., Golovnev N., Staroletov S. Towards methods to automatically identify the most common errors in Linux by analyzing git commit messages. In Proc. of the Spring/Summer Young Researchers' Colloquium on Software Engineering, 2023

[2]. Chou A., Yang, J. Chelf B., Hallem S., Engler D. An empirical study of operating systems errors. In Proc. of the eighteenth ACM symposium on Operating systems principles, 2001, pp. 73–88.

[3]. Palix N., Thomas G., Saha S., Calves C., Lawall J., Muller G. Faults in Linux: Ten years later. In Proc. of the sixteenth international conference on Architectural support for programming languages and operating systems, 2011, pp. 305–318.

[4]. Mutilin V., Novikov E., Khoroshilov A. Analysis of typical errors in Linux OS drivers (in Russian), Proceedings of the Institute for System Programming of the Russian Academy of Sciences, vol. 22, pp. 349–374, 2012.

[5]. Novikov E.M. Evolution of the Linux OS kernel (in Russian). Proceedings of the Institute for System Programming of the Russian Academy of Sciences, vol. 29, no. 2, pp. 77–96, 2017.

[6]. Lu L., Arpaci-Dusseau A.C., Arpaci-Dusseau R.H., Lu S. A study of Linux file system evolution. ACM Transactions on Storage (TOS), vol. 10, no. 1, pp. 1–32, 2014.

[7]. Tan L., Liu C., Li Z., Wang X., Zhou Y., Zhai C. Bug characteristics in open source software. Empirical software engineering, vol. 19, pp. 1665–1705, 2014.

[8]. 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, vol. 68, no. 4, pp. 1356–1383, 2019.

[9]. Kernel.org. Bugzilla. Available at: https://bugzilla.kernel.org, accessed Sep. 14, 2023.

[10]. Melo J., Flesborg E., Brabrand C., Wasowski A. A quantitative analysis of variability warnings in Linux. In Proc. of the Tenth International Workshop on Variability Modelling of Software-intensive Systems, 2016, pp. 3–8.

[11]. Hoang T., Lawall J., Tian Y., Oentaryo R.J., Lo D. PatchNet: Hierarchical deep learning-based stable patch identification for the Linux kernel. IEEE Transactions on Software Engineering, vol. 47, no. 11, pp. 2471–2486, 2019.

[12]. Tian Y., Lawall J., Lo D. Identifying Linux bug fixing patches. In Proc. of 2012 34th international conference on software engineering (ICSE). IEEE, 2012, pp. 386–396.

[13]. Acher M., Martin H., Pereira J.A., Blouin A., Khelladi D.E., Jezequel J.-M. Learning from thousands of build failures of Linux kernel configurations. Ph.D. dissertation, Inria; IRISA, 2019.

[14]. Levenshtein V.I. Binary codes with correction of dropouts, insertions and character substitutions (in Russian). Reports of the Academy of Sciences, vol. 163, no. 4. Russian Academy of Sciences, 1965, pp. 845–848.

[15]. Staroletov S. M. Researching the most common bugs in the Linux kernel by analysing commits in the git repository (in Russian). System Administrator, vol. 4(197), pp. 73–77, 2019 Available at: http://samag.ru/archive/article/3859, accessed Sep. 14, 2023.

[16]. Staroletov S. A survey of most common errors in Linux kernel. SYRCoSE Poster session, 2017.

[17]. Hann M. Towards an algorithmic methodology of lemmatization. Bulletin Association for Literary and Linguistic Computing, vol. 3, no. 2, pp. 140–150, 1975.
[18]. Categorial Variation Database (version 2.1). Available at: https://github.com/nizarhabash1/catvar, accessed Sep. 14, 2023.
[19]. Manning C.D., Surdeanu M., Bauer J., Finkel J. R., Bethard S., McClosky D. The Stanford CoreNLP natural language processing toolkit. In Proc. of 52nd annual meeting of the association for computational linguistics: system demonstrations, 2014, pp. 55–60.
[20]. Class StanfordCoreNLP. Available at: https://nlp.stanford.edu/nlp/javadoc/javanlp/edu/stanford/nlp/pipeline/StanfordCoreNLP.html, accessed Sep. 14, 2023.
[21]. Salton G., Fox E. A., Wu H. Extended boolean information retrieval Communications of the ACM, vol. 26, no. 11, pp. 1022– 1036, 1983.
[22]. H. Steinhaus et al. Sur la division des corps mate´riels en parties. Bull. Acad. Polon. Sci, vol. 1, no. 804, p. 801, 1956.
[23]. Ester M., Kriegel H.-P., Sander J., Xu X. et al. A density-based algorithm for discovering clusters in large spatial databases with noise. KDD, vol. 96, no. 34, 1996, pp. 226–231.
[24]. Ward J.H. Hierarchical grouping to optimize an objective function. Journal of the American statistical association, vol. 58, no. 301, pp. 236–244, 1963.
[25]. McKenney P.E. A Tour Through TREE_RCU's Grace-Period Memory Ordering. Available at: https://www.kernel.org/doc/html/latest/RCU/Design/Memory-Ordering/Tree-RCU-Memory-Ordering.html, accessed Sep. 14, 2023.
[26]. McKenney P.E., Fernandes J., Boyd-Wickizer S., Walpole J. RCU usage in the Linux kernel: Eighteen years later. ACM SIGOPS Operating Systems Review, vol. 54, no 1, pp. 47-63, 2020.
[27]. Linux kernel guide. Writing kernel-doc comments. Available at: https://docs.kernel.org/doc-guide/kernel-doc.html, accessed Sep. 14, 2023.
[28]. Staroletov. S., Chudov R. An anomaly detection and network filtering system for Linux based on Kohonen maps and variable-order Markov chains. In Proc. Conference of Open Innovations Association, vol. 32, pp. 280-290, 2022. – EDN NNASCK.
[29]. Linux kernel guide. Livepatch. Available at: https://www.kernel.org/doc/html/latest/livepatch/livepatch.html, accessed Sep. 14, 2023.
[30]. Linux kernel guide. Ftrace - Function Tracer. Available at: https://www.kernel.org/doc/html/latest/trace/ftrace.html, accessed Sep. 14, 2023.
[31]. Linux manual page. timer_create(2). Available at: https://man7.org/linux/man-pages/man2/timer_create.2.html, accessed Sep. 14, 2023.
[32]. Linux kernel guide. ktime accessors. Available at: https://www.kernel.org/doc/html/latest/core-api/timekeeping.html, accessed Sep. 14, 2023.
[33]. S. Boyd. Timekeeping in the Linux kernel. Available at: https://elinux.org/images/0/0e/Timekeeping_in_the_Linux_Kernel_0.pdf, accessed Sep. 14, 2023.
[34]. Linux kernel guide. Runtime locking correctness validator. Available at: https://www.kernel.org/doc/html/latest/locking/lockdep-design.html, accessed Sep. 14, 2023.
[35]. Linux kernel guide. Checkpatch. Available at: https://www.kernel.org/doc/html/latest/dev-tools/checkpatch.html, accessed Sep. 14, 2023.
[36]. Linux kernel guide. BPF (Berkeley Packet Filter) Documentation. Available at: https://www.kernel.org/doc/html/latest/bpf/index.html, accessed Sep. 14, 2023.
[37]. Linux kernel guide. R. Davoli, M. Di Stefano, 2019. Berkeley Packet Filter: theory, practice and perspectives (Doctoral dissertation, Master's thesis, Universita di Bologna). Available at: https://amslaurea.unibo.it/19622/1/berkeleypacketfilter_distefano.pdf, accessed Sep. 14, 2023.
[38]. Google. Syzkaller. Available at: https://github.com/google/syzkaller, accessed Sep. 14, 2023.
[39]. Linux kernel guide. RCU Torture Test Operation. Available at: https://www.kernel.org/doc/html/latest/RCU/torture.html, accessed Sep. 14, 2023.
[40]. Lawall J., Muller G. Automating Program Transformation with Coccinelle. In Proc. of NASA Formal Methods Symposium, pp. 71-87, 2022.
[41]. Serrano L., Nguyen V.A., Thung F., Jiang L., Lo D., Lawall J., Muller G. SPINFER: Inferring Semantic Patches for the Linux Kernel. In Proc. of 2020 USENIX Annual Technical Conference (USENIX ATC 20), pp. 235-248, 2020.
[42]. Linux kernel guide. Kernel Samepage Merging. Available at: https://www.kernel.org/doc/html/latest/admin-guide/mm/ksm.html, accessed Sep. 14, 2023.

[43]. Lameter C. Slab allocators in the Linux Kernel: SLAB, SLOB, SLUB. LinuxCon/Düsseldorf, 2014. Available at: https://events.static.linuxfound.org/sites/events/files/slides/slaballocators.pdf, accessed Sep. 14, 2023.
[44]. Linux kernel guide. Kernel Memory Leak Detector. Available at: https://www.kernel.org/doc/html/latest/dev-tools/kmemleak.html, accessed Sep. 14, 2023.
[45]. Linux kernel guide. Kernel Electric-Fence (KFENCE). Available at: https://docs.kernel.org/dev-tools/kfence.html, accessed Sep. 14, 2023.
[46]. Linux kernel guide. Transparent Hugepage Support. Available at: https://www.kernel.org/doc/html/latest/admin-guide/mm/transhuge.html, accessed Sep. 14, 2023.
[47]. Linux kernel guide. DAMON: Data Access MONitor. Available at: https://docs.kernel.org/mm/damon/index.html, accessed Sep. 14, 2023.
[48]. Linux kernel guide. Concepts overview. Available at: https://www.kernel.org/doc/html/latest/admin-guide/mm/concepts.html, accessed Sep. 14, 2023.
[49]. Corbet J. Clarifying memory management with page folios. Available at: https://lwn.net/Articles/849538/, accessed Sep. 14, 2023.
[50]. Linux kernel guide. The Kernel Address Sanitizer (KASAN). Available at: https://www.kernel.org/doc/html/latest/dev-tools/kasan.html, accessed Sep. 14, 2023.
[51]. Google. Kernel sanitizers. Available at: https://github.com/google/kernel-sanitizers, accessed Sep. 14, 2023.
[52]. Linux kernel guide. Deadline Task Scheduling. Available at: https://www.kernel.org/doc/html/latest/scheduler/sched-deadline.html, accessed Sep. 14, 2023.
[53]. Linux kernel guide. CFS Scheduler. Available at: https://www.kernel.org/doc/html/latest/scheduler/sched-design-CFS.html, accessed Sep. 14, 2023.
[54]. Linux kernel guide. Scheduler Domains. Available at: https://www.kernel.org/doc/html/latest/scheduler/sched-domains.html, accessed Sep. 14, 2023.
[55]. Menage. P. Linux kernel guide. Control Groups. Available at: https://www.kernel.org/doc/html/latest/admin-guide/cgroup-v1/cgroups.html, accessed Sep. 14, 2023.
[56]. Linux kernel guide. Real-Time group scheduling. Available at: https://www.kernel.org/doc/html/latest/scheduler/sched-rt-group.html, accessed Sep. 14, 2023.
[57]. Linux kernel guide. rfkill - RF kill switch support. Available at: https://docs.kernel.org/driver-api/rfkill.html, accessed Sep. 14, 2023.
[58]. Linux kernel guide. Using GPIO Lines in Linux. Available at: https://www.kernel.org/doc/html/latest/driver-api/gpio/using-gpio.html, accessed Sep. 14, 2023.
[59]. Bluetooth. Core Specification 5.4, 2023. Available at: https://www.bluetooth.com/specifications/specs/core-specification-5-4/, accessed Sep. 14, 2023.
[60]. Linux kernel guide. Distributed Switch Architecture (DSA). Available at: https://docs.kernel.org/next/networking/dsa/dsa.html, accessed Sep. 14, 2023.
[61]. Linux kernel guide. Linux generic IRQ handling. Available at: https://www.kernel.org/doc/html/latest/core-api/genericirq.html, accessed Sep. 14, 2023.
[62]. Barham P., Dragovic B., Fraser, et al. Xen and the art of virtualization. ACM SIGOPS operating systems review, vol. 37 no. 5, 164-177, 2013.
[63]. Arpaci-Dusseau R.H., Arpaci-Dusseau A.C. Operating systems: Three easy pieces. Arpaci-Dusseau Books, LLC, 2018. Available at: https://pages.cs.wisc.edu/~remzi/OSTEP/vm-tlbs.pdf, accessed Sep. 14, 2023.
[64]. Li S.W., Li X., Gu R., Nieh J. Hui J.Z. A secure and formally verified Linux KVM hypervisor. In Proc. of 2021 IEEE Symposium on Security and Privacy (SP), pp. 1782-1799, 2021.
[65]. Constantinescu C. AMD EPYC™ 7002 series–a processor with improved soft error resilience. In Proc. of 2021 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks-Supplemental Volume (DSN-S), pp. 33-36, 2021.
[66]. Linux. Work-arounds for many known PCI hardware bugs. Available at: https://elixir.bootlin.com/linux/latest/source/drivers/pci/quirks.c, accessed Sep. 14, 2023.
[67]. Linaro. The Devicetree Specification. Current Release. Available at: https://www.devicetree.org/specifications/, accessed Sep. 14, 2023.
[68]. Linux kernel guide. Linux and the Devicetree. Available at: https://www.kernel.org/doc/html/latest/devicetree/usage-model.html, accessed Sep. 14, 2023.
[69]. Linux manual page. vdso(7). Available at: https://man7.org/linux/man-pages/man7/vdso.7.html, accessed Sep. 14, 2023.

[70]. Linux kernel guide. Scalable Vector Extension support for AArch64 Linux. Available at: https://www.kernel.org/doc/Documentation/arm64/sve.txt, accessed Sep. 14, 2023.

Информация об авторах / Information about authors

Никита Александрович СТАРОВОЙТОВ – магистрант и ассистент кафедры прикладной математики. Сфера научных интересов: кластеризация, анализ текстов.

Nikita Alexandrovich STAROVOYTOV – master student and assistant at the department of Applied Mathematics. Research interests: clusterization, text analysis.

Николай Андреевич ГОЛОВНЕВ – магистрант кафедры прикладной математики. Сфера научных интересов: анализ текстов, JVM.

Nikolay Andreevich GOLOVNEV – master student at the department of Applied Mathematics. Research interests: text analysis, JVM.

Сергей Михайлович СТАРОЛЕТОВ – кандидат физико-математических наук, доцент. Сфера научных интересов: формальная верификация, model checking, киберфизические системы, операционные системы.

Sergey Mikhailovich STAROLETOV – Candidate of Physical-Mathematical Sciences (PhD), associate professor (docent). Research interests: formal verification, model checking, cyber-physical systems, operating systems.