**Syzkaller Investigation**

The tool investigation and code improvement working group focuses on application of tools and handling the tool results, improving the kernel based on the tools’ feedback.

In the endeavor of fulfilling that mission, we investigate the syzkaller tool. Syzkaller is a tool to do coverage-guided fuzzing on the kernel with the system call interface. Tbd: explain Fuzzing

* the most well-known and acknowledged fuzzing tool in the kernel community.

Syzkaller is developed, with various contributions and forks carrying experimental features from other companies and research departments around the world.

For our investigation on the syzkaller tool, we maintain a server machine, which runs two instances of the syzkaller tool. One instance does fuzzing on some recent kernel version of the long-term stable branch 5.10.y, which reports its results at https://elisa-builder-00.iol.unh.edu/syzkaller/. We manually update the kernel version to the latest release version and restart the fuzzing campaign a few times throughout the year. The second instance does fuzzing on linux-next and reports its results at https://elisa-builder-00.iol.unh.edu/syzkaller-next/. This instance automatically gets updates for the latest available version of linux-next and then restarts the fuzzing campaign when a new version is available. The two kernels are built with the build configuration ARCH=x86-64 defconfig and some further kernel build configuration additions to enable the most well-known sanitizers that are useful for identifying typical bug classes with the fuzzing campaigns.

These two kernels that are exposed to testing are continuously evolving. The latest state of linux-next changes very fast, as almost every reasonable patch proposed by anyone may appear in linux-next and linux-next is released on a close-to-daily basis. Roughly, that results in an average daily change rate of 170 commits, as about 12,000 changes end up in linux-next within 10 weeks (the common duration of one release cycle). For those more interested in the change rate of linux-next, the up-to-date statistics on the linux-next changes can be found at http://neuling.org/linux-next-size.html.

Also the 5.10.y stable version, changes with an average speed of XXX changes (i.e., XXX commits) per year, whereas those changes are largely bug fixes, but not new functionality, and rather smaller in size compared to the changes being added to linux-next.

Further, the fuzzing campaign is continuously running and there is really no final saturation of observed executions compared to all possible executions. So, every new execution of the campaign may potentially reveal a new bug; however, the longer these campaigns are running, the less likely it is to observe a new bug, simply because they become increasingly more compute-intense to discover.

So despite the fact that the kernels under test and the testing campaign is continuously evolving and the summary below may be soon outdated, here is a quick summary of the results at the time of writing, in the mid of February 2023:

For both instances, we have been running syzkaller for 20 months (from end of June 2021 to mid of February 2023) with 4 virtual machines each. This makes a total of 80 months of exposing the kernel to sequences of system calls to attempt to reach a high testing coverage and find as many different crashes as possible, each for linux-next and for the 5.10.y branch.

When we investigate the results from our two syzkaller instances, we notice that most identified syzkaller findings are non-reproducible, and only a few findings come with C reproducers. On the instance running recent v5.10 versions, we found XX with C reproducers among the total number YY, in the overall time this instance has been running.

On the instance running linux-next, we found XX with C reproducers among the total number YY.

A finding that comes with a C reproducer is a kernel crash with a specific type of stacktrace that could be repeatedly (i.e., at least twice, once during the campaign, and a second time when checking for the C reproducer) triggered with a specific C program generated by the fuzzing machinery. If after the first kernel crash, the syzkaller system is not able to trigger the kernel crash again with the recorded information from the first kernel crash, then this finding is considered non-reproducible. As the kernel’s system state is very large and changes in various non-deterministic ways, crashes may appear in specific system states, but it is simply practically impossible to re-run a sequence of operations that leads again to the specific system state where this kernel crash may be observed. In such a case, the chances of obtaining a C reproducer is low and the finding remains non-reproducible.

The reproducible findings and their corresponding C reproducers are of course interesting, as they allow much more informed analysis and debugging.

On the campaign with the v5.10.y LTS kernels, many C reproducers have most likely been fixed.

For a few findings, we did already confirm them being fixed with a few manual runs that show that some findings do not appear anymore since a specific version and are probably resolved with a bugfix with that version.

We also ran the two latest C reproducers. Tbd.

<https://elisa-builder-00.iol.unh.edu/syzkaller/report?id=34e8ccdc0c7fb916735e544ea99234de75455d32>

So, we know that the two issues identified with the v5.10.y LTS kernel still happen with the latest kernel version.

So, one might well ask why have the issues for the known C reproducers then not been fixed yet?

Just because a C reproducer may have been found that crashes the kernel, it is not necessarily a kernel bug. In short, the kernel provides enough rope to hang yourself. The long story is that a randomly generated program, which is running as root, may very well execute operations that setup the system in such a way that the system is essentially broken and invoking operation on such a broken kernel system setup will crash, as expected and as “intended” by such a setup. Given a fuzzing campaign that runs just long enough, it may very well at some point simply construct such programs and report the corresponding crash. The syzkaller tool already has some mechanisms implemented to avoid the obvious ways to create broken system setups, but there are numerous (if not even infinitely many) different creative ways to setup broken systems, and the syzkaller developers have not set up mechanisms and limits to avoid all of those numerous possibilities.

Further, the fuzzing campaign uses fault injections . **tbd! Explain: the issue with fault injections!**

So, for those findings, we really would need to understand if they point to a real issue that should be fixed. Additionally, we may possibly already understand better which functions in the code cause the issue and have a first idea how to possibly resolve the issue.

If we have reached that point of understanding, we can probably judge if it is worth reporting the findings to the kernel community and the specific maintainers, and then further follow up with patch proposals to discuss them with all experts involved.

This described investigation for the found C reproducers takes quite some time, especially time for learning how to debug such issues and getting a good understanding of the kernel code, in which the kernel crash is rooted.

So far, the members of our group simply did not yet find the time and priority to make real progress on such an investigation. Given that the two reproducers are just indicating a memory leak under a specific error path, caused by an artificially-injected fault, it might very well be that the error path cannot be actually triggered in an actual well-configured system or that the error path is effectively caused by another internal failure so severe to the system, e.g. the kernel would indicate that its virtual file system is not practically usable anymore, that a small memory leak really is not a problem anyone practically would consider relevant. At that point, the running kernel would need to be rebooted anyway; and the memory leak of course just disappears with this reboot.

However, we are really looking forward to new contributors that are willing to make a real attempt at such an investigation. If successful, we would have a kernel patch that fixes a long-standing issue in the kernel and generally, the kernel community would overall be grateful for going through the challenge to the point of creating a proper patch to fix an issue. Certainly, with the skills acquired, there are many more findings in further syzkaller campaigns by others in the larger community that could then be investigated as well. Within our group, we could also increase the computing resources, syzkaller instances and fuzzing campaigns to search for more findings and C reproducers that could then be further investigated. For sure, a contributor that can analyze syzkaller findings in depth and create proper bug-fix patches for the kernel is not going to run out of work soon!

So, we are looking forward to welcoming you as a new contributor in our group, and help you in getting acquainted with our previous work and our common endeavor of applying tools, handling the tool results, and improving the kernel based on the tools’ feedback.

As a first step, feel free to have a look yourself to see the latest state of our syzkaller campaign at the URLs above. As the results of our syzkaller fuzzing campaigns are continuously growing, new interesting findings may have been discovered in the meantime between the time of writing and you reading this blog post.