

Дикий Александр Борисович

Студент

РГУ нефти и газа (НИУ) имени И.М. Губкина

Журавлев Глеб Дмитриевич

Студент

РГУ нефти и газа (НИУ) имени И.М. Губкина

**АНАЛИЗ СОСТОЯНИЙ ГОНКИ (RACE CONDITIONS) ПРИ
ПАРАЛЛЕЛЬНОМ ДОСТУПЕ НА ЧТЕНИЕ/ЗАПИСЬ ЧЕРЕЗ /PROC:
СТАТИЧЕСКИЙ АНАЛИЗ И ПРАКТИЧЕСКАЯ ДЕМОНСТРАЦИЯ
УЯЗВИМОСТЕЙ**

Аннотация: В работе рассматриваются проблемы синхронизации доступа к данным при параллельном доступе на чтение и запись через виртуальную файловую систему /proc в ядре Linux. Цель работы – оценить, способен ли статический анализатор обнаруживать race conditions в подсистеме /proc Linux на синтетических и реальных примерах, а также предложить комбинированную методику для повышения эффективности обнаружения. С помощью Smatch выполнен анализ обработчиков /proc в ядре дистрибутива ALT Linux, а также создан специальный уязвимый модуль с преднамеренным отсутствием синхронизации и многопоточное приложение-эксплойт. Для подтверждения актуальности проблемы проведён эксперимент с реальной уязвимостью CVE-2025-39866 (use-after-free – использование после освобождения в подсистеме writeback). Результаты показывают, что Smatch способен выявлять отдельные классы ошибок синхронизации (например, missing unlock), но не обнаруживает гонки при полном отсутствии блокировок и в сложных межпроцедурных сценариях. Предложенная двухэтапная методика может быть использована при аудите безопасности модулей ядра и в образовательных целях.

Abstract: This paper addresses data access synchronization issues during concurrent read/write operations through /proc pseudo filesystem in the Linux kernel. The aim is to evaluate whether the static analyzers can detect race conditions in the /proc subsystem using synthetic and real-world examples, and to propose a combined methodology to improve detection effectiveness. Smatch is used to analyze /proc handlers in the ALT Linux kernel, and a deliberately unsynchronized kernel module with a multithreaded exploit is created. To demonstrate real-world relevance, an experiment with CVE-2025-39866 (a use-after-free in the writeback subsystem) is conducted. The results show that Smatch can find certain classes of synchronization errors (e.g., missing unlocks) but fails to detect races due to complete absence of locking or complex cross-function scenarios. The proposed two-step methodology can be used for kernel module security auditing and educational purposes.

Ключевые слова: Linux, ядро, состояние гонки, race condition, /proc, синхронизация, статический анализ, Smatch, CVE-2025-39866, уязвимость, ALT Linux.

Keywords: Linux, kernel, race condition, /proc, synchronization, static analysis, Smatch, CVE-2025-39866, vulnerability, ALT Linux.

Виртуальная файловая система /proc в операционной системе Linux предоставляет удобный механизм для обмена данными между ядром и пользовательским пространством. Многие драйверы и модули ядра экспортируют свои интерфейсы через файлы /proc, позволяя прикладным программам читать состояние подсистем или изменять параметры ядра с помощью стандартных системных вызовов read() и write(). Однако, если разработчик модуля явно не защищает разделяемые данные, при конкурентном доступе нескольких потоков пользовательского пространства к одному и тому же проc-файлу возникает состояние гонки.

Проблема усугубляется тем, что типичные обработчики read и write в модулях ядра часто работают с глобальными буферами, счётчиками и указателями. Отсутствие синхронизации приводит к повреждению данных ядра,

чтению несогласованной информации или, в худшем случае, к панике системы и отказу в обслуживании.

Для экспорта файла в виртуальную файловую систему /proc модуль ядра выполняет несколько стандартных действий:

- вызывает `proc_create()` с указанием имени файла, прав доступа и структуры `struct file_operations`;
- в `file_operations` заполняет указатели на функции-обработчики: `.read` (чтение из файла), `.write` (запись), иногда `.open` и `.release`;
- в этих обработчиках реализует логику обмена данными между ядром и пользовательским пространством.

Если два потока пользовательского процесса одновременно выполняют системные вызовы `read()` и `write()` к одному файлу `/proc/myfile`, то в ядре эти вызовы могут исполняться параллельно на разных ядрах процессора или быть прерваны по таймеру на одном ядре. Поскольку обработчики зачастую не используют блокировки, возможны несколько сценариев.

Первый сценарий — гонка чтения и записи. Писатель начинает копирование данных в `shared_buffer`, но в середине операции его вытесняет читатель. Читатель видит частично обновлённый буфер, содержащий новые и старые данные. При этом значение `buffer_len` может уже указывать новую длину, что приведёт к выходу за границы при копировании в пользовательское пространство.

Второй сценарий — гонка двух записей. Два писателя одновременно изменяют буфер и длину, в результате итоговое состояние оказывается несогласованным: длина от одного писателя, содержимое от другого.

Третий сценарий — гонка с частичным обновлением. Многие простые операции, например, `buffer_len = len`, не являются атомарными на многих архитектурах (требует нескольких инструкций), поэтому читатель может прочитать промежуточное значение.

Описанные сценарии являются следствием ошибочных представлений разработчиков, например, что файлы `/proc` обрабатываются последовательно или что атомарных операций достаточно для любых типов данных.

В реальности любое обращение к разделяемым изменяемым данным без блокировок создаёт состояние гонки. Блокировки, например, `mutex_lock/mutex_unlock`, гарантируют, что критическая секция выполняется только одним потоком в единицу времени, но все равно не защищают от неправильного использования самих блокировок, что может спровоцировать появление новых гонок.

Для демонстрации опасности состояния гонки проведены несколько экспериментов. На первом этапе эксперимента была предпринята попытка выявления реальных состояний гонки в коде ядра дистрибутива ALT Linux (версия ядра 6.12.74-6.12-alt1) с использованием статического анализатора Smatch.

Выбор инструмента обусловлен его способностью выполнять глубокий анализ потоков данных и межпроцедурный анализ. Это позволяет Smatch отслеживать состояние переменных, блокировок и указателей не только внутри одной функции, но и при их передаче между разными частями кода. Такие возможности критически важны для обнаружения состояний гонки, где некорректная синхронизация может происходить при вызове одной функции из другой.

Однако у него есть существенные ограничения. Smatch успешно ищет несоответствия в использовании примитивов синхронизации, например, `missing unlock`, двойная блокировка, нарушение порядка, но не моделирует разделяемую память без блокировок полностью. Таким образом, гонки, возникающие при полном отсутствии защиты, не являются целевой областью Smatch. Это ограничение принципиально для всех статических анализаторов, не имеющих аннотаций о разделяемых данных таких как `__shared__` или `guarded_by`.

На рисунке 1 представлен запуск smatch для анализа `/proc` и смежных с ней директорий и файлов.

```
[root@host-15 linux-6.12.74]# make CHECK="smatch --no-fatal-errors" C=1 fs/proc/
fs/seq_file.c kernel/ -j$(nproc) 2>&1 | tee ~/proc_smatch.log
DESCEND objtool
CALL scripts/checksyscalls.sh
INSTALL libsubcmd_headers
CC fs/proc/task_mmu.o
CC fs/proc/inode.o
CC fs/proc/root.o
CHECK fs/proc/root.c
CHECK fs/proc/inode.c
CC fs/proc/base.o
```

Рисунок 1 – Запуск smatch

При анализе полученных логов (рисунок 2), не было обнаружено явного указания на наличие потенциальной race condition.

```
[root@host-15 linux-6.12.74]# make C=1 CHECK="smatch -p=kernel --two-passes" fs/
proc/ > ~/proc_full_analysis.log 2>&1
[root@host-15 linux-6.12.74]# grep -E "(warn:|error:|info:)" ~/proc_full_analysi
s.log > ~/smatch_only_proc.log
[root@host-15 linux-6.12.74]# cat ~/smatch_only_proc.log
fs/proc/task_mmu.c:604 do_procmmap_query() warn: check that 'karg.size' doesn't l
eak information
fs/proc/proc_sysctl.c:1244 get_links() warn: unused return: link = (null)()
fs/proc/vmcore.c:1570 vmcore_init() warn: missing error code? 'rc'
fs/proc/bootconfig.c:38 copy_xbc_key_value_list() warn: unused return: val = (nu
ll)()
fs/proc/bootconfig.c:38 copy_xbc_key_value_list() warn: unused return: val = (nu
ll)()
[root@host-15 linux-6.12.74]#
```

Рисунок 2 – Анализ полученных ошибок и предупреждений

Это не означает отсутствия уязвимостей типа race conditions в ядре, а объясняется особенностями современной реализации прос-файлов.

Начиная с ядер версии 5.6, инициализация структур file_operations для прос-файлов выполняется с помощью макросов, таких как DEFINE_PROC_OPS(), DEFINE_PROC_SHOW_ATTRIBUTE() и PROC_OPS_RW(). Эти макросы раскрываются в определение структуры на этапе препроцессора, ограничивая тем самым использование многих статических анализаторов, использующих поиск синтаксических и семантических шаблонов в коде, как например Coccinelle.

Помимо синтаксических трудностей, связанных с анализаторами на подобии Coccinelle, использование рассмотренного в данной работе Smatch, также не приводит к обнаружению состояний гонки в подсистеме /proc современных ядер. Это обусловлено несколькими причинами.

Во-первых, /proc является одним из наиболее отлаженных компонентов ядра, в котором классические блокировки на основе мьютексов и спин-

блокировок были в значительной степени заменены механизмами без блокировок (lock-free), в частности RCU (Read-Copy-Update) и внутренней синхронизацией интерфейса seq_file [15].

Во-вторых, как упоминалось выше, Smatch в основном ищет несоответствия в использовании блокировок, но не анализирует полностью несинхронизированный доступ, так как не имеет модели «разделяемые данные без защиты». Вследствие этого даже при корректной предобработке макросов статический анализ реального кода /proc не выдаёт значимых предупреждений о состояниях гонки.

Чтобы продемонстрировать способность Smatch выявлять ошибки синхронизации, был создан тестовый модуль. Он содержит общий буфер и счётчик длины, а обработчики read и write намеренно не используют синхронизацию. В него также был добавлен мьютекс, но намеренно допущена ошибка, а именно, в одном из путей обработки ошибок отсутствует вызов mutex_unlock(). Писатель копирует данные в shared_buffer, обновляет buffer_len, а читатель одновременно обращается к тем же переменным, что и вызывает гонку.

Для эксплуатации уязвимости было разработано многопоточное приложение пользовательского пространства на языке C с использованием pthread. Эксплойт создаёт два потока:

- поток-писатель непрерывно записывает в /proc/test_race строку фиксированного размера;
- поток-читатель непрерывно читает из того же файла.

Для увеличения вероятности пересечения потоков использовались циклы без синхронизации и вызовы sched_yield().

Smatch успешно обнаружил проблемный модуль и выдал предупреждение (рисунок 3).

```

[root@host-15 ~]# make -C /lib/modules/$(uname -r)/build M=$(pwd) C=2 CHECK="smatch -p=kernel --data-dir=/root/lin_kernel/linux-6.12.74 --two-passes" modules 2>
&1 | tee ~/smatch_after_fix.log
make: вход в каталог «/usr/src/linux-6.12.74-6.12-alt1»
CC [M] /root/race_module.o
/root/race_module.c:15:9: warning: no previous prototype for 'proc_read' [-Wmissing-prototypes]
  15 | ssize_t proc_read(struct file *file, char __user *buf, size_t count, lof
f_t *ppos)
      |
      | ~~~~~
/root/race_module.c:40:9: warning: no previous prototype for 'proc_write' [-Wmissing-prototypes]
  40 | ssize_t proc_write(struct file *file, const char __user *buf, size_t cou
nt, loff_t *ppos)
      |
      | ~~~~~
CHECK /root/race_module.c
/root/race_module.c:65 proc_write() warn: inconsistent returns 'global &data_mutex'.
  Locked on : 55
  Unlocked on: 49,65
/root/race_module.c:76 race_init() warn: proc file "test_race" is world writable
MODPOST /root/Module.symvers
CC [M] /root/race_module.mod.o

```

Рисунок 3 – Обнаружение Smatch ошибки синхронизации (неразблокированный мьютекс) в модифицированном модуле

Далее, был проведен тестовый запуск созданного эксплойта. Запуск эксплойта продемонстрирован на рисунке 4.

```

[root@host-15 ~]# ./exploit

[!] Corruption detected at byte 1024 (value=3d)
[!] Corruption detected at byte 1024 (value=3d)
[!] Corruption detected at byte 1024 (value=3d)
[!] Corruption detected at byte 1024 (value=3d)
[!] Corruption detected at byte 1247 (value=0)
[!] Corruption detected at byte 1247 (value=0)
[!] Corruption detected at byte 1247 (value=0)
[!] Corruption detected at byte 1247 (value=0)
[!] Corruption detected at byte 1247 (value=0)
[!] Corruption detected at byte 1247 (value=0)
[!] Corruption detected at byte 1247 (value=0)
[!] Corruption detected at byte 1247 (value=0)
[!] Corruption detected at byte 1247 (value=0)
[!] Corruption detected at byte 1247 (value=0)
[!] Corruption detected at byte 1024 (value=3d)
[!] Corruption detected at byte 1024 (value=3d)
[!] Corruption detected at byte 1024 (value=3d)
[!] Corruption detected at byte 1024 (value=3d)
[!] Corruption detected at byte 1024 (value=3d)

```

Рисунок 4 – Запуск эксплойта

Вызов эксплойта против тестового модуля спровоцировал механизм защиты CONFIG_HARDENED_USERCOPY, присутствующий в современных версиях ядра (рисунок 5).

```

[ 9039.960300] race_module loaded
[ 9144.809485] usercopy: Kernel memory exposure attempt detected from SLUB object 'kmalloc-1k' (offset 0, size 1082)!
[ 9144.809710] -----[ cut here ]-----
[ 9144.809731] kernel BUG at mm/usercopy.c:102!
[ 9144.809734] usercopy: Kernel memory exposure attempt detected from SLUB object 'kmalloc-1k' (offset 0, size 1082)!
[ 9144.809755] usercopy: Kernel memory exposure attempt detected from SLUB object 'kmalloc-1k' (offset 0, size 1082)!
[ 9144.809778] Oops: invalid opcode: 0000 [#1] PREEMPT SMP NOPTI
[ 9144.809805] CPU: 1 UID: 0 PID: 57905 Comm: exploit Tainted: G      W  OE
6.12.74-6.12-alt1 #1
[ 9144.809812] Tainted: [W]=WARN, [O]=OOT_MODULE, [E]=UNSIGNED_MODULE
[ 9144.809814] Hardware name: innotek GmbH VirtualBox/VirtualBox, BIOS VirtualBox 12/01/2006
[ 9144.809898] -----[ cut here ]-----
[ 9144.809906] kernel BUG at mm/usercopy.c:102!
[ 9144.809817] RIP: 0010:usercopy_abort+0x68/0x80
[ 9144.809936] Code: 6f b4 51 48 c7 c2 17 0c 72 b4 41 52 48 c7 c7 98 73 77 b4 48 0f 45 d6 48 c7 c6 62 78 6f b4 48 89 c1 49 0f 45 f3 e8 68 7a d3 ff <0f> 0b 49 c7 c1 b0 64 6f b4 4d 89 ca 4d 89 c8 eb a8 0f 1f 80 00 00
[ 9144.809939] RSP: 0018:ffffd0ef01c43d58 EFLAGS: 00010246
[ 9144.809943] RAX: 0000000000000066 RBX: ffff8efb4436d800 RCX: 0000000000000000
[ 9144.809945] RDX: 0000000000000000 RSI: 0000000000000000 RDI: 0000000000000000
[ 9144.809947] R00: 0000000000000000 R01: 0000000000000000 R02: 0000000000000000 R03: 0000000000000000
[ 9144.809949] R04: 0000000000000000 R05: 0000000000000000 R06: 0000000000000000 R07: 0000000000000000
[ 9144.809951] R08: 0000000000000000 R09: 0000000000000000 R10: 0000000000000000 R11: 0000000000000000
[ 9144.809953] R12: 0000000000000000 R13: 0000000000000000 R14: 0000000000000000 R15: 0000000000000000
[ 9144.809955] R16: 0000000000000000 R17: 0000000000000000 R18: 0000000000000000 R19: 0000000000000000
[ 9144.809957] R20: 0000000000000000 R21: 0000000000000000 R22: 0000000000000000 R23: 0000000000000000
[ 9144.809959] R24: 0000000000000000 R25: 0000000000000000 R26: 0000000000000000 R27: 0000000000000000
[ 9144.809961] R28: 0000000000000000 R29: 0000000000000000 R30: 0000000000000000 R31: 0000000000000000
[ 9144.809963] R32: 0000000000000000 R33: 0000000000000000 R34: 0000000000000000 R35: 0000000000000000
[ 9144.809965] R36: 0000000000000000 R37: 0000000000000000 R38: 0000000000000000 R39: 0000000000000000
[ 9144.809967] R40: 0000000000000000 R41: 0000000000000000 R42: 0000000000000000 R43: 0000000000000000
[ 9144.809969] R44: 0000000000000000 R45: 0000000000000000 R46: 0000000000000000 R47: 0000000000000000
[ 9144.809971] R48: 0000000000000000 R49: 0000000000000000 R50: 0000000000000000 R51: 0000000000000000
[ 9144.809973] R52: 0000000000000000 R53: 0000000000000000 R54: 0000000000000000 R55: 0000000000000000
[ 9144.809975] R56: 0000000000000000 R57: 0000000000000000 R58: 0000000000000000 R59: 0000000000000000
[ 9144.809977] R60: 0000000000000000 R61: 0000000000000000 R62: 0000000000000000 R63: 0000000000000000
[ 9144.809979] R64: 0000000000000000 R65: 0000000000000000 R66: 0000000000000000 R67: 0000000000000000
[ 9144.809981] R68: 0000000000000000 R69: 0000000000000000 R70: 0000000000000000 R71: 0000000000000000
[ 9144.809983] R72: 0000000000000000 R73: 0000000000000000 R74: 0000000000000000 R75: 0000000000000000
[ 9144.809985] R76: 0000000000000000 R77: 0000000000000000 R78: 0000000000000000 R79: 0000000000000000
[ 9144.809987] R80: 0000000000000000 R81: 0000000000000000 R82: 0000000000000000 R83: 0000000000000000
[ 9144.809989] R84: 0000000000000000 R85: 0000000000000000 R86: 0000000000000000 R87: 0000000000000000
[ 9144.809991] R88: 0000000000000000 R89: 0000000000000000 R90: 0000000000000000 R91: 0000000000000000
[ 9144.809993] R92: 0000000000000000 R93: 0000000000000000 R94: 0000000000000000 R95: 0000000000000000
[ 9144.809995] R96: 0000000000000000 R97: 0000000000000000 R98: 0000000000000000 R99: 0000000000000000
[ 9144.809997] R100: 0000000000000000 R101: 0000000000000000 R102: 0000000000000000 R103: 0000000000000000
[ 9144.809999] R104: 0000000000000000 R105: 0000000000000000 R106: 0000000000000000 R107: 0000000000000000
[ 9144.810001] R108: 0000000000000000 R109: 0000000000000000 R110: 0000000000000000 R111: 0000000000000000
[ 9144.810003] R112: 0000000000000000 R113: 0000000000000000 R114: 0000000000000000 R115: 0000000000000000
[ 9144.810005] R116: 0000000000000000 R117: 0000000000000000 R118: 0000000000000000 R119: 0000000000000000
[ 9144.810007] R120: 0000000000000000 R121: 0000000000000000 R122: 0000000000000000 R123: 0000000000000000
[ 9144.810009] R124: 0000000000000000 R125: 0000000000000000 R126: 0000000000000000 R127: 0000000000000000
[ 9144.810011] R128: 0000000000000000 R129: 0000000000000000 R130: 0000000000000000 R131: 0000000000000000
[ 9144.810013] R132: 0000000000000000 R133: 0000000000000000 R134: 0000000000000000 R135: 0000000000000000
[ 9144.810015] R136: 0000000000000000 R137: 0000000000000000 R138: 0000000000000000 R139: 0000000000000000
[ 9144.810017] R140: 0000000000000000 R141: 0000000000000000 R142: 0000000000000000 R143: 0000000000000000
[ 9144.810019] R144: 0000000000000000 R145: 0000000000000000 R146: 0000000000000000 R147: 0000000000000000
[ 9144.810021] R148: 0000000000000000 R149: 0000000000000000 R150: 0000000000000000 R151: 0000000000000000
[ 9144.810023] R152: 0000000000000000 R153: 0000000000000000 R154: 0000000000000000 R155: 0000000000000000
[ 9144.810025] R156: 0000000000000000 R157: 0000000000000000 R158: 0000000000000000 R159: 0000000000000000
[ 9144.810027] R160: 0000000000000000 R161: 0000000000000000 R162: 0000000000000000 R163: 0000000000000000
[ 9144.810029] R164: 0000000000000000 R165: 0000000000000000 R166: 0000000000000000 R167: 0000000000000000
[ 9144.810031] R168: 0000000000000000 R169: 0000000000000000 R170: 0000000000000000 R171: 0000000000000000
[ 9144.810033] R172: 0000000000000000 R173: 0000000000000000 R174: 0000000000000000 R175: 0000000000000000
[ 9144.810035] R176: 0000000000000000 R177: 0000000000000000 R178: 0000000000000000 R179: 0000000000000000
[ 9144.810037] R180: 0000000000000000 R181: 0000000000000000 R182: 0000000000000000 R183: 0000000000000000
[ 9144.810039] R184: 0000000000000000 R185: 0000000000000000 R186: 0000000000000000 R187: 0000000000000000
[ 9144.810041] R188: 0000000000000000 R189: 0000000000000000 R190: 0000000000000000 R191: 0000000000000000
[ 9144.810043] R192: 0000000000000000 R193: 0000000000000000 R194: 0000000000000000 R195: 0000000000000000
[ 9144.810045] R196: 0000000000000000 R197: 0000000000000000 R198: 0000000000000000 R199: 0000000000000000
[ 9144.810047] R200: 0000000000000000 R201: 0000000000000000 R202: 0000000000000000 R203: 0000000000000000
[ 9144.810049] R204: 0000000000000000 R205: 0000000000000000 R206: 0000000000000000 R207: 0000000000000000
[ 9144.810051] R208: 0000000000000000 R209: 0000000000000000 R210: 0000000000000000 R211: 0000000000000000
[ 9144.810053] R212: 0000000000000000 R213: 0000000000000000 R214: 0000000000000000 R215: 0000000000000000
[ 9144.810055] R216: 0000000000000000 R217: 0000000000000000 R218: 0000000000000000 R219: 0000000000000000
[ 9144.810057] R220: 0000000000000000 R221: 0000000000000000 R222: 0000000000000000 R223: 0000000000000000
[ 9144.810059] R224: 0000000000000000 R225: 0000000000000000 R226: 0000000000000000 R227: 0000000000000000
[ 9144.810061] R228: 0000000000000000 R229: 0000000000000000 R230: 0000000000000000 R231: 0000000000000000
[ 9144.810063] R232: 0000000000000000 R233: 0000000000000000 R234: 0000000000000000 R235: 0000000000000000
[ 9144.810065] R236: 0000000000000000 R237: 0000000000000000 R238: 0000000000000000 R239: 0000000000000000
[ 9144.810067] R240: 0000000000000000 R241: 0000000000000000 R242: 0000000000000000 R243: 0000000000000000
[ 9144.810069] R244: 0000000000000000 R245: 0000000000000000 R246: 0000000000000000 R247: 0000000000000000
[ 9144.810071] R248: 0000000000000000 R249: 0000000000000000 R250: 0000000000000000 R251: 0000000000000000
[ 9144.810073] R252: 0000000000000000 R253: 0000000000000000 R254: 0000000000000000 R255: 0000000000000000
[ 9144.810075] R256: 0000000000000000 R257: 0000000000000000 R258: 0000000000000000 R259: 0000000000000000
[ 9144.810077] R260: 0000000000000000 R261: 0000000000000000 R262: 0000000000000000 R263: 0000000000000000
[ 9144.810079] R264: 0000000000000000 R265: 0000000000000000 R266: 0000000000000000 R267: 0000000000000000
[ 9144.810081] R268: 0000000000000000 R269: 0000000000000000 R270: 0000000000000000 R271: 0000000000000000
[ 9144.810083] R272: 0000000000000000 R273: 0000000000000000 R274: 0000000000000000 R275: 0000000000000000
[ 9144.810085] R276: 0000000000000000 R277: 0000000000000000 R278: 0000000000000000 R279: 0000000000000000
[ 9144.810087] R280: 0000000000000000 R281: 0000000000000000 R282: 0000000000000000 R283: 0000000000000000
[ 9144.810089] R284: 0000000000000000 R285: 0000000000000000 R286: 0000000000000000 R287: 0000000000000000
[ 9144.810091] R288: 0000000000000000 R289: 0000000000000000 R290: 0000000000000000 R291: 0000000000000000
[ 9144.810093] R292: 0000000000000000 R293: 0000000000000000 R294: 0000000000000000 R295: 0000000000000000
[ 9144.810095] R296: 0000000000000000 R297: 0000000000000000 R298: 0000000000000000 R299: 0000000000000000
[ 9144.810097] R300: 0000000000000000 R301: 0000000000000000 R302: 0000000000000000 R303: 0000000000000000
[ 9144.810099] R304: 0000000000000000 R305: 0000000000000000 R306: 0000000000000000 R307: 0000000000000000
[ 9144.810101] R308: 0000000000000000 R309: 0000000000000000 R310: 0000000000000000 R311: 0000000000000000
[ 9144.810103] R312: 0000000000000000 R313: 0000000000000000 R314: 0000000000000000 R315: 0000000000000000
[ 9144.810105] R316: 0000000000000000 R317: 0000000000000000 R318: 0000000000000000 R319: 0000000000000000
[ 9144.810107] R320: 0000000000000000 R321: 0000000000000000 R322: 0000000000000000 R323: 0000000000000000
[ 9144.810109] R324: 0000000000000000 R325: 0000000000000000 R326: 0000000000000000 R327: 0000000000000000
[ 9144.810111] R328: 0000000000000000 R329: 0000000000000000 R330: 0000000000000000 R331: 0000000000000000
[ 9144.810113] R332: 0000000000000000 R333: 0000000000000000 R334: 0000000000000000 R335: 0000000000000000
[ 9144.810115] R336: 0000000000000000 R337: 0000000000000000 R338: 0000000000000000 R339: 0000000000000000
[ 9144.810117] R340: 0000000000000000 R341: 0000000000000000 R342: 0000000000000000 R343: 0000000000000000
[ 9144.810119] R344: 0000000000000000 R345: 0000000000000000 R346: 0000000000000000 R347: 0000000000000000
[ 9144.810121] R348: 0000000000000000 R349: 0000000000000000 R350: 0000000000000000 R351: 0000000000000000
[ 9144.810123] R352: 0000000000000000 R353: 0000000000000000 R354: 0000000000000000 R355: 0000000000000000
[ 9144.810125] R356: 0000000000000000 R357: 0000000000000000 R358: 0000000000000000 R359: 0000000000000000
[ 9144.810127] R360: 0000000000000000 R361: 0000000000000000 R362: 0000000000000000 R363: 0000000000000000
[ 9144.810129] R364: 0000000000000000 R365: 0000000000000000 R366: 0000000000000000 R367: 0000000000000000
[ 9144.810131] R368: 0000000000000000 R369: 0000000000000000 R370: 0000000000000000 R371: 0000000000000000
[ 9144.810133] R372: 0000000000000000 R373: 0000000000000000 R374: 0000000000000000 R375: 0000000000000000
[ 9144.810135] R376: 0000000000000000 R377: 0000000000000000 R378: 0000000000000000 R379: 0000000000000000
[ 9144.810137] R380: 0000000000000000 R381: 0000000000000000 R382: 0000000000000000 R383: 0000000000000000
[ 9144.810139] R384: 0000000000000000 R385: 0000000000000000 R386: 0000000000000000 R387: 0000000000000000
[ 9144.810141] R388: 0000000000000000 R389: 0000000000000000 R390: 0000000000000000 R391: 0000000000000000
[ 9144.810143] R392: 0000000000000000 R393: 0000000000000000 R394: 0000000000000000 R395: 0000000000000000
[ 9144.810145] R396: 0000000000000000 R397: 0000000000000000 R398: 0000000000000000 R399: 0000000000000000
[ 9144.810147] R400: 0000000000000000 R401: 0000000000000000 R402: 0000000000000000 R403: 0000000000000000
[ 9144.810149] R404: 0000000000000000 R405: 0000000000000000 R406: 0000000000000000 R407: 0000000000000000
[ 9144.810151] R408: 0000000000000000 R409: 0000000000000000 R410: 0000000000000000 R411: 0000000000000000
[ 9144.810153] R412: 0000000000000000 R413: 0000000000000000 R414: 0000000000000000 R415: 0000000000000000
[ 9144.810155] R416: 0000000000000000 R417: 0000000000000000 R418: 0000000000000000 R419: 0000000000000000
[ 9144.810157] R420: 0000000000000000 R421: 0000000000000000 R422: 0000000000000000 R423: 0000000000000000
[ 9144.810159] R424: 0000000000000000 R425: 0000000000000000 R426: 0000000000000000 R427: 0000000000000000
[ 9144.810161] R428: 0000000000000000 R429: 0000000000000000 R430: 0000000000000000 R431: 0000000000000000
[ 9144.810163] R432: 0000000000000000 R433: 0000000000000000 R434: 0000000000000000 R435: 0000000000000000
[ 9144.810165] R436: 0000000000000000 R437: 0000000000000000 R438: 0000000000000000 R439: 0000000000000000
[ 9144.810167] R440: 0000000000000000 R441: 0000000000000000 R442: 0000000000000000 R443: 0000000000000000
[ 9144.810169] R444: 0000000000000000 R445: 0000000000000000 R446: 0000000000000000 R447: 0000000000000000
[ 9144.810171] R448: 0000000000000000 R449: 0000000000000000 R450: 0000000000000000 R
```

уязвимости CVE-2025-39866 (use-after-free в подсистеме writeback) [4]. Уязвимость проявляется в ядрах версий < 6.12.16 и возникает при конкурентном доступе к структуре `bdi_writeback` в функции `__mark_inode_dirty()` [13].

Гонка возникает между двумя соревнующимися потоками исполнения в подсистеме writeback, происходит это по следующему механизму:

1. Запись информации происходит при вызове функции `__mark_inode_dirty()`, который читает указатель на `inode->i_wb` (процесс записи информации о том какие «грязные» `inode` могут быть использованы для записи текущего потока)
2. Переключение и сбитие синхронизации исполнения потоков происходит при вызове `inode_switch_wbs_work_fn()` заменяет исполняемый поток записи по указателю на новый контекст writeback и освобождает старый через `wb_put_many()`.

При отсутствии надёжной рабочей синхронизации между этими двумя потоками, может произойти сценарий, что при попытке записи `__mark_inode_dirty()` может обратиться к структуре уже освобожденной функцией `wb_put_many()`, из-за чего возникает ситуация использования страницы после освобождения.

Для демонстрации и вызова уязвимости был создан упрощенный модуль ядра [2], создающий два соревнующихся потока, работающих по описанному ранее принципу. Логика модуля следующая:

1. Поток `cve_thread_1` циклически обращается к полю `list_lock` структуры `bdi_writeback`, предоставляя доступ к двум конкурирующим путям
2. Поток `cve_thread_1` вызывает `__mark_inode_dirty()`, который вызывает `wb_put_many`, который может создать уязвимость.

Модуль использует нативные средства и вызовы ядра Линукс из-за чего при попытке провести статический анализ обнаружение данной уязвимости может быть затруднено, потому что прямой работы с памятью и блокировками нет. Подобный паттерн может быть замаскирован под легитимный системный

модуль, например, оптимизатор записи fastwrite.ko, что демонстрирует серьёзность угрозы.

Сразу после загрузки модуля в ядро (рисунок 7) иницируются конкурентные потоки.

```
[root@host-15 CVE-2025-39866]# make
make -C /lib/modules/6.12.15/build M=/root/CVE-2025-39866 modules
make[1]: вход в каталог «/root/kernel-build/linux-6.12.15»
CC [M] /root/CVE-2025-39866/exploit.o
MODPOST /root/CVE-2025-39866/Module.symvers
CC [M] /root/CVE-2025-39866/exploit.mod.o
CC [M] /root/CVE-2025-39866/.module-common.o
LD [M] /root/CVE-2025-39866/exploit.ko
make[1]: выход из каталога «/root/kernel-build/linux-6.12.15»
[root@host-15 CVE-2025-39866]# ls
exploit.c  exploit.mod  exploit.mod.o  Makefile      Module.symvers
exploit.ko  exploit.mod.c  exploit.o      modules.order
[root@host-15 CVE-2025-39866]# insmod exploit.ko
[root@host-15 CVE-2025-39866]# dmesg > race_log.log
[root@host-15 CVE-2025-39866]#
```

```
XWAYLAND
[ 70.237395] 11:01:23.345095 main Service: VirtualBox host version check
[ 70.238009] 11:01:23.345747 main Daemonizing service ...
[ 70.275723] 11:01:23.383379 main Creating worker thread ...
[ 70.277535] 11:01:23.385281 main Service started
[ 70.310784] 11:01:23.418489 shcl Shared Clipboard: Host does not support
transfers
[ 70.310980] 11:01:23.418723 main Service started
[ 254.810895] hrtimer: interrupt took 6421632 ns
[ 788.566363] exploit: loading out-of-tree module taints kernel.
[ 788.566370] exploit: module verification failed: signature and/or required ke
y missing - tainting kernel
[ 788.568885] CVE-2025-39866: Starting race condition
[ 788.569370] CVE-2025-39866: Threads started
[ 788.571860] BUG: kernel NULL pointer dereference, address: 0000000000000030
[ 788.571860] #PF: error_code(0x0000) - not-present page
```

Рисунок 7 – Запуск эксплойта и инициализация потоков

Через несколько секунд выполнения система аварийно завершается (рисунок 8) с ошибкой NULL pointer dereference в функции __mark_inode_dirty() [14].

```
[ 788.571860] BUG: kernel NULL pointer dereference, address: 0000000000000030
[ 788.571868] #PF: supervisor read access in kernel mode
[ 788.571872] #PF: error_code(0x0000) - not-present page
[ 788.571875] PGD 0 P4D 0
[ 788.571880] Ooops: Ooops: 0000 [#1] PREEMPT SMP NOPTI
[ 788.571885] CPU: 2 UID: 0 PID: 3241 Comm: cve_thread2 Tainted: G OE 6.12.15 #1
[ 788.571890] Tainted: [O]=OOT_MODULE, [E]=UNSIGNED_MODULE
[ 788.571893] Hardware name: innotek GmbH VirtualBox/VirtualBox, BIOS VirtualBox 12/01/2006
[ 788.571895] RIP: 0010: __mark_inode_dirty+0x38/0x380
[ 788.571904] Code: 48 89 fd 53 89 f3 4c 8b 6f 28 0f 1f 44 00 00 f6 c3 18 0f 84 db 00 00 00 f6 85 91 00
00 00 08 0f 85 9a 02 00 00 0f 1f 44 00 00 <49> 8b 45 30 48 8b 40 18 48 85 c0 74 10 89 de 48 89 ef 81 e6 1
8 08
[ 788.571907] RSP: 0018:ffffafcdc3c5fec0 EFLAGS: 00010246
[ 788.571911] RAX: 0000000000000000 RBX: 0000000000000038 RCX: 0000000000000000
[ 788.571914] RDX: 0000000000000000 RSI: 0000000000000038 RDI: ffff9ed3038bd000
[ 788.571917] RBP: ffff9ed3038bd000 R08: 0000000000000000 R09: 0000000000000000
[ 788.571919] R10: 0000000000000000 R11: 0000000000000000 R12: ffffafcdc3ca7968
[ 788.571922] R13: 0000000000000000 R14: 0000000000000000 R15: ffffffffcc0941050
[ 788.571927] FS: 0000000000000000(0000) GS: ffff9ed31bd00000(0000) knlGS: 0000000000000000
[ 788.571930] CS: 0010 DS: 0000 ES: 0000 CR0: 0000000080050033
[ 788.571933] CR2: 0000000000000030 CR3: 0000000103c9a000 CR4: 00000000000106f0
```

Рисунок 8 – Kernel NULL pointer dereference в __mark_inode_dirty

Анализ дампа показывает:

- RIP: __mark_inode_dirty+0x30/0x380 точка сбоя в ядерной функции;
- CR2: 0x0000000000000030 обращение по нулевому адресу;
- PID: поток cve_thread2 явился причиной сбоя;
- Tainted: OE загружен внешний модуль.

Call Trace демонстрирует цепочку вызовов от загруженного модуля к ядерной функции (рисунок 9), подтверждая, что гонка возникает именно между указанными конкурирующими путями.

```

[ 788.571938] Call Trace:
[ 788.571943] <TASK>
[ 788.571947] ? __die+0x1f/0x70
[ 788.571953] ? page_fault_oops+0x163/0x5a0
[ 788.571959] ? srso_alias_return_thunk+0x5/0xfbef5
[ 788.571966] ? srso_alias_return_thunk+0x5/0xfbef5
[ 788.571970] ? search_bpf_extables+0x5b/0x90
[ 788.571978] ? exc_page_fault+0x7a/0x1a0
[ 788.571983] ? asm_exc_page_fault+0x22/0x30
[ 788.571990] ? __pfx_thread_fn2+0x10/0x10 [exploit]
[ 788.571998] ? __mark_inode_dirty+0x38/0x380
[ 788.572006] ? __pfx_thread_fn2+0x10/0x10 [exploit]
[ 788.572009] thread_fn2+0x1d/0x40 [exploit]
[ 788.572013] kthread+0xce/0x100
[ 788.572019] ? __pfx_kthread+0x10/0x10
[ 788.572023] ret_from_fork+0x30/0x50
[ 788.572028] ? __pfx_kthread+0x10/0x10
[ 788.572031] ret_from_fork_asm+0x1a/0x30
[ 788.572039] </TASK>
[ 788.572041] Modules linked in: exploit(0E) snd_seq_dummy snd_hrtimer af_packet qrtr rfkill edac_core i

```

Рисунок 9 – Call Trace путь от модуля к функции сбоя

Полученный сбой подтверждает наличие race condition в подсистеме writeback ядра 6.12.15. Поток, вызвавший mark_inode_dirty(), обратился к полю inode->i_wb, которое в момент вызова содержало недействительный указатель. Это произошло потому, что асинхронный механизм переключения writeback-контекста освободил старую структуру раньше, чем функция __mark_inode_dirty() завершила работу с ней.

Ввиду того что вызываемые функции и операции внутри описанного модуля являются легитимными, а сами функции не напрямую могут вызвать асинхронизацию и нарушение памяти, статический анализ существенных результатов не даст, как и в случае с прос-модулем. Нарушения работы модуля возможно обнаружить только при его фактическом запуске по определенному времени исполнения атомарных операций, что требует точности и работы с динамикой. Из чего следует вывод в необходимости анализа разрабатываемых модулей и программ в динамике. В данной работе основной фокус был на попытке обнаружения данного семейства уязвимостей статическим анализом, ввиду распространённости практики избыточности статистического анализа при проверке на уязвимости, который, как показывают наши опыты, не достаточны для выявления определенных типов уязвимостей.

Использование динамического анализа не гарантирует обнаружения всех типов гонок, но значительно увеличивает вероятность обнаружения данной

уязвимости, потому как даёт более полную картину о состоянии исполняемых процессов и больше признаков для нахождения ошибки.

Заключение

В работе рассмотрены проблемы состояний гонки при параллельном доступе на чтение/запись к данным через файлы виртуальной файловой системы /proc в ядре Linux, а также в подсистеме записи страничного кэша в постоянное хранилище writeback на примере CVE-2025-39866.

Проведённые эксперименты продемонстрировали недостаточность статических анализаторов при попытке обнаружения гонки при полной асинхронной работе функций, но успешно выявляет ошибки использования мьютексов. Уязвимость CVE-2025-39866 осталась не обнаружена ввиду сложного межпроцедурного характера гонки, все случаи запуска уязвимых исполняемых модулей приводили к падению ядра.

Эксперименты подтвердили необходимость использования динамического анализа для обнаружения несинхронизированных потоков и вызовов функций. Рекомендуется двухэтапный анализ модулей ядра: статический анализ эффективен для быстрого выявления типовых ошибок, а также неправильного использования примитивов синхронизации, по завершению статического анализа проводить динамическое тестирование для обнаружения гонок и других процессов (RCU, seq_file) и ошибок, не покрываемых статикой.

Литература

1. The kernel's command-line parameters [Электронный ресурс] // The Linux Kernel documentation. – URL: <https://www.kernel.org/doc/html/latest/admin-guide/kernel-parameters.html> (дата обращения: 13.04.2026).
2. Сборка модулей ядра [Электронный ресурс] // ALT Linux Wiki. – URL: https://www.altlinux.org/Сборка_модулей_ядра (дата обращения: 13.04.2026).
3. Linux Kernel Locking Documentation [Электронный ресурс] // The Linux Kernel documentation. – URL: <https://origin.kernel.org/doc/html/v6.6/locking/index.html> (дата обращения: 13.04.2026).

4. CVE-2025-39866 Detail // <https://nvd.nist.gov/> URL:
<https://nvd.nist.gov/vuln/detail/CVE-2025-39866> (дата обращения: 12.04.2026).
5. Уймин, А. Г. Цифровые двойники сетевых инфраструктур: точность, методы и практические решения / А. Г. Уймин // Радиотехнические и телекоммуникационные системы. – 2023. – № 3(51). – С. 44-52. – DOI 10.24412/2221-2574-2023-3-44-52. – EDN QUSITK.
6. Dossche N., Abrath B., Coppens B. A Context-Sensitive, Outlier-Based Static Analysis to Find Kernel Race Conditions //arXiv preprint arXiv:2404.00350. – 2024.
7. Андрианов П. С. Анализ корректности синхронизации компонентов ядра операционных систем // Труды ИСП РАН. 2019. №5. URL:
<https://cyberleninka.ru/article/n/analiz-korrektnosti-sinhronizatsii-komponentov-yadra-operatsionnyh-sistem> (дата обращения: 04.05.2026).
8. Андрианов П. С., Мутилин В. С., Хорошилов А. В. Метод легковесного статического анализа для поиска состояний гонок // Труды ИСП РАН. 2015. №5. URL: <https://cyberleninka.ru/article/n/metod-legkovesnogo-staticheskogo-analiza-dlya-poiska-sostoyaniy-gonok> (дата обращения: 04.05.2026)
9. Upadhyay A., Laxmi V., Naval S. Navigating the Concurrency Landscape: A Survey of Race Condition Vulnerability Detectors //arXiv preprint arXiv:2312.14479. – 2023.
10. Заборовский Н. В., Тормасов А. Г. Статическое обнаружение гонок в коде, содержащем ветвления и циклы // Прикладная информатика. 2011. №6 (36). URL:
<https://cyberleninka.ru/article/n/staticheskoe-obnaruzhenie-gonok-v-kode-soderzhaschem-vetvleniya-i-tsikly> (дата обращения: 04.05.2026).
11. Андрианов П. С., Мутилин В. С., Хорошилов А. В. Конфигурируемый метод поиска состояний гонок в операционных системах с использованием предикатных абстракций // Труды ИСП РАН. 2016. №6. URL:
<https://cyberleninka.ru/article/n/konfiguriruemyy-metod-poiska-sostoyaniy-gonok-v-operatsionnyh-sistemah-s-ispolzovaniem-predikatnyh-abstraktsiy> (дата обращения: 03.05.2026).

12. Li T. et al. SPATA: Effective OS bug detection with summary-based, alias-aware, and path-sensitive tpestate analysis // ACM Transactions on Computer Systems. – 2024. – Т. 42. – №. 3-4. – С. 1-40.
13. Vulnerability CVE-2025-39866: Information // ALT LINUX TEAM URL: <https://packages.altlinux.org/en/vuln/CVE-2025-39866> (дата обращения: 01.05.2026).
14. Oops! Debugging Kernel Panics // Linux Journal URL: <https://www.linuxjournal.com/content/oops-debugging-kernel-panics-0> (дата обращения: 01.05.2026).
15. What is RCU? // The Linux Kernel URL: <https://www.kernel.org/doc/html/latest/RCU/whatisRCU.html> (дата обращения: 01.05.2026).

Литература

1. The kernel's command-line parameters [Электронна ресурс] // The Linux Kernel documentation. - URL: <https://www.kernel.org/doc/html/latest/admin-guide/kernel-parameters.html> (атшазынархара амш: 13.04.2026).
2. Аядро амодульква радгалра [Электронна ресурс] / / ALT Linux Wiki. - URL: https://www.altlinux.org/Сборка_модулей_ядра (атшазынархара амш: 13.04.2026).
3. Linux Kernel Locking Documentation [Электронна ресурс] / / The Linux Kernel documentation. - URL: <https://origin.kernel.org/doc/html/v6.6/locking/index.html> (атшазынархара амш: 13.04.2026).
4. CVE-2025-39866 Detail // <https://nvd.nist.gov> / URL: <https://nvd.nist.gov/vuln/detail/CVE-2025-39866> (атшазынархара амш: 12.04.2026).
5. Уймин, а.Г. Ацифра аҗвазаква асетъ инфраструктуракв: атамамра, аҗамальква игъи апрактика щаквыргылраква / а. Г. Уймин // Радиотехническа игъи телекоммуникационна системакв. – 2023. – № 3(51). – 44-52 ан. – DOI 10.24412/2221-2574-2023-3-44-52. – EDN QUSITK.
6. Dossche N., Abrath B., Coppens B. A Context-Sensitive, Outlier-Based Static Analysis to Find Kernel Race Conditions // arXiv preprint arXiv:2404.00350. – 2024.

7. Андрианов П.С. операцията системава рквта акомпонентква синхронизация тамамта Алыргара // ИСП РАН Анхараква. 2019. №5. URL: <https://cyberleninka.ru/article/n/analiz-korrektnosti-sinhronizatsii-komponentov-yadra-operatsionnyh-sistem> (атшазынархара амш: 04.05.2026).
8. Андрианов П.С., Мутилин В. С., Хорошилов А. В. апхъанысраква раъаща азыпшгара ахъаз йласу астатика алыргара Агламалъ // ИСП РАН Анхараква. 2015. №5. URL: <https://cyberleninka.ru/article/n/metod-legkovesnogo-staticheskogo-analiza-dlya-poiska-sostoyaniy-gonok> (атшазынархара амш: 04.05.2026)
9. Upadhyay A., Laxmi V., Naval S. Navigating the Concurrency Landscape: A Survey of Race Condition Vulnerability Detectors // arXiv preprint arXiv:2312.14479. – 2023.
10. Заборовский Н. В., Тормасов А. Г. Статическое обнаружение гонок в коде, содержащем ветвления и циклы // Прикладная информатика. 2011. №6 (36). URL: <https://cyberleninka.ru/article/n/staticheskoe-obnaruzhenie-gonok-v-kode-soderzhaschem-vetvleniya-i-tsikly> (дата обращения: 04.05.2026).
11. Андрианов П.С., Мутилин В. С., Хорошилов А. В. Конфигурировать зчауа агламалъ апхъанысраква раъаща азыпшгара операцията системава рпны предикатна абстракцияква гладысабапуамца // ИСП РАН Анхараква. 2016. №6. URL: <https://cyberleninka.ru/article/n/konfiguriruemyy-metod-poiska-sostoyaniy-gonok-v-operatsionnyh-sistemah-s-ispolzovaniem-predikatnyh-abstraktsiy> (атшазынархара амш: 03.05.2026).
12. Li T. et al. SPATA: Effective OS bug detection with summary-based, alias-aware, and path-sensitive typestate analysis // ACM Transactions on Computer Systems. – 2024. - Т.42. – №. 3-4. - АН.1-40.
13. Vulnerability CVE-2025-39866: Information // ALT LINUX TEAM URL: <https://packages.altlinux.org/en/vuln/CVE-2025-39866> (атшазынархара амш: 01.05.2026).
14. Oops! Debugging Kernel Panics // Linux Journal URL: <https://www.linuxjournal.com/content/oops-debugging-kernel-panics-0> (атшазынархара амш: 01.05.2026).

15. What is RCU? // The Linux Kernel URL:
<https://www.kernel.org/doc/html/latest/RCU/whatisRCU.html> (атшазынархара амш:
01.05.2026).