ИПМ им.М.В.Келдыша РАН

Абрау-2023 • Труды конференции

А.В. Никешин, В.З. Шнитман

Подходы к тестированию модуля OpenvSwitch сетевой подсистемы ядра Linux

Рекомендуемая форма библиографической ссылки

Никешин А.В., Шнитман В.З. Подходы к тестированию модуля OpenvSwitch сетевой подсистемы ядра Linux // Научный сервис в сети Интернет: труды XXV Всероссийской научной конференции (18-21 сентября 2023 г., онлайн). — М.: ИПМ им. М.В.Келдыша, 2023. — С. 271-282.
https://doi.org/10.20948/abrau-2023-4
https://keldysh.ru/abrau/2023/theses/4.pdf

Видеозапись выступления

Презентация к докладу

Подходы к тестированию модуля OpenvSwitch сетевой подсистемы ядра Linux

А.В. Никешин, В.З. Шнитман

Институт системного программирования им. В.П. Иванникова РАН

Аннотация. В данной работе представлен опыт исследования безопасности модуля OpenvSwitch сетевой подсистемы ядра Linux. Сегодня Linux является одной из самых популярных операционных систем в мире. Качество кода этой ОС (и в частности его ядра) непосредственно влияет на надежность и безопасность всего спектра продуктов на его основе. Международное сообщество разработчиков Linux прилагает огромные усилия для повышения его надежности, используя всевозможные методы тестирования и поиска ошибок. Однако количество обнаруживаемых ежегодно новых ошибок и количество неисправленных старых ошибок показывает, что этих усилий явно недостаточно. В работе использовались методы архитектурного анализа и фаззинг-тестирования модуля OpenvSwitch. Для фаззинг-тестирования применялся инструмент syzkaller с открытым исходным кодом от компании Google. Найдена одна ошибка, принятая в основную ветку ядра. Работа выполнена в рамках проекта Технологического центра исследования безопасности ядра Linux (ИСП РАН).

Ключевые слова: ядро Linux, тестирование, фаззинг, OpenvSwitch, syzkaller

Testing the OpenvSwitch module of the Linux kernel network subsystem

A.V. Nikeshin, V.Z. Shnitman

Ivannikov Institute for System Programming of the Russian Academy of Sciences

Abstract. This paper presents the experience of researching the security of the OpenvSwitch module of the Linux kernel network subsystem. Today Linux is one of the most popular operating systems in the world. The quality of the code of this OS (and in particular its kernel) directly affects the reliability and security of the entire range of products based on it. The international community of Linux developers is making great efforts to improve its reliability, using all kinds of testing methods and error detections. However, the number of new errors detected annually and the number of old errors that have

not been corrected shows that these efforts are clearly insufficient. The methods of architectural analysis and fuzzing testing of the OpenvSwitch module were applied in the work. The syzkaller tool was used for fuzzing testing. One error, accepted into the main branch of the kernel, was found. The work is part of the project of the Linux Verification Center (ISP RAS).

Keywords: linux kernel, testing, fuzzing, OpenvSwitch, syzkaller

1. Введение

Сегодня Linux является одной из самых популярных операционных систем в мире. Несомненно, качество кода этой ОС (и в частности его ядра) непосредственно влияет на надежность и безопасность всего спектра продуктов на его основе. Международное сообщество разработчиков Linux прилагает огромные усилия для повышения его надежности, используя всевозможные методы тестирования и поиска ошибок.

В данной работе представлен опыт исследования безопасности модуля OpenvSwitch сетевой подсистемы ядра Linux. Работа выполняется в рамках научных работ Технологического центра исследования безопасности ядра Linux [1].

Структура статьи:

- во втором разделе представлен международный проект по разработке ядра Linux,
- в третьем разделе обозначены применяемые методы исследования,
- в четвертом разделе описан фаззер syzkaller,
- в пятом разделе кратко описан модуль OpenvSwitch сетевой подсистемы ядра,
- в шестом разделе описана организация тестового стенда,
- в седьмом разделе представлены результаты тестирования.

2. Международный проект по разработке ядра Linux

Согласно отчету Linux Foundation операционные системы семейства Linux занимают ведущие позиции в мире по многим направлениям [2]:

- более 96% веб-серверов работают на Linux;
- более 85% смартфонов в мире используют Android, основанный на ядре Linux;
- все 100% суперкомпьютеров из списка 500 самых производительных используют ОС на базе ядра Linux;
- более 90% облачных сервисов, в том числе Google, Amazon, Ebay, Paypal, Walmart, Yandex, работают на Linux;
- большинство бирж, включая NYSE, NASDAQ, London Exchange, Tokyo Stock Exchange, работают на Linux.

Проект по разработке ядра Linux является одним из самых крупных и долгоживущих среди свободного программного обеспечения. К

настоящему времени разработка ядра Linux длится более 30 лет, количество строк кода ядра достигло 22 миллионов. Согласно статистике, собранной за 10 последних лет, в работе над каждой версией ядра принимает участие более 1600 различных разработчиков.

На данный момент сложилась четкая циклическая схема работы по развитию ядра Linux. Каждые 9-10 недель происходит выпуск новой версии основной ветки ядра. Первые две недели отводятся под «окно слияний» (merge window), в ходе которого в основную ветку попадает новая функциональность, прошедшая предварительное обсуждение и тестирование в рамках тематических веток разработки ядра. Затем выпускается первый кандидат на релиз очередной версии ядра (например, 5.10-rc1). С этого момента в основную ветку ядра принимаются только исправления ошибок (так называемый период стабилизации, 7-8 недель). В процессе стабилизации регулярно выпускаются очередные кандидаты на релиз (rc2, rc3 и т. д.), которые проходят тестирование у заинтересованных участников процесса. Процесс завершается выпуском очередной версии ядра, давая начало следующему циклу.

Поддержка предыдущих стабильных веток ядра продолжается некоторое время, в этот период регулярно выпускаются обновления с исправлениями обнаруженных ошибок. Стандартная длительность поддержки стабильной версии ядра немного превышает время работы над следующей версией. Например, в обычных условиях поддержка версии 5.10 была бы прекращена вскоре после того как была выпущена версия 5.11. Но для некоторых версий принимается решение о долгосрочной поддержке, которая может продлиться несколько лет. Например, для версии 5.10 было принято решение о поддержке до конца 2026 года.

Огромная распространенность ядра Linux как основы многочисленных программных систем требует непрерывных усилий со стороны разработчиков и производителей по обеспечению надежности и безопасности ядра. Так, в составе поставок ядра представлены инструменты для статической проверки кода (checkpatch, coccinelle, sparse), набор автоматических тестов kselftest, среда модульного тестирования KUnit, средства для сбора покрытия кода (kcov, gcov), средства для обнаружения различных ошибок выполнения и утечек памяти (KASAN, KFENCE, Kmemleak, UBSAN, KCSAN), средства для отладки кода ядра kgdb и kdb. Дополнительно поддерживаются специализированные наборы тестов (xfstests, blktests, FIO, NFS Connectathon testsuite).

Отдельные компании, использующие ядро Linux в своих продуктах, разрабатывают тестовые наборы и поддерживают инфраструктуры для проверки обновлений отдельных веток ядра Linux. Так, компания Intel представила тестовый набор LKP (Linux Kernel Performance tests) [3]. Google в рамках разработки ОС Android тестирует на своих облачных

ресурсах обновления соответствующих веток ядра Linux, а также предоставляет ресурсы в своем облаке для фаззинг-тестирования ядра Linux с использованием инструмента syzkaller [4].

Кроме того, существуют различные совместные проекты с открытым исходным кодом по разработке инструментов для проведения анализа и тестирования кода ядра Linux (Linux Test Project, KernelCI, Linaro Automation and Validation Architecture и др.).

Тем не менее, несмотря на прилагаемые общие усилия международного сообщества, обеспечение надежности и безопасности ядра Linux остается нерешенной актуальной задачей. Согласно только одному из отчетов количество открытых ошибок постоянно растет, при том, что в год исправляется до 400 ошибок [5]. Объем кода ядра постоянно требует огромных ресурсов на его анализ, которых на данный момент явно недостаточно, несмотря на совместное участие в этой работе многих сторон, в том числе, крупных компаний.

Ядро Linux пользуется популярностью и у многочисленных отечественных разработчиков, в первую очередь, различных операционных систем общего и специального назначения (Альт, AstraLinux, РОСА, РедОС, Аврора и др.). При этом многие из этих решений применяются в государственных информационных системах, на объектах критической инфраструктуры и в других ответственных областях.

Понятно, что отечественные компании не сидят сложа руки и также решают задачи по обеспечению надежности и безопасности своих продуктов, предлагая, в том числе оригинальные решения. Но их возможности явно меньше по сравнению с международными гигантами. Поэтому напрашивается решение – объединить усилия всех отечественных компаний для широкого анализа безопасности всех компонентов ядра Linux, чтобы решать эту задачу сообща. При этом конкурентные преимущества участников никак не затрагиваются, поскольку лежат совсем в других областях. С этой целью при поддержке отечественного регулятора создан Технологический центр исследования безопасности ядра Linux, главной задачей которого заявлено повышение уровня безопасности отечественных Linux-систем [1].

3. Основные методы исследования ядра Linux

Архитектурный анализ

Основная цель архитектурного анализа заключается в определении поверхности атаки, т.е. набора модулей и функций, которые обрабатывают данные, пришедшие из источников, потенциально контролируемых нарушителем согласно рассматриваемой модели угроз. Такие источники принято называть недоверенными.

Статический анализ

Анализ кода ядра без его реального выполнения. Цель данного анализа – обнаружение проблемных мест в коде, которые могут приводить к различным ошибкам и потенциальным уязвимостям во время реального выполнения.

Системное и модульное тестирование

Системное и модульное тестирование нацелено на проверку корректности функционирования ядра в ходе заданного набора тестовых сценариев. Модульное тестирование проводится для отдельных функций ядра (подмножества функций) на инструментальной машине в специально подготовленном окружении. Системное тестирование нацелено на проведение тестирования ядра на целевой машине (физической или виртуальной), где ядро работает в режиме, приближённом к режиму нормальной работы.

Фаззинг-тестирование

Фаззинг-тестированием называют динамический метод анализа кода, при котором на вход исследуемому программному обеспечению подаются случайные данные в попытке выявить дефекты.

В 1988 году Бартон Миллер (University of Wisconsin-Madison) предложил подавать на вход приложениям неструктурированные случайные данные [6]. Данный метод, названный автором Fuzz-генератором, был опробован на системных утилитах ОС Unix и показал хорошие результаты: значительная часть приложений завершилась с фатальными ошибками или "зависли". Дальнейшее развитее данного подхода привело к созданию более умных Fuzz-генераторов: на вход подаются более-менее структурированные данные, соответствующие конкретной тестируемой системе, в которых отдельные части или поля меняются или случайным образом, или в соответствии со схемой генератора тестов.

В случае с ядром Linux в качестве входа, как правило, выступают системные вызовы и пакеты, попадающие в ядро через те или иные устройства, например, сетевые.

Анализ помеченных данных

Целью анализа помеченных данных является проверка того, что чувствительные данные (например, пароли или приватные ключи) удаляются из памяти, как только исчезает необходимость в их наличии. Проверка производится динамически при помощи полносистемных эмуляторов на выбранных сценариях, в которых выделяется место

появления чувствительных данных и момент времени, где эти данные перестают быть необходимыми.

4. Инструмент для фаззинг-тестирования syzkaller

При фаззинг-тестировании на вход системе подаются случайные данные с целью выявить не запланированное поведение системы. Однако для эффективной работы на сложных системах фаззер должен обладать рядом возможностей:

- знание об интерфейсах тестируемого программного обеспечения (применительно к ядру – это знание семантики системных вызовов), позволяющее не тратить время на бесперспективные вызовы;
- обратная связь по покрытию кода, позволяющая эффективно сокращать множество используемых входных данных;
- механизм мутации входных данных, на основе знаний о покрытии кода.

В данном проекте в качестве основного инструмента для фаззинг-тестирования ядра Linux используется инструмент syzkaller – продукт с открытым исходным кодом от компании Google, обладающий указанными выше качествами [7].

Данный фаззер поддерживает следующие ядра операционных систем: Linux, Windows (с ограничениями), NetBSD, OpenBSD, FreeBSD, Android, Fuchsia, Akaros. В процессе своей работы syzkaller конструирует случайные программы на основе грамматик системных вызовов, запускает их на исследуемой платформе, получает обратную связь в виде достигнутого покрытия кода ядра, проверяет лог ядра на предмет записей об ошибках. Случайные программы конструируются таким образом, чтобы увеличивать размер покрытия.

Для эффективного выявления ошибок в ядре исследуемой ОС должны быть включены отладочные опции и опции самопроверки. В таком случае при возникновении ошибок сообщения о них в лог-файл. Основными опциями диагностики ядра являются проверки KASAN, KUBSAN, KMSAN, KCSAN.

KASAN (kernel address sanitizer) – детектор ошибок доступа к памяти. Способен обнаруживать ошибки вида use-after-free (использование после освобождения ресурса) и out-of-bounds (доступ за границами объекта).

KUBSAN (kernel undefined behavior sanitizer) – используется для обнаружения ошибок неопределенного поведения.

KMSAN (kernel memory sanitizer) – используется для обнаружения ошибок доступа к неинициализированной памяти.

KCSAN (kernel concurrency sanitizer) – используется для обнаружения состояния гонок.

Кроме этого, в ядре Linux существует множество других опций для диагностики утечек памяти, некорректного использования блокировок,

вызова планировщика в критических секциях, переполнения счетчиков ресурсов и др.

Для измерения покрытия используется механизм KCOV, позволяющий измерять покрытие кода ядра с точностью до отдельного системного вызова. На его основе syzkaller конструирует множество программ, пытаясь увеличить покрытие.

При обнаружении ошибки в ядре операционной системы, syzkaller пытается сгенерировать минимальную версию программы на языке Си, которая бы надежным образом вызывала срабатывание данной ошибки. В большом количестве случаев это удается сделать, в том числе для ошибок, срабатывающих в мультипоточной среде.

Полезным качеством syzkaller является возможность строить «фаззинг-фермы» – схема, в которой фаззингом занимаются несколько машин с одинаковой конфигурацией и которые обмениваются между собой информацией о покрытии кода.

В целом, самим разработчиком выявлено несколько тысяч ошибок в коде ядра [4,8]. Тем не менее, по оценкам разработчика степень покрытия кода ядра в целом остается не большой, что оставляет большой простор для дальнейшей работы.

Таким образом, к достоинствам инструмента syzkaller можно отнести поддержку разработчика в лице крупной компании, открытый исходный код с документацией, значительный набор готовых описаний интерфейсов ядра Linux. Оценка покрытия кода является важным свойством фаззера и требуется не только для эффективной работы самого инструмента, но и помогает разработчикам вносить изменения в описания вызовов, чтобы достигнуть новых участков тестируемого кода. Инструмент работает в режиме непрерывного цикла тестирования.

5. Модуль OpenvSwitch ядра Linux

В рамках работ Технологического центра было выделено направление исследования сетевой подсистемы ядра Linux как наиболее значимого и распространенного интерфейса для атаки. Был выполнен обзор этой подсистемы для версии ядра 5.10.56 (каталог net/ дистрибутива) и проведен анализ поверхности атаки нескольких модулей. На рис. 1 представлена общая схема сетевой подсистемы, более подробное ее описание с интерактивной схемой доступно на сайте центра [9].

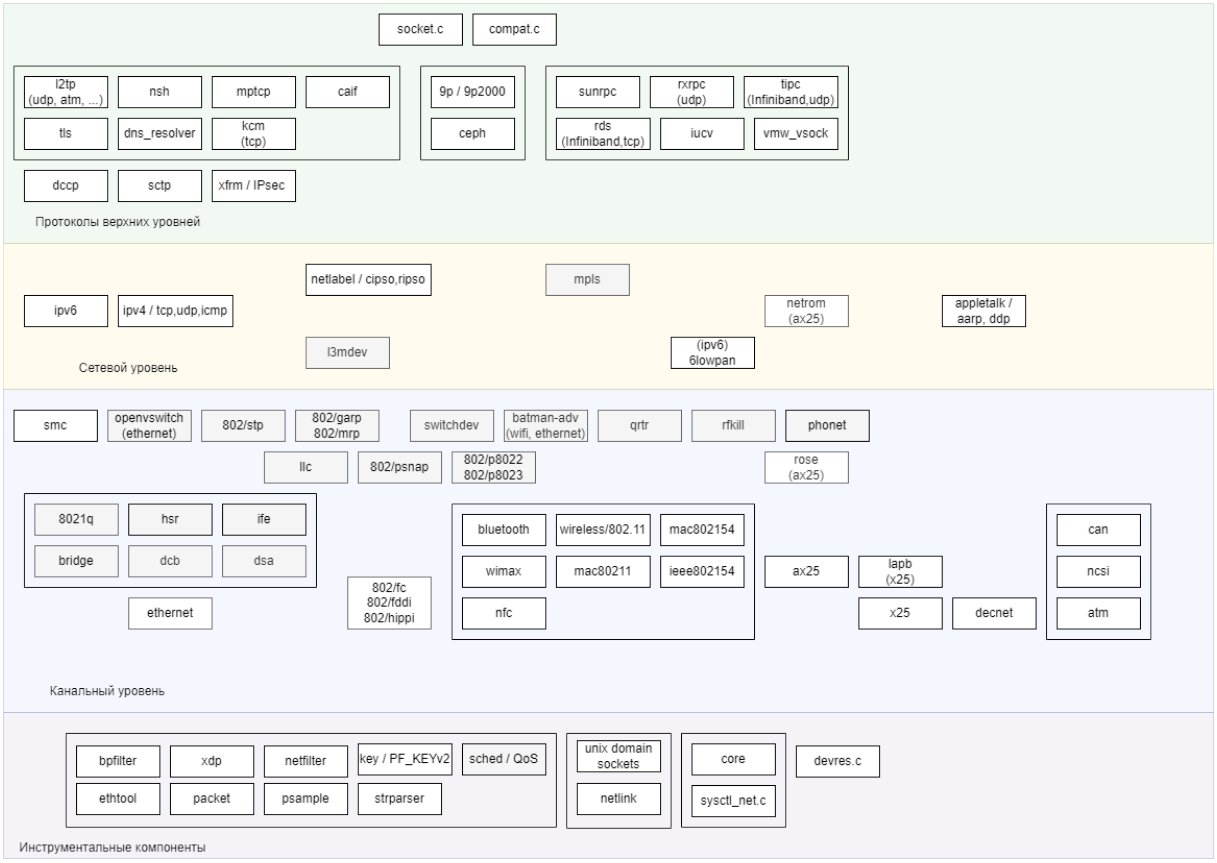

Рис. 1. Сетевая подсистема ядра Linux

Для дальнейшего расширения возможностей фаззера syzkaller был выбран модуль OpenvSwitch, покрытие кода для которого остается незначительным.

OpenvSwitch – широко используемая программная реализация многоуровневого Ethernet коммутатора, применяется для управления трафиком в системах виртуализации. Помимо поддержки различных функций аппаратных коммутаторов OpenvSwitch использует дополнительные программируемые расширения и управление сетью на основе потоков. Дополнительные компоненты обеспечивают возможность преобразования файлов конфигурации различных форматов в правила обработки пакетов. Имеется поддержка нескольких типов туннелирования: GRE, VXLAN, Geneve. Модуль может использоваться для реализации простого коммутатора Ethernet, соединения сетевых устройств, обработки VLAN, управления доступом к сети, управления сетью на основе потоков и др.

OpenvSwitch реализует множество путей “datapaths” (“путь передачи данных”, аналог моста / bridge), каждый из которых может иметь несколько виртуальных портов “vports” (аналог портов внутри моста). К каждому пути (datapath) привязана собственная “таблица потоков” (flow table), которую заполняет пользователь (userspace). Каждый поток (flow)

сопоставляет ключи (на основе заголовков пакетов и метаданных) с наборами действий (actions). В модуле реализовано множество действий (actions), наиболее распространенное – перенаправление пакета на другой vport.

Реализация OpenvSwitch имеет хорошо проработанный пользовательский интерфейс. Однако непосредственно модуль ядра взаимодействует с внешним окружением по протоколу netlink, схема обмена сообщениями которого не документирована. Именно этот интерфейс использует syzkaller, что создает дополнительные сложности при описании тестов.

6. Тестовый стенд

Тестовый стенд состоит из несколько хостов, на каждом из которых работает syzkaller. Хосты объединены сетью в кластер и могут обмениваться информацией. Целевая система представляет собой образ операционной системы с ядром Linux заданной версии и минимальным окружением, необходимым для работы ядра и взаимодействия с ним. Ядро сконфигурировано со всеми опциями, требуемыми для тестирования. Данный образ операционной системы загружается в виртуальной машине под управлением эмулятора QEMU [10]. Таким образом, syzkaller на тестовом хосте запускает одну или несколько виртуальных машин с образом ядра Linux, подключается к ним по протоколу ssh, выполняет внутри них тестовые программы, собирает данные о покрытии кода и информацию об ошибках, дополнительно может обмениваться полученными данными с другими экземплярами syzkaller на хостах кластера. Количество виртуальных машин на каждом хосте определяется вычислительными возможностями этого хоста.

Ветка ядра, поддерживаемая Технологическим центром, формируется на основе одной из стабильных версий ядра. В настоящий момент, по результатам опроса заинтересованных компаний, в качестве базовой ветки ядра выбрана версия 5.10, которая запланирована к поддержке международным сообществом разработчиков до 2026 года.

7. Результаты

В рамках исследования безопасности модуля OpenvSwitch сетевой подсистемы ядра Linux проведен анализ поверхности атаки этого модуля [11].

Выполнено частичное описание интерфейса взаимодействия сданным модулем для инструмента syzkaller.

Найдена одна ошибка, принятая в основную ветку ядра [12].

Как отмечалось выше, отсутствие описания информационного обмена по протоколу netlink для данного модуля, создает трудности при описании тестовых воздействий, что, несомненно, сказывается на результативности работы фаззера.

8. Заключение

В данной работе представлен опыт исследования безопасности модуля OpenvSwitch сетевой подсистемы ядра Linux.

Сегодня Linux является одной из самых популярных операционных систем в мире. Несомненно, качество кода этой ОС (и в частности его ядра) непосредственно влияет на надежность и безопасность всего спектра продуктов на его основе. Международное сообщество разработчиков Linux прилагает огромные усилия для повышения его надежности, используя всевозможные методы тестирования и поиска ошибок. Однако количество обнаруживаемых ежегодно новых ошибок и количество неисправленных старых ошибок показывает, что этих усилий явно недостаточно. Поэтому любой дополнительный вклад в эту общую работу является актуальным.

В работе использованы методы архитектурного анализа и фаззинг-тестирования указанного модуля ядра. Для фаззинг-тестирования использовался инструмент syzkaller с открытым исходным кодом от компании Google.

Найдена одна ошибка, принятая в основную ветку ядра.

Разработка тестового набора и тестирование продолжается.

Работа выполнена в рамках научных работ Технологического центра исследования безопасности ядра Linux. Следует отметить, что главной задачей центра является повышение уровня безопасности отечественных Linux-систем, поэтому основная цель работ не столько в непосредственном нахождение ошибок, сколько в создании комплекса сертифицированных испытаний, обеспечивающих, насколько это возможно, отсутствие каких-либо ошибок в коде ядра Linux.

Технологический центр исследования безопасности ядра Linux создан на базе ИСП РАН под эгидой ФСТЭК России в целях реализации федерального проекта «Информационная безопасность» национальной программы «Цифровая экономика Российской Федерации» в 2022 году.

Литература

1. Технологический центр исследования безопасности ядра Linux. — ИСП им. В.П. Иванникова РАН. — https://portal.linuxtesting.ru
2. New Horizons for Open Source: 2021 linux foundation annual report. — The Linux Foundation, 2021. —

https://www.linuxfoundation.org/resources/publications/linux-foundation-annual-report-2021/
3. Intel Linux Kernel Performance tests — https://github.com/intel/lkp-tests
4. https://syzkaller.appspot.com/upstream
5. Kees Cook. Linux Kernel Security Done Right. August 3, 2021 — https://security.googleblog.com/2021/08/linux-kernel-security-done-right.html
6. B. P. Miller, L. Fredriksen, and B. So. An empirical study of the reliability of UNIX utilities // Commun. ACM, vol. 33, pp. 32–44, December 1990.
7. Syzkaller — https://github.com/google/syzkaller
8. Dmitry Vyukov. Syzbot and the tale of thousand kernel bugs. — Linux Security Summit, 2018. — http://events19.linuxfoundation.org/wp-content/uploads/2017/11/Syzbot-and-the-Tale-of-Thousand-Kernel-Bugs-Dmitry-Vyukov-Google.pdf
9. Сетевая подсистема ядра Linux версии 5.10. — https://portal.linuxtesting.ru/LVCArch.html#main
10. QEMU — https://www.qemu.org/
11. Модуль коммутатора Open vSwitch. — https://portal.linuxtesting.ru/net-openvswitch.html
12. https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/commit/?id=0c598aed445eb45b0ee7ba405f7ece99ee349c30

References

1. Verification center of the operating system Linux. — ISP RAS. — https://portal.linuxtesting.ru
2. New Horizons for Open Source: 2021 linux foundation annual report. — The Linux Foundation, 2021. — https://www.linuxfoundation.org/resources/publications/linux-foundation-annual-report-2021/
3. Intel Linux Kernel Performance tests — https://github.com/intel/lkp-tests
4. https://syzkaller.appspot.com/upstream
5. Kees Cook. Linux Kernel Security Done Right. August 3, 2021 — https://security.googleblog.com/2021/08/linux-kernel-security-done-right.html
6. B. P. Miller, L. Fredriksen, and B. So. An empirical study of the reliability of UNIX utilities // Commun. ACM, vol. 33, pp. 32–44, December 1990.
7. Syzkaller — https://github.com/google/syzkaller
8. Dmitry Vyukov. Syzbot and the tale of thousand kernel bugs. — Linux Security Summit, 2018. — http://events19.linuxfoundation.org/wp-content/uploads/2017/11/Syzbot-and-the-Tale-of-Thousand-Kernel-Bugs-Dmitry-Vyukov-Google.pdf
9. Linux kernel 5.10 network subsystem — https://portal.linuxtesting.ru/LVCArch.html#main

10. QEMU — https://www.qemu.org/
11. OpenvSwitch Linux module. — https://portal.linuxtesting.ru/net-openvswitch.html
12. https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/commit/?id=0c598aed445eb45b0ee7ba405f7ece99ee349c30