DiffKemp: Automatically Analysing Differences in Kernel Parameters

Viktor Malík
Red Hat
January 25, 2019
What is the goal?

• The goal is to check if some part of the kernel behaves the same between 2 different kernel versions

• Generally, we are comparing behavior of functions
  • For parameters (global variables), we check all functions using the variable

• This is difficult to test since we would need tests that cover all possible paths through the function
Why Analysing Differences?

Main reasons: **stability** and **compatibility**

- It is required that the same setting has the same effect in different versions of kernel.
  - sysctl options (built-in parameters)
  - module parameters
- Red Hat guarantees stability of Kernel Application Binary Interface (KABI)
Why can’t we just use diff?

Are these functions same?

RHEL 7.5 (linux-3.10.0-862)

```c
int bio_add_page(struct bio *bio, struct page *page,
                 unsigned len, unsigned offset)
{
    struct request_queue *q = get_queue(bio->bi_bdev);
    unsigned int max_sectors;

    max_sectors = max_size_offset(q, bio->bi_sector);
    if ((max_sectors < (len >> 9)) && !bio->bi_size)
        max_sectors = len >> 9;
    return __bio_add_page(q, bio, page, len,
                          offset, max_sectors);
}
```

RHEL 7.6 (linux-3.10.0-957)

```c
int bio_add_page(struct bio *bio, struct page *page,
                 unsigned len, unsigned offset)
{
    struct request_queue *q = get_queue(bio->bi_bdev);
    unsigned int max_sectors;

    max_sectors = max_size_offset(q, bio->bi_sector);
    if ((max_sectors < (len >> 9)) && !bio->bi_size)
        max_sectors = len >> 9;
    return __bio_add_page(q, bio, page, len,
                          offset, max_sectors);
}
```

According to diff: YES
Why can’t we just use diff?

Are these functions same?

RHEL 7.5 (linux-3.10.0-862)

```c
int bio_add_page(struct bio *bio, struct page *page,
                 unsigned len, unsigned offset)
{
    struct request_queue *q = get_queue(bio->bi_bdev);
    unsigned int max_sectors;
    max_sectors = max_size_offset(q, bio->bi_sector);

    static inline unsigned max_size_offset(
        struct request_queue *q, sector_t offset)
    {
        if (!q->limits.chunk_sectors)
            return q->limits.max_sectors;

        return q->limits.chunk_sectors -
            (offset & (q->limits.chunk_sectors - 1));
    }

    return __bio_add_page(q, bio, page, len,
                           offset, max_sectors);
}
```

RHEL 7.6 (linux-3.10.0-957)

```c
int bio_add_page(struct bio *bio, struct page *page,
                 unsigned len, unsigned offset)
{
    struct request_queue *q = get_queue(bio->bi_bdev);
    unsigned int max_sectors;
    max_sectors = max_size_offset(q, bio->bi_sector);

    static inline unsigned max_size_offset(
        struct request_queue *q, sector_t offset)
    {
        if (!q->limits.chunk_sectors)
            return q->limits.max_sectors;

        return min_t(unsigned int, q->limits.chunk_sectors -
                      (offset & (q->limits.chunk_sectors - 1)),
                      q->limits.max_sectors);
    }

    return __bio_add_page(q, bio, page, len,
                           offset, max_sectors);
}
```

In reality: NO
Why can’t we just use diff?

Are these functions same?

RHEL 7.5 (linux-3.10.0-862)

```c
int bio_add_page(struct bio *bio, struct page *page, unsigned len, unsigned offset)
{
    struct request_queue *q = get_queue(bio->bi_bdev);
    unsigned int max_sectors;
    max_sectors = max_size_offset(q, bio->bi_sector);

    static inline unsigned max_size_offset(
        struct request_queue *q, sector_t offset)
    {
        if (!q->limits.chunk_sectors)
            return q->limits.max_sectors;
        return q->limits.chunk_sectors -
            (offset & (q->limits.chunk_sectors - 1));
    }

    return q->limits.chunk_sectors -
        (offset & (q->limits.chunk_sectors - 1));
}
```

RHEL 7.6 (linux-3.10.0-957)

```c
int bio_add_page(struct bio *bio, struct page *page, unsigned len, unsigned offset)
{
    struct request_queue *q = get_queue(bio->bi_bdev);
    unsigned int max_sectors;
    max_sectors = max_size_offset(q, bio->bi_sector);

    static inline unsigned max_size_offset(
        struct request_queue *q, sector_t offset)
    {
        if (!q->limits.chunk_sectors)
            return q->limits.max_sectors;

        return min_t(unsigned int, q->limits.chunk_sectors -
            (offset & (q->limits.chunk_sectors - 1)),
            q->limits.max_sectors);
    }

    return q->limits.chunk_sectors -
        (offset & (q->limits.chunk_sectors - 1));
}
```

The problem requires checking called functions.
Why can’t we just use diff?
Are these functions equal?

```c
static __init int tcpprobe_init(void)
{
    init_waitqueue_head(&tcp_probe.wait);
    spin_lock_init(&tcp_probe.lock);
    if (bufsize == 0)
        return -EINVAL;
    bufsize = roundup_pow_of_two(bufsize);
    tcp_probe.log = kcalloc(bufsize, sizeof(struct tcp_log), GFP_KERNEL);
    if (!tcp_probe.log)
        return -ENOMEM
    return 0;
}
```

```
static __init int tcpprobe_init(void)
{
    BUILD_BUG_ON(__same_type(tcp_rcv_established, jtcp_rcv_established) == 0);
    init_waitqueue_head(&tcp_probe.wait);
    spin_lock_init(&tcp_probe.lock);
    if (bufsize == 0)
        return -EINVAL;
    bufsize = roundup_pow_of_two(bufsize);
    tcp_probe.log = kcalloc(bufsize, sizeof(struct tcp_log), GFP_KERNEL);
    if (!tcp_probe.log)
        return -ENOMEM
    return 0;
}
```

No
Why can’t we just use diff?

Are these functions equal?

```c
static __init int tcpprobe_init(void)
{
    init_waitqueue_head(&tcp_probe.wait);
    spin_lock_init(&tcp_probe.lock);

    if (bufsize == 0)
        return -EINVAL;

    bufsize = roundup_pow_of_two(bufsize);
    tcp_probe.log = kcalloc(bufsize, sizeof(struct tcp_log), GFP_KERNEL);
    if (!tcp_probe.log)
        return -ENOMEM
    return 0;
}
```

With respect to `bufsize`: **YES**
Why can’t we just use diff?

Are these functions equal?

```c
static __init int tcpprobe_init(void)
{
    init_waitqueue_head(&tcp_probe.wait);
    spin_lock_init(&tcp_probe.lock);

    if (bufsize == 0)
        return -EINVAL;

    bufsize = roundup_pow_of_two(bufsize);
    tcp_probe.log = kcalloc(bufsize, sizeof(struct tcp_log), GFP_KERNEL);
    if (!tcp_probe.log)
        return -ENOMEM
    return 0;
}
```

```c
static __init int tcpprobe_init(void)
{
    BUILD_BUG_ON(__same_type(tcp_rcv_established, jtcp_rcv_established) == 0);
    init_waitqueue_head(&tcp_probe.wait);
    spin_lock_init(&tcp_probe.lock);

    if (bufsize == 0)
        return -EINVAL;

    bufsize = roundup_pow_of_two(bufsize);
    tcp_probe.log = kcalloc(bufsize, sizeof(struct tcp_log), GFP_KERNEL);
    if (!tcp_probe.log)
        return -ENOMEM
    return 0;
}
```

The problem requires some understanding of the code.
We can go even further...

Are these functions equal?

dietlibc

```c
char *strpbrk(const char *s,
const char *accept) {

    register unsigned int i;
    for (; (*s); s++)
        for (i = 0; accept[i]; i++)
            if (*s == accept[i])
                return (char *)s;
    return 0;
}
```

openbsd

```c
char *strpbrk(const char *s1,
const char *s2) {

    const char *scanp;
    int c, sc;

    while (((c = *s1++) != 0)) {
        for (scanp = s2; ((sc = *scanp++) != 0);)
            if (sc == c)
                return ((char *)(s1 - 1));
    }
    return (NULL);
}
```
We can go even further…

Are these functions equal?

Actually YES

For every possible arguments, the effect of both functions will be the same.
We can go even further…
Are these functions equal?

The problem requires deep understanding of the code.
How do we do it?

- Using `diff`
How do we do it?

- Using `diff` Not sufficient
How do we do it?

- **Using** `diff` Not sufficient
- Analysing C code
  - It's simpler than C
  - But it contains much more information than assembly

Solution: use a compiler's internal representation and run the analysis over it.
How do we do it?

- **Using diff** Not sufficient
- **Analysing C code** It’s too complex (and we would need to write a parser)

Solution: use a compiler’s internal representation and run the analysis over it

- It’s simpler than C
- But it contains much more information than assembly

We use Clang/LL VM
How do we do it?

- **Using `diff`**  Not sufficient
- **Analysing C code**  It’s too complex (and we would need to write a parser)
- **Analysing assembly**
How do we do it?

• Using **diff** Not sufficient
• Analysing C code It’s too complex (and we would need to write a parser)
• Analysing assembly It’s too unstructured (too much information is lost)

Solution: use a compiler’s internal representation and run the analysis over it

• It’s simpler than C
• But it contains much more information than assembly

We use Clang/LLVM
How do we do it?

• Using `diff` Not sufficient
• Analysing C code It’s too complex (and we would need to write a parser)
• Analysing assembly It’s too unstructured (too much information is lost)

Solution: use a compiler’s internal representation and run the analysis over it
• It’s simpler than C
• But it contains much more information than assembly
How do we do it?

- Using `diff` Not sufficient
- Analysing C code It’s too complex (and we would need to write a parser)
- Analysing assembly It’s too unstructured (too much information is lost)

Solution: use a compiler’s internal representation and run the analysis over it
- It’s simpler than C
- But it contains much more information than assembly

We use **Clang/LLVM**
Benefits of using LLVM

• Well-structured and human-readable internal representation (LLVM IR)

```c
unsigned abs(int x) {
    if (x < 0)
        return -x;
    else
        return x;
}
```

• Good infrastructure containing many useful **built-in analyses and code transformations**.
• Has a nice API, can be used as a library.
• Many static analysers are built upon LLVM.
DiffKemp architecture

Kernel param/
KABI function
DiffKemp architecture

Kernel param/KABI function → Source finder

C source
DiffKemp architecture

Kernel param/KABI function → Source finder → C source → Compiler → LLVM IR
DiffKemp architecture

Kernel param/KABI function → Source finder → C source → Compiler → Code slicing and simplifying

LLVM IR
DiffKemp architecture

Kernel param/KABI function → Source finder → C source

Compiler → LLVM IR → Syntax diff

LLVM IR → Syntax diff

Code slicing and simplifying

equal

not equal

EQUAL
DiffKemp architecture

Kernel param/KABI function → Source finder → Compiler → Code slicing and simplifying → LLVM IR

C source

LLVM IR

Syntax diff → Semantic diff

equal → EQUAL

not equal

not equal

+ diff

NOT EQUAL
Source finding and compilation

Kernel param/ KABI function → Source finder → LLVM IR → Syntax diff → equal

Compiler → C source → LLVM IR → not equal

Code slicing and simplifying

Semantic diff → not equal

LLVM IR

EQUAL

NOT EQUAL
Source finding and compilation

- **Source finder** - using cscope
- **Source compilation**
  - `clang` is not supported for kernel building
  - We retrieve the command run by `Kbuild` and then run it with `clang` instead of `gcc` to get the LLVM IR
  - No optimisations are on by default
  - Some hacks are needed to overcome `clang` limitations (e.g. no support for `asm goto`)
  - The produced code is for analysis only (cannot be compiled into an executable)
Code slicing and simplifying

Kernel param/KABI function → Source finder → Compiler → LLVM IR → Syntax diff → Semantic diff

C source → LLVM IR

Syntax diff → equal or not equal

Semantic diff → equal or not equal

Equal → EQUAL

Not equal → NOT EQUAL
Code slicing and simplifying

- **Slicing** – removing all code irrelevant for the analysis

    ```c
    static __init int tcpprobe_init(void)
    {
        init_waitqueue_head(&tcp_probe.wait);
        spin_lock_init(&tcp_probe.lock);

        if (bufsize == 0)
            return -EINVAL;

        bufsize = roundup_pow_of_two(bufsize);
        tcp_probe.log = kcalloc(bufsize, sizeof(struct tcp_log), GFP_KERNEL);
        if (!tcp_probe.log)
            return -ENOMEM

        return 0;
    }
    ```

- **Code simplification** – removing content of printing functions, dead code elimination,...

    ```c
    static __init int tcpprobe_init(void)
    {
        if (bufsize == 0)
            return -EINVAL;

        bufsize = roundup_pow_of_two(bufsize);
        tcp_probe.log = kcalloc(           
            bufsize, sizeof(struct tcp_log), GFP_KERNEL);
        if (!tcp_probe.log)
            return -ENOMEM

        return 0;
    }
    ```
Syntax diff

Kernel param/KABI function → Source finder → C source → Compiler → LLVM IR

Code slicing and simplifying

LLVM IR → Syntax diff → not equal → Semantic diff → not equal

equal

EQUAL

NOT EQUAL

+ [diff]
Syntax diff

• Based on LLVM’s FunctionComparator
• Goes instruction-by-instruction and compares them for equality
• Handles some changes that do not affect semantics
  • Variable renaming
  • Changes in structure layout
  • ...
• In case of non-equality, returns a list of functions that are not syntactically equal and that need further analysis.
Semantic diff

Kernel param/KABI function → Source finder → Compiler → LLVM IR → Code slicing and simplifying

C source → LLVM IR → Syntax diff

LLVM IR → Syntax diff

Semantic diff

equal

not equal

equal

not equal

EQUAL

NOT EQUAL
Semantic diff (here comes the science)

• Based on an academic tool llreve
• Compared programs are translated into a logical formula – the formula expresses the effect of the program
• We use a tool for solving logical formulas (a so-called SMT solver) and ask it:

  Is there an input $i$ such that executing the first program with $i$ yields a different result than executing the second program with $i$?

• If yes → programs are not equal
• If no → programs are equal
Demo
Do you want to try DiffKemp?

- We have a Docker image at Docker Hub with everything prepared: viktormalik/diffkemp
- GitHub repo: https://github.com/viktormalik/diffkemp
- Feedback, issue reports, PRs welcome!

Thank you for the attention!