Finding Double-Unlock Bugs with Shape-and-Effect Analysis

Anders Fischer-Nielsen

December 16th 2019

Contents

Todo list

Problem definition statement - should be very clear in the introduction, using bold font, a label "problem" a box, or something like this. 4

1 Introduction

The Linux kernel supports a vast array of computer architectures and runs on a multitude of devices from personal computers to servers and embedded devices from everything from wireless access points to smart TVs, smartphones and refrigerators. Errors in the Linux kernel therefore affect a multitude of devices and therefore have a potential significant negative impact.

An important aspect of kernel programming is management and manipulation of resources, be it devices, file handles, memory blocks, and locks. Locks are used extensively in the C source code of the Linux kernel in order to allow parallelization of subsystems within the kernel while at the same time avoiding race conditions. Static analysers allow detection of errors in the C source code of the Linux kernel by reasoning about this resource manipulation. A control flow graph can be found for the components of the kernel, which can then in turn be statically analysed to detect possible ressource manipulation errors.

One such resource manipulation error is a *double unlock* error. A thread holding a lock and then releasing this lock more than once will result in undefined behaviour, according to the POSIX standard. This standard is an attempt to generate a standard version of UNIX to facilitate application portability and defines how C constructs should be implemented by UNIX OS vendors. The `pthread.h` file defines the spinlock constructs which are used in the Linux kernel and the accompanying specification is of note here, since this file describes how the structs found in the header should behave. The section describing `pthread_spin_unlock` defines the behaviour of the `spin_unlock` unlocking operation of a spinlock observed in the kernel code, and is defined as:

> "The results are undefined if the lock is not held by the calling thread. [...] The results are undefined if this function is called with an uninitialized thread spin lock." [2]

If a thread wanting to unlock a lock does not currently hold that lock, the lock has either been unlocked already or has never been locked. This will in both instances lead to undefined behaviour at a kernel level, possibly making the operating system behave in unexpected ways.

Undefined behaviour is problematic since a program depending on undefined behaviour might not break today, but could break in the future. E.g. if a program depending on undefined behaviour has been implemented on an assumption that the output of the undefined behaviour will always be within a certain interval, but due to the nature of undefined behaviour this changes in a compiler update, the program suddenly breaks, leading to a software panic. Undefined behaviour is exactly that - *undefined* - and assumptions can therefore not be made about its output.

Developing a way to detect such errors is desirable in order to allow developers to detect errors in their code, leading to safer programs. An implementation for detecting such errors can furthermore be used as a tool for evaluating the performance of other code analysis tools, such as a tool for automated software repair. Patching the inverse error, a double lock error, could potentially introduce an unwanted double unlock error in the code. An implementation of a double unlock checker could serve as a correctness evaluation tool, and as a test harness for automated double lock repair.

This report will detail our approach for detecting such double unlock errors based on previous work, give an overview of the concrete implementation of our approach and finally evaluate results of validating files in the Linux kernel components.

Problem definition statement - should be very clear in the introduction, using bold font, a label "problem" a box, or something like this.

2 Background

The shape-and-effect inference system described by Abal et. al. [1] enables

> "[...] efficient and scalable inter-procedural reasoning about resource manipulation"

Abal et. al. describe a method for detecting double-lock bugs in the kernel source code using the EBA analyzer.

3 Finding Double Unlock Bugs

4 Results

Given the open source nature of the Linux kernel, the project sees contributions from many developers. Contributions are integrated into the code base using a Distributed Version Control System (DVCS), namely *git*.

The log of contributions, *commits*, can be queried in order to find patches to errors discovered in the code base. Here we are interested in identifying patches that fix double unlock bugs, accomplished by querying the complete log of all commits to the code base.

This allows extracting a set of confirmed bugs, that is bugs which have been fixed by developers. Furthermore, picking a point in time before the patch was merged with the code base allows finding the bug in the code base before it was fixed, allowing verification of our proposed analyzer on an implicitly *confirmed* bug. If a patch has been submitted and accepted in the log, it must implicitly have been confirmed as being an actual bug by the maintainers of the code base.

The accuracy of our proposed code analyzer can be determined by extracting n actual double unlock bugs and seeing whether these true positives are detected by the analyzer.

The following sections will detail the evaluation of the proposed analyzer and highlight interestring cases of bugs that either were or were not detected.

4.1 True Positives

4.1.1 4dd75b33

The patch 4dd75b33 patches a double unlock error in the file fs/ubifs/orphan.c, part of the file system components of the kernel.

The function orphan_delete contains two if statements which, if evaluating to true will result in a double unlock of &c->orphan_lock, since the function ubifs_delete_orphan calls orphan_delete and then unlock the same pointer value.

The full code snippet for this function can be seen in Fig. 1.

```
void ubifs_delete_orphan(struct ubifs_info *c, ino_t inum)
{
    [...]

    orphan_delete(c, orph);

    spin_unlock(&c->orphan_lock);
}


static void orphan_delete(struct ubifs_info *c,
    struct ubifs_orphan *orph)
{
    if (orph->del) {
        spin_unlock(&c->orphan_lock);
        dbg_gen("deleted twice ino %lu", orph->inum);
        return;
    }
    if (orph->cmt) {
        orph->del = 1;
        orph->dnext = c->orph_dnext;
        c->orph_dnext = orph;
        spin_unlock(&c->orphan_lock);
        dbg_gen("delete later ino %lu", orph->inum);
        return;
    }
}
```

Figure 1: The orphan_delete function containing a double unlock.

4.2 False Negatives

Several false negatives were found during evaluation of the proposed error checker.

5 Future Work

6 Conclusion

References

[1] Iago Abal, Claus Brabrand, and Andrzej Wasowski. Effective bug finding in c programs with shape and effect abstractions. In *VMCAI*, 2017.

[2] IEEE and The Open Group. pthread_spin_unlock - unlock a spin lock object, 2017.

7 Appendix

7.1 Confirmed Double Unlock Bugs

File	Present in	Patched in
drivers/block/drbd/drbd_main.c	b0814361	8e9c5230
fs/ubifs/orphan.c	7542c6de	4dd75b33
drivers/gpu/drm/nouveau/nouveau_svm.c	5fbcf501	de4ee728
fs/btrfs/file.c	78e03651	f49aa1de
drivers/staging/wilc1000/wilc_wlan.c	ca641bae	fea69916
drivers/staging/kpc2000/kpc_dma/fileops.c	d4c596eb	c85aa326
fs/nfs/client.c	a46126cc	c260121a
fs/btrfs/file.c	2b90883c	8fca9550
drivers/media/dvb-core/dvbdev.c	ded71626	122d0e8d
mm/memory_hotplug.c	6376360e	e3df4c6e
sound/soc/codecs/pcm512x.c	fd270fca	28b698b7
drivers/target/target_core_user.c	807cf197	f0e89aae
drivers/rpmsg/qcom_smd.c	fb416f69	c3388a07
drivers/scsi/aacraid/commsup.c	09624645	d844752e
drivers/staging/rtl8188eu/os_dep/usb_intf.c	612e1c94	23bf4042
block/blk-cgroup.c	e0223003	bbb427e3

7.2 Evaluation Results

File	Status
drivers/block/drbd/drbd_main.c	Parser error
fs/ubifs/orphan.c	Detected
drivers/gpu/drm/nouveau/nouveau_svm.c	Parser error
fs/btrfs/file.c	Parser error
drivers/staging/wilc1000/wilc_wlan.c	Parser error
drivers/staging/kpc2000/kpc_dma/fileops.c	Parser error
fs/nfs/client.c	Parser error
fs/btrfs/file.c	Parser error
drivers/media/dvb-core/dvbdev.c	Parser error
mm/memory_hotplug.c	Parser error
sound/soc/codecs/pcm512x.c	Parser error
drivers/target/target_core_user.c	Parser error
drivers/rpmsg/qcom_smd.c	Parser error
drivers/scsi/aacraid/commsup.c	Parser error
drivers/staging/rtl8188eu/os_dep/usb_intf.c	Parser error
block/blk-cgroup.c	Parser error