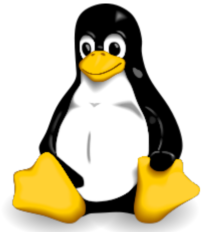
Using Dynamic Analysis to Hunt Down Problems in Kernel Modules

Eugene A. Shatokhin

Institute for System Programming of Russian Academy of Sciences (ISPRAS)

http://www.ispras.ru/en/



Loadable Kernel Modules (LKMs)

Kernel Modules in Linux:

- Device drivers
- File systems
- Networking stack and firewalls
- ... and much more (virtualization, RPC, ...)

Kernel 3.0-rc1 – 3.1-rc5: more than **1200** errors fixed.
Among these are:

- more than **100** concurrency-related errors (race conditions, deadlocks)
- more than **90** memory leaks
- more than **60** problems in "error path" (incorrect handling of rarely occuring situations)

Digression: Detecting Errors in Kernel Modules for Microsoft Windows

Certification of kernel modules:

- Driver Verifier
- Device Fundamental Tests
- Class-specific Tests

Development of kernel modules:

- all of the above
- Windows - checked build
- PreFAST for Drivers (PFD)
- Static Driver Verifier (SDV)

Details: Microsoft Windows Logo Program
http://msdn.microsoft.com/library/windows/hardware/gg463010

Digression: Driver Verifier for Microsoft Windows

Recommended for development and testing, **required** for certification.

The first version - in Windows 2000; continuously enhanced since then.

Runtime analysis of the kernel modules chosen by the user:

- verification of operations with kernel objects and handles, etc.
- simulation of low memory conditions and of other "uncommon" situations
- detection of memory leaks, double free, "use-after-free" errors, etc.
- … and much more

Main usage areas:

- driver development: testing, debugging, ...
- certification of kernel-mode software: checking basic requirements
- technical support: analysis of failures on the users' systems

N.B. Driver Verifier itself does not make requests to the target driver.

Kernel Modules: Analysis Tools

- *Static analysis:* source or binary code is analyzed without execution.
- *Dynamic analysis:* runtime, post factum on a trace, etc. The code under analysis is executed.

<u>Automated dynamic analysis of kernel modules in Linux:</u>

- Systemtap, LTTng, Ftrace
- Kmemcheck, Kmemleak, Fault Injection framework, ...
- Mmiotrace (for Nouveau graphics drivers, etc.)
- User Mode Linux + (Valgrind, GDB, …)
- "API Swapping" ("Imposter") facilities from Novell YES Tools
- KEDR framework (ISPRAS)
- ...

Novell YES Tools: "API Swapping"

Analysis of kernel modules chosen by the user:

- detection of memory leaks of several kinds
- detection of "write-past-end" and "write-before-begin" errors
- simulation of memory allocation failures (kmalloc() only, hard-coded scenarios)

Implementation:

Interception (replacement) of function calls: about 80 functions.
Instrumentation: changing names of imported functions in the object file.
Supported architectures: x86 (32- and 64-bit), IA64, PPC (32- and 64-bit)

Used only as a part of the certification test suites, not a standalone product.
Last updated in 2007 (?).

Details:
Novell YES Certified Program: http://developer.novell.com/devnet/yes/
Test Tools: http://www.novell.com/developer/ndk/storage_test_tools.html

KEDR Framework

KEDR: KErnel-mode Drivers in Runtime

Developed since April 2010, version 0.3 — June 17, 2011.
License: GPL v.2

Features:

- memory leak detection
- fault simulation using customizable scenarios ("what to make fail when")
- call monitoring (call tracing)
- interface for creating custom analysis tools

Implementation:

Interception (replacement) of function calls: about 75 functions.

Supported architectures: x86 (32- and 64-bit).

Relies on several in-kernel facilities: x86 instruction decoder (from KProbes), notification system, ring buffer implementation, …

Details:
KEDR project site: http://code.google.com/p/kedr/

KEDR Framework: Call Replacement

In the memory image of the module:

e8 bc 33 f1 c7
call __kmalloc

→

e8 2c 26 02 00
call repl___kmalloc

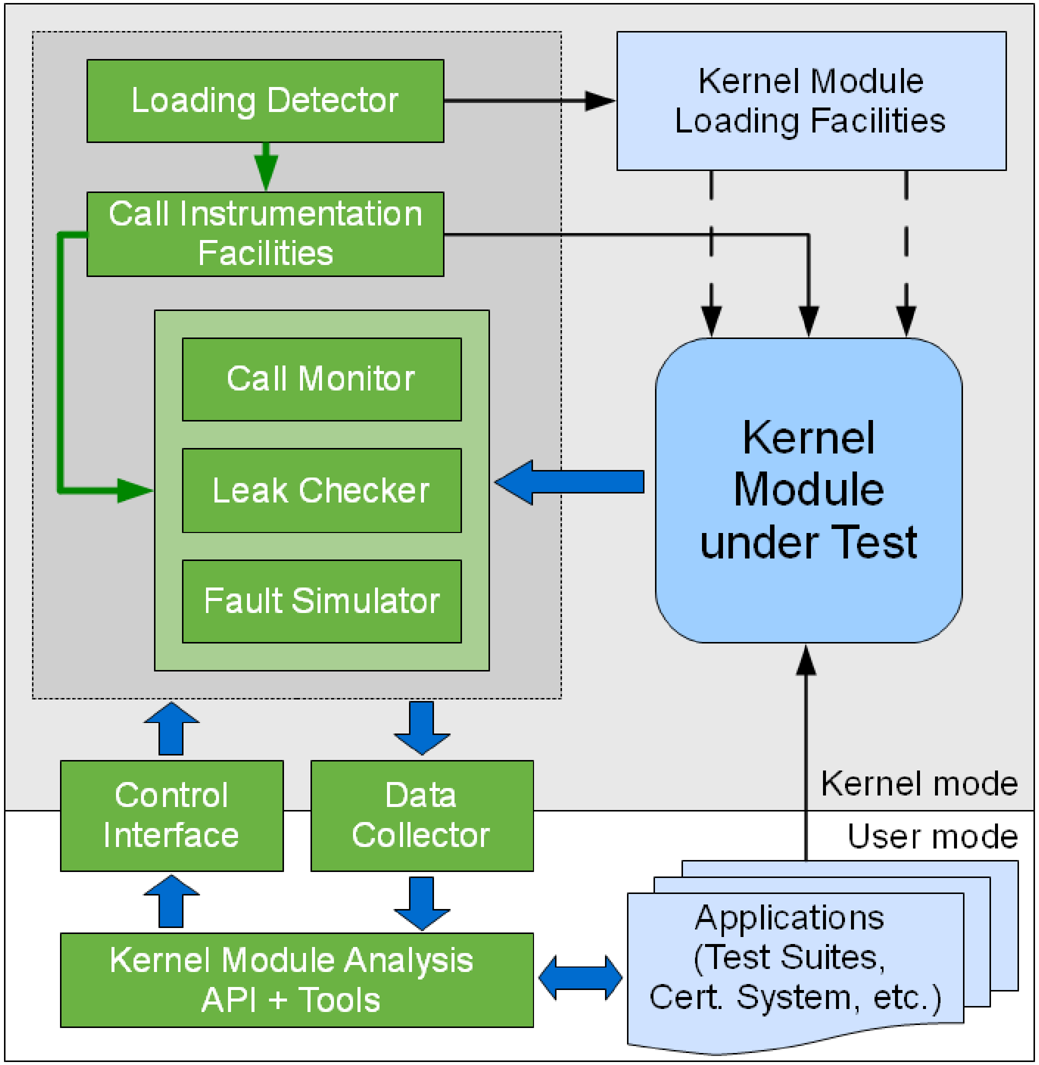
"Target" function:

```
void *__kmalloc(size_t size, gfp_t flags);
```

"Replacement" function:

```
void *repl___kmalloc(size_t size, gfp_t flags)
{
        void *result;
        if (do_fault_simulation("__kmalloc", size, flags))
                result = NULL;
        else
                result = __kmalloc(size, flags);
        return result;
}
```

KEDR Tool = KEDR Core + Plugin(s)



KEDR Framework: Memory Leak Detection (LeakCheck)

Function calls being processed:

- SLAB allocs / frees: __kmalloc*(), kfree(), kmem_cache_alloc*(), ...
- Page allocs / frees: alloc_pages*(), free_pages(), ...
- vmalloc / vfree family
- Other exported functions calling the ones above: kstrdup(), posix_acl_alloc(), ...

Usage with the test suites and benchmarks:

- Autotest
- Linux Test Project
- Phoronix Suite
- ...

Other information:
Integration with Autotest: http://code.google.com/p/kedr/wiki/HowTo_Autotest_Basics
Comparison with Kmemleak: http://code.google.com/p/kedr/wiki/KEDR_And_Kmemleak

KEDR Framework: Analysis of Real-World Kernel Modules

VirtualBox Guest Additions 4.0.2,
from the report generated by LeakCheck:

```
Block at 0xf659a000, size: 4096;
stack trace of the allocation:
[<fe2ab904>] sf_follow_link+0x34/0xa0 [vboxsf]
[<c0303caf>] link_path_walk+0x79f/0x910
[<c0303f19>] path_walk+0x49/0xb0
[<c0304089>] do_path_lookup+0x59/0x90
[<c03042bd>] user_path_at+0x3d/0x80
[<c02f8825>] sys_chdir+0x25/0x90
[<c0203190>] sysenter_do_call+0x12/0x22
[<ffffe430>] 0xffffe430
[<ffffffff>] 0xffffffff
+8 more allocation(s) with the same call stack.
```

KEDR Framework: Analysis of Real-World Kernel Modules

VirtualBox Guest Additions 4.0.2, file *lnkops.c*:

```
static void *
sf_follow_link(struct dentry *dentry, struct nameidata *nd)
{
<...>
  int error = -ENOMEM;
  unsigned long page = get_zeroed_page(GFP_KERNEL);
  if (page) {
    error = 0;
    rc = vboxReadLink(&client_handle,
         &sf_g->map, sf_i->path, PATH_MAX, (char *)page);
    if (RT_FAILURE(rc)) {
        LogFunc(("vboxReadLink failed <...>"));
        error = -EPROTO;
    }
  }
  nd_set_link(nd, error ? ERR_PTR(error) : (char *)page);
  return NULL;
}
```

KEDR Framework: Fault Simulation

Simulation of failures and other "uncommon" conditions

Affected operations: memory allocation, copy_*_user(), capable(), ...

Scenarios (key features):

- can be changed in runtime (via writing to files in debugfs)
- control expressions:

  "!in_init && size > 224 && flags != GFP_ATOMIC"

  "cap == CAP_SYS_ADMIN || cap == CAP_SYS_RAWIO"

  ...
- support for random failures (*rnd100* and *rnd10000* parameters)
- restriction by call site address (*caller_address* parameter)
- restriction by process ID

Other information: Comparison with Fault Injection framework:
http://code.google.com/p/kedr/wiki/KEDR_And_Fault_Injection

KEDR Framework: Results

Errors detected by KEDR tools:

- EXT4 FS - use-after-free and invalid free due to the problems in error handling
- FAT FS - kernel oops in low memory conditions
- Ath5k (wireless networking) - 3 memory leaks
  N.B. 2 of these were found by KEDR working with Autotest on Chromium OS
- VirtualBox Guest Additions - 3 memory leaks

All these errors are now confirmed and fixed by the maintainers.

Other projects KEDR was applied to: Open-MX, KNEM (INRIA)

Details:
http://code.google.com/p/kedr/wiki/Problems_Found

KEDR Framework: Advantages and Limitations

Advantages:

- Source code of the analyzed module is not needed; KEDR can be used to analyze closed-source modules too
- Extensibility
- Work "out-of-the-box" on many modern Linux-based systems
- Low requirements on system resources

Limitations and drawbacks:

- Analysis at the level of binary code => macros and inlines are "invisible" (+ kernel ABI is even more unstable than API)
- Analysis of function calls only; only the calls directly made by the given module are processed
- Analysis of only one module at a time
- For the present, only x86 and x86-64

KEDR Framework: Future Directions

High priority:

- Support for tracking memory read and write operations; working with offline data race detectors like ThreadSanitizer (Google)
- Enhanced integration with Autotest (more groups of tests to cover)

Medium priority:

- Making memory leak detector more accurate with the help of tracking callback operations
- Support for fault simulation for disk I/O
- Support for analysis of several modules at once
- Opportunities to work with OpenQA
- Revisiting KEDR to improve its portability (eventually - support for ARM, etc.)

KEDR Framework: Our Related Projects

- **KEDR-COI** — tracking the calls to callback operations (file operations, inode operations, fault handlers, ...).
http://code.google.com/p/kedr-callback-operations-interception/
Status: beta

- **Kernel Strider** — tracking memory read/write operations in addition to function calls + preparing data for offline data race detectors (ThreadSanitizer, ...).

  The project is supported by a Google Research Award (2011): "Instrumentation and Data Collection Framework for Dynamic Data Race Detection in Linux Kernel Modules"

http://code.google.com/p/kernel-strider/
Status: pre-alpha / prototyping

Thank you!