УДК 004.056

ПРИМЕНЕНИЕ ТЕХНОЛОГИИ eBPF ДЛЯ МОНИТОРИНГА БЕЗОПАСНОСТИ ОС НА БАЗЕ ЯДРА LINUX

Д.А. Кузьмин[1], А.С. Ершов[1]

[1]Саратовский государственный технический университет им. Гагарина Ю.А., г. Саратов
e-mail: mr.dmint+ssau@mail.ru

Аннотация

В данной статье рассматривается использование технологии eBPF в ядре Linux для мониторинга системных вызовов, сетевой активности, взаимодействия процессов, операций с файлами, работы контейнеров и пользовательских действий. Отслеживая различные события системы через eBPF-программы, возможно проводить анализ на наличие подозрительных действий и превентивно предотвращать их на низком программном уровне, с привилегиями ядра.

Ключевые слова: ядро Linux, eBPF, информационная безопасность, мониторинг, операционные системы.

Введение

eBPF (от англ. Extended Berkeley Packet Filter) [1] – это технология в ядре Linux (а также в меньшей степени в Windows), позволяющая запускать программы в изолированной среде в привилегированном контексте. Она расширяет возможности ядра в сфере безопасности и мониторинга ресурсов без необходимости остановки и перекомпиляции ядра.

eBPF позволяет в реальном времени отслеживать системные вызовы, сетевую активность, взаимодействие между процессами, операции с файлами. На этой основе реализуются механизмы обнаружения вторжений (HIDS), анализа поведения процессов, контроля сетевых потоков (NDR) и аудита действий пользователей.

Вместе с этим eBPF формирует новую поверхность атаки [2]. Существуют сценарии [3] внедрения вредоносных eBPF-программ для перехвата системных вызовов, маскировки активности и создания rootkit. Эффективное и безопасное использование eBPF [4] требует строгого контроля загрузки eBPF-программ, ограничения привилегий, аудита активности, связанной с eBPF, и регулярного обновления ядра для устранения актуальных уязвимостей.

В условиях растущей доли Linux-устройств [5], их подавляющего доминирования в мобильном, серверных сегментах и среди сетевого оборудования, а также в процессе выполнения постановлений Указа Президента №166 и работы с отечественными дистрибутивами GNU/Linux, защита и мониторинг ОС на базе ядра Linux с помощью технологии eBPF является значимым элементом комплексного процесса безопасности ОС.

1. Принцип работы eBPF

Удобство и революционность eBPF состоит в реализации внутри ядра Linux посредника для работы с сами ядром, а также точек привязки посредника, с которыми он может взаимодействовать. Наличие посредника, которым и является eBPF, обеспечивает возможность работы с ядром без его пересборки. Это обеспечивается внутренней песочницей с ограниченной областью памяти, правилами попадания в песочницу, условиями нахождения и работы в ней. Общий процесс работы с eBPF показан на рисунке 1:

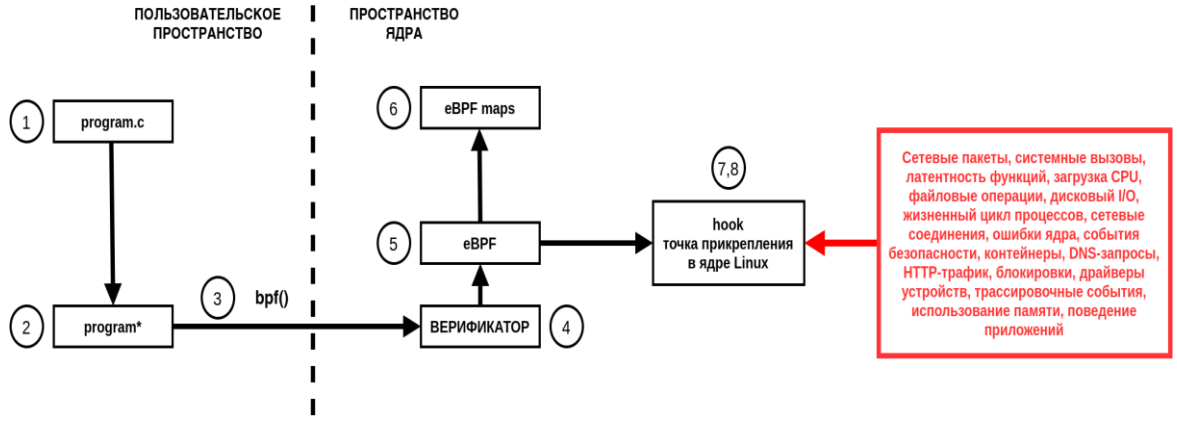

Рисунок 1 – Схема работы eBPF

В процессе работы с eBPF можно выделить основные этапы [6]: Разработка программы (описано пунктом 1 ниже и на рисунке 1). Загрузка и верификация программы (пункты 2 – 4 ниже). Инициализация и привязка (пункты 5 – 7 ниже). Исполнение, обмен данными, штатный жизненный цикл программы до завершения.

Необходимо детализировать описанную выше последовательность [7]:

1) eBPF–программа компилируется через clang/LLVM в eBPF-байткод (ELF-объект с секциями под разные типы программ и maps);

2) пользовательское приложение загружает объект в ядро с помощью системного вызова bpf();

3) ядро перед загрузкой передаёт программу верификатору, который выполняет статический анализ: проверяет корректность обращения с памятью, контроль потока исполнения, допустимость выполняемых операций и завершимость (отсутствие бесконечных циклов, доказуемая выполнимость программы);

4) если программа проходит верификацию, она принимается ядром. Если нет – загрузка отклоняется;

5) при успешной загрузке создаются и инициализируются eBPF maps – структуры для хранения состояния и обмена данными с пользовательским пространством;

6) программа прикрепляется к конкретной точке ядра – hook (например, XDP, kprobe, tracepoint, LSM);

7) ядро регистрирует эту привязку, после чего программа становится частью соответствующего пути исполнения.

При наступлении события в точке привязки ядро вызывает связанную с этой точкой eBPF-программу. Программа выполняется в контексте ядра и обрабатывает входные данные. По завершении возвращает результат, который влияет на дальнейшее поведение ядра. Пользовательское приложение параллельно может читать данные из maps или буферов (perf/ring buffer), получая результаты работы программы.

2. Мониторинг безопасности

Мониторинг безопасности можно осуществлять на основе утилиты bpftrace, реализовав однострочные скрипты:

1. Отслеживание запросов к определенным IP: `sudo bpftrace -e 'tracepoint:sock:inet_sock_set_state /args->protocol == 6 && args->newstate == 1/ { printf("[%lld] pid=%d uid=%d comm=%s TCP_SYN_SENT %s:%d -> %s:%d\n", nsecs, pid, uid, comm, ntop(args->saddr), args->sport, ntop(args->daddr), args->dport); }' | grep '<IP_ADDRESS>'`;

2. Отслеживание команд терминала, содержащих перечень потенциально опасных операций: `sudo /usr/sbin/execsnoop-bpfcc | grep -Ei 'apt|apt-get|dpkg|yum|dnf|pacman|rpm|install|remove|purge|bash -c|sh -c|python|perl|ruby|curl|wget|chmod|chown|rm|history|\.bash_history|\.zsh_history|truncate|echo .* >|tee'`;

Также можно реализовать более сложный механизм контроля на основе eBPF. Например, программа, загружаемая в ядро с использованием библиотеки BCC (BPF Compiler Collection) посредством python-скрипта, может перехватывать системный вызов openat через механизм kprobe и анализировать обращения к файловой системе. В процессе выполнения извлекаются имя файла из пользовательского пространства (с использованием helper-функций, таких как `bpf_probe_read_user_str`), идентификатор процесса (PID/TGID) и имя процесса (comm), после чего выполняется проверка на соответствие заданным шаблонам (например, `/etc/shadow`, `/etc/passwd`). Проверка реализуется на уровне байтового сравнения строки пути, что обусловлено ограничениями eBPF. Процесс работы обработчика eBPF-программы показан на рисунке 2:

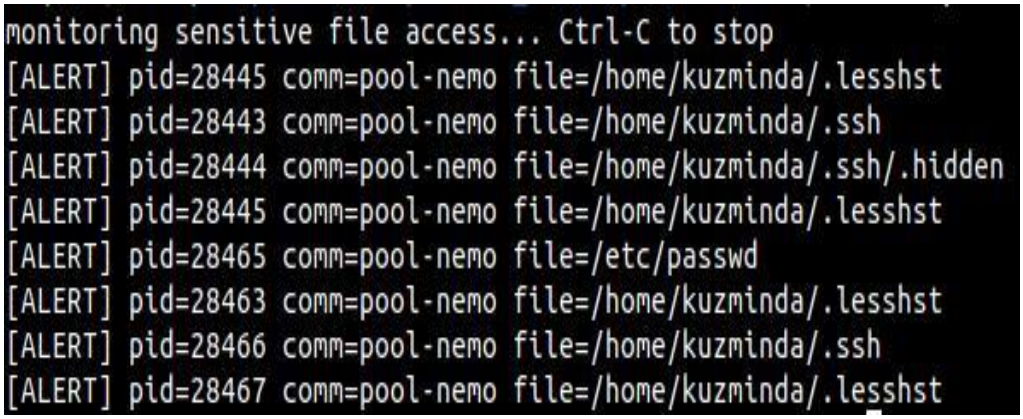

Рисунок 2 – Схема работы eBPF

При обнаружении совпадения формируется структурированное событие, содержащее контекст выполнения, которое передаётся в пользовательское пространство

через механизм perf buffer (`BPF_PERF_OUTPUT`). В пользовательской части реализуется обработчик, который принимает эти события, декодирует структуру и выполняет вывод информации в терминал.

Заключение

В статье рассмотрены практические сценарии применения eBPF: мониторинг сетевой активности, отслеживание терминальных команд, перехват системных вызовов для контроля доступа к чувствительным файлам. eBPF обеспечивает наблюдаемость ОС GNU/Linux в режиме реального времени с низкими накладными расходами и может использоваться для построения систем аудита и обнаружения аномалий.

Список использованных источников

1. BPF: профессиональная оценка производительности / Грегг Брендан – СПб.: Питер, 2024. – 880 с.

2. eBPF глазами хакера. Часть 3. – URL: https://www.securityvision.ru/blog/ebpf-glazami-khakera-chast-3/ (дата обращения: 22.03.2026).

3. eBPF: E for Evil. – URL: https://rt-solar.ru/solar-4rays/blog/4918/ (дата обращения: 22.03.2026).

4. Аудит по-взрослому. Применяем Tetragon на практике (и убеждаемся в силе eBPF). – URL: https://xakep.ru/2025/04/09/ebpf-tetragon/ (дата обращения: 22.03.2026).

5. Desktop Operating System Market Share Worldwide. – URL: https://gs.statcounter.com/os-market-share/desktop/worldwide (дата обращения: 22.03.2026).

6. Learning eBPF / Liz Rice. – O'Reilly Media, Inc., 2023. – 236 p.

7. eBPF Docs. – URL: https://docs.ebpf.io/ (дата обращения: 22.03.2026).

APPLICATION OF eBPF TECHNOLOGY FOR MONITORING OS SECURITY BASED ON THE LINUX KERNEL

D.A. Kuzmin[1], A.S. Ershov[1]

[1]Yuri Gagarin State Technical University of Saratov, Saratov
e-mail: mr.dmint+ssau@mail.ru

Abstract

This article discusses the use of eBPF technology in the Linux kernel to monitor system calls, network activity, process interactions, file operations, container operation, and user actions. By monitoring various system events through eBPF programs, it is possible to analyze for suspicious actions and prevent them proactively at a low software level, with kernel privileges.

Keywords: Linux kernel, eBPF, information security, monitoring, operating systems.