DOI: 10.15514/ISPRAS-2026-38(4)-20

CC BY

# Сравнительная количественная оценка механизмов усиления защищенности ядер операционных систем

*Д.В. Ефремов, ORCID: 0000-0002-9916-056X <efremov@ispras.ru>*

*Институт системного программирования им. В.П. Иванникова РАН,*
*Россия, 109004, г. Москва, ул. А. Солженицына, д. 25.*

**Аннотация.** В работе предложена методика количественного сопоставления механизмов усиления защищенности ядер монолитных и микроядерных систем, а также встраиваемых операционных систем (ОС). Существующие руководства и инструменты обычно предназначены для отдельных операционных систем и позволяют проверить главным образом наличие механизмов и их включение; единая методика количественного сопоставления разнородных ядер отсутствует. Сформирована таксономия, включающая 76 механизмов, объединенных в семь категорий. Каждый механизм оценивается по пяти измерениям: наличию и применимости, использованию по умолчанию, стойкости реализации, степени архитектурной интеграции и зрелости. На основе частных оценок вычисляется интегральная оценка с учетом настраиваемых профилей весов измерений и весов категорий, определяемых моделью угроз. Интегральная оценка дополняется показателями охвата таксономии и качества реализованных механизмов, разделяющими широту и глубину защиты. Методика применена к девяти системам, представляющим три аналитические группы: Linux, OpenBSD, NetBSD, Fuchsia (Zircon), GNU Hurd (GNU Mach), seL4, Redox, Tock и Zephyr. Результаты показывают, что разброс оценок внутри одного архитектурного класса может превышать различия между классами. Следовательно, интегральная оценка зависит не только от архитектуры ядра, но и от полноты реализации механизмов, их использования по умолчанию, стойкости, зрелости и активности сопровождения. Полный набор данных с результатами оценки размещен в открытом доступе.

**Ключевые слова:** усиление защищенности; ядро операционной системы; количественная оценка; метрика защищенности; микроядро; монолитное ядро; встраиваемые системы; модель угроз; защита памяти; контроль целостности потока управления; архитектурная изоляция; зрелость механизмов защиты.

**Для цитирования:** Ефремов Д.В. Сравнительная количественная оценка механизмов усиления защищенности ядер операционных систем. Труды ИСП РАН, 2026, том 38, вып. 4, часть 2, стр. 81–108. DOI: 10.15514/ISPRAS-2026-38(4)-20.





# Comparative Quantitative Evaluation of Kernel Hardening Mechanisms in Operating Systems

*D.V. Efremov, ORCID: 0000-0002-9916-056X <efremov@ispras.ru>*

*Ivannikov Institute for System Programming of the Russian Academy of Sciences,*
*25, Alexander Solzhenitsyn st., Moscow, 109004, Russia.*

**Abstract.** This paper presents a methodology for quantitatively comparing hardening mechanisms in operating system kernels across monolithic, microkernel-based, and embedded systems. Existing guidance and tools are generally limited to individual operating systems and mostly verify whether mechanisms are present or enabled; no unified methodology is available for quantitatively comparing heterogeneous kernels. We develop a taxonomy of 76 mechanisms organized into seven categories. Each mechanism is scored along five dimensions: presence and applicability, default posture, implementation strength, depth of integration into the kernel design, and maturity. Individual scores are aggregated into a composite score using configurable profiles of dimension weights and threat-model-dependent category weights. The composite score is complemented by coverage and implemented-quality metrics that separate breadth from depth of protection. The methodology is applied to nine systems representing three analytical groups: Linux, OpenBSD, NetBSD, Fuchsia (Zircon), GNU Hurd (GNU Mach), seL4, Redox, Tock, and Zephyr. The results show that variation within an architectural class may exceed the differences between classes. Thus, the composite score depends not only on kernel architecture but also on completeness of implementation, default posture, implementation strength, maturity, and maintenance activity. The complete evaluation dataset is made publicly available.

**Keywords:** security hardening; operating system kernel; quantitative evaluation; security metric; microkernel; monolithic kernel; embedded systems; threat model; memory protection; control-flow integrity; architectural isolation; hardening mechanism maturity.

**For citation:** Efremov D.V. Comparative Quantitative Evaluation of Kernel Hardening Mechanisms in Operating Systems. Trudy ISP RAN/Proc. ISP RAS, 2026, vol. 38, issue 4, part 2, pp. 81-108 (in Russian). DOI: 10.15514/ISPRAS-2026-38(4)-20.

## *1. Введение*

Для усиления защищенности ядра операционной системы (ОС) применяют как отдельные механизмы, так и архитектурные решения. В монолитных и микроядерных системах, а также во встраиваемых ОС привилегированный код распределен по-разному; различаются и способы изоляции компонентов, и аппаратные средства защиты. Одни и те же задачи защиты в них решаются разными способами. Само по себе наличие механизма не позволяет судить о том, используется ли он по умолчанию, насколько трудно его обойти и насколько тесно он связан с архитектурой системы. Для корректного сравнения разнородных ядер нужна единая система понятий и критериев, учитывающая эти различия.

Актуальность этой задачи обусловлена тем, что значительная часть уязвимостей системного программного обеспечения связана с ошибками при работе с памятью [1-2]. Перенос существующего кода ядер на языки с безопасной работой с памятью требует времени и ресурсов. Поэтому по-прежнему важны механизмы, затрудняющие эксплуатацию ошибок и ограничивающие последствия компрометации. В то же время принципы обеспечения защищенности на этапе проектирования и курс на создание программного обеспечения, защищенность которого поддается измерению, делают систематическую оценку таких механизмов еще более необходимой [3-4].

Отношение к таким механизмам менялось поэтапно. Пионерами их систематического внедрения стали проект PaX и операционная система OpenBSD, которая раньше других систем включила средства защиты в стандартную конфигурацию [5-6]. Примерно до 2014 года защиты развивались преимущественно усилиями энтузиастов, после чего эта деятельность институционализировалась. В 2014 году компания Google сформировала команду Project Zero, поставившую поиск уязвимостей и приемов эксплуатации на

систематическую основу [7], а в 2015 году создан Kernel Self-Protection Project, координирующий разработку механизмов самозащиты в основной ветви Linux [8]. Атаки на спекулятивное исполнение с 2018 года расширили предметную область [9], а с начала 2020-х годов требования к механизмам самозащиты входят в предписывающие документы [10, 1]. Количественно эта динамика прослеживается на хронологии внедрения механизмов в оцениваемых системах, приведенной в разделе 6.5.

Строгого определения усиления защищенности ядра в литературе и нормативных документах не сложилось: в разных источниках под этим термином понимаются различные по объему совокупности мер. В настоящей работе к механизмам усиления защищенности ядра отнесены архитектурные решения, приемы реализации, параметры сборки и настройки системы, которые сокращают поверхность атаки, затрудняют эксплуатацию ошибок, защищают целостность кода и данных, регулируют расширяемость ядра привилегированным кодом и ограничивают последствия компрометации – автоматически и с приемлемыми потерями производительности. Рассматриваются механизмы самозащиты ядра – те механизмы усиления защищенности, которые могут применяться в штатных конфигурациях и не требуют детальной настройки политик для отдельных приложений. За пределами исследования остаются отладочные средства и меры защиты, реализуемые вне ядра. Эти ограничения существенны: средства обнаружения ошибок памяти обычно не включаются в промышленной эксплуатации из-за высоких накладных расходов, а в решениях, требующих трудоемкой настройки политик, объектом атаки становится сама конфигурация. За рамками исследования остается и криптографическое усиление защищенности: выбор и стойкость криптографических алгоритмов, а также реализация криптографических подсистем ядра не оцениваются, а криптографические примитивы учитываются лишь в составе механизмов целостности и цепочки доверия. Подробный обзор механизмов приведен в [11].

Существующие руководства и специализированные анализаторы преимущественно отвечают на вопрос, имеется ли заданный механизм в конкретной ОС и включен ли он [12, 13, 10, 8, 14]. По результатам такой проверки нельзя судить о стойкости, степени архитектурной интеграции и зрелости механизма, а разнородные показатели не сводятся к единой сопоставимой оценке. При сравнении ядер необходимо различать две ситуации: либо механизм отсутствует при наличии соответствующей поверхности атаки, либо данный класс атак неприменим к системе. Наконец, формальная верификация и языки с безопасной работой с памятью дают строгие гарантии относительно отдельных свойств, но не позволяют построить единую шкалу для систем, в которых разные подходы применяются частично или совместно.

Цель работы состоит в разработке воспроизводимой и не зависящей от архитектуры методики количественного сопоставления механизмов усиления защищенности ядер ОС. Для достижения этой цели поставлены три исследовательских вопроса:

1) Как построить не зависящую от архитектуры таксономию механизмов усиления защищенности для монолитных и микроядерных систем, а также для встраиваемых ОС?
2) Как оценивать реализацию этих механизмов с учетом не только их наличия, но и использования по умолчанию, стойкости, архитектурной интеграции и зрелости, а затем объединять частные оценки в интегральную оценку с учетом модели угроз?
3) Насколько результаты сопоставления зависят от архитектурного класса и насколько – от полноты реализации механизмов и качества их сопровождения?

Чтобы ответить на эти вопросы, сформирована таксономия из 76 механизмов, объединенных в семь категорий. Каждый механизм оценивается по пяти измерениям: наличию и применимости, использованию по умолчанию, стойкости реализации, степени архитектурной интеграции и зрелости. Частные оценки объединяются с помощью настраиваемых профилей весов измерений и весов категорий, определяемых моделью угроз.





Каждую оценку сопровождают обоснование и ссылки на исходный код, документацию и историю изменений. Методика применена к девяти системам: трем монолитным ОС – Linux, OpenBSD и NetBSD; четырем системам на основе микроядра – Fuchsia (Zircon), GNU Hurd (GNU Mach), seL4 и Redox; двум встраиваемым ОС – Tock и Zephyr.

Результаты показывают, что разброс оценок внутри одного архитектурного класса может превышать различия между классами. Архитектура влияет на доступные способы изоляции и размер доверенной кодовой базы, но сама по себе не определяет интегральную оценку. Интегральная оценка зависит также от полноты реализации механизмов защиты, их использования по умолчанию, стойкости и зрелости, а также от того, насколько активно они сопровождаются. Предлагаемая методика не дает абсолютной оценки защищенности ОС: она показывает, насколько полно и последовательно в ядре реализованы и сопровождаются механизмы самозащиты в рамках заданной таксономии и модели угроз. Оценочные данные и их обоснования опубликованы в открытом наборе данных [15].

Дальнейшее изложение построено следующим образом. В разделе 2 рассмотрены существующие подходы, в разделе 3 – таксономия механизмов, в разделе 4 – методика оценки, а в разделе 5 – объекты оценки. Результаты и их обсуждение приведены в разделах 6 и 7, угрозы валидности – в разделе 8, заключение – в разделе 9.

## *2. Обзор литературы*

На практике для усиления защищенности чаще всего используют контрольные списки и руководства по настройке. Руководства Центра интернет-безопасности [12] и Агентства оборонных информационных систем США [13] предписывают конкретные настройки для отдельных ОС; профили защиты по Общим критериям для операционных систем общего назначения [10] задают формализованные качественные требования к функциям безопасности. В экосистеме Linux рекомендации проекта самозащиты ядра [8] и автоматический анализатор конфигурации kernel-hardening-checker [14] позволяют проверить включение отдельных механизмов. Эти инструменты позволяют установить, реализованы ли отдельные механизмы и включены ли они по умолчанию, но не оценивают стойкость реализации, степень архитектурной интеграции и зрелость. Кроме того, они ориентированы на конкретные ОС и не сводят результаты к сопоставимой числовой оценке.

Обобщенная таксономия метрик защищенности систем [16] задает понятийный аппарат для описания измеримых свойств. Однако конкретная процедура оценки механизмов усиления защищенности ядра в этой работе не предложена.

Наиболее близки к предлагаемому подходу количественные модели оценки защищенности ОС. В иерархической модели [17] сотни механизмов распределены по уровням, заданным экспертами. Эта модель предназначена для оценки узла и административных политик и охватывает аутентификацию, управление доступом и криптографию, но не детализирует механизмы противодействия эксплуатации уязвимостей ядра и не учитывает их зрелость и степень архитектурной интеграции. Модель [18] также оценивает защищенность на уровне узла без декомпозиции отдельных механизмов ядра. В исследовании [19] сопоставляется качество подсистем управления доступом различных ОС, однако оно охватывает только одну категорию механизмов. Масштабируемая система оценки механизмов противодействия эксплуатации уязвимостей ядра [20] апробирована только на ядрах Linux и не использует многомерную систему оценивания. В работе [21] формализована метрика поверхности атаки на основе статических графов вызовов, а в исследовании [22] предложена методика генерации и оценки защитных мер на уровне сети. Эти подходы позволяют получить отдельные измеримые характеристики, но не интегральную оценку усиления защищенности разнородных ядер. Кроме того, оценку по опубликованным программам эксплуатации уязвимостей трудно воспроизвести: результат зависит от конкретной реализации программы, которую можно доработать для обхода проверяемого механизма; для многих систем такие

программы не опубликованы. После включения механизма по умолчанию новые атаки строятся в обход него, поэтому число опубликованных программ эксплуатации и записей реестра общеизвестных уязвимостей (CVE) не отражает вклада механизма – такая выборка подвержена систематической ошибке выжившего. Косвенные рыночные индикаторы, например размеры вознаграждений за обнаруженные уязвимости, в значительной степени определяются бюджетами организаций и потому могут быть несопоставимы между проектами, поддерживаемыми компаниями, и проектами, развиваемыми сообществом.

Отдельная группа работ посвящена архитектурной изоляции и разделению ядра на изолированные домены. В обзорах [23-24] выделяются два основных назначения изоляции: «песочница» защищает ядро от недоверенного компонента, а защищенный домен – критически важный компонент от остальной части ядра. Работа BULKHEAD [25] демонстрирует масштабируемую внутриядерную изоляцию с использованием ключей защиты. Выводы этих исследований учтены при формировании категории архитектурной изоляции в предлагаемой таксономии, но предложенные в них подходы не решают задачу количественного сопоставления ОС в целом.

Формальная верификация обеспечивает строгие гарантии относительно доказанных свойств в пределах допущений выбранной модели. Формальное доказательство корректности микроядра seL4 [26] учитывается при определении стойкости соответствующих механизмов изоляции в системах семейства L4.

Оценка стойкости отдельных механизмов опирается на исследования способов их обхода. При ранжировании реализаций используются как эмпирические, так и формальные результаты. В частности, учитываются исследования практических способов обхода рандомизации размещения адресного пространства ядра с помощью микроархитектурных каналов [27], реализаций контроля целостности потока управления на основе аутентификации указателей ARM [28] и защиты памяти ядра Linux с помощью формальных методов [29].

В исследовании [30] систематически сопоставлены накладные расходы и энергопотребление исправлений уязвимостей и защит от аппаратных уязвимостей ядра Linux. Полученные данные используются для отдельного анализа издержек применения механизмов.

Обзоры механизмов усиления защищенности [11, 31] и обзоры угроз мобильных платформ [32-33] описывают отдельные механизмы защиты и классы атак. При построении таксономии прежде всего использовался обзор [11], в котором подробно описаны механизмы, оцениваемые в настоящей работе количественно.

Существующие подходы либо качественно оценивают одну ОС, либо количественно измеряют отдельные свойства, включая поверхность атаки, качество управления доступом и накладные расходы, либо систематизируют отдельные классы механизмов. Ни один из них не предлагает единой количественной методики, которая не зависела бы от архитектуры, была применима к ядрам монолитных, микроядерных и встраиваемых ОС и позволяла свести разнородные измерения к сопоставимой интегральной оценке. Именно такая методика предлагается в настоящей работе.

## *3. Таксономия механизмов усиления защищенности*

Таксономия сформирована на основе анализа исходного кода ядер, конфигураций сборки, официальной документации, открытых обсуждений разработчиков и исследований способов обхода защит. Единицей таксономии служит не конкретная настройка отдельной ОС, а обобщенное защитное свойство. Например, запрет исполнения кода из пользовательских страниц в режиме ядра может обеспечиваться предотвращением исполнения в режиме супервизора (SMEP) на процессорах x86 или запретом привилегированного исполнения (PXN) на процессорах ARM, но в обоих случаях оценивается одно свойство. Благодаря этому





реализации для разных архитектур можно сопоставлять по назначению, а не по совпадению названий.

Таксономия имеет три уровня: семь функциональных категорий C1–C7, подкатегории по классам защищаемых свойств и отдельные механизмы. Всего в нее включено 76 механизмов. Границы между отдельными механизмами отчасти определяются выбранной степенью детализации: аппаратную и программную реализации одного свойства можно объединить либо рассматривать отдельно, если различаются их гарантии и применимость. Чтобы обеспечить воспроизводимость, перечень и идентификаторы механизмов зафиксированы в машиночитаемом файле таксономии. Все указанные далее количества относятся именно к этому перечню.

**C1 – защита от повреждения памяти** – наиболее крупная категория, включающая 26 механизмов в шести подкатегориях. Пространственная безопасность памяти охватывает обнаружение переполнения стека, сторожевые страницы вокруг стека и кучи, защищенное копирование между ядром и пользовательским пространством, проверки размеров буферов и аппаратное тегирование памяти. Временная безопасность памяти включает противодействие использованию памяти после освобождения, защиту аллокатора памяти, защиту счетчиков ссылок и связных списков. Отдельно учитываются безопасная инициализация памяти, рандомизация размещения стека, кучи и структур, защита отображений и метаданных (принцип «запись либо выполнение» (W^X), неизменяемые области, SMEP/PXN, память только для исполнения), а также использование языков с безопасной работой с памятью.

**C2 – контроль целостности потока управления** – объединяет восемь механизмов. Для защиты прямых переходов используются программный контроль целостности потока управления (CFI), отслеживание косвенных переходов (IBT) и идентификация цели перехода (BTI); для защиты возвратов – теневой стек, защита адресов возврата (RETGUARD), технология контроля целостности потока управления (CET), код аутентификации указателей (PAC) и защищенный стек управления (GCS). Отдельно оценивается защита динамической компиляции «на лету» (JIT), например в расширенном фильтре пакетов Беркли (eBPF).

**C3 – предотвращение утечек информации** – содержит девять механизмов. В категорию входят рандомизация базового адреса ядра (KASLR) и функций ядра, изоляция таблиц страниц ядра (KPTI), ограничение доступа к пользовательским отображениям из привилегированного кода, очистка освобожденной памяти, стека и регистров, а также ограничение раскрытия адресов и диагностических данных.

**C4 – разграничение привилегий** – включает шесть механизмов, описывающих границу между субъектами и ресурсами. К ним отнесены модель объектных возможностей, сокращение поверхности системных вызовов, ограничения загрузки модулей и расширений, изоляционные домены для процессов и ресурсов, защита операций с учетными данными и связями между процессами, а также контроль точек, из которых разрешены системные вызовы. Последний механизм запрещает выполнять системные вызовы из произвольных участков кода и реализуется, например, привязкой вызовов к доверенным точкам входа или требованием выполнять их только через виртуальный динамический разделяемый объект.

**C5 – архитектурная изоляция** – состоит из 13 механизмов в пяти подкатегориях. Различаются «песочница», защищающая ядро от недоверенного драйвера, модуля или внутриядерной виртуальной машины, и защищенный домен, обеспечивающий сохранность доверенного компонента или ключевого материала после частичной компрометации ядра [23-24]. Остальные подкатегории охватывают изоляцию прямого доступа к памяти (DMA) и периферии, внутриядерные домены памяти, разделение служб и ограничение распространения компрометации между компонентами. Для встраиваемых систем и систем с компонентами разных уровней критичности добавлена подкатегория C5e: обнаружение и локализация сбоев, временная изоляция и квотирование ресурсов, направленные на обеспечение устойчивости к отказу в обслуживании.

**C6 – целостность и цепочка доверия** – объединяет пять механизмов: доверенную загрузку, измеряемую загрузку и удаленную аттестацию, обязательную проверку цифровой подписи кода ядра, контроль целостности во время исполнения и проверку целостности исполняемых файлов и иных объектов. Категория характеризует непрерывность цепочки доверия от загрузочной среды до работающего ядра и загружаемых им компонентов.

**C7 – защита от атак на спекулятивное исполнение** – включает девять механизмов защиты от Spectre версий 1, 2 и 4, линейной спекуляции, Meltdown, терминального отказа кэша первого уровня, микроархитектурной выборки данных, асинхронного отката транзакции и утечек между потоками. Подкатегория C7d расширяет исходный перечень защитой от устойчивых каналов утечки по времени: разделением кэша, его очисткой при переключении домена и другими средствами защиты по времени. Это свойство особенно важно для микроядер, где логическая изоляция компонентов сама по себе не устраняет совместное использование микроархитектурного состояния.

Состав категорий приведен в табл. 1. Указанные классы уязвимостей приведены в качестве примеров: один механизм может противодействовать нескольким классам, а часть архитектурных свойств не имеет точного соответствия одной позиции классификации типовых недостатков программного обеспечения (CWE).

*Табл. 1. Категории механизмов усиления защищенности ядра.*
*Table 1. Categories of kernel hardening mechanisms.*

| Категория | Краткое название | Количество | Примеры механизмов | Основные классы |
|---|---|---|---|---|
| C1 | Повреждение памяти | 26 | стековые «канарейки», сторожевые страницы, защита аллокатора, W^X | CWE-120, CWE-121, CWE-122, CWE-416 |
| C2 | Поток управления | 8 | CFI, теневой стек, PAC, защита JIT | CWE-843, цепочки возвратов и переходов |
| C3 | Утечки информации | 9 | KASLR, KPTI, очистка памяти | CWE-200, CWE-226 |
| C4 | Привилегии | 6 | объектные возможности, фильтрация системных вызовов | CWE-269, CWE-94 |
| C5 | Архитектурная изоляция | 13 | изоляция драйверов и DMA, домены, квоты | CWE-250, CWE-284, CWE-400, CWE-829 |
| C6 | Цепочка доверия | 5 | загрузка, подпись кода, контроль объектов | CWE-494, CWE-829 |
| C7 | Спекулятивные атаки | 9 | Spectre, Meltdown, очистка состояния, временная изоляция | CVE-2017-5753/5715/5754, CWE-385 |

Не каждый механизм применим к любой системе. При отсутствии динамического выделения памяти неприменима часть механизмов защиты кучи; при отсутствии внутриядерной динамической компиляции JIT не требуется ее дополнительная защита; аппаратные





механизмы, включая расширение тегирования памяти (MTE), PAC, BTI и CET, применимы только на платформах, где они поддерживаются. Во всех перечисленных случаях механизм неприменим; это не следует отождествлять с отсутствием необходимой защиты при наличии соответствующей поверхности атаки. Различие между неприменимостью и отсутствием реализации учитывается в измерении наличия и применимости $P$, описанном в разделе 4.2.

## *4. Методика количественной оценки*

### 4.1 Единица оценки и источники данных

Для каждого механизма $M$ из фиксированной таксономии отдельно оценивается его реализация в зафиксированной версии и конфигурации системы. Объект сравнения задается не только названием ОС, но и версией ядра, конфигурацией по умолчанию и аппаратной платформой. Это существенно для настраиваемых механизмов и аппаратно-зависимых средств защиты: режим использования механизмов в одном и том же ядре Linux различается при выборе стандартной конфигурации defconfig, конфигурации дистрибутива или типового образа ядра Android (GKI), а PAC и CET неприменимы на архитектурах без их поддержки.

Первичные данные извлекаются из исходного кода и системы сборки ядра, стандартных конфигураций, официальной документации, истории изменений, списков рассылки и публикаций об устройстве механизмов и способах их обхода. Для каждого сочетания «механизм–ОС» набор данных содержит состояние оценки, значения $P/D/S/A/T$, описание реализации, сведения об использовании по умолчанию, параметры сборки, пути к исходным файлам, записи истории изменений, связанные с механизмом, ссылки на внешние источники и текстовое обоснование. Такая структура позволяет отделить выставленный балл от материалов, на основании которых он получен.

В наборе данных состояние `scored` присваивается записи с самостоятельной оценкой, `na` – записи о неприменимом механизме, а `xref` – записи, ссылающейся на уже учтенную реализацию.

### 4.2 Измерения

Вместо непосредственной оценки вероятности успешной эксплуатации уязвимости при наличии того или иного механизма используются пять измерений. Даже исследователь, не являющийся экспертом в области эксплуатации уязвимостей, может воспроизводимо проверить их значения по открытым источникам и критериям, приведенным в табл. 2, без отдельного изучения каждого из 76 механизмов. Эти измерения характеризуют не абсолютный уровень защищенности системы, а полноту реализации, использование в стандартной конфигурации и качество инженерного сопровождения предусмотренных таксономией средств защиты. Обозначение N/A означает «неприменимо».

**Наличие и применимость $P$.** Значение N/A присваивается, если в оцениваемой конфигурации отсутствует сама поверхность атаки: например, в ядре нет динамически распределяемой кучи, загружаемых модулей или внутриядерного JIT, либо процессор не подвержен рассматриваемому классу атак. Значение N/A присваивается и в том случае, когда аппаратная платформа не предоставляет расширений, необходимых для реализации механизма, например ARM MTE или PAC на x86-64; правила учета таких случаев приведены в разделе 4.6. Значение $P$=0 означает, что поверхность атаки имеется, но защита не реализована; $P$=1 – что имеется хотя бы один поддерживаемый вариант реализации. Для разграничения этих случаев применяется следующее правило: если отсутствие подсистемы само по себе устраняет соответствующую поверхность атаки, механизму можно присвоить N/A; если же подсистема выполняет необходимую функцию, но не защищена, выставляется $P$=0.

*Табл. 2. Измерения оценки отдельного механизма.*
*Table 2. Per-mechanism assessment dimensions.*

| Измерение | Шкала | Критерий |
|---|---|---|
| $P$, наличие и применимость | N/A, 0, 1 | имеется ли поверхность атаки и реализован ли механизм |
| $D$, использование по умолчанию | 0-3 | недоступен, требует включения, включен по умолчанию или обязателен |
| $S$, стойкость | 0-3 | насколько трудно обойти механизм на практике |
| $A$, степень архитектурной интеграции | 0-3 | внешний набор изменений, опция основной ветви или неотделимое свойство архитектуры |
| $T$, зрелость | 0-3 | минимум из возраста механизма и активности его сопровождения |

**Использование по умолчанию** $D$**.** $D$=0 присваивается недоступному механизму; $D$=1 – механизму, который необходимо явно включить при сборке, загрузке или во время исполнения; $D$=2 – механизму, включенному в выбранную стандартную конфигурацию; $D$=3 – обязательному свойству, которое нельзя отключить без изменения архитектуры или доверенной сборки. Например, механизм контроля целостности потока управления ядра kCFI, доступный, но не включенный в базовую конфигурацию, получает $D$=1, а механизм W^X в NetBSD, не отключаемый ни опцией сборки, ни настройкой `sysctl`, ни параметром загрузки, – $D$=3. Тот же механизм W^X в Linux получает лишь $D$=2: документированный параметр загрузки `rodata=off` образует штатный путь его отключения. Таким образом, наличие реализации оценивается отдельно от того, используется ли она в поставляемой конфигурации.

**Стойкость** $S$**.** Значение 1 обозначает защиту, которая повышает сложность атаки, но допускает практический обход при сравнительно небольшом числе дополнительных условий; 2 – защиту, для которой неизвестны тривиальные способы обхода, хотя применимы более сложные методы; 3 – отсутствие известных практических способов обхода либо устранение защищаемого класса дефектов на уровне архитектуры. Значение 0 используется для отсутствующего механизма. При оценке учитываются различия между реализациями, полнота охвата и опубликованные сведения о способах обхода. Так, микроархитектурные методы раскрытия размещения ядра ограничивают стойкость базовой KASLR [27]; для CFI существенны число допустимых целей перехода и устойчивость к подмене указателей [28]; для W^X одной настройки сборки недостаточно без проверки всех путей создания отображений [29]. Если формально доказанное свойство охватывает тот же вектор атаки, оценка стойкости принимается не ниже $S$=2, однако границы применимости формальной модели необходимо анализировать отдельно [26]. В наборе данных фиксируется исходная стойкость $S_{raw}$, оценивающая механизм в отдельности, а в формуле используется эффективная стойкость $S_{eff}$, учитывающая заданные в таксономии жесткие предпосылки механизма. Стойкость механизма не может превышать стойкость самой слабой предпосылки более чем на один уровень: $S_{eff}$=$min(S_{raw}, min_d(S_d+1))$ по предпосылкам $d$, а отсутствующая предпосылка ограничивает эффективную стойкость единицей. Предпосылка со статусом N/A оценку не ограничивает: отсутствие поверхности атаки у самой предпосылки не ослабляет зависящий от нее механизм. Прочие взаимодействия механизмов, включая конфликты между ними, обсуждаются только качественно.

**Степень архитектурной интеграции** $A$**.** Значение 1 получает внешний набор изменений, сопровождаемый отдельно от основной ветви; 2 – механизм, находящийся в основной ветви, но включаемый и отключаемый через конфигурацию; 3 – базовое свойство архитектуры ядра,





например обязательная модель объектных возможностей или вынесение всех драйверов в изолированные процессы. Шкала учитывает не только расположение кода в репозитории, но и обязательность установленной механизмом границы для остальных подсистем. Различие между внешним набором изменений и основной ветвью отражает и владение кодом: включение механизма в основную ветвь означает, что разработчики ядра принимают ответственность за его сопровождение, тогда как сторонний набор изменений, применяемый без доработки, такой ответственности не создает. Поэтому $A$ не совпадает с $D$: механизм может находиться в основной ветви, но быть выключенным по умолчанию.

**Зрелость** $T$**.** Зрелость характеризуют два показателя: возраст $T_{age}$ и активность сопровождения $T_{maintained}$. Оценки 1, 2 и 3 по возрасту соответствуют экспериментальному механизму моложе года, стабильной реализации возрастом от одного года до пяти лет и механизму старше пяти лет с опытом эксплуатации. Чтобы оценить активность сопровождения, анализируют историю изменений за 18 месяцев: отсутствие содержательных изменений соответствует 1, эпизодические исправления – 2, регулярные изменения, тестирование и работа с выявленными дефектами – 3. Итоговое значение равно минимуму двух составляющих. Поэтому давно реализованный, но более не сопровождаемый механизм и активно развиваемая, но еще экспериментальная реализация не получают максимального балла. Шкалы $D$, $A$ и $T$ совместно отражают апробацию механизма опытом эксплуатации: обычно новый механизм добавляется выключенным и включается по умолчанию лишь после того, как его ценность подтверждена, накладные расходы снижены, а реализация прошла длительное тестирование.

## 4.3 Оценка механизма и агрегирование

Представление защищенности в виде набора измеримых свойств согласуется с общей системой понятий для метрик безопасности [16]. Для применимого механизма оценка задается формулой

$$H(M)=P \cdot (w_d D+w_s S+w_a A+w_t T),$$

где все веса измерений неотрицательны, а их сумма равна 1. При $P$=0 оценка равна нулю независимо от остальных измерений; для N/A оценка не вычисляется, а соответствующая запись исключается из дальнейшего усреднения. Максимальное значение $H(M)$ равно 3.

Некоторые защитные свойства пересекаются. Например, одна реализация KPTI одновременно препятствует непосредственному чтению памяти ядра и защищает от Meltdown. Если для конкретной ОС обе записи ссылаются на одну реализацию, вторичная запись помечается `xref` и не учитывается при агрегировании. Если же ОС реализует различимые свойства независимыми средствами, их можно оценивать отдельно; решение фиксируется в обосновании оценки.

Оценка категории $H(C_k)$ равна среднему значению $H(M)$ по самостоятельно оцениваемым механизмам этой категории:

$$H(C_k)=\frac{1}{|M_k|}\sum_{M\in M_k} H\,(M),$$

где $M_k$ – множество самостоятельно оцениваемых применимых механизмов категории $C_k$, а $H(M)$ вычисляется с эффективной стойкостью $S_{eff}$, учитывающей предпосылки механизма (раздел 4.2). Отсутствующий, но применимый механизм с $P$=0 входит в среднее и снижает его; N/A и `xref` исключаются.

Интегральная оценка $H_{total}$ является взвешенной суммой оценок категорий:

$$H_{total}=\sum_{k\in K} W_k' H(C_k), W_k'=\frac{W_k}{\sum_{j\in K} W_j},$$

где $K$ – множество категорий, содержащих хотя бы один самостоятельно оцениваемый применимый механизм. Если в некоторой категории таких механизмов нет, она исключается из $K$, а ее базовый вес $W_k$ пропорционально распределяется между оставшимися категориями. В частности, отсутствие целого класса аппаратных атак не уменьшает оценку ОС, но и не приносит ей дополнительных баллов. Значения $H(C_k)$ и $H_{total}$ лежат в диапазоне от 0 до 3.

Для точной воспроизводимости порядок округления промежуточных и итоговых значений зафиксирован в вычислительной реализации методики. Полученное число не выражает вероятность компрометации и не является сертификатом безопасности: оно характеризует полноту и качество реализации механизмов в заданной конфигурации при фиксированных таксономии и весах.

## 4.4 Профили весов измерений

Веса позволяют явно задать относительную важность использования по умолчанию, стойкости, архитектурной интеграции и зрелости. Используются четыре профиля из табл. 3, обозначенные в наборе данных как `balanced`, `security_first`, `pragmatic` и `architectural`. Основным является сбалансированный профиль; остальные предназначены для анализа чувствительности. Как показывает анализ в разделе 6.2, итоговое ранжирование слабо зависит от выбора весов измерений. Поэтому четыре профиля используются не как конкурирующие точки зрения, а как инструмент проверки чувствительности и сопоставимости с опубликованным набором данных.

*Табл. 3. Профили весов измерений.*
*Table 3. Dimension-weight profiles.*

| Профиль | D | S | A | T | Приоритет |
|---|---|---|---|---|---|
| Сбалансированный (`balanced`) | 0,25 | 0,35 | 0,25 | 0,15 | учет всех свойств с повышенным весом стойкости |
| Приоритет защищенности (`security_first`) | 0,20 | 0,45 | 0,25 | 0,10 | непосредственная эффективность защиты |
| Практический (`pragmatic`) | 0,30 | 0,25 | 0,15 | 0,30 | использование по умолчанию и проверенность временем |
| Архитектурный (`architectural`) | 0,15 | 0,25 | 0,45 | 0,15 | свойства, неотделимые от архитектуры ядра |

## 4.5 Модели угроз и веса категорий

Профили весов измерений не определяют, какие классы атак важнее в конкретном сценарии применения. Для этого отдельно задаются веса категорий $W_k$, приведенные в табл. 4. Сумма весов в каждой строке равна 1; после исключения полностью неприменимых категорий веса заново нормируются для каждой ОС. Один из профилей предназначен для встраиваемых систем и устройств Интернета вещей (IoT).

В модели угроз возможности нарушителя упорядочены по возрастанию: удаленный неаутентифицированный доступ; исполнение локального кода без привилегий; частичные привилегии или отдельные полномочия; управление соседней виртуальной машиной; возможность атак через DMA или отладочный интерфейс при физическом доступе. Компрометация цепочки поставок соответствует следующему уровню возможностей нарушителя, находящемуся за рамками исследования. К целям атак относятся чтение памяти ядра, запись в нее или исполнение размещенного в ней кода, повышение привилегий, закрепление в системе и отказ в обслуживании; к точкам входа – системные вызовы,





драйверы, сетевой стек, файловые системы, внутриядерные виртуальные машины и обращения, инициируемые аппаратурой.

Профиль сервера общего назначения учитывает прежде всего угрозы со стороны удаленного и локального непривилегированного нарушителя, поэтому категориям C1 и C4 присвоены повышенные веса. Профиль для систем с повышенными требованиями к защищенности дополнительно учитывает частично привилегированного нарушителя и нарушителя, управляющего соседней виртуальной машиной, поэтому в нем повышены веса C2 и C5. В профиле для встраиваемых систем физический доступ и необходимость восстановления устройства после сбоя увеличивают веса архитектурной изоляции и цепочки доверия. Выбор весов является параметром сценария, а не оценкой вероятности конкретной атаки; поэтому следует сопоставлять результаты, полученные при использовании нескольких профилей. Формализация поверхностей атаки и их зависимости от конфигурации опирается на подход [21].

*Табл. 4. Веса категорий для моделей угроз.*
*Table 4. Category weights for the threat models.*

| Модель угроз | C1 | C2 | C3 | C4 | C5 | C6 | C7 |
|---|---|---|---|---|---|---|---|
| Сервер общего назначения | 0,20 | 0,15 | 0,15 | 0,20 | 0,10 | 0,10 | 0,10 |
| Повышенные требования к защищенности | 0,20 | 0,20 | 0,15 | 0,15 | 0,15 | 0,10 | 0,05 |
| Встраиваемые системы и IoT | 0,15 | 0,10 | 0,10 | 0,15 | 0,25 | 0,15 | 0,10 |

В отличие от весов измерений, веса категорий действительно влияют на порядок систем, поэтому для их обоснования привлечены три независимых источника статистики уязвимостей. Это распределение классов дефектов (CWE) по 870 записям с проставленным CWE из 3 249 записей CVE ядра Linux за 2024–2026 годы из набора данных Vulnrichment Агентства по кибербезопасности и защите инфраструктуры США [34], перечень CWE Top 25 2025 года [35] и 37 задокументированных случаев эксплуатации ранее неизвестных уязвимостей ядра Linux и драйверов его поставщиков в реальных атаках за 2014–2026 годы [36, 15]. Во всех трех источниках – для перечня CWE Top 25 среди применимых к ядру классов – доминируют ошибки работы с памятью: к категории C1 относятся 32 из 37 уязвимостей, эксплуатировавшихся в реальных атаках; сопоставимый по охвату анализ 1 858 CVE ядра Linux за 2010–2020 годы приведен в [37]. При этом ни один из источников статистики по ядрам не дает оснований для назначения весов категориям C2, C6 и C7: контроль целостности потока управления противодействует технике эксплуатации, а не первопричине уязвимости, цепочка доверия защищает от подмены кода при поставке и физическом доступе, а атаки на спекулятивное исполнение среди рассмотренных задокументированных случаев эксплуатации ядра отсутствуют. Поэтому веса этих трех категорий заданы явным решением, отражающим модель угроз, а не выведены из данных; разделение весов на измеримую и декларируемую части зафиксировано в опубликованном наборе данных [15].

Использованные веса при этом расположены вблизи центра области, внутри которой порядок систем не меняется, а не на ее границе. Отклонение ключевого веса каждой модели угроз на ±0,05 с пропорциональной перенормировкой остальных меняет итоговые оценки не более чем на 0,096 по шкале от 0 до 3 и приводит не более чем к двум перестановкам соседних систем.

## 4.6 Учет архитектурной и аппаратной асимметрии

Для сравнения систем разных классов применяются три правила. Во-первых, если поверхность атаки отсутствует, механизму присваивается N/A: ядру без загружаемых

модулей не требуется механизм их изоляции; это относится и к случаям, когда отсутствие защищаемого класса ошибок формально доказано, – такие записи не снижают оценку системы. Во-вторых, при наличии функционального эквивалента оценивается фактическая реализация общего свойства, даже если ее интерфейс и название отличаются от реализации в другой ОС. В-третьих, свойство, обеспечиваемое самой архитектурой, оценивается по тем же $D/S/A/T$ и не получает дополнительных баллов. Например, обязательное вынесение драйверов за пределы микроядра дает высокую степень архитектурной интеграции, но режим использования этого механизма, его стойкость и зрелость должны быть подтверждены источниками.

Применимость аппаратных механизмов определяется для платформы, выбранной для оцениваемой версии системы. Отсутствие ARM MTE на x86 или Intel CET на Cortex-M означает N/A; наличие подходящего оборудования при отсутствии поддержки в ядре означает $P$=0. Аналогично программная защита от Spectre не требуется для процессора, в котором соответствующая форма спекуляции отсутствует, но на уязвимом процессоре отсутствие такой защиты считается недостатком.

### 4.7 Диагностические показатели и воспроизводимость

Помимо $H_{total}$ вычисляются пять дополнительных диагностических показателей. Коэффициент охвата равен доле самостоятельно оцениваемых применимых механизмов с $P$=1; охват по умолчанию – доле механизмов с $D \geq 2$; охват обязательными механизмами – доле механизмов с $P$=1 и $D$=3. Взвешенный по стойкости охват равен отношению суммы эффективной стойкости $S_{eff}$ реализованных механизмов к максимально возможной сумме по всем применимым механизмам. Качество реализованных механизмов равно среднему значению $H(M)$ по механизмам с $P$=1 и характеризует глубину реализации отдельно от широты охвата таксономии. Значения диагностических показателей для рассмотренных систем приведены в разделе 6.3. Количество записей N/A и `xref` публикуется отдельно, что позволяет определить долю таксономии, исключенную из знаменателя.

Накладные расходы не входят в $H(M)$. Их включение в виде штрафа приводило бы к тому, что система без стойкой, но требующей значительных ресурсов защиты могла бы получить преимущество перед системой, в которой эта защита применяется. В текущей версии набора данных расходы не представлены в виде формализованного измерения.

## *5. Объекты оценки*

Основная выборка включает девять систем, различающихся устройством привилегированной части, моделью изоляции, языком реализации, зрелостью и целевой аппаратной платформой. Это целенаправленно сформированная контрастная выборка, а не статистическая выборка из генеральной совокупности ОС. Ее назначение – проверить, применима ли единая таксономия к системам, в которых одинаковые защитные свойства обеспечиваются настройками, аппаратной изоляцией, языком с безопасной работой с памятью или архитектурным разделением компонентов. Системы с закрытым исходным кодом, в частности Windows, в выборку не включены: они не относятся к целевому семейству рассматриваемых систем и привлекаются лишь как сравнительные ориентиры при анализе отдельных механизмов, поскольку без доступа к исходному коду невозможно воспроизводимо подтвердить реализацию механизмов и полноту раскрытых сведений о них.

Системы условно разделены на три аналитические группы: монолитные ОС общего назначения, системы на основе микроядра и встраиваемые ОС. Эти группы не образуют строгого взаимоисключающего деления по архитектуре: например, Tock и Zephyr отнесены к группе встраиваемых ОС с учетом области применения и ограничений микроконтроллеров, хотя их привилегированный код не организован как классическое микроядро. В частности, ядро Tock сочетает единое адресное пространство привилегированного кода с изоляцией





драйверных капсул средствами системы типов языка программирования Rust и изоляцией процессов средствами блока защиты памяти (MPU), занимая промежуточное положение между монолитными и микроядерными архитектурами. Zephyr представляет собой операционную систему реального времени (ОСРВ). Поэтому в табл. 5 наряду с группой указана конкретная платформа, с учетом которой определяется применимость аппаратных механизмов.

*Табл. 5. Объекты оценки и их базовые конфигурации.*
*Table 5. Evaluated systems and baseline configurations.*

| Система и группа | Версия и базовая конфигурация | Основание для включения |
|---|---|---|
| Linux, монолитная | Fedora 44, ядро 7.1.4-200.fc44, x86-64 | широкий набор настраиваемых механизмов в распространенной ОС общего назначения |
| OpenBSD, монолитная | OpenBSD 7.9, GENERIC, amd64 | последовательное включение средств защиты по умолчанию и собственные реализации механизмов |
| NetBSD, монолитная | NetBSD 11.0, GENERIC, amd64 | сравнение с другой системой семейства BSD и иной охват механизмов |
| Fuchsia/Zircon, микроядерная | Fuchsia F30, стандартная сборка Zircon, x86-64 | модель объектных возможностей, драйверы в пространстве пользователя и активная разработка |
| GNU Hurd/GNU Mach, микроядерная | GNU Mach, ветвь `master`, Debian GNU/Hurd 2025, amd64 | классическое разделение Mach с пользовательскими серверами |
| seL4, микроядерная | seL4 15.0.0, `881de507f`, AArch64, конфигурация для верификации | формальная верификация, модель объектных возможностей и небольшая доверенная вычислительная база |
| Redox OS, микроядерная | Redox OS 0.9.0, ядро 0.5.12 (`4531e15b`), выпуск для x86-64 | сочетание микроядра и реализации на Rust |
| Tock, встраиваемая | Tock 2.2, `9554639b1`, nRF52840, Cortex-M4F с MPU | язык с безопасной работой с памятью, изоляция процессов и отсутствие общей кучи ядра |
| Zephyr, встраиваемая | Zephyr 4.4.1, `1f6485eca`, Cortex-M33, MPU, TrustZone-M, MCUboot | настраиваемая ОСРВ с аппаратными и программными средствами защиты |

В качестве основной конфигурации Linux выбрано ядро Fedora 44, а не конфигурация `defconfig`: последняя предназначена преимущественно для разработки и не отражает настройки ядра, поставляемого пользователям.

OpenBSD выбрана потому, что механизмы усиления защищенности последовательно включаются в основную ветвь разработки и стандартную конфигурацию ядра GENERIC. К характерным реализациям относятся перекомпоновка ядра со случайным расположением объектов при каждой загрузке (KARL), RETGUARD, память, доступная только для

исполнения, неизменяемые отображения и привязка системных вызовов к доверенным точкам входа [38, 39]. NetBSD отличается от остальных систем набором механизмов и их использованием по умолчанию. Сравнение этих ОС позволяет оценить разброс внутри одного архитектурного класса без учета различий, связанных с микроядерной организацией.

Четыре микроядерные системы представляют разные подходы к обеспечению гарантий. Fuchsia/Zircon использует модель объектных возможностей, небольшое ядро и вынесение драйверов в процессы пользовательского пространства. GNU Hurd сочетает GNU Mach с набором пользовательских серверов и позволяет оценить классическую архитектуру Mach на примере зафиксированной версии Debian GNU/Hurd. В seL4 гарантии изоляции обеспечиваются моделью объектных возможностей и формальными доказательствами корректности, проверенными в системе интерактивного доказательства теорем Isabelle/HOL [26]. В качестве базовой выбрана конфигурация AArch64, для которой определен набор верифицируемых свойств. В Redox архитектурное разделение дополняется гарантиями безопасной работы с памятью, которые дает Rust, но система оценивается по тем же отдельным механизмам: выбор языка сам по себе не повышает интегральную оценку.

На примере встраиваемых ОС можно проследить влияние иной аппаратной платформы. Tock использует Rust для ядра и драйверных капсул, изолирует процессы с помощью MPU; общая динамически распределяемая куча в ядре отсутствует. Zephyr преимущественно реализован на языке C и предоставляет широкий набор настраиваемых средств защиты; для воспроизводимости зафиксирована платформа Cortex-M33 с MPU, TrustZone-M, MCUboot и включенной поддержкой пользовательского режима. При сравнении этих систем с ядрами x86-64 необходимо учитывать правила присвоения N/A и различия между MPU микроконтроллера и блоком управления памятью (MMU) процессора общего назначения.

Основные конфигурации систем общего назначения оцениваются преимущественно на x86-64, seL4 – в конфигурации AArch64 с формальной верификацией, а Tock и Zephyr – на микроконтроллерах Cortex-M с MPU. Таким образом, сравнение архитектурных классов не проводится в условиях единой архитектуры набора команд (ISA): измерение $P$ позволяет исключить заведомо неприменимые записи, однако при интерпретации результатов необходимо учитывать аппаратную платформу. Для Linux, NetBSD, Redox и Fuchsia набор данных дополнительно содержит сведения о конфигурациях AArch64; они используются для анализа чувствительности к платформе и не образуют отдельных объектов основной выборки.

## *6. Результаты*

### 6.1 Итоговые оценки

Оценки вычислены по методике раздела 4 для версий и конфигураций, зафиксированных в табл. 5. В качестве основного варианта используются сбалансированный профиль весов измерений и модель угроз сервера общего назначения; остальные профили и модели угроз применяются для анализа чувствительности. Значения оценок категорий $H(C_k)$ и интегральной оценки $H_{total}$ приведены на рис. 1. Прочерк означает, что все механизмы категории неприменимы к системе; в этом случае вес категории перераспределяется между остальными, как описано в разделе 4.3. Диаграммы по категориям и обоснования отдельных оценок доступны в интерактивном представлении опубликованного набора данных [15].

Наибольшие интегральные оценки получили Fuchsia (1,72) и Linux (1,67), за ними следуют OpenBSD (1,46) и Tock (1,35). Средние позиции занимают Redox (1,04) и Zephyr (1,00); наименьшие значения получили seL4 (0,87), NetBSD (0,85) и GNU Hurd (0,60). Даже лидеры достигают лишь около 57% максимума шкалы: ни одна система не реализует и не включает по умолчанию стойкие варианты всех применимых к ней механизмов.





В монолитной группе Linux демонстрирует наиболее широкий и равномерный профиль: высокие оценки защиты от повреждения памяти (1,91), предотвращения утечек информации (1,84), цепочки доверия (2,02) и защиты от атак на спекулятивное исполнение (2,02) сочетаются с минимальным для этой системы значением архитектурной изоляции (0,90), отражающим ограниченное разделение монолитного ядра на изолированные домены. OpenBSD получает наибольшую среди всех систем оценку категории предотвращения утечек информации (1,90) и подкатегории защиты отображений и метаданных памяти (2,34 против 1,84 у Linux), что отражает последовательное включение по умолчанию KARL, памяти, доступной только для исполнения, и неизменяемых отображений. OpenBSD также опережает Linux в категории разграничения привилегий (2,16 против 1,68) благодаря привязке системных вызовов к доверенным точкам входа. Отставание OpenBSD от Linux по интегральной оценке (0,21) складывается из категорий цепочки доверия (0,53 против 2,02) и защиты от атак на спекулятивное исполнение (1,23 против 2,02) и лишь частично компенсируется перечисленными преимуществами. Среднее качество реализованных механизмов у OpenBSD при этом практически совпадает со значением Linux (2,04 против 2,02), то есть разрыв определяется охватом таксономии, а не качеством реализованных средств защиты; стоящие за этими пробелами решения проекта обсуждаются в разделе 7. NetBSD получает низкие значения в большинстве категорий, а контроль целостности потока управления в ее базовой конфигурации отсутствует. В результате разрыв между Linux и NetBSD внутри монолитного класса составляет 0,82, тогда как лидеры монолитной и микроядерной групп практически не различаются: разрыв между Fuchsia и Linux составляет 0,05.

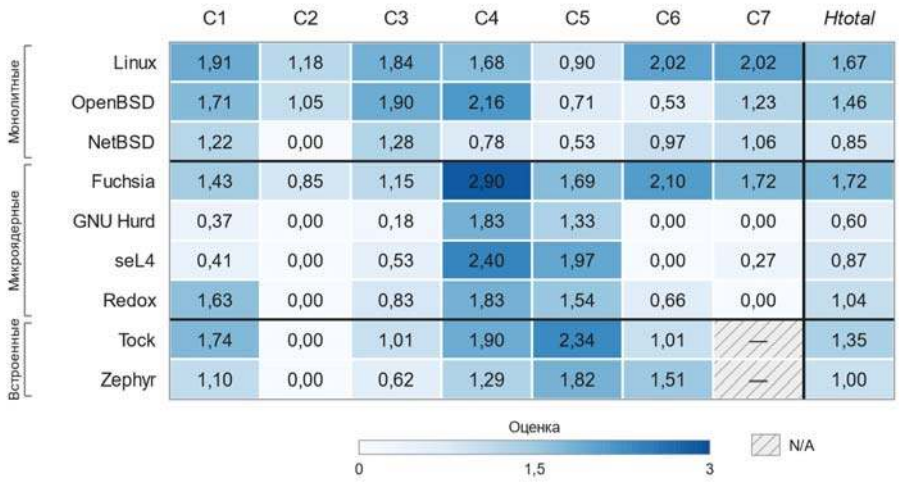


*Рис. 1. Оценки C1–C7 и H_total: сбалансированный профиль, модель угроз сервера общего назначения; цвет кодирует значение от 0 до 3; штриховка – N/A.*
*Fig. 1. Scores C1–C7 and H_total: balanced profile and general-purpose server threat model; color encodes values from 0 to 3 and hatching denotes N/A.*

Профиль микроядерных и встраиваемых систем противоположен по структуре профилю монолитной группы. Обязательная модель объектных возможностей и вынесение драйверов в изолированные домены дают Fuchsia наибольшую в выборке оценку разграничения привилегий (2,90), а seL4 – близкую к ней (2,40); максимальная оценка недостижима, поскольку модель возможностей ограничивает, какие объекты доступны вызову, но не точки

кода, из которых системные вызовы могут выполняться. Наибольшую оценку архитектурной изоляции получает Tock (2,34) благодаря доменам MPU и капсульной изоляции драйверов. Одновременно в этой группе наблюдается системный пробел в категориях контроля целостности потока управления и предотвращения утечек информации: оценка C2 равна нулю у всех систем группы, кроме Fuchsia (0,85), в которой реализованы программные средства защиты прямых переходов и возвратов. Рандомизация базового адреса ядра не реализована ни в одной из систем этой группы, использующих виртуальную память, включая Fuchsia, а для микроконтроллерных Tock и Zephyr она неприменима. Ядра, написанные на языке программирования Rust, частично закрывают категорию повреждения памяти: оценки Tock (1,74) и Redox (1,63) сопоставимы со значением Linux (1,91), а по подкатегории пространственной безопасности памяти Tock (2,63) превосходит все системы выборки. GNU Hurd представляет предельный случай: унаследованное от Mach разграничение привилегий на основе портов (1,83) сочетается с близкими к нулю значениями остальных категорий, кроме архитектурной изоляции (1,33). Схожая асимметрия свойственна seL4: почти максимальное разграничение привилегий, высокая архитектурная изоляция, подкрепленная формальной верификацией [26], и реализация защиты по времени (C7d) сочетаются с узким охватом остальных категорий – реализован лишь 31% применимых механизмов. При этом среднее качество реализованных механизмов у seL4 наивысшее в выборке (2,55); этот случай подробно разбирается в разделе 7.

## 6.2 Анализ чувствительности

Значения $H_{total}$ для всех профилей весов измерений и моделей угроз приведены на рис. 2. Ранжирование по $H_{total}$ устойчиво к выбору профиля весов измерений: при фиксированной модели угроз состав первой тройки и последнее место совпадают во всех четырех профилях, а различия сводятся к перестановкам соседних систем. Во всех двенадцати сценариях последнее место занимает GNU Hurd, тогда как состав первой тройки определяется только моделью угроз: Fuchsia, Linux и OpenBSD в моделях сервера общего назначения и повышенных требований и Fuchsia, Linux и Tock в модели для встраиваемых систем. Минимальная попарная ранговая корреляция Спирмена между ранжированиями двенадцати сценариев – четырех профилей в сочетании с тремя моделями угроз – составляет 0,90 и достигается на паре «практический профиль с моделью встраиваемых систем» и «архитектурный профиль с моделью сервера общего назначения», то есть выводы сравнения не определяются выбором весов измерений. Смена модели угроз, напротив, может менять состав лидеров. В модели сервера общего назначения впереди Fuchsia (1,72) и Linux (1,67), третье место занимает OpenBSD (1,46); в модели повышенных требований этот порядок сохраняется (1,61; 1,58; 1,38). В модели для встраиваемых систем с повышенными весами архитектурной изоляции и цепочки доверия отрыв Fuchsia (1,76) увеличивается, а Tock (1,54) поднимается с четвертого места на третье, отставая от сохранившего второе место Linux (1,57) на 0,03. Одновременно OpenBSD опускается с третьего места на четвертое (1,26), а Zephyr поднимается с шестого на пятое (1,23), опережая Redox (1,09).

Устойчивость ранжирования дополнительно проверена случайным перебором: для каждой проверки сгенерировано по 50 000 допустимых комбинаций весов – неотрицательных наборов с суммой 1. Веса измерений на порядок систем влияют слабо: при переборе всего симплекса ранговая корреляция с базовым ранжированием не опускается ниже 0,90 в 99,8% комбинаций как для модели сервера общего назначения, так и для модели повышенных требований, а ее наименьшее значение составляет 0,87. Точный базовый порядок воспроизводится при этом в 32% и 45% комбинаций соответственно, а остальные различия приходятся на системы с близкими оценками. Веса категорий, напротив, являются действительным рычагом сравнения: произвольная их комбинация воспроизводит точный базовый порядок лишь в 1,8% случаев. В пределах ±0,05 от заданных весов устойчивость заметно выше, но не абсолютна: базовый порядок сохраняется в 46% комбинаций для модели





сервера общего назначения, в 49% – для модели повышенных требований и в 49% – для модели встраиваемых систем, а ранговая корреляция не опускается ниже 0,93 в первых двух моделях и ниже 0,77 – в третьей. Неустойчивость сосредоточена в парах с близкими оценками: при случайном выборе весов категорий взаимный порядок в парах OpenBSD–Tock, NetBSD–seL4 и Linux–Fuchsia меняется почти равновероятно (47–57% комбинаций). В паре Redox–Zephyr смена порядка является не исключением, а правилом: в 70% комбинаций выше оказывается Zephyr, а не Redox, как в базовом сценарии, поэтому положение систем внутри этих пар не следует трактовать как строгий результат. По той же причине не следует ранжировать системы, интегральные оценки которых различаются менее чем на 0,01: такое различие не превышает погрешности округления вычислительной схемы. Так, в практическом профиле с моделью повышенных требований округленные оценки Linux и Fuchsia совпадают (1,64), и различить эти системы в данном сценарии невозможно.

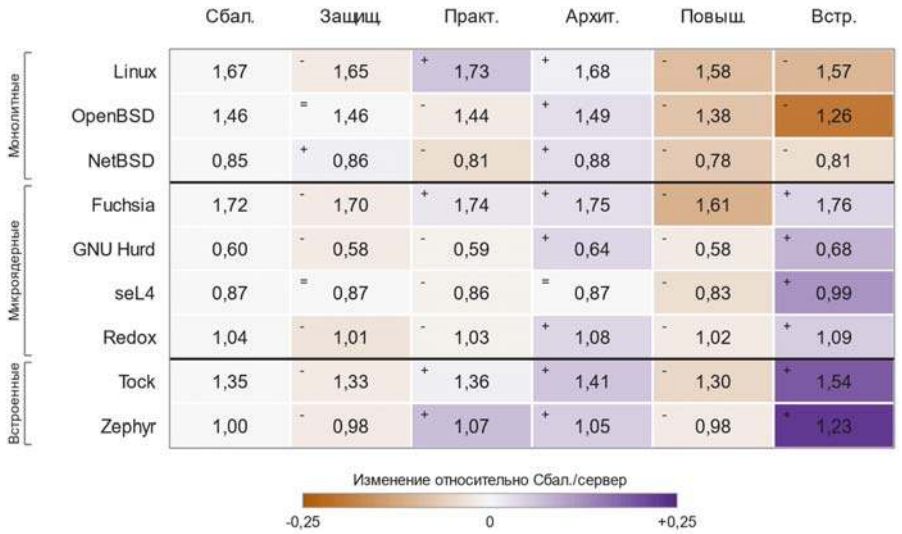


*Рис. 2. Чувствительность H_total: числа показывают оценки, цвет – изменение относительно сбалансированного профиля и модели угроз сервера общего назначения.*
*Fig. 2. Sensitivity of H_total: numbers show scores, while color shows change from the balanced profile and general-purpose server threat model.*

## 6.3 Производные показатели

Диагностические показатели раздела 4.7 для всех систем показаны на рис. 3; дополнительно приведены доля включенных по умолчанию среди реализованных механизмов и доля записей N/A.

Показатели демонстрируют, что охват таксономии и фактическое использование механизмов не совпадают. Коэффициент охвата Zephyr составляет 65%, однако охват по умолчанию равен лишь 31%: в зафиксированной стандартной конфигурации включено менее половины (47%) реализованных механизмов. У Tock при сопоставимом охвате (65%) охват по умолчанию достигает 60%, поскольку включено 92% реализованных механизмов; для Linux эти показатели составляют 83% и 64%. Таким образом, наличие реализации не

предопределяет режим ее применения, что эмпирически подтверждает целесообразность отдельного измерения $D$.


*Рис. 3. Производные показатели, %: охват, охват по умолчанию, доля включенных среди реализованных, взвешенный по стойкости и обязательный охват, доля N/A; длина полосы кодирует процент.*

*Fig. 3. Derived metrics, %: coverage, default coverage, enabled share among implemented mechanisms, strength-weighted and mandatory coverage, and N/A share; bar length encodes the percentage.*

Охват обязательными механизмами лишь слабо связан с коэффициентом охвата (ранговая корреляция −0,15) и образует самостоятельную ось сравнения. У Tock (50%), Redox (35%) и Fuchsia (23%) значительная часть защит гарантируется архитектурой и не может быть отключена, тогда как Linux (3%) достигает большего охвата средствами управляемой конфигурации. OpenBSD занимает промежуточное положение (18%): проект поставляет единственную конфигурацию ядра GENERIC, и для многих защит выключатель просто не предусмотрен, что делает их фактически обязательными. Взвешенный по стойкости охват практически повторяет интегральную оценку (корреляция 0,93 с $H_{total}$) и приводится для полноты картины. Доля записей N/A практически не связана с интегральной оценкой (корреляция 0,07 с $H_{total}$) и характеризует не защищенность, а применимость таксономии к классу системы.

Качество реализованных механизмов упорядочивает системы почти противоположно охвату: наивысшие значения получают seL4 (2,55), Redox (2,40) и Tock (2,38), тогда как Linux при наибольшем охвате показывает умеренное качество (2,02). Пара «охват – качество» дает сжатое описание крайних случаев: seL4 реализует немногое, но почти безупречно (31%; 2,55), Fuchsia сочетает широту с качеством (73%; 2,25), Linux дает самый широкий охват при умеренном качестве (83%; 2,02).

Оцениваемые ядра различаются объемом привилегированного кода на три порядка (табл. 6): от 11 тыс. строк у seL4 до 11,2 млн строк в поставляемом ядре Fedora. Для Linux, OpenBSD, NetBSD, seL4, Redox и GNU Hurd подсчитаны только те единицы трансляции, которые





реально собираются в зафиксированной конфигурации; для Fuchsia, Tock и Zephyr использована более грубая оценка по каталогам исходного кода.

Отношение $H_{total}$ к размеру кода различает системы, достигающие сопоставимой интегральной оценки принципиально разной ценой доверенного кода: у seL4 плотность защит на строку кода на два–три порядка выше, чем у монолитных ядер общего назначения. Показатель является описательным и не входит в интегральную оценку: сам по себе размер доверенной кодовой базы – неточная характеристика поверхности атаки [21], однако без него сравнение ядра из 11 тыс. строк с ядром из миллионов строк было бы неполным.

*Табл. 6. Размер привилегированной кодовой базы (число строк исходного кода без пустых строк и комментариев) и плотность защиты (H_total на тыс. строк).*

*Table 6. Trusted computing base size (SLOC, excluding blank lines and comments) and hardening density (H_total per thousand SLOC).*

| Система | Строк кода ядра, тыс. | H_total на тыс. строк |
|---|---|---|
| Linux | 11 158 (1 883 без модулей) | 0,00015 |
| NetBSD | 2 269 (1 617 без модулей) | 0,00037 |
| OpenBSD | 1 909 | 0,00076 |
| Fuchsia | 234 | 0,0074 |
| GNU Hurd | 54 | 0,011 |
| Tock | 33 | 0,041 |
| Redox | 29 | 0,036 |
| Zephyr | 25 | 0,040 |
| seL4 | 11 | 0,079 |

## 6.4 Зависимости между шкалами

Для проверки того, что измерения методики не дублируют друг друга, вычислены ранговые корреляции Спирмена между шкалами. Корреляции между измерениями вычислены по реализованным механизмам – 306 парам «система–механизм» с $P$=1 – и приведены на рис. 4а. Записи с $P$=0 исключены: у них все измерения одновременно равны нулю, и общий ноль завысил бы каждый попарный коэффициент до 0,84 и выше, подменяя связь шкал фактом наличия механизма.

Попарные связи измерений умеренные, что подтверждает их содержательную самостоятельность. Сильнее всего связаны использование по умолчанию и архитектурная интеграция (0,65): встроенные в архитектуру механизмы чаще включены изначально. Зрелость $T$ слабо связана с остальными измерениями (0,19–0,23) и вносит информацию, не выводимую из прочих шкал. Эффективная стойкость почти совпадает с исходной (корреляция 0,99): ограничение по предпосылкам изменяет лишь 3 из 306 оценок – по одной у NetBSD, Fuchsia и Tock, – поэтому результаты не являются артефактом этого правила.

Корреляции между оценками категорий $H(C_k)$, вычисленные по девяти системам, приведены на рис. 4б. Для пар с участием C7 системы с неприменимой категорией исключены. Категории защиты от повреждения памяти, контроля целостности потока управления, предотвращения утечек информации и защиты от атак на спекулятивное исполнение образуют группу взаимно связанных направлений (C1, C2, C3, C7; попарные коэффициенты 0,61–0,87), к которой примыкает цепочка доверия (C6; 0,72 с C7). Разграничение привилегий и архитектурная изоляция связаны между собой (C4 и C5; 0,43), а с группой точечных программных мер противодействия эксплуатации преимущественно не коррелируют либо коррелируют отрицательно (коэффициенты от −0,57 до +0,24; единственная заметная положительная связь – 0,24 между C2 и C4). Наращивание программных мер

противодействия и архитектурная изоляция выступают в выборке как два слабо связанных направления развития защиты, и одно интегральное число не могло бы передать это различие. Пересчет по полному опубликованному набору данных – 19 конфигураций, включающих 15 систем и четыре варианта AArch64, – сохраняет обе группы, а отрицательные связи между ними при этом ослабевают.

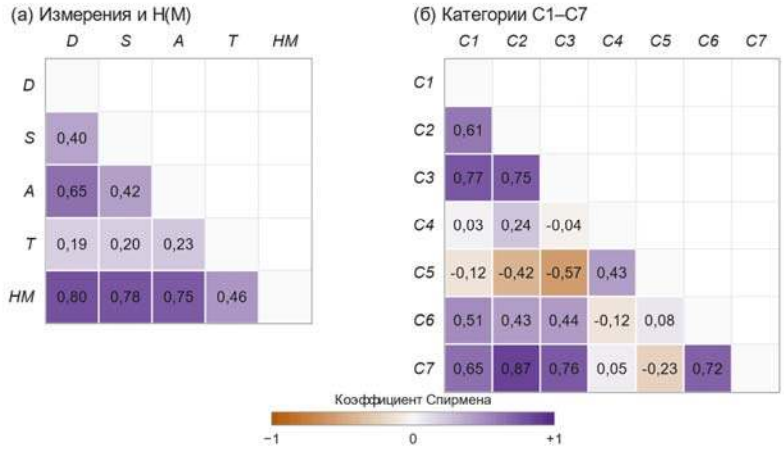


*Рис. 4. Ранговые корреляции Спирмена: (а) между измерениями и H(M); (б) между категориями C1–C7; цвет кодирует коэффициент от −1 до +1.*
*Fig. 4. Spearman rank correlations: (a) between dimensions and H(M); (b) between categories C1–C7; color encodes coefficients from −1 to +1.*

### 6.5 Хронология внедрения механизмов

Годы внедрения механизмов, определенные по самым ранним зафиксированным изменениям в опубликованном наборе данных, показаны на рис. 5.

Linux и OpenBSD наращивали механизмы постепенно, на протяжении двух–трех десятилетий, причем плотность внедрений заметно возрастает после 2015 года, что согласуется с институционализацией работ по самозащите ядра, отмеченной в разделе 1. Системы более позднего происхождения – Fuchsia, Redox, Tock и Zephyr – демонстрируют плотные серии внедрений в первые годы разработки, в том числе до первого официального выпуска. Хронология показывает смену модели развития: для новых ядер усиление защищенности перестает быть последовательностью доработок и закладывается при исходном проектировании.

## *7. Обсуждение*

Полученные результаты позволяют сформулировать несколько выводов, не сводящихся к сопоставлению итоговых оценок.

Во-первых, архитектура без инженерного сопровождения не обеспечивает защиту. GNU Hurd построен по классической микроядерной схеме с разграничением привилегий через порты Mach, однако занимает последнее место при всех рассмотренных моделях угроз: категории противодействия повреждению памяти и утечкам информации получают близкие к нулю оценки (0,37 и 0,18), а защита от атак на спекулятивное исполнение и цепочка доверия не





реализованы. Архитектурно выверенное решение без систематической реализации и сопровождения механизмов самозащиты получает низкую интегральную оценку.

Во-вторых, язык с безопасной работой с памятью является лишь частичным решением. Rust устраняет значительную часть поверхности атаки, связанной с повреждением памяти, и по пространственной безопасности памяти Tock превосходит Linux. Однако выбор языка не закрывает остальные категории: в Redox отсутствуют рандомизация базового адреса ядра и защита от атак на спекулятивное исполнение. Аналогично модель объектных возможностей обеспечивает высокие оценки разграничения привилегий, но сама по себе не защищает от эксплуатации ошибок в реализации ядра. Незакрытые категории образуют наиболее привлекательные для нарушителя направления атаки, поэтому одного универсального решения недостаточно.

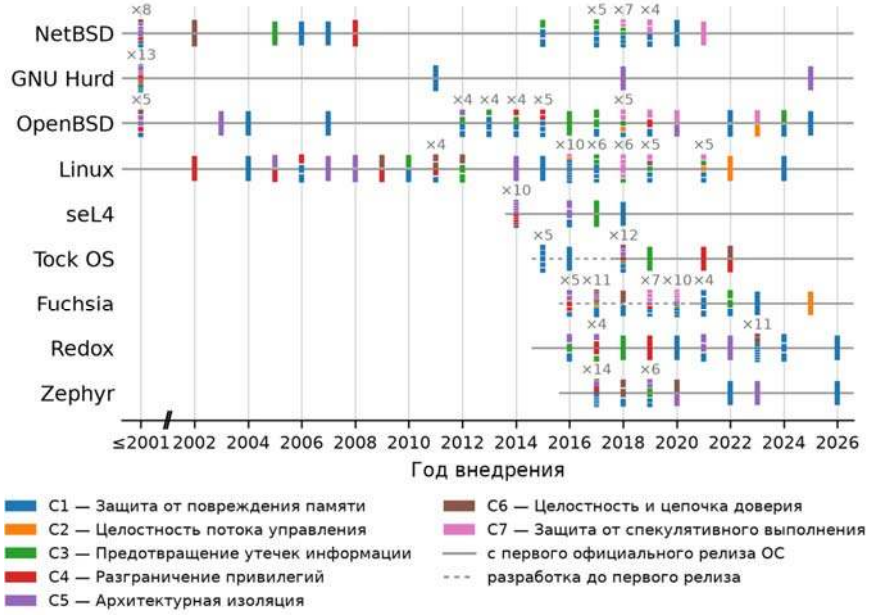


*Рис. 5. Хронология внедрения механизмов усиления защищенности: полоса соответствует системе, засечка – году самого раннего зафиксированного изменения, реализующего механизм; цвет обозначает категорию таксономии, сплошная часть полосы начинается с первого официального выпуска системы.*
*Fig. 5. Timeline of hardening-mechanism introduction: one lane per system, one notch per mechanism at the year of its earliest implementing commit; color encodes the taxonomy category, and the solid part of each lane starts at the system's first official release.*

В-третьих, интегральная оценка измеряет широту охвата таксономии механизмов и потому не видит гарантий, которые механизмами не являются. Показательный случай – seL4: седьмое место из девяти (0,87) при наивысшем в выборке качестве реализованных механизмов (2,55) и охвате лишь 31%. Из 37 отсутствующих применимых механизмов действительными пробелами являются лишь восемь. Задачу одиннадцати механизмов, включая стековые «канарейки» и разделение записываемой и исполняемой памяти (W^X), решает формальное доказательство корректности: механизм обнаружения ошибки не нужен там, где доказано, что ошибка невозможна. Еще четыре механизма несовместимы с самой верификацией: рандомизация адресов и структур обесценила бы доказательство,

зафиксированное для конкретной раскладки памяти, поэтому отсутствие KASLR в seL4 – не упущение, а осознанно исключенное свойство. Остальные отсутствующие механизмы замещены структурными альтернативами, перенесены в пользовательское окружение или неприменимы на зафиксированной аппаратной платформе. Ядро, добавившее недостающие механизмы, подняло бы свою интегральную оценку, но увеличило бы объем неверифицированного кода и тем самым ослабило бы гарантии, обеспечиваемые формальной верификацией. Сами эти гарантии не являются безусловными: формальная верификация имеет собственные предпосылки и ограничения, которые в настоящей работе не оцениваются. Существенное исключение – каналы утечки по времени: доказательство корректности их явно не охватывает, а защита по времени реализована только в виде исследовательского прототипа вне основной ветви seL4, поэтому оценка C7 составляет лишь 0,27 [40]. Дефицит охвата не устраняется выбором весов – ни один из двенадцати сценариев не поднимает seL4 выше седьмого места; читать его результат следует парой показателей «охват – качество» (раздел 6.3).

В-четвертых, отсутствие механизма не всегда означает отсутствие защиты: случай OpenBSD показывает, что защищенность системы не сводится к перечню реализованных в ядре механизмов усиления защищенности. Из двенадцати механизмов, реализованных в Linux и отсутствующих в OpenBSD, лишь три отвергнуты проектом сознательно, четыре замещены иными, не эквивалентными средствами. Характерный пример замещения – обнаружение переполнения буфера на этапе компиляции (FORTIFY_SOURCE): ту же задачу проект решает на уровне исходного кода дисциплиной безопасных строковых интерфейсов strlcpy и strlcat [41]. Значительная часть отставания в защите от атак на спекулятивное исполнение объясняется принципиальной позицией проекта: наиболее надежной мерой против микроархитектурных утечек между аппаратными потоками считается отключение одновременной многопоточности (SMT), действующее по умолчанию с 2018 года, поскольку аппаратные потоки разделяют буфер трансляции адресов (TLB) и кэш первого уровня [42]. Отключение SMT снимает необходимость в целых классах точечных мер противодействия, которые вынужден поддерживать Linux, – барьерах предсказателя переходов для отдельных процессов, режимах защиты от Spectre v4, инфраструктуре параметра `mitigations=` – и именно эти отсутствующие строки таксономия фиксирует как нули. Пробел в цепочке доверия также является следствием решений, а не небрежности: проект не стал встраиваться в экосистему ключей безопасной загрузки на основе единого расширяемого интерфейса прошивки (UEFI) [43], не реализовал проверку целостности исполняемых файлов во время исполнения, отклонив соответствующие предложения [44-45], и компенсирует это подписью выпусков и пакетов на этапе распространения средствами signify [46]. Измеряемая загрузка, кроме того, структурно конфликтует с KARL: ядро со случайной компоновкой при каждой загрузке несовместимо со статическими эталонными контрольными значениями. Методика сознательно оценивает механизмы ядра, а не результат замещения: эквивалентные альтернативы учтены в других строках таксономии, тогда как неэквивалентные замещения и настоящие пробелы остаются нулями – именно они и составляют содержание разрыва между OpenBSD и Linux.

В-пятых, наличие реализации не означает ее использования. Zephyr реализует 65% применимых механизмов, однако в стандартной конфигурации активны лишь 47% из них, тогда как в OpenBSD и Tock включено 85% и 92% реализованных механизмов соответственно. Разрыв между потенциальной и фактической защищенностью характеризует и Linux: часть механизмов основной ветви, включая kCFI и очистку освобожденной памяти и стека, в конфигурации Fedora требует явного включения.

В-шестых, оценки категории контроля целостности потока управления существенно зависят от аппаратной платформы. Аппаратные IBT и CET доступны только на x86, BTI и PAC – только на AArch64, а совместное применение аутентификации указателей и идентификации целей переходов (PACBTI) – начиная с ARMv8.1-M; на микроконтроллерах без таких





расширений соответствующие записи получают N/A, и категория может быть закрыта только программными средствами. Поэтому достижимый уровень защиты потока управления определяется не только зрелостью программной реализации, но и выбранной платформой, которая фиксирует применимость $P$ аппаратных механизмов.

В-седьмых, корреляционный анализ раздела 6.4 показывает, что наращивание программных мер противодействия и архитектурная изоляция развиваются в выборке как во многом независимые направления защиты; единственное исключение – Fuchsia, сочетающая оба направления.

В-восьмых, производные показатели разделяют два способа обеспечить применение защиты: архитектурную обязательность (Tock, Redox, Fuchsia) и дисциплину сопровождения конфигурации (Linux; раздел 6.3), что продолжает разграничение измерений $D$ и $A$.

Устойчивость ранжирования к сценариям весов и к случайному перебору комбинаций (раздел 6.2) показывает, что выводы сравнения не определяются точными значениями весов измерений; действительным параметром сравнения остаются веса категорий, то есть выбор модели угроз. Слабая связь зрелости $T$ с остальными измерениями и расхождение охвата с охватом по умолчанию эмпирически оправдывают выделение $T$ и $D$ в самостоятельные шкалы.

Главный вывод состоит в том, что разброс оценок внутри одного архитектурного класса может превышать различия между классами: архитектура задает доступные способы изоляции, но интегральный результат определяется полнотой реализации механизмов, режимом их применения по умолчанию, стойкостью и зрелостью. При этом интегральная оценка характеризует качество инженерного сопровождения средств самозащиты, а не «безопасность» в абсолютном смысле: корректность реализации и наличие незакрытых уязвимостей ею не измеряются. Такое сопровождение может коррелировать с фактической защищенностью, однако подмена оценки защищенности показателем его качества была бы методологической ошибкой. Интегральную оценку при этом следует читать вместе с показателями охвата и качества реализованных механизмов: одно число не различает узкую, но почти безупречную реализацию и широкую, но умеренную (раздел 6.3).

Для ОС, служащих платформой для разнородных применений, детальные модели угроз будущих применений предусмотреть невозможно, поэтому общий набор механизмов самозащиты закладывается заранее и конфигурируется под конкретный сценарий. Самозащиту ядра следует отличать от управления уязвимостями: последнее устраняет выявленные дефекты, тогда как механизмы самозащиты исходят из допущения, что неизвестные уязвимости в коде присутствуют всегда. Методика пригодна и для самопроверки перед внешним аудитом: низкая оценка отдельной категории служит не итоговым выводом, а сигналом для целенаправленного исследования.

Наконец, полученные профили указывают направление сближения классов: перспективной представляется интеграция архитектурной изоляции, свойственной микроядерным и встраиваемым системам, с механизмами рандомизации, контроля целостности потока управления и цепочки доверия, развитыми в монолитных ядрах. Fuchsia, сочетающая изолированные драйверы с программной защитой потока управления и доверенной загрузкой, показывает, что эти свойства совместимы в одной системе.

## *8. Угрозы валидности*

Основная угроза конструктной валидности связана с экспертным характером рангов стойкости $S$ и архитектурной интеграции $A$. Для снижения субъективности используются явные критерии шкал, опубликованные исследования способов обхода [27, 28, 29] и открытые текстовые обоснования каждой оценки, допускающие независимую проверку. Тем не менее оценки выставлены одним исследователем; протокол независимого дублирующего оценивания с измерением согласованности остается предметом дальнейшей работы. Кроме

того, механизмы существенно различной сложности вносят в оценку одинаковый вклад: элементарная проверка в аллокаторе памяти и контроль целостности потока управления в равной мере получают $P$=1, а различия между развитым и упрощенным вариантами одного механизма лишь частично отражает шкала стойкости. Это решение принято осознанно, поскольку и простые механизмы могут заметно повышать стоимость эксплуатации; приоритетность внедрения выражена в методике косвенно – через зрелость механизмов и их использование по умолчанию. Внутри отдельной системы измерения могут вырождаться: у Linux все реализованные механизмы имеют $A$=2, поэтому корреляционный анализ раздела 6.4 опирается на межсистемную вариацию, а различия механизмов внутри одной системы определяются главным образом измерениями $D$, $S$ и $T$.

Оценки привязаны к зафиксированным версиям ядер и стандартным конфигурациям, перечисленным в табл. 5. Состав механизмов и настройки по умолчанию со временем меняются, поэтому приведенные значения представляют собой срез на момент оценивания, а не постоянные характеристики систем. Аналогично таксономия фиксирует классы угроз, известные на момент ее составления: появление принципиально новых классов атак, как произошло с атаками на спекулятивное исполнение [9], или перенос систем в непредусмотренные области применения может потребовать ее расширения. Методика сознательно измеряет воспроизводимые признаки реакции на известные угрозы, а не способность предвидеть новые.

Накладные расходы исключены из интегральной оценки: система не штрафуется за применение ресурсоемкой, но стойкой защиты, однако $H_{total}$ не отражает эксплуатационную цену механизмов.

Итоговые значения зависят от выбора весов измерений и категорий. Для весов измерений эта угроза снята эмпирически: при случайном переборе допустимых комбинаций ранговая корреляция с базовым ранжированием почти всегда остается не ниже 0,90 (раздел 6.2). Веса категорий действительно влияют на порядок систем, однако их измеримая часть согласована с тремя независимыми источниками статистики уязвимостей, устойчивость к малым отклонениям весов показана в разделе 4.5, а спорные пары и порог различимости оценок указаны явно в разделе 6.2. Веса категорий C2, C6 и C7 принципиально не выводимы из статистики уязвимостей и заданы решением, отражающим модель угроз; поэтому результаты следует интерпретировать только совместно с указанием профиля и модели угроз.

Наконец, методика не оценивает защищенность в абсолютном смысле и корректность реализации механизмов (раздел 7): высокая оценка не гарантирует отсутствия уязвимостей. Интегральная оценка, кроме того, взвешена охватом таксономии: полное устранение поверхности атаки не приносит баллов, тогда как усиление оставшейся поверхности оценивается вплоть до максимальной оценки. Гарантии, не являющиеся механизмами ядра, – формальная верификация, культура аудита кода, усиление защищенности пользовательского окружения – учитываются лишь частично через шкалу стойкости либо остаются за рамками таксономии. Методика также не содержит вычислимой модели нарушителя: списочная форма таксономии удобна для проверки, но уступает формализованной модели атакующих воздействий в описательной силе; построение такой модели остается предметом дальнейшей работы. Воспроизводимость результатов обеспечивается публикацией машиночитаемых данных и обоснований оценок.

## *9. Заключение*

В работе предложены не зависящая от архитектуры таксономия, включающая 76 механизмов усиления защищенности ядра в семи категориях, и многомерная методика их количественной оценки по наличию и применимости, использованию по умолчанию, стойкости, архитектурной интеграции и зрелости. Настраиваемые профили весов измерений и веса категорий, задаваемые моделью угроз, позволяют сводить частные оценки к





воспроизводимой интегральной оценке. Методика применена к девяти системам из трех аналитических групп: монолитным ОС, системам на основе микроядра и встраиваемым ОС.

Главный результат состоит в том, что интегральная оценка определяется не архитектурным классом как таковым, а полнотой реализации механизмов, режимом их применения по умолчанию, стойкостью и зрелостью; состав лидеров при этом зависит от модели угроз. Поэтому выбор системы для конкретного применения должен определяться приоритетами модели угроз, а не архитектурной принадлежностью.

Направления дальнейшей работы включают автоматизацию выставления и актуализации оценок, расширение набора рассматриваемых ОС и конфигураций, развитие таксономии как расширяемого каталога классов угроз, уточнение шкал стойкости и архитектурной интеграции с независимым дублирующим оцениванием, а также исследование совместного применения архитектурной изоляции и механизмов рандомизации и контроля целостности потока управления. Опубликованные данные о датах появления механизмов (рис. 5) и их использовании по умолчанию позволяют построить эмпирическую приоритизацию: механизмы, раньше других принятые и включенные по умолчанию в большинстве систем, образуют естественную последовательность внедрения защит в новых ядрах. Машиночитаемые таксономия, оценочные данные и обоснования опубликованы в открытом наборе данных [15].

## Список литературы / References


[1]. National Security Agency, Software Memory Safety. Cybersecurity Information Sheet, 2022. Available at: https://media.defense.gov/2022/Nov/10/2003112742/-1/-1/0/CSI_SOFTWARE_MEMORY_SAFETY.PDF, accessed 09.07.2026.

[2]. Szekeres L., Payer M., Tao Wei, Song D. SoK: Eternal War in Memory. In Proc. of the 2013 IEEE Symposium on Security and Privacy, 2013. pp. 48-62. DOI: 10.1109/SP.2013.13.

[3]. Cybersecurity and Infrastructure Security Agency, Shifting the Balance of Cybersecurity Risk: Principles and Approaches for Secure by Design Software. CISA and partner agencies, 2023. Available at: https://www.cisa.gov/resources-tools/resources/secure-by-design, accessed 09.07.2026.

[4]. Office of the National Cyber Director, Back to the Building Blocks: A Path Toward Secure and Measurable Software. The White House, Office of the National Cyber Director, 2024. Available at: https://bidenwhitehouse.archives.gov/wp-content/uploads/2024/02/Final-ONCD-Technical-Report.pdf, accessed 03.08.2026.

[5]. PaX Team, Address Space Layout Randomization. PaX project documentation, 2001. Available at: https://pax.grsecurity.net/docs/aslr.txt, accessed 03.08.2026.

[6]. de Raadt T. Exploit Mitigation Techniques in OpenBSD. OpenCON, presentation foils, 2005. Available at: https://www.openbsd.org/papers/ven05-deraadt/index.html, accessed 03.08.2026.

[7]. Evans C. Announcing Project Zero. Google Online Security Blog, 2014. Available at: https://security.googleblog.com/2014/07/announcing-project-zero.html, accessed 29.07.2026.

[8]. Kernel Self-Protection Project. Available at: https://kspp.github.io/, accessed 03.08.2026.

[9]. Kocher P., Horn J., Fogh A., Genkin D., Gruss D., Haas W., Hamburg M., Lipp M., Mangard S., Prescher T., Schwarz M., Yarom Y. Spectre Attacks: Exploiting Speculative Execution. In Proc. of the 2019 IEEE Symposium on Security and Privacy (SP), 2019. pp. 1-19. DOI: 10.1109/SP.2019.00002.

[10]. National Information Assurance Partnership, Protection Profile for General Purpose Operating Systems, Version 4.3. 2022. Available at: https://www.niap-ccevs.org/static_html/protection-profile/469/OS%204.3%20PP/index.html, accessed 03.08.2026.

[11]. Ефремов Д. В., Петренко А. К., Позин Б. А., Семенов В. А. Обзор механизмов усиления защищенности операционных систем и пользовательских приложений. Труды Института системного программирования РАН, 2025, том 37, вып. 3, стр. 325-354. DOI: 10.15514/ISPRAS-2025-37(3)-23. Available at: https://www.mathnet.ru/php/archive.phtml?wshow=paper&jrnid=tisp&paperid=1006&option_lang=eng, accessed 03.08.2026.

[12]. Center for Internet Security, CIS Benchmarks. Available at: https://www.cisecurity.org/cis-benchmarks, accessed 09.07.2026.

[13]. Defense Information Systems Agency, Security Technical Implementation Guides (STIGs). Available at: https://public.cyber.mil/stigs/, accessed 09.07.2026.
[14]. Popov A. kernel-hardening-checker: A tool for checking the security hardening options of the Linux kernel. 2024. Available at: https://github.com/a13xp0p0v/kernel-hardening-checker, accessed 09.07.2026.
[15]. Efremov D. V. OS Kernel Hardening Evaluation: Taxonomy, Evaluation Data and Scoring Pipeline. 2026. Available at: https://evdenis.github.io/os-hardening-evaluation/, accessed 29.07.2026.
[16]. Pendleton M., Garcia-Lebron R., Cho J., Xu S. A Survey on Systems Security Metrics. ACM Computing Surveys, 2016, vol. 49, issue 4, pp. 62:1-62:35. DOI: 10.1145/3005714.
[17]. Afzali H., Mokhtari H. A Quantitative Model of Operating System Security Evaluation. In Proc. of the Advances in Computer Science, Engineering & Applications, 2012. pp. 345-353. DOI: 10.1007/978-3-642-30111-7_33.
[18]. Song J., Hu G., Xu Q. Operating System Security and Host Vulnerability Evaluation. In Proc. of the 2009 International Conference on Management and Service Science, 2009. pp. 1-4. DOI: 10.1109/ICMSS.2009.5302077.
[19]. Cheng L., Zhang Y., Han Z., Deng Y., Sun X., Feng D. Evaluating and comparing the quality of access control in different operating systems. Computers & Security, 2014, vol. 47, pp. 26-40. DOI: 10.1016/j.cose.2014.05.001.
[20]. Chen S., Li D., Wu C., Zhan J. Evaluating Kernel Anti-Exploitation Capabilities: A Scalable and General Framework Based on Evaluatology. In Proc. of the Benchmarking, Measuring, and Optimizing – 16th BenchCouncil International Symposium, Bench 2024, 2025. pp. 127-143. DOI: 10.1007/978-981-96-5032-3_8.
[21]. Kurmus A., Tartler R., Dorneanu D., Heinloth B., Rothberg V., Ruprecht A., Schröder-Preikschat W., Lohmann D., Kapitza R. Attack Surface Metrics and Automated Compile-Time OS Kernel Tailoring. In Proc. of the 20th Annual Network and Distributed System Security Symposium (NDSS), 2013. Available at: https://www.ndss-symposium.org/ndss2013/attack-surface-metrics-and-automated-compile-time-os-kernel-tailoring.
[22]. Enoch S. Y., Moon C. Y., Lee D., Ahn M. K., Kim D. S. A Practical Framework for Cyber Defense Generation, Enforcement and Evaluation. Computer Networks, 2022, vol. 208, pp. 108878. DOI: 10.1016/j.comnet.2022.108878.
[23]. Lim S. Y., Agrawal S., Han X., Eyers D., O'Keeffe D., Pasquier T. Securing Monolithic Kernels using Compartmentalization. 2024. arXiv:2404.08716.
[24]. Lefeuvre H., Dautenhahn N., Chisnall D., Olivier P. SoK: Software Compartmentalization. In Proc. of the 46th IEEE Symposium on Security and Privacy (S&P), 2025. pp. 3107-3126. DOI: 10.1109/SP61157.2025.00075.
[25]. Guo Y., Wang Z., Bai W., Zeng Q., Lu K. BULKHEAD: Secure, Scalable, and Efficient Kernel Compartmentalization with PKS. In Proc. of the 32nd Annual Network and Distributed System Security Symposium (NDSS), 2025. DOI: 10.14722/ndss.2025.230328.
[26]. Klein G., Andronick J., Elphinstone K., Murray T., Sewell T., Kolanski R., Heiser G. Comprehensive Formal Verification of an OS Microkernel. ACM Transactions on Computer Systems, 2014, vol. 32, issue 1, pp. 2:1-2:70. DOI: 10.1145/2560537.
[27]. Gruss D., Lipp M., Schwarz M., Fellner R., Maurice C., Mangard S. KASLR is Dead: Long Live KASLR. In Proc. of the Engineering Secure Software and Systems (ESSoS 2017), 2017. pp. 161-176. DOI: 10.1007/978-3-319-62105-0_11.
[28]. Yoo S., Park J., Kim S., Kim Y., Kim T. In-Kernel Control-Flow Integrity on Commodity OSes using ARM Pointer Authentication. In Proc. of the 31st USENIX Security Symposium, 2022. pp. 89-106. Available at: https://www.usenix.org/conference/usenixsecurity22/presentation/yoo.
[29]. Liakh S., Grace M., Jiang X. Analyzing and Improving Linux Kernel Memory Protection: A Model Checking Approach. In Proc. of the 26th Annual Computer Security Applications Conference (ACSAC), 2010. pp. 271-280. DOI: 10.1145/1920261.1920301.
[30]. Rauscher F., Herzog B., Hönig T., Gruss D. Systematic Analysis of Kernel Security Performance and Energy Costs. In Proc. of the 20th ACM Asia Conference on Computer and Communications Security, 2025. pp. 1676-1689. DOI: 10.1145/3708821.3736197.
[31]. Bhat P., Dutta K. A Survey on Various Threats and Current State of Security in Android Platform. ACM Computing Surveys, 2019, vol. 52, issue 1, pp. 21:1-21:35. DOI: 10.1145/3301285.
[32]. Ahmed O. M., Sallow A. B. Android Security: A Review. Academic Journal of Nawroz University, 2017, vol. 6, issue 3, pp. 135-140. DOI: 10.25007/ajnu.v6n3a97.





[33]. Zakaria S. N., Zolkipli M. F. Review on Mobile Attacks: Operating System, Threats, and Solution. Borneo International Journal, 2021, vol. 4, issue 2, pp. 8-16.
[34]. Cybersecurity and Infrastructure Security Agency, Vulnrichment: CVE enrichment with CWE, CVSS and SSVC decision points. 2024. Available at: https://github.com/cisagov/vulnrichment, accessed 02.08.2026.
[35]. MITRE Corporation, 2025 CWE Top 25 Most Dangerous Software Weaknesses. 2025. Available at: https://cwe.mitre.org/top25/, accessed 02.08.2026.
[36]. Google Project Zero, 0-days In-the-Wild: root cause analyses and tracking spreadsheet. Available at: https://googleprojectzero.github.io/0days-in-the-wild/, accessed 02.08.2026.
[37]. Shameli-Sendi A. Understanding Linux kernel vulnerabilities. Journal of Computer Virology and Hacking Techniques, 2021, vol. 17, issue 4, pp. 265-278. DOI: 10.1007/s11416-021-00379-x.
[38]. de Raadt T. KARL - Kernel Address Randomized Link. 2017. Available at: https://marc.info/?l=openbsd-tech&m=149732026405941&w=2, accessed 17.06.2025.
[39]. de Raadt T. Amd64 Kernel W^X. 2015. Available at: https://marc.info/?l=openbsd-tech&m=142120787308107&w=2, accessed 17.06.2025.
[40]. Ge Q., Yarom Y., Chothia T., Heiser G. Time Protection: The Missing OS Abstraction. In Proc. of the Fourteenth EuroSys Conference (EuroSys '19), 2019. DOI: 10.1145/3302424.3303976.
[41]. Miller T. C., de Raadt T. strlcpy and strlcat – Consistent, Safe, String Copy and Concatenation. In Proc. of the USENIX Annual Technical Conference, FREENIX Track, 1999. Available at: https://www.usenix.org/legacy/publications/library/proceedings/usenix99/full_papers/millert/millert.pdf, accessed 03.08.2026.
[42]. Kettenis M. Disable SMT (Simultaneous Multi-Threading) by default and introduce the hw.smt sysctl. OpenBSD src repository, commit 96c1135, 2018. Available at: https://github.com/openbsd/src/commit/96c11352863a7f6240b4e5e388052f414b75f95b, accessed 02.08.2026.
[43]. Varghese S. OpenBSD's de Raadt slams Red Hat, Canonical over 'secure' boot. iTWire, interview with Theo de Raadt, 2012. Available at: https://itwire.com/business-it-news/open-source/openbsds-de-raadt-slams-red-hat-canonical-over-secure-boot, accessed 03.08.2026.
[44]. Unangst T. Re: veriexec in OpenBSD?. Post to the openbsd-misc mailing list, 2010. Available at: https://marc.info/?l=openbsd-misc&m=128337583106955&w=2, accessed 02.08.2026.
[45]. Guenther P. Re: Verified Executables for OpenBSD?. Post to the openbsd-misc mailing list, 2016. Available at: https://marc.info/?l=openbsd-misc&m=147327081816169&w=2, accessed 02.08.2026.
[46]. Unangst T. signify: Securing OpenBSD From Us To You. BSDCan, 2015. Available at: https://www.openbsd.org/papers/bsdcan-signify.html, accessed 02.08.2026.

## *Информация об авторах / Information about authors*

Денис Валентинович ЕФРЕМОВ – старший научный сотрудник. Сфера научных интересов: формальная верификация, статический и динамический анализ.

Denis Valentinovich EFREMOV – senior researcher. Research interests: formal verification, static and dynamic analysis.