ISBN 978-981-11-3671-9
Proceedings of 2017 the 7th International Workshop on Computer Science and Engineering (WCSE 2017)
Beijing, 25-27 June, 2017, pp. 244-248
doi: 10.18178/wcse.2017.06.042

Fault Simulation and Memory Leak Detection over Custom Kernel Module by Using KEDR

Syeda Huma Jabeen, Gaoshou Zhai[+] and Ruixia Zhai

School of Computer and Information Technology, Beijing Jiaotong University, Beijing, 100044, China

Abstract. Fault simulation and memory leak detection of Linux kernel modules, especially of device drivers are a critically important task. However, due to the special nature of the kernel operation, it is very challenging to perform runtime analysis of kernel modules of interest without adverse influence on the rest of the kernel. In this paper, we have tested our own compiled custom kernel modules for fault simulation and memory leak detection by using KEDR framework, an extensible runtime analysis system for Linux kernel modules, which employs various kinds approaches to perform different kinds of analysis.

Keywords: device driver testing, fault simulation, kernel module, memory leak detection.

1. Introduction

One of the critical and very important tasks is verification of Linux kernel modules, especially under system crash or failure situations. The most commonly used examples includes device drivers, file systems, audio video software's, networks, and various other facilities which are important to operating system and specifically employed as kernel modules [1-4].

Some memory areas are indirect access to kernel modules which are used by other kernel parts, and it's opposite to user-space applications. At lower level kernel modules, can communicate with hardware, may also have capabilities that might be rarely possible in outside kernel space. Furthermore, memory occupied will not be freed automatically unless module is unloaded so resource allocation is important if any problem happened in kernel module can produce alarming situation [1-2, 5].

KEDR (Kernel Drivers in Runtime) framework system [6] is used for dynamic analysis of kernel modules (device drivers, file system modules, etc.) in Linux on x86 systems. KEDR tools are very well developed and they are easy to be understood than any other tools developed for fault simulation or fault injection [7-9].

KEDR operates only on the special modules which are chosen by the user and doesn't affect the other kernel parts. It can detect memory leak, perform fault simulation as well as other kinds of data collection and analysis. KEDR-based tools are very efficient and have already proven their efficiency by finding errors in several widely-used kernel modules [10].

2. General Process of Fault Injection Testing Based on KEDR

KEDR provides powerful mechanisms to simulate various faults and the essence of related techniques is to force some of the calls made by the target module to fail. Perhaps, KEDR simulates the failure deprived of indeed calling the target function. User can modify and take a control over the scenarios (in which condition which function may face the failure). In general, the overall flow of fault injection testing based on KEDR can be divided into installation of KEDR package, loading of target module, setting of fault types and parameters, starting of testing, finally check and verify the results (refer to Fig. 1).

[+] Corresponding author. Tel.: +86-10-51684177; fax: +86-10-51684177.
E-mail address: gszhai@bjtu.edu.cn

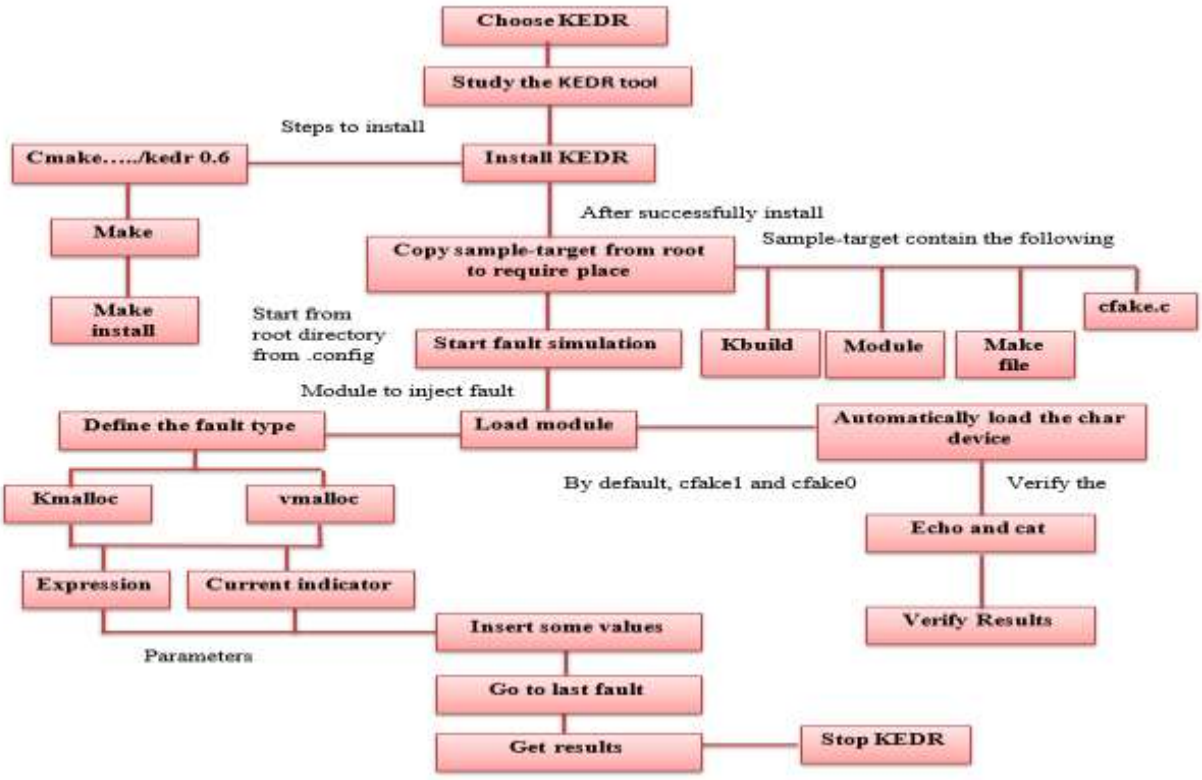

Fig. 1: A diagrammatic overview of KEDR fault simulation and memory leak detection over kernel modules

In this paper, KEDR fault simulation is verified over two aspects:

- Fault is injected for *kmalloc, vmalloc* and character devices (*cfake0, cfake1*).
- Compare results as for the kedr-sample-module under different conditions.

3. Platform for KEDR Tests

To implement the fault injection by using KEDR, VMware station 10.0 is installed on our machine and a virtual machine is built and Ubuntu 15.04 with Linux-header-3.19.0-15-generic is installed on that virtual machine. Then source codes of KEDR version 0.6 are downloaded, compiled and installed on the virtual machine. Furthermore, it's mandatory to know about its basic requirements as listed below before working with KEDR:

- Kernel version ≥ 3.2, can be checked by uname -r
- Currently supported for x86 and x86-64 architectures
- *Cmake* version should be ≥ 2.8
- C++ and GNU C compiler version should be ≥ 4.0
- GNU Make
- Some other packages and tools to build kernel modules

4. Runtime Tests on KEDR

4.1. Fault Simulation

KEDR framework for fault simulation provides the facility to inject a fault on user custom kernel module, and it also provides the example sample-target. In this paper, a custom kernel module called *ExampleLKM* is used to verify a KEDR fault simulation and memory leak functions.

To do so, we have used the same makefile and kbuild of sample-target but after making change in the module name as custom kernel module name. Fault simulation point is selected as *kmalloc* and current-indicator and expression are selected as fault simulation indicators [11].

4.1.1. Fault Simulation for *kmalloc*

To inject a fault over the custom module, firstly, we start KEDR fault simulation by the command "**kedr** start *ExampleLKM* -f *fsim.conf*". Hence, KEDR is successfully loaded and associated with our custom kernel module. At the same time, KEDR loads various fault simulation modules along with it. In addition, we have make the custom kernel module as ***ExampleLKM.ko***.

To inject a kmalloc, we have set a "kmalloc" as a current indicator. Meanwhile, we need set a value 1 to expression, which made the custom kernel module unable to allocate memory at time of loading. By setting a value 1 over the expression means always, make it fail to allocate memory. To verify the fault simulation, we have tried to load the module with ***insmod*** command such as "*insmod ExampleLKM.ko*", provided us an error as expected memory allocation fail "cannot allocate memory". After that, we have tried to make simulation failed by setting a value 0 over the expression, that means never do a fault simulation. This time, the module has been successfully loaded with the same command and it has efficaciously occupied the memory allocation.

4.1.2. Fault Simulation for *copy-from-user* and *copy-to-user*

In this part, we have tried to simulate fault over devices drivers associated with custom kernel module for copy-from-user and copy-to-user. KEDR frameworks provides the two device drivers (*cfake0* and *cfake1*) along with its example, we have used the same devices but make them associated with our custom kernel module (*ExampleLKM*).

We started the KEDR fault simulation and do make module as we do before. As we know after the successful start of KEDR it loads various fault simulation module along with it, but for this example, we have not specified any specific fault simulation module such as *kedr_fsim_cmm.ko* etc. and using the basic functionalities provided by KEDR fault simulation.

Firstly, we have set the value common as a current indication for copy-from-user and copy-to-user simulation points and load the module with *insmod*, write some string and read that string from custom module. It successfully writes and read the string. At this time, we need to simulate fault for function to copied data to and from user device drivers (*cfake0* and *cfake1*). We have tried to use the read and write operation of device driver to write data to and to read data from custom kernel module associated with these device drivers.

As it is already defined above that by setting a value 1 to expression make the function fail. So, to simulate a fault we have set the value 1 to expression of copy-from-user and copy-to-user files. Now again we write some string to *cfake0* and read *(/dev/cfake0*) it gave us error "Bad address", mean fault has been simulated successfully. We also write and read the device (*/dev/cfake1*) and got the same error message "Bad address". To make this fault simulation failed set value 0 in both file and this time no error message while write and read of devices.

4.1.3. About Verification of Fault Simulation

Besides the direct results about executing related command, both kernel messages (via the command *dmesg*) and system log files (via the command *syslog*) can be used to verify fault simulation. In addition, kedr framework provides the facility to verify the results via *defugfs* directory (refer to Fig. 2).

```
root@ubuntu:/home/me/Desktop/example# cat /sys/kernel/debug/kedr_fault_simulation/last_fault
__kmalloc at [<f871c0a0>] cfake_init_module+0xa0/0x1000 [ExampleLKM]
root@ubuntu:/home/me/Desktop/example#
```

Fig. 2: Last fault position of ExampleLKM

4.2. Memory Leaks Detection

KEDR framework provides the facility to detect a memory leak, and we have tried to detect a memory leak for custom kernel. Firstly, we started the kedr memory leak detection by the command "**kedr** start *ExampleLKM* -f *leak_check.conf*". KEDR is successfully started but load various memory leak modules along with it such as *kedr_leak_check.ko* and *kedr_lc_common_mm.ko*. Just as above, our custom kernel module is required to be built at the same time. And the module is successfully loaded with the command ***insmod*** and character devices (*cfake0* and *cfake1*) are created accordingly. After that we have done some work with character devices (*dd if=/dev/zero of=/dev/cfake1 bs=1 count=8*) which prompted

8+0 records in

8+0 records out

8 bytes (8 B) copied, 0.000748022 s, 10.7 kB/s

Now at this time, write something to cfake1 character device and remove the module with *rmmod* command. After doing *dmesg -c* we got the results of memory leak detection (refer to Fig. 3).

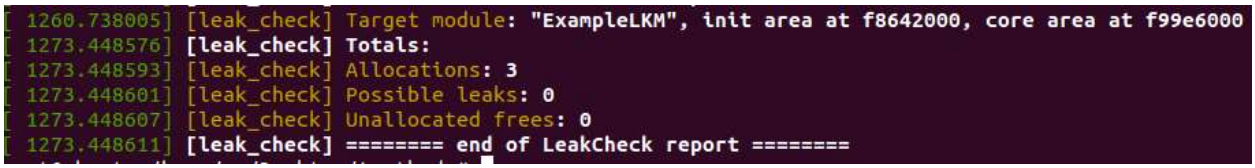

Fig. 3: Screenshot about execution of memory leak detection

5. Summary and Discussion

The system's availability and reliability can be increased by testing the device drivers by fault simulation tools or various fault injection techniques. In this paper, we have used KEDR to inject faults into custom kernel module related for device read and write operation along with memory allocation. We found KEDR framework is efficient and very helpful to inject and simulate a fault.

Starting from version 2.6.20, LFIF (Linux Fault Injection Framework) [5] has been built into the kernel and can be used to inject various faults into memory and block devices. Furthermore, page allocation errors, slab errors and disk I/O errors can be simulated as for newer Linux kernel. In addition, a special test tool with SCSI fault injection [12-13] can be used to inject faults in the processing of the target SCSI command. At the same time, Linux Test Project (LTP) [14-15] provides various test suites to test different aspects of the Linux operating systems. And LTP can be combined with LFIF so as to improve related test strength and extents.

Comparative experiments have been done on the basis of the *kedr-sample-target* module over normal conditions without LFIF, and conditions with LFIF or with KEDR (refer to Tab. 1).

Table I: Results about comparative experiments for kedr-sample-target under NORMAL, LFIF or KEDR

Platform requirements	Successfully load module	Load the char devices (cfake0, cfake1)	Verify the devices	Successfully injected fault
NORMAL	Yes	Yes	Yes	No
LFIF	Yes	Yes	Yes	Do not support module-level injection
KEDR	Yes	Yes	Yes	Yes

As for faults supported by LFIF, KEDR and SCSI injector, results about corresponding comparative study can be concluded as Tab. 2.

Table II. Results about comparative study among LFIF, KEDR and SCSI injector

Tools	kmalloc faults	Failslab faults	Disk I/O faults	RAID faults	Module-Level Injection
LFIF	Yes	Yes	Yes	No	No
KEDR	Yes	No	No	No	Yes
SCSI injector	No	No	Yes	Yes	No

According to the above analysis, a better fault injector tool ought to support as many as possible faults including not only faults about memory such as *kmalloc* and *failslab* faults but also faults about disk I/O faults and RAID faults and even other device operation faults such as network I/O faults. Moreover, it ought

to support module-level fault injection so that a special kernel module or device driver can be tested flexibly while different faults can be specified at each time. Obviously, available fault injection framework and tools don't satisfy these requirements completely and perfectly. And there are many work need to be done.

6. Acknowledgements

Part of the research presented in this paper was performed with the support of the National Natural Science Fund of China (No.6167209). And we would express our great thanks to Dr. Eugene A. Shatokhin (Russian academy of Sciences) for his pertinent advices.

7. References

[1] V.V. Rubanov and E.A. Shatokhin. Runtime verification of Linux kernel modules based on call interception. Software Testing, Verification and Validation (ICST), Fourth International Conference on Software Testing, Verification and Validation. 2011. pp. 180-189. IEEE.

[2] S. D. Shekar, B. B. Meshram and P. Vashapriya. Device driver fault simulation using kedr. International Journal of Advanced Research in Computer Engineering & Technology Volume 1, Issue 4, June 2012:580-584.

[3] K. Cong, L. Lei, Z. Yang, and F. Xie. 2015, July. Automatic fault injection for driver robustness testing. In Proceedings of the 2015 International Symposium on Software Testing and Analysis (pp. 361-372). ACM.

[4] K. Tsirunyan, V. Martirosyan and A. Tsyvarev. The Spruce System: quality verification of Linux file systems drivers.

[5] Linux Kernel Documentation (Online). http://www.mjmwired.net/kernel/Documentation/

[6] KEDR Project, http://linuxtesting.org/kedr.

[7] T. Naughton, W. Bland, G. Vallee, C. Engelmann and S. L. Scott. Fault injection framework for system resilience evaluation: fake faults for finding future failures. In Proceedings of the 2009 workshop on Resiliency in high performance. pp. 23-28. ACM.

[8] Kernel memory Leak Detector, http://www.mjmwired.net/kernel/Documentation/kmemleak.txt.

[9] J. J. Bai, H.Q. Liu, Y.P. Wang, and S. M. Hu. 2016, June. Testing error handling code in device drivers using characteristic fault injection. In 2016 USENIX Annual Technical Conference (USENIX ATC 16) pp. 635-647. USENIX Association.

[10] Problems in Kernel Modules Found by KEDR. http://openfacts2.berlios.de/wikien/index.php/BerliosProject: KEDR.

[11] https://github.com/euspectre/kedr/wiki/kedr_manual_getting_started.

[12] Tanaka, K., Hamaguchi, M., Sato, T. and Tatsukawa, K. SCSI Fault Injection Test. Proceedings of the Linux Symposium Ottawa, Ontario Canada, July 23rd–26th, 2008:205-214.

[13] https://github.com/dwalkes/scsi_fault_injection_test_tool

[14] Larson, P. Testing Linux® with the Linux Test Project. Ottawa Linux Symposium June 26th–29th, 2002 Ottawa, Ontario Canada: 265-273.

[15] https://github.com/linux-test-project/ltp