DOI: https://doi.org/10.15276/ict.03.2026.05
УДК 004.056.53:004.451.9

Архітектурні рішення мережевого екранування в операційних системах Android та iOS: порівняльний аналіз рівнів ядра, системних розширень та застосунка

Задерейко Олександр Владиславович[1)]
ORCID: https://orcid.org/0000-0003-0497-9861; zadereyko@onua.edu.ua. Scopus Author ID: 57196005917
Гура Володимир Ігоревич[1)]
ORCID: https://orcid.org/0000-0001-6415-7865; hura@onua.edu.ua
Блажко Олександр Анатолійович[2)]
ORCID: https://orcid.org/0000-0001-7413-275X; blazhko@op.edu.ua. Scopus Author ID: 57195410976
Гайда Анатолій Юліанович[3)]
ORCID: https://orcid.org/0000-0002-8217-5044; cetus@ukr.net
[1)] Національний університет «Одеська юридична академія», Фонтанська дорога, 23. Одеса, 65009, Україна
[2)] Національний університет «Одеська політехніка», пр. Шевченка, 1. Одеса, 65044, Україна
[3)] Національний університет кораблебудування імені адмірала Макарова, пр. Центральний 3, Миколаїв, 54000, Україна

АНОТАЦІЯ

Стаття присвячена порівняльному аналізу архітектурних рішень, що лежать в основі мережевого екранування в операційних системах Android та iOS. Актуальність дослідження зумовлена домінуючим становищем цих двох операційних систем на ринку мобільних комунікаційних пристроїв і принциповими відмінностями їх архітектурної організації, що визначають можливості та обмеження сторонніх засобів контролю мережевих з'єднань. Встановлено, що архітектура мережевого екранування операційної системи Android на основі модуля Netfilter/iptables і технології extended Berkeley Packet Filter достатньо детально висвітлена у наукових публікаціях, тоді як архітектура мережевого екранування операційної системи iOS, а саме фреймворк NetworkExtension (разом із системним пакетним фільтром PF ядра XNU та тунельним інтерфейсом utun) розглядаються існуючими дослідженнями лише фрагментарно, у загальному ряду характеристик безпеки, без виділення як самостійного предмета аналізу. Для усунення виявленої прогалини виконано послідовну декомпозицію архітектур обох операційних систем на порівнюємі рівні – ядро, системні служби та розширення, користувацький застосунок. Показано, що мережевий екран операційної системи Android реалізується трьома архітектурними моделями. На основі Netfilter/iptables/eBPF, на основі eBPF у складі платформи Android Open Source Project та на основі непривілейованого програмного інтерфейсу VpnService. Тоді як в операційній системі iOS функції мережевого екрана розподілені між конфігураційними та виконавчими модулями фреймворку NetworkExtension (NEFilterManager, NEFilterProviderConfiguration, NEFilterDataProvider, NEFilterControlProvider), а застосування вердиктів здійснюється через системний тунельний інтерфейс utun. Визначено, що архітектурні відмінності між операційними системами виявляються на трьох рівнях. Відкритості ядра, кількості та спеціалізації точок розширення для сторонніх розробників і моделі ізоляції виконання фільтрувального коду. Спільним для обох операційних систем обмеженням виступає неможливість одночасної роботи декількох незалежних мережевих фільтрів. Запропонована декомпозиційна модель яка симетрично відображає архітектуру мережевого екранування обох операційних систем на порівнюваних рівнях. Отримані результати декомпозиції архітектур формують основу для подальшої розробки єдиної методології та системи критеріїв порівняльного аналізу мережевих екранів операційних систем Android та iOS.

Ключові слова: Android, iOS; Netfilter/iptables; NetworkExtension; архітектура безпеки; мережевий екран; мережеві з'єднання

Для цитування: **Задерейко О. В., Гура В. І., Блажко О. А., Гайда А. Ю. "Архітектурні рішення мережевого екранування в операційних системах Android та iOS: порівняльний аналіз рівнів ядра, системних розширень та застосунка".** ***Інформатика. Культура. Техніка.*** **2026; Том 3 № 1 (3): 58–74. DOI: https://doi.org/10.15276/ict.03.2026.05**

Актуальність. Комунікаційні пристрої під керуванням операційних систем (ОС) Android та iOS є основним засобом взаємодії користувачів з комунікаційним мережевим середовищем. Їхнє повсюдне поширення безпосередньо пов'язане зі зростанням кількості цілеспрямованих мережевих атак, що робить контроль мережевих з'єднань одним із фундаментальних завдань комунікаційної безпеки, оскільки ці комунікаційні пристрої зберігають і передають значні обсяги персональних даних.

Фундаментальною основою їхнього захисту є мережевий екран, реалізація якого в ОС спирається на принципово різні архітектурні засоби:

– ОС Android використовує модифіковане ядро Linux із модулем Netfilter та системну службу користувацького простору netd, що спирається на нього;

– ОС iOS успадкувала від BSD пакетний фільтр PF у закритому ядрі XNU та винесений в ізольовані розширення модуль фреймворка NetworkExtension.

Ця архітектурна відмінність породжує різні можливості, обмеження та моделі загроз для мережевих екранів сторонніх виробників, що працюють у межах стандартних, не розширених користувачем прав доступу, та зумовлює наукову й практичну проблему відсутності єдиної методології їхнього порівняння.

Архітектура мережевого екрану в ОС Android на основі модуля Netfilter/iptables досить детально висвітлена в наукових публікаціях. Дослідження [1] описує принцип функціонування модуля Netfilter/iptables, призначення окремих ланцюжків правил і наводить приклад реалізації мережевого екрану, що ефективно запобігає зовнішнім атакам. Більш складним класом задач є формальна верифікація правил, а не лише їх налаштування [2], [3]. У цих дослідженнях представлено повністю верифіковану систему аналізу наборів правил iptables, яка обчислює мінімальні сервісні матриці допустимих мережевих з'єднань. Коректність запропонованих алгоритмів перетворення правил у спрощену модель доведено за допомогою системи Isabelle, а дослідження [3] визначає процес формування правил функціонування мережевих екранів. У дослідженні [4] запропоновано інтелектуальне розширення функціональності мережевого екрану за рахунок програмної архітектури системи виявлення мережевих аномалій, у якій Netfilter перехоплює мережеві з'єднання та передає їх у загальний буфер, звідки їх зчитують конвеєри аналізу для детектування аномалій на основі марковських ланцюгів змінного порядку та самоорганізованих карт Кохонена.

Однак дослідження [1], [2], [3], [4] розглядають функціонал Netfilter/iptables виключно в контексті Linux-подібних систем, не адаптуючи висновки до специфіки функціонування ОС Android (обмежень щодо енергоспоживання, фонової роботи процесів та моделі дозволів мережевих з'єднань для застосунків).

Одне з фундаментальних системних досліджень [5] порівнює ОС Android та iOS за загальними параметрами безпеки такими як модель дозволів для застосунків, їх ізоляцію та шифрування. При цьому тематика мережевих екранів порушується лише побічно, через категорію дозволів на мережеві з'єднання. У дослідженні [6] проведено порівняння компонентів безпеки, а саме рандомізації пам'яті, пісочниці застосунків, ізоляції, шифрування, вбудованого антивіруса та зберігання даних. На підставі отриманих даних зроблено висновок, що ОС iOS забезпечує вищий рівень безпеки завдяки більшій закритості. Дослідження [7] порівнює засоби обробки даних застосунками для ОС Android та iOS з вимогами політик конфіденційності, також не розглядаючи архітектуру мережевих екранів як окремий об'єкт аналізу. Дослідження [8] порівнює користувачів ОС Android та iOS за рівнем обізнаності щодо безпеки та конфіденційності на основі онлайн-опитування по над 700 респондентів, зміщуючи пріоритет у бік поведінкових, а не архітектурно-технічних аспектів. Таким чином, дослідження [5], [6], [7], [8] розглядають контроль мережевих з'єднань лише опосередковано, у загальному ряді характеристик безпеки, не виділяючи архітектуру мережевих екранів як самостійний предмет аналізу.

Порівняння наведених досліджень виявляє суттєві відмінності в аналізі архітектури мережевих екранів ОС Android, включаючи її формальну верифікацію [2], [3] та інтелектуальні доповнення [4]. Водночас кількість досліджень, присвячених аналізу архітектури системного модуля мережевого екранування в ОС iOS (фреймворку NetworkExtension) та реалізованих на його основі способів тунелювання й фільтрації мережевих з'єднань, є невеликою. Дослідження [5], [6], [8] розглядають контроль мережевих з'єднань лише у загальному контексті таких характеристик, як дозволи, пісочниця та шифрування, а дослідження [7], яке є найближчим до розглянутої тематики, обмежується аналізом відповідності доступу застосунків встановленим політикам конфіденційності.

Слід уточнити критерій, покладений в основу даного порівняння. Мова йде не про відсутність досліджень безпеки ОС iOS, а про відсутність досліджень, що системно

розглядають архітектуру мережевого екранування (фреймворку NetworkExtension та пакетного фільтра PF, що лежить в його основі) як самостійний предмет аналізу, за аналогією з тим, як дослідження [1], [2], [3], [4] аналізують архітектуру Netfilter/iptables. Саме за цим критерієм дослідження [5]. [6], [7], [8], зберігаючи наукову цінність як порівняльний аналіз безпеки ОС Android та iOS загалом, не заповнюють виявлений пробіл у дослідженнях. Вони розглядають мережеві екрани як один з елементів організації безпеки (дозвіл, ізоляція, шифрування, відповідність політикам конфіденційності), а не як об'єкт цілісного архітектурного аналізу.

Таким чином, існуючі теоретичні та практичні напрацювання щодо організації мережевих екранів в ОС Android [1], [2], [3], [4] різко контрастують із фактичною відсутністю досліджень архітектури мережевих екранів в ОС iOS, а існуючі порівняльні дослідження з аналізу безпеки ОС Android та iOS [5], [6], [7], [8] торкаються питань мережевого екранування лише фрагментарно.

Відсутність систематизованого порівняння ОС Android та iOS не дозволяє обґрунтовано обирати архітектурні рішення під час розробки кросплатформних засобів контролю мережевих з'єднань застосунків, зокрема рішень, що забезпечують захист персональних даних користувача шляхом обмеження їх неконтрольованої передачі у комунікаційну мережу.

Також слід зазначити, що актуальність порівняльного аналізу підходів до організації мережевого екранування ОС Android та iOS визначається трьома взаємопов'язаними факторами:

– масштабом поширення – на сьогодні ОС Android та iOS практично повністю контролюють ринок мобільних ОС. Станом на початок 2026 року на частку цих двох платформ припадає понад 99% активних комунікаційних пристроїв у світі, при цьому ОС Android займає близько 70% ринку, а ОС iOS – близько 29 %, тоді як системи, що раніше конкурували з ними (Symbian, BlackBerry OS, Windows Phone) фактично припинили існування. Така ситуація означає, що будь-яке технічне або методологічне рішення у сфері мобільної мережевої безпеки, яке ігнорує специфіку хоча б однієї з ОС, від самого початку не може претендувати на універсальність та практичну застосованість у масштабах галузі мобільної мережевої комунікації;

– через відмінності в архітектурній організації – для контролю мережевих з'єднань застосунків використовують принципово різні технічні підходи. ОС Android допускає як рішення на рівні ядра, засновані на Netfilter/iptables, так і прикладні рішення на базі VpnService, тоді як ОС iOS надає єдиний, суворо регламентований і сертифікований Apple фреймворк NetworkExtension, що функціонує виключно в ізольованому системному розширенні;

– зростаючою значимістю мережевого екранування, як засобу захисту персональних даних користувачів. Домінуючим чинником актуальності є посилення нормативних вимог до захисту персональних даних (GDPR, CCPA, LGPD, PIPL, PDPA та ін.), які дедалі більше передбачають технічний контроль каналів передачі даних з боку застосунків на комунікаційному пристрої користувача. У цьому контексті мережевий екран мобільної ОС перестає бути виключно засобом захисту від зовнішніх атак і перетворюється на інструмент забезпечення прозорості та керованості вихідних з'єднань застосунків. У тому числі мережевих з'єднань, пов'язаних зі збором даних для подальшого використання, включаючи навчання моделей машинного навчання на серверній стороні. Подальше посилення вимог до контролю персональних даних відбулося в Європейському Союзі. З 2 серпня 2026 року набули чинності положення EU AI Act, що встановлюють вимоги до прозорості походження даних (ст. 10) та зобов'язують розробників розкривати інформацію про використання програмних модулів відстеження (SDK), прив'язаних до апаратних ідентифікаторів комунікаційного пристрою. Тим самим технічний контроль мережевих з'єднань застосунків стає не лише засобом захисту від зовнішніх загроз, а й інструментом забезпечення відповідності новому класу регуляторних вимог, що додатково підвищує практичну значущість порівняльного аналізу мережевих екранів ОС Android та iOS.

Таким чином, сукупність перелічених вище факторів підтверджує актуальність і необхідність проведення порівняльного аналізу підходів до організації роботи мережевих екранів в ОС Android та iOS з подальшою розробкою єдиної системи критеріїв їхнього порівняння.

Мета дослідження полягає у проведенні порівняльного аналізу архітектурних рішень мережевого екранування в ОС Android та iOS шляхом декомпозиції їхніх архітектур за рівнями «ядро – системні служби та розширення – користувацький застосунок», що усуває прогалину у системному описі архітектури мережевого екранування ОС iOS.

Для досягнення поставленої мети необхідно вирішити такі завдання:

– проаналізувати стан наукових публікацій щодо організації мережевих екранів в ОС Android та iOS і обґрунтувати прогалину в архітектурному аналізі мережевого екранування ОС iOS;

– виконати декомпозицію архітектури обробки мережевих з'єднань кожної ОС за рівнями – від модуля Netfilter в ОС Android і закритого ядра XNU в ОС iOS через системні служби та фреймворки (VpnService, ConnectivityManager – для ОС Android, NetworkExtension – для ОС iOS) до користувацького прикладного API;

– виявити точки розширення, доступні стороннім розробникам на кожній ОС, та оцінити їхні функціональні й експлуатаційні обмеження.

Аналіз останніх досліджень та публікацій. Питання порівняльного аналізу підходів до організації мережевих екранів в ОС Android та iOS привертає увагу дослідників з огляду на принципову відмінність у їхній архітектурній організації. ОС Android побудована на ядрі ОС Linux і успадкувала від неї відкриту та модульну модель безпеки з можливістю глибокого контролю над мережевим стеком, тоді як ОС iOS реалізує централізовану, жорстко контрольовану архітектуру з обмеженим доступом сторонніх застосунків до системних ресурсів [9].

Одним із провідних досліджень способів організації мережевого захисту в ОС Android є робота [9], де проведено методологічний якісний аналіз ризиків фреймворку ОС Android. Він показав, що ядро Linux надає вбудовані засоби пакетної фільтрації через підсистему Netfilter та утиліту iptables, що дозволяє реалізовувати повноцінне мережеве екранування на рівні ядра ОС. Це закладає принципову відмінність від ОС iOS, де подібний прямий доступ до мережевого стеку ядра стороннім розробникам не надається.

У дослідженні [10] запропоновано архітектуру WallDroid, як хмароорієнтоване рішення для реалізації мережевих екранів, специфічних для кожного застосунка на ОС Android. Воно поєднує технології тунелювання VPN (зокрема PPTP) із хмарною базою репутації застосунків для динамічного блокування шкідливих мережевих з'єднань. Це дослідження демонструє характерний для ОС Android підхід у використанні VPN-сервісу як програмного інтерфейсу для перехоплення та фільтрації мережевих з'єднань без необхідності root-доступу, оскільки прямий доступ до iptables на рівні ядра для звичайних застосунків із розвитком ОС Android був обмежений.

Порівняльний аналіз архітектурних принципів безпеки обох ОС детально викладено в дослідженні [11], де на основі систематизованих даних про вразливості за період 2015–2019 рр. показано, що відкрита, фрагментована за версіями та виробниками комунікаційних пристроїв ОС Android виявляється більш вразливою до атак і шкідливого програмного забезпечення. Централізована модель ізоляції ОС iOS забезпечує вищий рівень захищеності, де кожен застосунок працює в рамках декларативного профілю пісочниці (App Sandbox), яка примусово обмежує його доступ до системних ресурсів, а оновлення контролюються єдиним виробником, що узгоджується з висновками дослідження [6]. При цьому автори дослідження підкреслюють, що архітектурна відкритість ОС Android компенсується гнучкістю сторонніх засобів мережевого захисту, тоді як контрольованість ОС iOS обмежує можливості сторонніх розробників щодо створення повноцінних систем мережевого екранування.

Специфіку реалізації фільтрації мережевих з'єднань в ОС iOS відображає дослідження [12], де розкрито особливості фреймворку PSiOS, який пропонує реалізацію гнучкого,

визначеного користувачем або адміністратором контролю мережевого доступу застосунків. Такий підхід частково компенсує відсутність в ОС iOS штатного мережевого екрану (аналога Netfilter в ОС Android). Це дослідження ілюструє характерну для ОС iOS стратегію, яка ґрунтується на використанні програмних засобів контролю над системними викликами та профілями пісочниці (App Sandbox), що керуються через фреймворк Network Extension, замість фільтрації на рівні мережевих пакетів.

Практичні варіанти реалізації персональних мережевих екранів на основі вбудованого API VpnService в ОС Android розглянуто у дослідженні [13], де показано, що VPN-тунелювання є основним інструментом для перехоплення та аналізу мережевих з'єднань сторонніми застосунками в мобільних ОС, що не надають прямого доступу до фільтрації пакетів ядра ОС iOS.

У дослідженні [14] проведено масштабний аудит 281 популярних мобільних VPN-застосунків, що використовують вбудований API VpnService Android для реалізації функцій фільтрації та тунелювання мережевих з'єднань. Дослідження виявило суттєві недоліки у такої реалізації. Встановлено, що значна частина застосунків не забезпечує повного тунелювання мережевих з'єднань і використовує незахищені канали зв'язку та вразливі конфігурації безпеки. Це ставить під сумнів надійність програмних рішень, що працюють на рівні застосунків, для організації мережевого захисту в ОС Android порівняно з рішеннями на рівні ядра.

Таким чином, аналіз розглянутих вище наукових публікацій показує, що дослідження переважно стосуються розгляду організації мережевих екранів в ОС Android та iOS у контексті фундаментальної відмінності архітектурних моделей ядро-орієнтованого підходу з використанням Netfilter/iptables та VPN-API в ОС Android [9], [10], [13], [14] проти моделі примусового розмежування доступу на основі профілів пісочниці (App Sandbox) та фреймворку Network Extension в ОС iOS [11], [12]. Разом з тим поза полем зору більшості досліджень залишається комплексний порівняльний аналіз ефективності саме мережевих екранів в ОС Android та iOS з урахуванням сучасних версій API, що додатково зумовлює актуальність подальших досліджень у цьому напрямку.

Виклад основного матеріалу дослідження. З огляду на виявлену вище прогалину в аналізі архітектури мережевого екранування ОС iOS, необхідно послідовно розглянути архітектурні рішення мережевого екранування кожної з ОС від рівня ядра через системні служби та розширення до рівня користувацького застосунка. Це становить необхідну основу для подальшої розробки єдиної методології їхнього порівняння.

1. Архітектура мережевого екрану в ОС Android

ОС Android інтегрувала в себе модифіковане ядро Linux, що зумовило архітектуру вбудованих засобів фільтрації мережевих з'єднань. Мережевий екран на рівні ядра реалізується через підсистему Netfilter – компонент ядра Linux, що реалізує функції пакетної фільтрації та відстеження мережевих з'єднань [15].

Зміна політик фільтрації традиційно вимагала root-доступу для керуючої програми, що практично реалізовувалося через утиліту iptables ядра Linux 2.6, яка використовує фреймворк Netfilter для фільтрації, модифікації та відстеження стану мережевих з'єднань, включаючи [16]:

- фільтрацію вхідних, вихідних та транзитних пакетів;
- зіставлення за полями протоколу (IP-адреси відправника/одержувача, тип протоколу);
- інспекцію з урахуванням стану мережевого з'єднання;
- різні варіанти обробки пакета (відкидання, відхилення з ICMP-повідомленням, дозвіл).

Таким чином, архітектурно мережевий екран ОС Android спочатку проектувався як дворівнева система: ядро (Netfilter/iptables – функціональне ядро фільтрації) та користувацький простір (системні служби, що формують правила). Загальна схема такої взаємодії представлена на Рис. 1.

Ключовим елементом простору користувача є системна служба простору користувача netd, через яку компоненти ОС Android формують і застосовують мережеві політики для кожного застосунка. У рамках цієї архітектури системна служба простору користувача netd

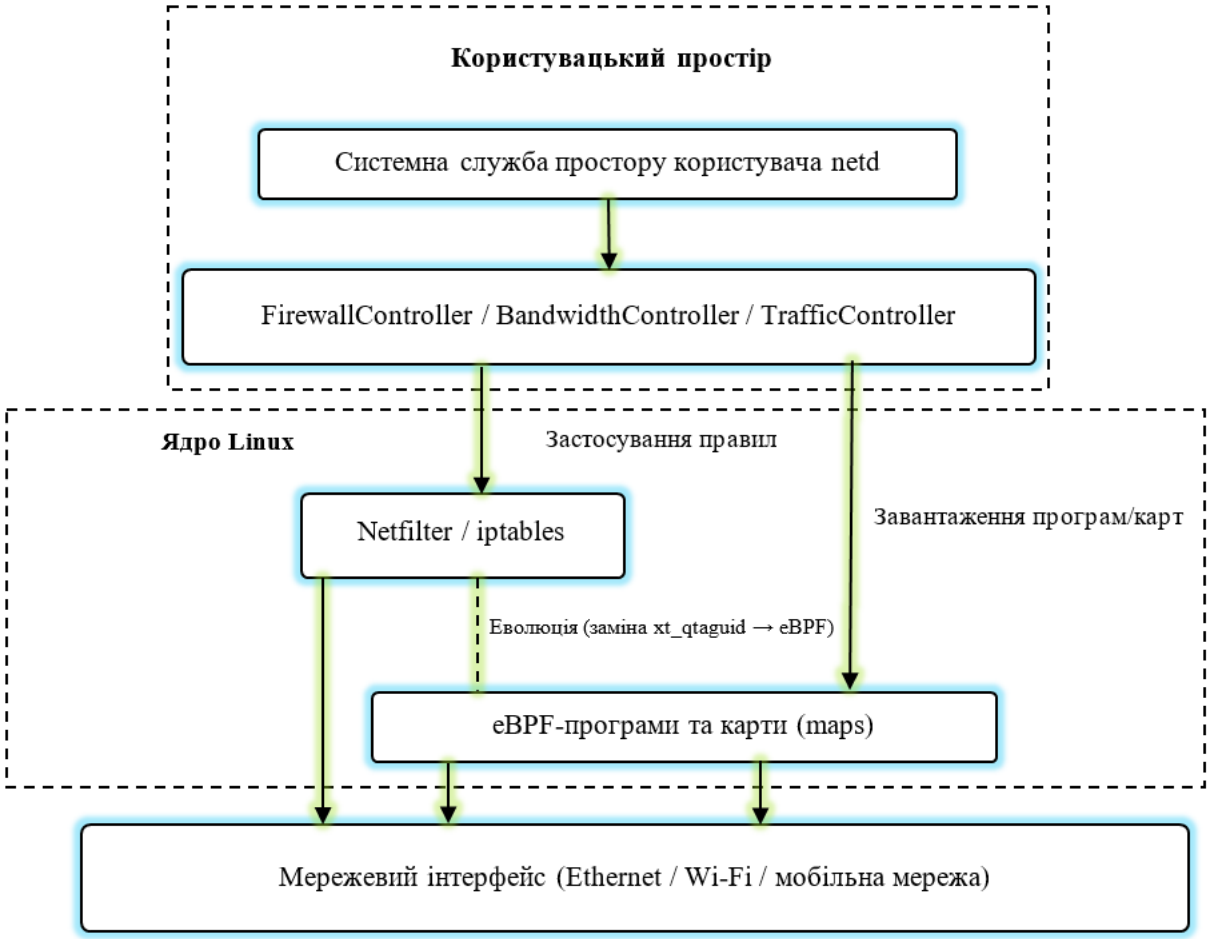

Рис. 1. **Дворівнева архітектура мережевого екрану ОС Android (ядро / користувацький простір)**

Джерело: **розроблено авторами**

використовує технологію eBPF для реалізації пакетної фільтрації та обліку мережевих з'єднань на рівні сокета, що дозволяє застосовувати політики фільтрації у прив'язці до кожного застосунка без необхідності привілейованого налаштування на рівні ядра ОС [17]. Така організація дозволяє ОС Android застосовувати диференційовані мережеві обмеження не до комунікаційного пристрою в цілому, а до його окремих застосунків – наприклад, блокувати фонові мережеві з'єднання в режимі енергозбереження або обмежувати мережевий доступ у режимі роумінгу. Загальна схема архітектури фільтрації мережевих з'єднань за UID застосунка наведена на Рис. 2.

Важливим етапом розвитку архітектури фільтрації мережевих з'єднань стала відмова від модуля ядра xt_qtaguid ОС Android. Спочатку облік кількості мережевих з'єднань та застосування мережевих політик для кожного застосунка спиралися саме на модуль xt_qtaguid у складі Netfilter, який відстежує мережеві з'єднання на рівні сокета для кожного застосунка за його унікальним UID, доповнений модулем quota2 для застосування іменованих квот мережевих з'єднань [18]. Однак цей модуль становив значний обсяг коду поза основним деревом ядра ОС Android, що створювало суттєве навантаження на його супровід, стабільність та продуктивність [19]. Це зумовило подальший перехід на технологію extended Berkeley Packet Filter (eBPF). У результаті застарілий, специфічний для ОС Android модуль ядра xt_qtaguid було замінено збором статистики мережевих з'єднань за допомогою eBPF [19]. Контролер TrafficController та eBPF-програма, що підключається ядром до точок

обробки мережевих операцій сокета групи процесів, об’єднаних загальним обмеженням ресурсів на рівні ядра, окрім статистики мережевих з’єднань, також відповідають за їх блокування для окремих UID застосунків залежно від налаштувань комунікаційного пристрою. Цей режим блокування налаштовується за допомогою окремих карт для режимів енергозбереження, очікування та «сплячого» стану застосунків [19]. Ці зміни відображені пунктирною лінією (див. Рис. 1) і повністю ілюструються схемою (див. Рис. 2), де показано, як єдина точка входу (netd) розподіляє управління між кількома спеціалізованими контролерами.

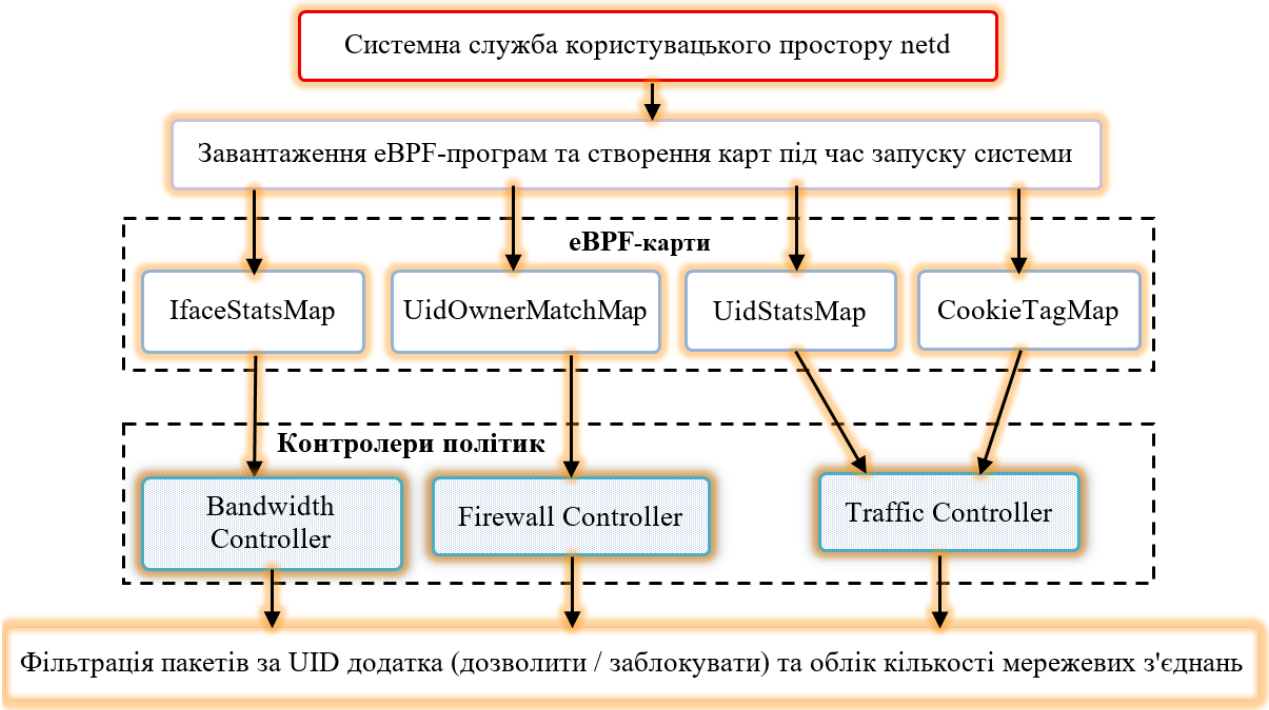

Рис. 2. **Архітектура фільтрації мережевих з'єднань за UID застосунка на основі netd та eBPF**

Джерело: **розроблено авторами**

Окрему архітектурну гілку утворюють мережеві екрани, що працюють без root-доступу. Оскільки пряме керування правилами iptables недоступне звичайним застосункам, для цього класу рішень використовується системний клас android.net.VpnService. Використання VpnService дозволяє реалізувати перехоплення мережевих з’єднань на незміненій ОС без спеціальних прав користувача, однак накладає обмеження. Одночасно на комунікаційному пристрої може бути активною лише одна така служба, а перехоплені IP-пакети застосунок змушений повторно інкапсулювати в TCP/UDP-пакети для передачі в комунікаційну мережу, а не транслювати їх безпосередньо [20]. Технічно цей підхід ґрунтується на створенні віртуального мережевого інтерфейсу (TUN), у який система спрямовує мережеві з’єднання для подальшого аналізу та фільтрації застосунком у просторі користувача. Практична реалізація такої архітектури розглянута в дослідженні [21], де детально описано процес моніторингу мережевого доступу застосунків в ОС Android, яка не містить вбудованої функції керування їхніми мережевими з’єднаннями. Контроль мережевого доступу застосунків у ОС Android, яка не надає штатних засобів керування їхніми з’єднаннями, може реалізовуватися двома шляхами, через ручне обмеження системних служб із використанням root-доступу або через непривілейовані застосунки, що створюють VPN-підключення та надають користувачеві інтерфейс для індивідуального налаштування дозволів на мережевий доступ кожного застосунка.

Цей підхід лежить в основі широко використовуваних непривілейованих рішень (NetGuard, Mobiwol, NoRoot Firewall, InternetGuard, RethinkDNS та аналогічних) і схематично представлений на Рис. 3.

Більш повну реалізацію такої моделі, що поєднує перехоплення мережевих з'єднань із гнучким управлінням правилами, демонструє архітектура, запропонована у дослідженні [22]. В основі цієї архітектури лежить конфігурація програмних компонентів, кожен з яких

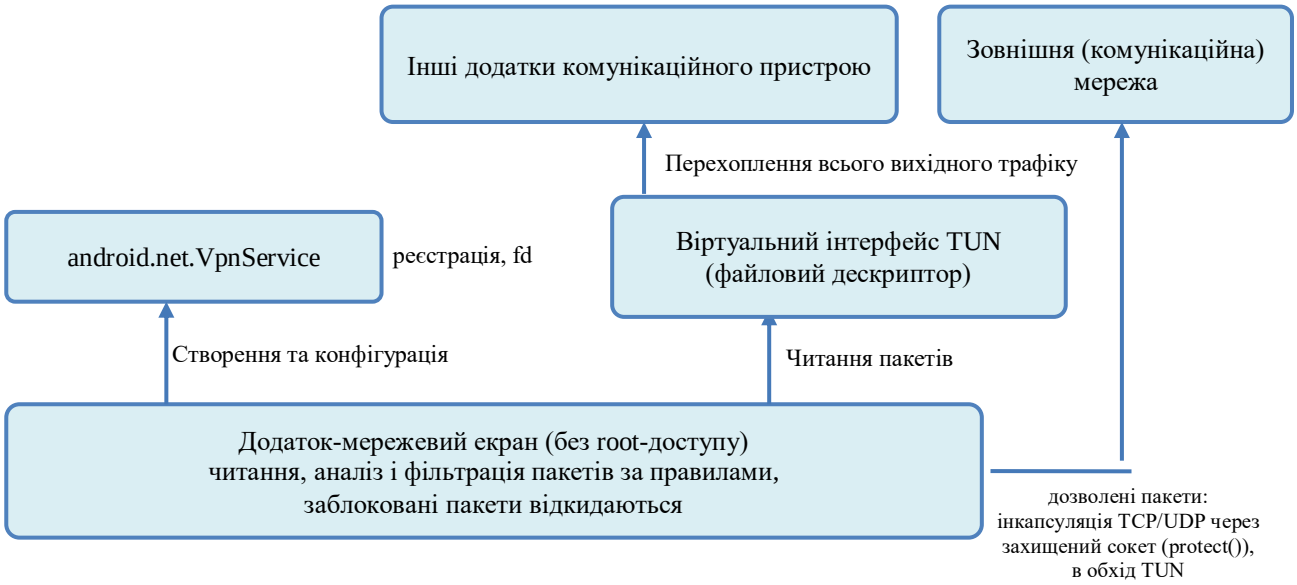

Рис. 3. **Архітектура непривілейованого мережевого екрану на основі VpnService**
Джерело: **розроблено авторами**

реалізує окрему функцію обробки пакета, що дозволяє гнучко переносити логіку перехоплення та прийняття рішень між рівнем ядра ОС та рівнем управління правилами у користувацькому застосунку. Стосовно персональних мережевих екранів ОС Android такий підхід реалізується у вигляді гібридної архітектури, що поєднує елементи обох розглянутих вище моделей – як із root-доступом (пряма взаємодія з підсистемою Netfilter), так і без таких обмежень на рівні користувацького управління правилами.

Узагальнюючи розглянуті підходи, можна виділити три архітектурні моделі мережевого екрану в ОС Android:

– модель на основі Netfilter/iptables/eBPF (Рис. 1), що забезпечує максимальну продуктивність і повноту контролю за рахунок необхідності привілейованого доступу або інтеграції на рівні ОС;

– модель на основі технології eBPF у складі ОС (Рис. 2) є сучасним підходом AOSP, що забезпечує фільтрацію та облік мережевих з'єднань за UID застосунка без модифікації ядра ОС [17], [18],[19];

– непривілейовану модель на основі VpnService (Рис. 3), як підхід, доступний стороннім розробникам на комунікаційних пристроях без root-доступу, але обмежений продуктивністю інкапсуляції пакетів та неможливістю одночасної роботи кількох подібних служб [20], [21], [22].

Слід окремо підкреслити межу доступності розглянутих моделей. Пряма модифікація правил Netfilter, модуль xt_qtaguid та керовані ядром eBPF-карти належать до системного рівня і рівня системних служб (netd, TrafficController), недоступного застосунку стороннього розробника без root-доступу. Єдиним рівнем, відкритим для стороннього розробника без розширених прав, є непривілейований API VpnService.

2. Архітектура мережевого екрану в ОС iOS

На відміну від класичних ОС, де мережевий екран реалізується як самостійний компонент ядра (наприклад, iptables/netfilter в ОС Linux), архітектура ОС iOS не передбачає

доступу користувача до таблиць маршрутизації та фільтрації пакетів на рівні ядра. Ядром ОС iOS є XNU (X is Not Unix) – гібридне ядро, що поєднує мікроядро Mach, яке відповідає за керування процесами, потоками та їхньою взаємодією, і шар BSD (Berkeley Software Distribution), що реалізує класичну Unix-підсистему (файлову систему, мережевий стек, модель користувачів і процесів) [23]. Саме в шарі BSD ядра XNU розташована інфраструктура контролю доступу. Це зумовлено загальною моделлю безпеки, побудованою на принципі мінімально необхідних привілеїв та суворої ізоляції застосунків [24]. Функції мережевого екрану в ОС iOS реалізуються не як окрема підсистема, а розподілені між кількома взаємопов'язаними рівнями архітектури ОС.

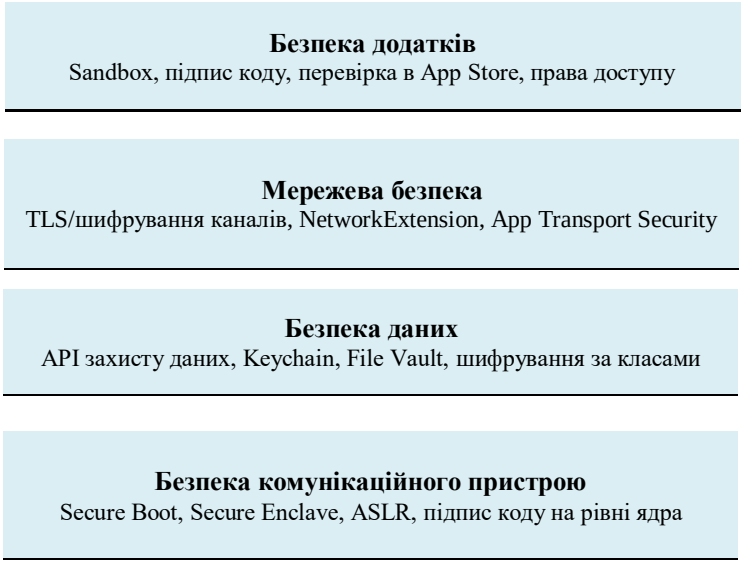

Рис. 4. **Чотирирівнева модель безпеки ОС iOS**
Джерело: **розроблено авторами**

Згідно з загальноприйнятою класифікацією, захист даних користувача в ОС iOS забезпечується чотирма взаємодоповнюючими рівнями [25] (Рис. 4):

- безпекою застосунків;
- мережевою безпекою;
- безпекою даних;
- безпекою комунікаційного пристрою.

Рівень безпеки застосунків відповідає за ізоляцію процесів один від одного за допомогою пісочниці (App Sandbox).

Рівень мережевої безпеки відповідає за шифрування мережевих з'єднань, що передаються комунікаційними каналами зв'язку.

Взаємодія рівня безпеки застосунків та рівня мережевої безпеки формує функціональний аналог мережевого екрану таким чином, що контроль вихідних мережевих з'єднань здійснюється не централізованим модулем фільтрації пакетів, а сукупністю компонентів обов'язкового контролю доступу, вбудованих у ядро XNU та реалізованих через профілі пісочниці (App Sandbox).

Рівень безпеки даних захищає дані безпосередньо на комунікаційному пристрої, де кожен файл шифрується індивідуальним ключем, похідним від пароля користувача та апаратного ідентифікатора чіпа Secure Enclave – ізольованого співпроцесора з власним незалежним циклом безпечного завантаження, який виконує криптографічні операції Data Protection самостійно, навіть якщо зловмисник отримає контроль над основним ядром ОС [26].

Рівень безпеки комунікаційного пристрою забезпечує довіру до самого пристрою як фізичного та програмного об'єкта – через біометричну/парольну автентифікацію, ланцюжок довіреного завантаження, яка поетапно перевіряє криптографічний підпис кожного компонента програмного стека під час увімкнення комунікаційного пристрою, починаючи з

незмінного коду початкового завантаження, записаного в постійну пам'ять процесора (Boot ROM) [26], та засоби дистанційного керування (блокування, стирання даних у разі втрати).

Ядро XNU об'єднує два компоненти – мікроядро Mach, що відповідає за низькорівневе керування процесами та їхньою взаємодією, і шар BSD, який реалізує класичний мережевий стек – протоколи TCP/IP і сокети, через які додатки встановлюють мережеві з'єднання [23]. Однак, перш ніж застосунок зможе скористатися цим мережевим стеком, він проходить перевірку на вищому рівні. Система перевіряє цифровий підпис коду та політику пісочниці (App Sandbox), призначену кожному процесу. Саме ця політика визначає, які системні виклики дозволено виконувати даному застосунку, – зокрема, чи може він взагалі створювати мережеві сокети та які мережеві з'єднання встановлювати. Звідси випливає ключове відмінність від класичного мережевого екрану – рішення «пропустити чи заблокувати» приймається не тоді, коли вже сформований пакет проходить через мережевий стек ядра ОС, а набагато раніше – у момент, коли застосунок лише намагається виконати системний виклик для відкриття мережевого з'єднання. Якщо політика пісочниці (App Sandbox) це забороняє, виклик блокується ядром ОС XNU ще до того, як будь-які дані взагалі покинуть комунікаційний пристрій.

Слід уточнити співвідношення згаданих компонентів, щоб уникнути враження їх прямої тотожності. Пакетний фільтр PF є компонентом шару BSD ядра XNU і безпосередньо стороннім розробникам недоступний. Фреймворк NetworkExtension є прикладним API, що дозволяє розробнику зареєструвати власне розширення-фільтр, проте не надає доступу до самого PF і не є його «обгорткою». Системний VPN-тунель utun – це механізм перехоплення мережевих з'єднань на рівні мережевого стеку, через який фізично проходять пакети вже після винесення вердикту модулями NetworkExtension. Таким чином, PF, NetworkExtension та utun є трьома різними, паралельно співіснуючими елементами архітектури, а не послідовними рівнями одного конвеєра обробки мережевих з'єднань.

Повноцінна реалізація користувацького мережевого екрану в ОС iOS стала можливою лише з появою фреймворку NetworkExtension [25] (Рис. 5).

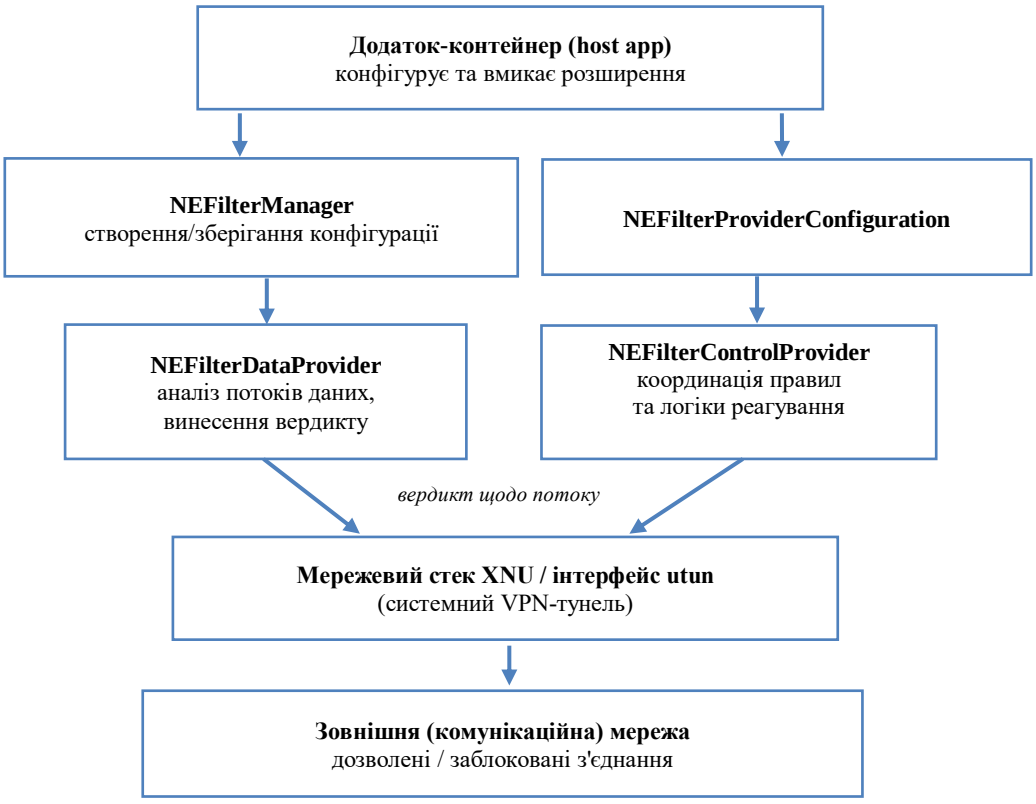

Рис. 5. **Архітектура фреймворка NetworkExtension**
Джерело: **розроблено авторами**

Структурна схема взаємодії компонентів фреймворку NetworkExtension відображає життєвий цикл мережевого фільтра. Від ініціалізації мережевого з'єднання у застосунку до винесення позитивного вердикту з дозволом виходу у зовнішню (комунікаційну) мережу.

Запускає процес застосунок-контейнер, який налаштовує та вмикає розширення фільтрації через системний API. Від нього управління розподіляється між двома модулями: NEFilterManager відповідає за реєстрацію розширення та вмикання/вимикання фільтрації загалом, а NEFilterProviderConfiguration відповідає за параметри самого фільтра (які мережеві з'єднання підлягають аналізу). Модуль NEFilterManager керує життєвим циклом розширення, модуль NEFilterProviderConfiguration керує змістовними правилами. Обидва налаштування ініціюються застосунком-контейнером host app, але обробляються незалежно. Далі кожен модуль передає управління своєму виконуючому компоненту. NEFilterDataProvider аналізує потоки даних і виносить щодо них рішення – дозволити чи заблокувати. Паралельно NEFilterControlProvider координує правила між фільтрами та відповідає за реакцію системи під час блокування (повідомлення, перенаправлення). Обидва виходи обох модулів сходяться в загальній точці – «вердикт щодо потоку», що формує сукупний результат (дозволити/заблокувати).

Винесений вердикт передається у мережевий стек XNU через інтерфейс utun (системний VPN-тунель). Це відображає технічну реалізацію перехоплення, де фреймворк NetworkExtension не вбудовується безпосередньо у мережевий стек ядра, а використовує віртуальний тунельний інтерфейс, через який фізично проходять усі мережеві з'єднання застосунка або ОС і на цьому етапі або пропускаються, або блокуються відповідно до винесеного вердикту. До зовнішньої (комунікаційної) мережі в результаті доходять лише дозволені мережеві з'єднання.

Таким чином, фільтрація в ОС iOS розподілена між конфігураційними модулями (NEFilterManager, NEFilterProviderConfiguration) та виконавчими модулями (NEFilterDataProvider, NEFilterControlProvider) фреймворку NetworkExtension, а рішення застосовується не на рівні пакета під час його пересилання через мережевий стек, а на рівні тунельного інтерфейсу utun, вбудованого в мережевий стек ядра XNU.

Практично мережевий екран в ОС iOS реалізується шляхом створення системного VPN-тунелю (інтерфейс utun), через який перенаправляються всі мережеві з'єднання комунікаційного пристрою для подальшого аналізу на рівні застосунка або локального проксі-сервера [27] (Рис. 6).

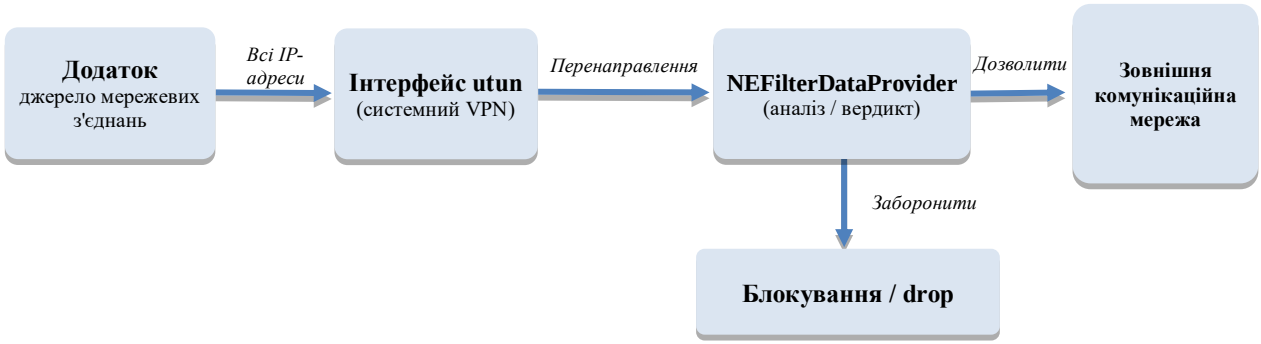

Рис. 6. **Проходження мережевих з'єднань застосунка через VPN-тунель під час фільтрації**

Джерело: **розроблено авторами**

У новітніх версіях iOS API розширено підтримкою фільтрації за повним URL, а не лише за іменем хоста, з використанням технологій приватного пошуку інформації (Private Information Retrieval) та Privacy Pass, що зменшує обсяг метаданих, доступних самому фільтру.

Аналогічно щодо ОС Android, і в ОС iOS слід розмежувати рівень, повністю закритий для стороннього розробника (ядро XNU, пакетний фільтр PF, безпосереднє керування таблицями маршрутизації), та рівень, доступний через публічний API (реєстрація

розширення NEFilterManager, налаштування правил NEFilterProviderConfiguration, обробка потоків NEFilterDataProvider та NEFilterControlProvider). Лише останній рівень може використовуватися розробниками застосунків мережевого захисту, тоді як сам пакетний фільтр PF і мережевий стек ядра залишаються виключно системним ресурсом.

Починаючи з ОС iOS 26, API розширено підтримкою фільтрації за повним URL, а не лише за іменем хоста, з використанням технологій приватного пошуку інформації (Private Information Retrieval) та Privacy Pass, що зменшує обсяг метаданих, доступних самому фільтру.

Така архітектура принципово відрізняється від класичних програмних мережевих екранів, де фільтрація пакетів, перевірка стану мережевих з'єднань та робота на рівні шлюзу застосунків виконуються єдиним модулем із прямим доступом до мережевого стеку [26], [27]. В ОС iOS усі рішення щодо мережевих з'єднань приймаються у просторі користувача, в ізольованому процесі-розширенні, що знижує ризик отримання зловмисником контролю над ядром ОС, але водночас обмежує продуктивність і глибину аналізу. Зокрема, відсутня можливість прямого зчитування таблиць маршрутизації для перевірки того, що всі мережеві з'єднання дійсно проходять через VPN-тунель [27].

3. Порівняльний аналіз архітектур мережевого екранування ОС Android та iOS

Узагальнення результатів проведеної декомпозиції архітектур мережевого екранування ОС Android та ОС iOS систематизує отримані результати за трьома розглянутими рівнями: ядром, системними службами та розширеннями, а також користувацьким застосунком (Таблиця 1).

Таблиця 1. **Порівняння архітектурних рівнів організації мережевого екранування в ОС Android та iOS**

Рівень архітектури	Архітектурний рівень контролю (Android)	Архітектурний рівень контролю (iOS)	Привілеї	Модель ізоляції	Рівень доступу та контролю	Типові сценарії використання
Ядро ОС	Netfilter/ iptables/ eBPF	Пакетний фільтр PF (BSD-шар XNU)	Системні/ root (Android); тільки ОС (iOS)	Код ядра	Прямого доступу немає (без системних прав)	Системні політики, енергозбереження, MDM
Системні служби / розширення	netd, FirewallController, TrafficController, eBPF - карти	NetworkExtension (NEPacket TunnelProvider, NEFilterData Provider, NEAppProxy Provider)	Системні (netd) або спеціальні entitlements	Окремий системний процес/розширення	Обмежений доступ через публічні API	Корпоративний захист, системна фільтрація
Застосунок користувача	VpnService (TUN-інтерфейс)	Host -застосунок + ізольоване розширення NetworkExtension	Без root (VpnService); без root + підпис Apple (Network Extension)	Пісочниця UID (Android); окрема пісочниця App Sandbox розширення (iOS)	Повне перехоплення трафіку в рамках одного активного тунелю	Персональний firewall, VPN, батьківський контроль, DPI-рішення

Джерело: **розроблено авторами на основі результатів декомпозиції, викладених у розділах 1 та 2**

Для кожного з цих рівнів (див. Таблицю 1) зіставлено доступні компоненти та привілеї, модель ізоляції виконання, наявні точки розширення та пов'язані з ними обмеження, а також типові сценарії практичного застосування. Таке узагальнення дає змогу наочно простежити, яким чином архітектурні особливості кожної ОС визначають можливості та обмеження розробників застосунків мережевого захисту на кожному з рівнів.

Висновки. У дослідженні виконано декомпозицію архітектур мережевого екранування ОС Android та iOS на порівнянні рівні – ядро, системні служби та розширення, користувацький застосунок, що дозволило розглянути архітектуру фреймворку NetworkExtension ОС iOS як самостійний предмет аналізу, а не в загальному ряді характеристик безпеки, як це робилося в дослідженнях [5], [6], [7], [8]. Тим самим усунуто

прогалину, зафіксовану у постановці проблеми, а саме різницю між детальною опрацьованістю архітектури мережевого екрану ОС Android [1], [2], [3], [4] та фрагментарністю аналогічних досліджень для ОС iOS.

Науковим результатом дослідження є запропонована триярусна декомпозиційна модель мережевого екранування («ядро – системні служби та розширення – користувацький застосунок»), яка вперше застосована симетрично до обох ОС і дозволила виокремити архітектуру фреймворку NetworkExtension ОС iOS як самостійний предмет аналізу, а не як один з елементів загального переліку характеристик безпеки, що мало місце у дослідженнях [5], [6], [7], [8].

Для забезпечення зіставності результатів декомпозиція обох архітектур виконана за такою системою критеріїв:

– відкритість ядра (можливість дослідження та модифікації коду фільтрації);

– кількість і спеціалізація точок розширення, доступних стороннім розробникам;

– модель ізоляції виконання фільтрувального коду (у складі ядра, системної служби або окремого процесу-розширення);

– необхідний рівень привілеїв (root-доступ, привілейована системна служба, непривілейований застосунок);

– можливість стороннього (не апаратного чи системного) контролю мережевих з'єднань;

– наявність обмежень на одночасну роботу кількох незалежних фільтрів.

Саме за цими критеріями далі послідовно розглядаються архітектури мережевого екранування ОС Android та ОС iOS і виконано їх зведене зіставлення.

Показано, що архітектура мережевого екрану ОС Android передбачає три моделі реалізації:

– ядро-орієнтовану модель на основі Netfilter/iptables/eBPF, яка забезпечує максимальну повноту контролю за рахунок необхідності привілейованого доступу;

– модель на основі технології eBPF у складі платформи AOSP, що застосовується системним сервісом netd для фільтрації та обліку мережевих з'єднань за UID застосунка без модифікації ядра;

– непривілейовану модель на основі VpnService, доступну стороннім розробникам на пристроях без root-доступу, але обмежену продуктивністю інкапсуляції пакетів та неможливістю одночасної роботи кількох таких служб.

Встановлено, що архітектура мережевого екрану ОС iOS принципово відрізняється від класичної моделі пакетної фільтрації. Рішення «пропустити чи заблокувати» приймається не під час проходження пакета через мережевий стек, а на рівні системного виклику, у момент спроби застосунка встановити мережеве з'єднання, на підставі політики пісочниці (App Sandbox), вбудованої в ядро XNU. Повноцінна реалізація користувацького мережевого екрану можлива лише через фреймворк NetworkExtension, у якому конфігураційні модулі (NEFilterManager, NEFilterProviderConfiguration) та виконавчі модулі (NEFilterDataProvider, NEFilterControlProvider) розподіляють функції реєстрації фільтра та винесення вердикту щодо кожного потоку, а рішення застосовується на рівні тунельного інтерфейсу utun.

Порівняння розглянутих архітектур мережевих екранів в ОС Android та iOS дозволяє зробити висновок, що їхня відмінність проявляється на трьох рівнях.

По-перше, на рівні ядра ОС Android використовує відкриту та задокументовану в проекті AOSP підсистему Netfilter/eBPF, тоді як ядро XNU в ОС iOS та його фільтр PF закриті для зовнішнього дослідження.

По-друге, на рівні API для розробників ОС Android надає єдину недиференційовану точку розширення (VpnService), тоді як ОС iOS диференціює три спеціалізовані типи розширень під різні сценарії використання (тунелювання, проксіювання, фільтрація контенту).

По-третє, на рівні ізоляції виконання модель ОС iOS спочатку проектувалася з окремим процесом-розширенням та обмежувальною пісочницею (App Sandbox), тоді як в ОС Android фільтруючий код повністю покладається на стандартну модель ізоляції застосунків за UID.

Спільним для обох ОС архітектурним обмеженням є заборона на одночасну роботу декількох незалежних застосунків-фільтрів, що перехоплюють усі мережеві з'єднання комунікаційного пристрою. В ОС Android – через обмеження на кількість активних екземплярів VpnService, в ОС iOS – через обмеження на кількість активних провайдерів кожного типу розширення на профіль конфігурації. Це системне обмеження необхідно враховувати під час проектування кросплатформних рішень мережевого захисту комунікаційних пристроїв.

Технічно в ОС Android це обмеження випливає з того, що вона дозволяє активувати лише один TUN-інтерфейс, створений через VpnService, одночасно. Встановлення другого VPN-застосунку автоматично розриває з'єднання, встановлене першим. В ОС iOS обмеження діє на рівні профілю конфігурації NetworkExtension. Одночасно, активним може бути лише один провайдер, кожного типу розширення (наприклад, лише один Content Filter Provider), що унеможливлює паралельну роботу двох незалежних застосунків-фільтрів одного класу.

Практичним наслідком для користувача є необхідність обирати між функціями (наприклад, VPN для приватності та застосунок батьківського контролю) у межах одного пристрою, а для розробника потрібно або інтегрувати декілька функцій мережевого захисту у єдиний застосунок, або передбачати коректну обробку відмови в реєстрації через активний провайдер.

Отримані результати декомпозиції архітектур ОС Android та iOS формують необхідну та достатню основу для подальшої розробки єдиної методології та системи критеріїв їх порівняльного аналізу, а також для порівняльної оцінки існуючих реалізацій мережевих екранів за цими критеріями, що є предметом окремого дослідження.

Зокрема, подальше дослідження передбачає:

– формалізацію наведеної вище системи критеріїв у вигляді кількісно та якісно оцінюваної порівняльної моделі;

– емпіричний аналіз репрезентативної вибірки застосунків трьох типів – персональних мережевих екранів (NetGuard, Mobiwol, RethinkDNS та аналогічних), корпоративних рішень контролю мережевих з'єднань та застосунків батьківського контролю;

– використання таких метрик ефективності та безпеки, як повнота тунелювання мережевих з'єднань (частка з'єднань, що обходять фільтр), затримка обробки пакета, енергоспоживання фонового процесу фільтрації та стійкість до технік обходу фільтра, описаних у джерелах [14] для ОС Android та [27] для ОС iOS.

Практичним наслідком виявлених архітектурних відмінностей для розробки кросплатформних рішень мережевого захисту є розмежування функцій на дві групи. До функцій, які можливо реалізувати уніфіковано на обох ОС, належать перехоплення мережевих з'єднань через віртуальний тунельний інтерфейс (TUN в ОС Android, utun в ОС iOS) та ухвалення рішення про блокування на рівні застосунка-посередника без модифікації ядра чи root/jailbreak-доступу. До функцій, що є принципово платформозалежними, належать пряма робота з правилами Netfilter/iptables та збір статистики мережевих з'єднань засобами eBPF за UID застосунка (доступно лише в ОС Android), а також диференціація типів розширень NetworkExtension під сценарії використання, зокрема тунелювання, проксіювання, фільтрація вмісту (доступно лише в ОС iOS). Відповідно, ядро кросплатформного рішення доцільно проєктувати на основі спільного знаменника VPN-тунелювання, тоді як платформоспецифічні можливості слід виносити в окремий, необов'язковий модуль.

Конфлікт інтересів: Автори декларують, що не мають конфлікту інтересів, зокрема фінансового, особистого, авторського чи будь-якого іншого характеру, який міг би вплинути на дослідження, а також на результати, опубліковані в цій статті

Фінансування: Дослідження проводилося без фінансової підтримки

Доступність даних: Усі дані доступні в цифровій або графічній формі в основному тексті рукопису

Використання засобів штучного інтелекту (ШІ): Автори підтверджують, що не застосовували інструментальні засоби ШІ для написання цієї роботи

СПИСОК ЛІТЕРАТУРИ

1. Wang, B., Lu, K. & Chang, P. “Design and implementation of Linux firewall based on the frame of Netfilter/Iptable”. *11th International Conference on Computer Science & Education (ICCSE)*. Nagoya Universiti, Japan. 2016. p. 949–953. DOI: https://doi.org/10.1109/ICCSE.2016.7581711.

2. Diekmann, C., Michaelis, J., Haslbeck, M. W. & Carle, G. “Verified iptables firewall analysis”. *IFIP Networking Conference, Networking and Workshops*. Vienna, Austria. 2016. p. 252–260. DOI: https://doi.org/10.1109/IFIPNetworking.2016.7497196.

3. Diekmann, C., Hupel, L., Michaelis, J., Haslbeck, M. W. & Carle, G. “Verified iptables Firewall Analysis and Verification”. *Journal of Automated Reasoning*. 2018; 61: 191–242. DOI: https://doi.org/10.1007/s10817-017-9445-1.

4. Staroletov, S. “Software architecture for an intelligent firewall based on Linux netfilter”. *25th Conference on Innovation in Clouds, Internet and Networks (ICIN)*. 2022. p. 160–162. DOI: https://doi.org/10.1109/ICIN53892.2022.9758121.

5. Mohamed, I. & Patel, D. “Android vs iOS security: A comparative study”. *12th International Conference on Information Technology – New Generations*. 2015. p. 725–730. DOI: https://doi.org/10.1109/ITNG.2015.123.

6. Yousif, A., Haider, M., Thirunavukkarasu, K. & Nand, P. “A review of Android and iOS operating system security”. *International Conference on Emerging Technologies for Sustainable Industry*. 2022. DOI: https://doi.org/10.1109/ICETSIS55481.2022.9888847.

7. Kununka, S., Mehandjiev, N. & Sampaio, P. “A comparative study of Android and iOS mobile applications' data handling practices versus compliance to privacy policy”. *Privacy and Identity Management. The Smart Revolution. IFIP Advances in Information and Communication Technology*. *Springer*. 2018; 526: 301–313. DOI: https://doi.org/10.1007/978-3-319-92925-5_20.

8. Reinfelder, L., Benenson, Z. & Gassmann, F. “Android and iOS users' differences concerning security and privacy”. *Extended Abstracts on Human Factors in Computing Systems*. 2013. p. 817–822. DOI: https://doi.org/10.1145/2468356.2468502.

9. Shabtai, A., Fledel, Y., Kanonov, U., Elovici, Y., Dolev, S. & Glezer, C. “Google Android: A comprehensive security assessment”. *IEEE Security & Privacy*. 2010: 8 (2): 35–44. DOI: https://doi.org/10.1109/MSP.2010.2.

10. Kilinc, C., Booth, T. & Andersson, K. “WallDroid: Cloud assisted virtualized application specific firewalls for the Android OS”. *Proceedings of the 11th Int. Conf. on Trust, Security and Privacy in Computing and Communications (TrustCom)*. 2012. p. 877–883. DOI: https://doi.org/10.1109/TrustCom.2012.298.

11. Garg, S. & Baliyan, N. “Comparative analysis of Android and iOS from security viewpoint”. *Computer Science Review*. 2021; 40: 100372. DOI: https://doi.org/10.1016/j.cosrev.2021.100372.

12. Werthmann, T., Hund, R., Davi, L., Sadeghi, A.-R. & Holz, T. “PSiOS: bring your own privacy & security to iOS devices”. *Proceedings of the 8th ACM SIGSAC Symposium on Information, Computer and Communications Security (ASIA CCS '13)*. 2013. p. 13–24. DOI: https://doi.org/10.1145/2484313.2484316.

13. Liu, Z. “Application and security analysis of virtual private network (VPN) in network communication”. *Academic Journal of Computing & Information Science*. 2023; 6 (11): 52–59. DOI: https://doi.org/10.25236/AJCIS.2023.061108.

14. Wang, W., Ortwein, A., Sobrados, E., Stanley, R., Sharma, P. K., Anwar, A. & Ensafi, R. “MVPNalyzer: An investigative framework for auditing the security & privacy of mobile VPNs”. *Network and Distributed System Security Symposium*. 2026. p. 1–17. DOI: https://doi.org/10.14722/ndss.2026.231573.

15. Shabtai, A., Mimran, D. & Elovici, Y. “Evaluation of security solutions for Android systems”. arXiv. 2015. DOI: https://doi.org/10.48550/arXiv.1502.04870.

16. Shabtai, A., Fledel, Yu., Kanonov, U., Elovici, Yu. & Dolev, S. “Google Android: A state-of-the-art review of security mechanisms”. arXiv. 2009. p. 1–42. DOI: https://doi.org/10.48550/arXiv.0912.5101.

17. Scholz, D., Raumer, D., Emmerich, P., Kurtz, A., Lesiak, K. & Carle, G. “Performance implications of packet filtering with Linux eBPF”. *30th International Teletraffic Congress (ITC 30), IEEE*. 2018; 1: 209–217. DOI: https://doi.org/10.1109/ITC30.2018.00039.

18. Sufatrio, Tan D. J. J., Chua T. & Thing V. L. L. “Securing Android: A survey, taxonomy, and challenges”. *ACM Computing Surveys*. 2015; 47 (4): 58. DOI: https://doi.org/10.1145/2733306.

19. Vieira. M. A. M., Castanho, M. S., Pacífico, R. D. G., Santos, E. R. S., Câmara Júnior, E. P. M. & Vieira, L. F. M. “Fast packet processing with eBPF and XDP: Concepts, code, challenges, and applications”. *ACM Computing Surveys*, 2020; 53 (1): 1–36. DOI: https://doi.org/10.1145/3371038.

20. Song, Y. & Hengartner, U. “PrivacyGuard: A VPN-based platform to detect information leakage on android devices”. *Proceedings 5th Annual ACM CCS Workshop on Security and Privacy in Smartphones and Mobile Devices (SPSM)*. 2015. p. 15–26. DOI: https://doi.org/10.1145/2808117.2808120.

21. Le, A., Varmarken, J., Langhoff, S., Shuba, A., Gjoka, M. & Markopoulou A. “AntMonitor: A system for monitoring from mobile devices” *Proceedings ACM SIGCOMM Workshop on Crowdsourcing and Crowdsharing of Big (Internet) Data (C2BID)*. 2015. p. 15–20. DOI: https://doi.org/10.1145/2787394.2787396.

22. Kohler, E., Morris, R., Chen, B., Jannotti, J. & Kaashoek, M. F. “The click modular router. *ACM Transactions on Computer Systems*, 2000; 18 (3): 263–297. DOI: https://doi.org/10.1145/354871.354874.

23. Watson, R. N. M. A. “Decade of OS access-control extensibility”. *Communications of the ACM*. 2013; 56 (2): 52–63. DOI: https://doi.org/10.1145/2408776.2408792.

24. Halilovic, M. & Subasi, A. “Intrusion detection on smartphones”. *International Burch University*. Sarajevo. 2012. DOI: https://doi.org/10.48550/arXiv.1211.6610.

25. Maier, E.-M., Tanczer, L. M. & Klausner, L. D. “Surveillance disguised as protection: A comparative analysis of sideloaded and in-store parental control apps”. *Proceedings on Privacy Enhancing Technologies*. 2025; 2: 107–124. DOI: https://doi.org/10.56553/popets-2025-0052.

26. Shepherd, C., Markantonakis, K., Van Heijningen, N., Aboulkassimi, D., Gaine, C., Heckmann, T. & Naccache, D. “Physical fault injection and side-channel attacks on mobile devices: A compre-hensive analysis”. *Computers & Security*. 2021; 111 (2): 102471. DOI: https://doi.org/10.1016/j.cose.2021.102471.

27. Wilson, J., McLuskie, D. & Bayne, E. “Investigation into the security and privacy of iOS VPN applications”. *Proceedings of the 15th International Conference on Availability, Reliability and Security (ARES 2020). ACM*. 2020. p. 1–9. DOI: https://doi.org/10.1145/3407023.3407029.

Отримано 05.06.2026
Отримано після перегляду 26.08.2026
Прийнято 04.09.2026

DOI: https://doi.org/10.15276/ict.03.2026.05
UDC 004.056.53:004.451.9

Architectural approaches to network shielding in Android and iOS operating systems: a comparative analysis of the kernel, system extensions, and application levels

Olexander V. Zadereyko[1)]
ORCID: https://orcid.org/0000-0003-0497-9861; zadereyko@onua.edu.ua. Scopus Author ID: 57196005917
Volodymyr I. Hura[1)]
ORCID: https://orcid.org/0000-0001-6415-7865; hura@onua.edu.ua

Oleksandr A. Blazhko[2)]
ORCID: https://orcid.org/0000-0001-7413-275X; blazhko@op.edu.ua. Scopus Author ID: 57195410976
Anatoliy Y. Gaida[3)]
ORCID: https://orcid.org/0000-0002-8217-5044; cetus@ukr.net
[1)] National university "Odesa Law Academy", 23, Fontanska doroha. Odesa, 65009, Ukraine
[2)] Odesa Polytechnic National University, 1, Shevchenko Av. Odesa, 65044, Ukraine
[3)] National Admiral Makarov National University of Shipbuilding, 3, Tsentralnyi Av. Mykolaiv, 54000, Ukraine

ABSTRACT

This article is devoted to a comparative analysis of the architectural solutions underlying network filtering in the Android and iOS operating systems. The relevance of this research stems from the dominant position of these two operating systems in the mobile communications device market and the fundamental differences in their architectural organization, which determine the capabilities and limitations of third-party tools for controlling network connections. It has been established that the network filtering architecture of the Android operating system, based on the Netfilter/iptables module and the extended Berkeley Packet Filter technology, is covered in considerable detail in scientific publications, whereas the network filtering architecture of the iOS operating system—specifically the NetworkExtension framework (along with the XNU kernel's PF system packet filter and the utun tunneling interface)—is addressed only fragmentarily in existing studies, generally within the context of security characteristics, without being singled out as a separate subject of analysis. To address this identified gap, a systematic decomposition of the architectures of both operating systems was performed at comparable levels—the kernel, system services and extensions, and user applications. It is shown that the Android operating system's network firewall is implemented using three architectural models: One is based on Netfilter/iptables/eBPF, another is based on eBPF as part of the Android Open Source Project, and the third is based on the unprivileged VpnService software interface. In contrast, in the iOS operating system, firewall functions are distributed among the configuration and execution modules of the NetworkExtension framework (NEFilterManager, NEFilterProviderConfiguration, NEFilterDataProvider, NEFilterControlProvider), and the enforcement of rules is carried out via the utun system tunnel interface. It has been determined that architectural differences between the operating systems manifest at three levels: kernel openness, the number and specialization of extension points for third-party developers, and the model of execution isolation for filtering code. A limitation common to both operating systems is the inability to run multiple independent network filters simultaneously. A decomposition model is proposed that symmetrically reflects the network filtering architecture of both operating systems at comparable levels. The results of this architectural decomposition form the basis for the further development of a unified methodology and a system of criteria for the comparative analysis of network filters in the Android and iOS operating systems.

Keywords: Android, iOS, Netfilter/iptables, NetworkExtension, security architecture, firewall, network connections

ІНФОРМАЦІЯ ПРО АВТОРІВ

Задерейко Олександр Владиславович - кандидат техн. наук, доцент кафедри Штучного інтелекту та математичного моделювання. Національний університет «Одеська юридична академія», Фонтанська дорога, 23. Одеса, 65009, Україна
Галузь досліджень: системний аналіз, аналіз та захист цифрових систем, штучний інтелект

Olexander V. Zadereyko - PhD, Associate Professor, Department of Artificial Intelligence and Mathematical Modeling. National university "Odesa Law Academy", 23, Fontanska doroha. Odesa, 65009, Ukraine
ORCID: https://orcid.org/0000-0003-0497-9861; zadereyko@onua.edu.ua
Research field: systems analysis, analysis and security of digital systems, artificial intelligence

Гура Володимир Ігорович - кандидат техн. наук, доцент кафедри Штучного інтелекту та математичного моделювання. Національний університет «Одеська юридична академія», Фонтанська дорога, 23. Одеса, 65009, Україна
***Галузь досліджень*:** програмна інженерія, мережеві технології, штучний інтелект

Volodymyr I. Hura - PhD, Associate Professor, Department of Artificial Intelligence and Mathematical Modeling. National university "Odesa Law Academy", 23, Fontanska doroha. Odesa, 65009, Ukraine
ORCID: https://orcid.org/0000-0001-6415-7865; hura@onua.edu.ua
Research field: software engineering, networking technologies, artificial intelligence

Блажко Олександр Анатолійович - кандидат техн. наук, доцент, доцент кафедри Інженерії програмного забезпечення. Національний університет «Одеська політехніка», пр. Шевченка, 1. Одеса, 65044, Україна
***Галузь досліджень*:** програмна інженерія

Oleksandr A. Blazhko - PhD, Associate Professor, Department of Software Engineering. Odesa Polytechnic National University, 1, Shevchenko Av. Odesa, 65044, Ukraine
ORCID: https://orcid.org/0000-0001-7413-275X; blazhko@op.edu.ua. Scopus Author ID: 57195410976
Research field: software engineering

Гайда Анатолій Юліанович - кандидат технічних наук, доцент кафедри Інформаційних управляючих систем та технологій. Національний університет кораблебудування імені адмірала Макарова, пр. Центральний 3, Миколаїв, 54000, Україна
***Галузь досліджень*:** управління проектами наукомістких виробництв

Anatoliy Yu. Gaida - PhD, Associate Professor, Department of Information Management Systems and Technologies. National Admiral Makarov National University of Shipbuilding, 3, Tsentralnyi Av. Mykolaiv, 54000 Ukraine
ORCID: https://orcid.org/0000-0002-8217-5044; cetus@ukr.net
Research field: project management in knowledge-intensive industries