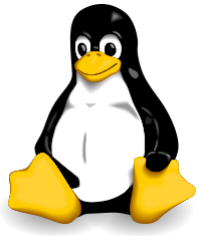
Динамический анализ и верификация модулей ядра ОС Linux
Проект KEDR

Евгений Шатохин
ИСП РАН

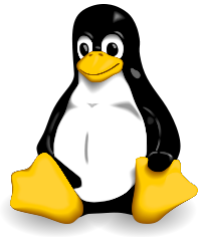


Модули ядра Linux

Wikipedia:
A loadable kernel module (or LKM) is an object file that contains code to extend the running kernel <...> of an operating system.
LKMs are typically used to add support for new hardware and/or filesystems, or for adding system calls.

Модули ядра Linux:

- Драйверы устройств
- Файловые системы
- Реализация сетевых протоколов и firewalls
- Многое другое (KVM, RPC, ...)



Некоторые ошибки в ядре Linux (версии 2.6.35 - 2.6.35.5)

Около 2000 исправлений ошибок, из них:

- Более 100 concurrency issues (race conditions, deadlocks)
- Более 130 ошибок, связанных с некорректной обработкой ошибок
- Десятки ошибок других видов, не специфичных для конкретных классов модулей (переполнение буфера, утечки ресурсов,и пр.)

Подробнее: Linux kernel change logs,
http://www.kernel.org/pub/linux/kernel/v2.6/

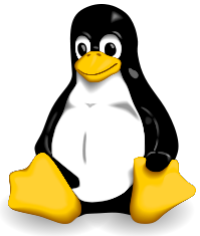
Взгляд в сторону: опыт Microsoft

Driver Verifier (1999-н.в.)

Динамический анализ поведения модулей ядра, выбранных пользователем:

- выявление утечек памяти, use-after-free и переполнений буферов;
- low resource simulation;
- проверка корректности операций с объектами ядра, DMA, locking, с памятью из пользовательского пространства и т.д.;
- сбор статистики о действиях драйвера;
- многое другое.

Подробнее:
http://www.microsoft.com/whdc/devtools/tools/DrvVerifier.mspx



Средства динамического анализа модулей ядра Linux

"Общесистемный" автоматизированный анализ:

- Systemtap, LTTng (базовые блоки: Kernel Probes, Tracing)
- Kmemcheck, Kmemleak, Fault Injection framework, ...
- User Mode Linux + (GDB, Valgrind, …)

"Ручной" анализ с элементами автоматизации:

- GDB, KGDB, Crash, Kexec, …

Автоматизированный анализ модулей, выбранных пользователем:

(в т.ч. с fault/low resource simulation - a la Driver Verifier для Windows):

- "API Swapping"-средства из Novell YES Tools

Novell YES Tools ("API Swapping")

Анализ модуля ядра, выбранного пользователем:

- Проверка выделения/освобождения памяти + special pool.
- Fault simulation по фиксированным (hard-coded) сценариям.

Реализация:

- Средства для подмены вызовов функций в файле модуля (user space).

Особенности:

- Не является отдельным программным продуктом.
- (?) Не развивается с 2006-2007.

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

KEDR Framework - динамический анализ выбранного модуля ядра

KEDR: **KE**rnel-mode **D**rivers in **R**untime

Разрабатывается с апреля 2010 г.
Версия 0.1 - ноябрь 2010 г., 0.2 (готовится к выпуску) - март (?) 2011 г.
Лицензия: GPL v.2

Возможности:

- Call monitoring с сохранением трассы вызовов
- Выявление утечек памяти
- Fault simulation по настраиваемым сценариям

Реализация: перехват (подмена) вызовов функций

v. 0.2 - около 70 функций: memory management, mutexes, spinlocks, wait queues, user space access, ...

Подробнее:
KEDR на forge.ispras.ru: http://forge.ispras.ru/projects/kedr
KEDR на BerliOS Developer: http://kedr.berlios.de/

KEDR Framework - Call Replacement (пример)

Исходная функция:

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

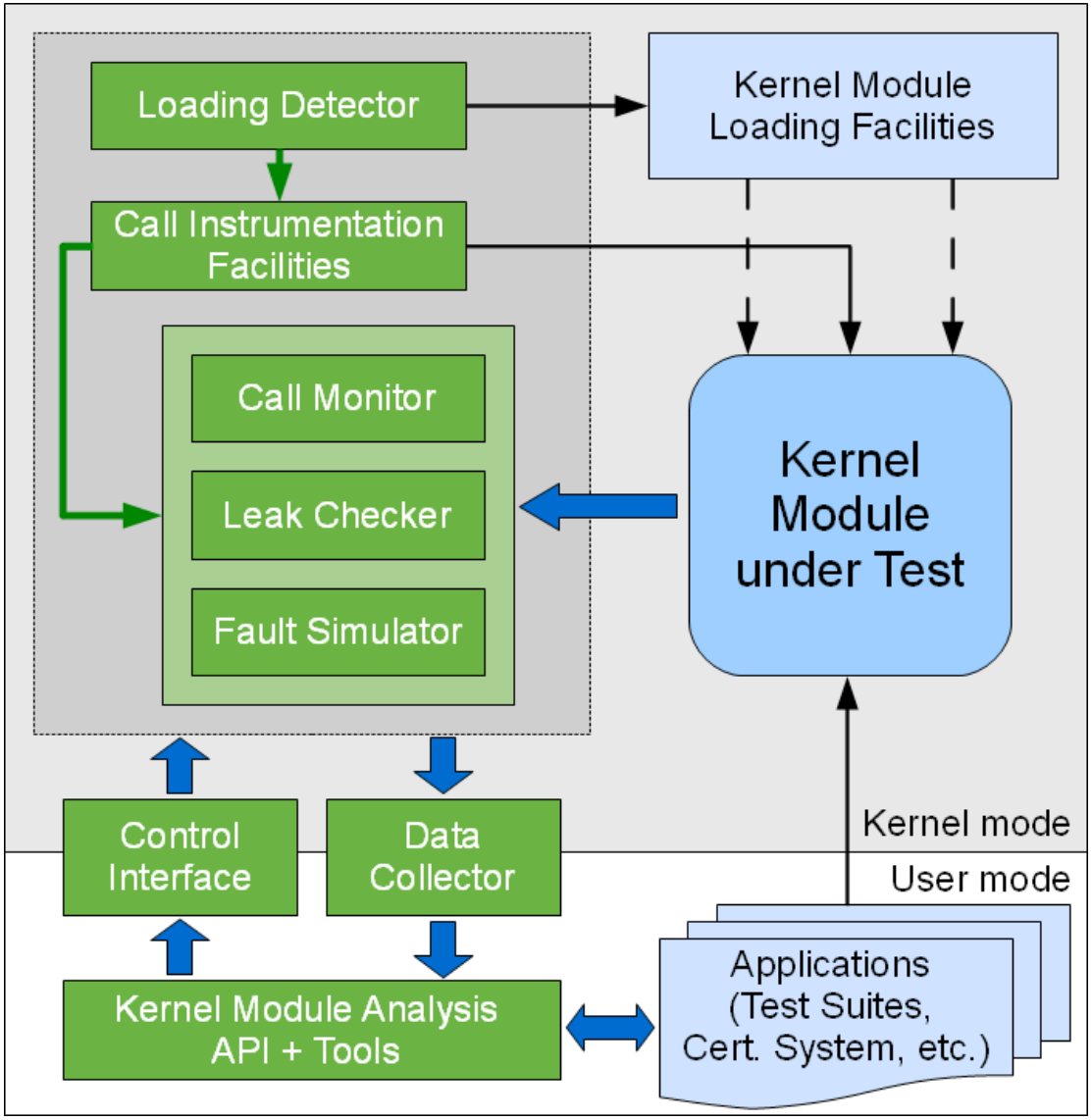
"Replacement"-функция:

```
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);
        }

        /* Вывод аргументов и результата в трассу */
        trace_called___kmalloc(size, flags, result);

        return result;
}
```

KEDR Tool = KEDR Core + Plugin(s)



KEDR Framework - Use Case

Типичный сценарий использования:

- Запуск KEDR (загрузка базовых компонентов и plugin'ов). Настройка KEDR: имя анализируемого модуля, сценарии fault simulation, ...
- Загрузка анализируемого модуля (target).
- Работа пользователя с target-модулем (обычные операции и/или запуск спец. тестов).
- Выгрузка target-модуля.
- Завершение работы KEDR (+ сохранение результатов).
- Анализ данных, собранных KEDR (трасса вызовов, отчёт об утечках памяти и т.д.)

KEDR Framework - примеры ошибок

Найдены ошибки в:

- VirtualBox Guest Additions (модуль vboxsf.ko, v.3.2.10, v.4.0.3) - утечки памяти при работе с общими каталогами (shared folders), 2 из 3 - критические.
- FAT FS (модуль fat.ko, ядро 2.6.34.7) - kernel oops при нехватке памяти: не везде проверяется успешность выделения памяти.
- EXT4 FS (модуль ext4.ko, ядро 2.6.37) - kernel oops при нехватке памяти: ошибка вида "use after free".

KEDR Framework - примеры ошибок

VirtualBox Guest Additions 4.0.2,
из отчёта KEDR о выявленных утечках памяти:

```
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 - примеры ошибок

VirtualBox Guest Additions 4.0.2, файл 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

Достоинства:

- Для анализа не нужен исходный код ни ядра, ни соотв. модулей.
- Возможности расширения: пользовательские сценарии fault simulation, другие виды анализа, ...
- Работа "out-of-the-box" на основных современных дистрибутивах Linux.
- Нетребовательность к системным ресурсам.

Ограничения и недостатки:

- Анализ на уровне бинарного кода => macros и inlines "невидимы".
- Анализ только вызовов функций, причём только тех вызовов, которые сделаны непосредственно данным модулем.
- Анализ одного модуля за раз.
- Поддержка только x86 и x86-64 на данный момент.

KEDR Framework - что дальше?

Направления дальнейшего развития:

- Special pool и выявление ошибок переполнения буфера.
- Механизм отслеживания операций чтения/записи в память + интеграция с инструментами для выявления data races.
- Поддержка анализа нескольких модулей ядра одновременно.
- Поддержка одновременного выполнения нескольких видов анализа, например, fault simulation и отслеживания утечек памяти.
- Обобщение механизма fault simulation.
- Средства для отслеживания callback-функций, таких, как file operations для устройств и т.п.
- ...

Спасибо за внимание!