DOI: https://doi.org/10.15276/aait.09.2026.34
UDC 004.056.53:004.451.9

Methodology and criteria for a comparative analysis of network firewalls in Android and iOS operating systems

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
Alona Y. Pavlenko[3)]
ORCID: https://orcid.org/0000-0002-3776-8377; aliona.pavlenko@nuos.edu.ua
[1)] National university "Odesa Law Academy", Fontanska doroha, 23. 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

Relevance. Network firewalls in Android and iOS rely on architecturally heterogeneous solutions, and existing comparative studies treat the firewall only as one element of the overall security architecture, with no reproducible comparison procedure. **The aim** is to develop a reproducible methodology and a system of criteria for comparing network firewalls in Android (ninth to seventeenth versions) and iOS (fifteenth to twenty-seventh versions). **Tasks:** to analyze prior studies; to build the methodology as a sequence of methods; to formalize the system of criteria; to apply it to three approaches and derive recommendations. **Methods:** systems analysis of architectures (decomposition into kernel, system services/extensions, and user application levels), comparative analysis of primary technical documentation, criteria-based comparison, and synthesis. **Scientific novelty** lies in the ordered combination of standard methods with explicit rules for reducing heterogeneous architectures to a comparable form, selecting criteria with defined data sources and admissible values, and aggregating results. **Practical significance:** a "usage scenario – recommended approach" matrix for corporate security, personal privacy control, antivirus solutions, and parental control. **Results.** Four groups of criteria (architectural, functional, operational, security and trust) were developed and applied qualitatively to the kernel-oriented and VpnService-oriented approaches in Android and to NetworkExtension in iOS; the platform differences are systemic for criteria assessable without measurements and appear at the kernel, programming interface, and execution isolation levels. **Conclusions.** The study is qualitative, without measurements of latency, throughput, or power consumption; open-source implementations only verify criterion values; further research covers experimental evaluation, external expert assessment, and a cross-platform prototype.

Keywords: Network firewall; Android; iOS; comparison methodology; criteria system; NetworkExtension

For citation: **Zadereyko O. V., Hura V. I., Blazhko O. A., Pavlenko A. Yu. "Methodology and criteria for a comparative analysis of network firewalls in Android and iOS operating systems". *Applied Aspects of Information Technology*. 2026; Vol. 9 No. 4: 527–546. DOI: https://doi.org/10.15276/aait.09.2026.34**

INTRODUCTION

The Android and iOS operating systems (OSs) dominate the market of mobile communication devices. Therefore, technical and methodological solutions in the field of mobile network security that are intended for cross-platform use must take into account the specifics of both OSs.

At the same time, the architectural approaches to network firewalling on these OSs differ substantially. Android supports both kernel-level solutions, in particular those using Netfilter/eBPF, and application-level solutions based on VpnService, whereas in iOS the implementation of network control is largely governed by the NetworkExtension framework and system extensions [1], [2]. The general principles of the Android security model, within which these solutions are implemented, are described in [3].

Network firewalling, i.e., control of applications' network connections on the user's device, is one approach to protecting personal data against unauthorized transmission. Recent studies show that tracking and the transmission of unique identifiers are widespread in apps on both platforms [4], and that a considerable share of mobile VPN apps, which are themselves network control tools, contain flaws that cause network connections to leave the tunnel [5], [6]. At the same time, network firewalling tools for Android and iOS are built on architecturally different principles, with different sets of programming interfaces available to comparison, generalization and synthesis) and to

© Zadereyko O., Hura V., Blazhko O., Pavlenko A., 2026

This is an open access article under the CC BY license (https://creativecommons.org/licenses/by/4.0/deed.uk)

third-party developers, which complicates the design of cross-platform solutions and the comparison of their capabilities.

LITERATURE REVIEW

Existing comparative studies of Android and iOS security do not create a reproducible comparison procedure applied specifically to network firewalling as an independent object of analysis. A number of studies compare Android and iOS in terms of general security parameters, namely permission, isolation, and encryption models, and touch on network firewalls only indirectly, through the network-connection permission category [1], [2], [18]. Other studies analyze in detail the filtering architecture based on Netfilter/iptables – the subsystem underlying network firewalling in the Android kernel – mostly outside the mobile context, and therefore provide no basis for comparing it with the architecturally different approach implemented in iOS [19], [20], [21]. The analysis of existing studies did not reveal a single, previously substantiated methodology and a corresponding system of criteria by which architecturally heterogeneous network firewalling solutions in Android and iOS could be compared on common grounds rather than by an arbitrary set of technical characteristics.

Empirical studies of mobile VPN apps form a separate group, such apps are in effect implementations of the VpnService-oriented and NetworkExtension-oriented approaches. The study [6] analyzed 283 Android apps with the VPN permission and revealed leaks of DNS and IPv6 connections as well as cases of TLS interception. The study [5] proposed the MVPNalyzer framework, applied it to 281 apps from Google Play, and established that 29 of them transmit connections, including DNS, outside the tunnel. The study [22] found, among other things, that most iOS VPN apps do not route additional user data, such as DNS queries, through the tunnel. The study [23] showed that different VPN apps may share operators and vulnerabilities. The study [24] proposed blocking advertising requests from all apps on a device by intercepting packets at the network interface level, with real-time classification of each packet. The study [4] compared approximately 24 thousand Android and iOS apps and established that neither platform is clearly better with respect to privacy. These studies assess the quality of individual apps or ecosystems but do not propose a common system of criteria for comparing the architectural approaches to network firewalling themselves in both OSs. The results of this study therefore complement them, and the system of criteria makes it possible to link shortcomings identified in practice (for example, leaks outside the tunnel) to platform properties (the single-interception-point model, Always-on VPN with blocking of non-tunnel connections in Android and the includeAllNetworks parameter in iOS, restrictions imposed by the supervised/managed modes).

PROBLEM STATEMENT, AIM AND OBJECTIVES OF THE RESEARCH

The absence of such a methodology and system of criteria hinders the correct comparison of network firewalling tools and the well-founded choice of architectural solutions for cross-platform tools that control applications' network connections. This applies, in particular, to solutions that protect users' personal data against unauthorized transmission and are implemented for various usage scenarios (corporate security, personal privacy control, antivirus/DPI solutions, and parental control). These circumstances determine the practical significance of this study. It involves developing a reproducible methodology for the comparative analysis of Android and iOS network firewalls, together with a system of criteria applied to three architectural approaches on the basis of primary technical documentation.

The aim of the research is to develop a reproducible methodology for the comparative analysis of network firewalling tools in Android and iOS as an integral system that combines comparison principles, methods of analysis, a system of criteria, and a procedure for applying the criteria. The methodology must make it possible to compare architecturally heterogeneous approaches on common, previously substantiated grounds and to provide a basis for choosing an architectural solution in typical scenarios of controlling applications' network connections, in particular for personal data protection.

To achieve this aim, the following objectives must be accomplished:

– to analyze existing comparative studies of Android and iOS security and to substantiate the absence of a reproducible methodology for comparing network firewalls as an independent object of analysis;

– to develop a comparison methodology as a sequence of interrelated methods (systems analysis of architectures, comparative analysis of developers' primary technical documentation, criteria-based establish the relationship between the architecture decomposition levels and the approaches being compared;

– on the basis of the results of the architecture decomposition and the comparative analysis of primary technical documentation, to form a system of criteria (architectural, functional, operational, and security and trust criteria) and to formalize the procedure for its formation (criteria selection rules, data sources, admissible values, and the order of aggregating results);

– to apply the system of criteria to three approaches to network firewalling in Android and iOS on the basis of primary technical documentation, to carry out a partial verification of criterion values on open-source network firewall implementations, to formulate practical recommendations for choosing architectural solutions for typical usage scenarios, and to identify the limitations of the system of criteria and directions for further research.

The comparative analysis of Android and iOS network firewalls was carried out sequentially. First, the comparison methodology was formed (Section 1). Then, on its basis, the system of criteria was developed (Section 2) and applied to three approaches on the basis of primary technical documentation (Section 3).

1. METHODOLOGY FOR COMPARING NETWORK FIREWALLS IN ANDROID AND iOS

As stated in the problem statement, the absence of a single methodology for comparing network firewalling architectures accounts for the fragmentary nature of existing work. Therefore, this study builds a comprehensive methodology that makes it possible to compare Android and iOS using a single, predefined, and substantiated system of criteria.

A distinctive feature of the subject of this study is that network firewalls in Android and iOS are implemented by fundamentally different architectural means, ranging from a kernel module to an isolated system extension. This makes a direct comparison of individual technical parameters incorrect without first reducing them to a comparable form. Therefore, the study proposes a methodology of comparative analysis as an integral system that combines principles for reducing heterogeneous architectures to a comparable form, a set of analysis methods, a system of criteria, and a procedure for applying them.

For the sake of specificity, in this study a network firewall (a network firewalling tool) is understood as software component that, on the user's device, intercepts applications' network connections (at the packet or flow level) and applies allow or block rules to them. This working definition covers Netfilter/eBPF, VpnService, and NetworkExtension but does not assume their equivalence, because these technologies differ in implementation level (kernel, service application, system extension), in the set of available operations (allow and block, modification, tunneling), and in conditions of use (root access, supervised or managed device). Therefore, what is compared is not the technologies as such but the approaches to implementing network firewalling that are based on these technologies, using a single system of criteria. The object of analysis is the architectural features of the platforms, specific applications, perimeter network firewalls, and tools operating only at the browser level are outside the scope of consideration.

The limits of applicability of the results are as follows. The analysis covers Android from version 9 (API 28) to version 17 (API 37) and iOS from version 15 to version 27, i.e., up to the current stable releases at the time of analysis. The conclusions do not extend to older OS versions.

The lower bound is determined by the appearance of key components:

– eBPF-based network traffic monitoring and Private DNS in Android 9 [15], [25];

– Screen Time API/Family Controls in iOS 15 and the per-app mode for managed devices in iOS 16 [8].

The architectural capabilities and restrictions of the platforms change with new OS releases, therefore, the results are valid for documentation as of May 2026. Programming interfaces that appeared only in the latest releases and have no analogue in the other OS (for example, the URL filter provider available since iOS 26 [8]) are not included in the system of criteria, in accordance with the exclusion rule (see below), and the minimum OS versions for individual platform functions are given in Table 2.

In this study, the methodology is understood as an integral system consisting of:

– principles for comparing architecturally heterogeneous OSs;

– a set of methods that implement these principles;

– a system of criteria as the formalized result of methods 1-2;

– a procedure for applying the criteria to three approaches (qualitative verification against documentation).

A method is a separate way of obtaining comparable data (decomposition, documentation analysis, comparison, synthesis). The methodology combines these methods into a single ordered

procedure, defining the sequence of actions for their implementation. Each method relies on the result of the previous one.

Each of the methods taken separately is well known (systems analysis, documentation analysis, criteria-based comparison, generalization and synthesis). The novelty of the methodology lies not in the individual methods but in their ordered combination with rules for reducing heterogeneous architectures to a comparable form (three-level decomposition, the concepts of functionally equivalent levels and of an approach), criteria selection rules, defined data sources, and an order of aggregating results. The systematization of known platform characteristics (Table 2) is separated from the methodology. It is a result of applying the methodology, not a component of it.

1. The systems analysis method applied to Android and iOS architectures provides their decomposition into comparable levels involved in processing network connections:

– OS kernel level;
– system services/extensions level;
– user application level.

For Android, these are the Linux kernel level (Netfilter), the level of the Android Framework system service (VpnService, ConnectivityManager), and the level of the filter application. For iOS, these are the XNU kernel level (closed to third-party code), the level of the NetworkExtension system framework with its provider types (Packet Tunnel Provider, App Proxy Provider, Filter Data/Control Provider), and the level of the main app that manages the extension configuration. Such decomposition makes it possible to compare the OSs not as a whole but at functionally equivalent levels.

The result of this decomposition of Android and iOS network firewalling architectures is shown in Fig. 1.

The labels between the blocks in Fig. 1 indicate which approaches are implemented at the corresponding levels. The label “kernel + VpnService” in the Android column means that two approaches are available for this OS:

– kernel-oriented (at the kernel level, third-party code is allowed access to Netfilter/eBPF only with root access);

– VpnService-oriented (system services level and filter application, without root access).

The label “NetworkExtension only” in the iOS column means that, because the XNU kernel is closed to third-party code, the only approach is the NetworkExtension framework (the system extensions level and the managing app). The label “netd – system-level, not an API” indicates that the system intermediary service netd belongs to the system level and is not a programming interface available to third-party developers.

Functional equivalence of levels is understood as the juxtaposition of those layers of the two OSs that play the same role in processing a network connection (interception, decision-making, rule enforcement), even if their technical implementation differs. The decomposition levels define the framework of the comparison, while the approaches are specific implementations of a network firewall at

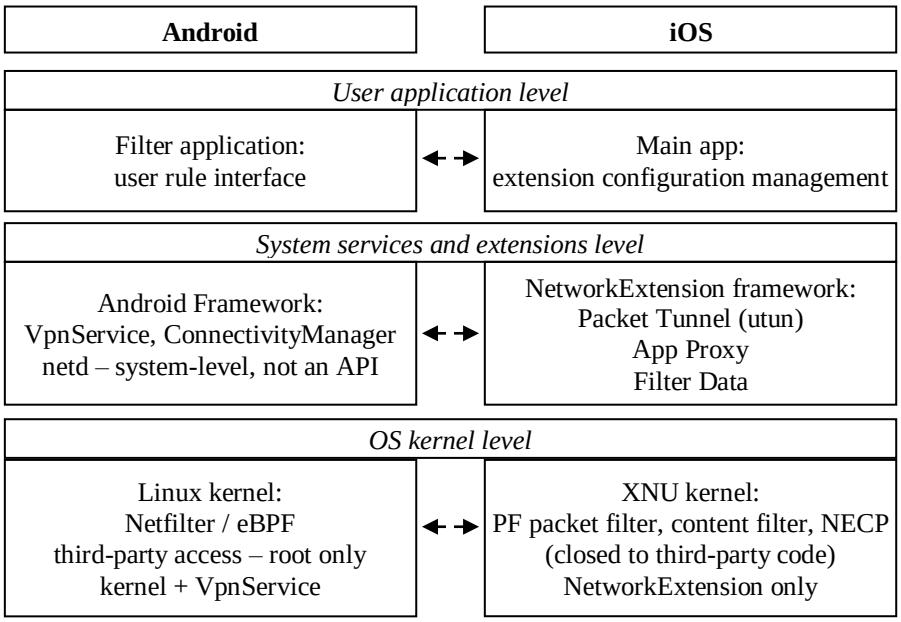


Fig. 1. **Three-level decomposition of Android and iOS network firewalling architectures**
Source: developed by the authors

these levels. An approach is understood as the use of the means of a particular decomposition level as the main way of implementing network firewalling. For Android, the kernel-oriented approach (Netfilter/eBPF) is considered at the kernel level, and the VpnService-oriented approach at the level of system services and the application. For iOS, architectural restrictions (the closed kernel) rule out a kernel-oriented approach for a third-party developer. The approach under comparison is localized at the level of the NetworkExtension system extensions and the managing app. This is why what is compared below is not "the OS as a whole" but three approaches: the kernel-oriented approach in Android, the VpnService-oriented approach in Android, and the NetworkExtension approach in iOS.

To ensure the correct use of terms in the subsequent sections, the following definitions are fixed:

– utun – a user-space virtual network interface in iOS/macOS through which the Packet Tunnel Provider receives IP packets;

– single-interception-point model – an OS restriction under which only one component of a given class can intercept all network connections of the device at any one time (one VpnService in Android and one provider of each NetworkExtension type per configuration profile in iOS);

– VpnService-oriented approach – an implementation of a network firewall in Android without root access through the official VpnService API;

– extension points – official APIs and interfaces available to third-party developers for implementing network firewalling logic (VpnService, NetworkExtension provider types, eBPF interception facilities given the appropriate privileges);

– NetworkExtension provider names – in this study, they are given by the class names NEPacketTunnelProvider (Packet Tunnel Provider), NEAppProxyProvider (App Proxy Provider), NEFilterDataProvider and NEFilterControlProvider (Filter Data/Control Provider), and NEDNSProxyProvider (DNS Proxy Provider) [7], [8], [9], [10], [11], [12].

2. The method of comparative analysis of primary technical documentation of the Android and iOS developers (Apple Developer documentation on NetworkExtension [7], [8], [9], [10], [11], [12], Linux kernel documentation on Netfilter and eBPF [26], [27], and Android Open Source Project specifications on VpnService [13], [14], [15]), as well as of scientific publications that describe practical implementations and effectiveness assessments of similar approaches [4], [5], [6], [22], [23], [24]. This ensures the objectivity of the comparison by relying on primary sources rather than on secondary review materials. It increases the reproducibility and validity of the comparison, since the characteristics of the architectures are determined primarily from official technical sources, while the results of practical studies are determined from scientific publications. The analysis was performed basing on the versions of documentation current as of May 2026 (Android 9-17, iOS 15-27), and the minimum OS versions from which individual programming interfaces and modes are available are given in Table 2. Primary documentation is the main data source for each criterion (the third column of Table 1), and scientific publications are used to check whether the documented behavior of the programming interface is confirmed by the operation of real apps.

3. The method of comparative analysis allows forming a single system of criteria basing on the architecture decomposition (method 1) and the comparative analysis of developers' primary technical documentation as well as on scientific publications (method 2), the three approaches (kernel-oriented Android, VpnService-oriented Android, and NetworkExtension in iOS) is evaluated separately against each criterion.

The result is a system of criteria that is applied in this study qualitatively (descriptively), while the quantitative assessment of latency, throughput, and power consumption lies outside the scope of the study.

The following procedure for forming the criteria is proposed:

a) the decomposition results are used to fix the comparable attributes of each level – interception point, privileges, isolation, number of simultaneous instances, available extension points, and source code openness;

b) the results of the comparative analysis of developers' primary technical documentation are used to verify that these attributes are present in all three approaches;

c) attributes without a common basis for comparison are discarded;

d) the remainder are grouped into four blocks – architectural, functional, operational, and security and trust criteria.

The criteria selection rules are as follows.

Inclusion – an attribute becomes a criterion if all of the following hold:

1) its value is defined for each of the three approaches in at least one primary source (the third column of Table 1);

2) it affects the performance of the main functions of a network firewall (interception, decision-making, rule enforcement) or the conditions of its operation and the trust placed in it;

3) its value can be established either from primary documentation or by measurements following a known procedure.

Exclusion – an attribute is discarded if its value is determined not by the architecture of the approach but by the features of a particular application or device manufacturer (user interface, commercial model), or if it is meaningful for only one of the approaches.

Deferral – attributes whose values require measurements (latency, throughput, power consumption) remain in the system of criteria but are not assessed in this study.

The result of the procedure is the system of criteria (Table 1), against which each approach is assessed separately.

For each criterion in Table 1, the data sources and the set of admissible values are defined (the third column of Table 1). The criteria are nominal (for example, the interception level) or ordinal (for example, the condition of use may be not required / required in certain device modes / always required) in nature, therefore, numerical aggregation is not applied, and the values are recorded descriptively in Table 2 with a reference to the source.

4. The method of generalization and synthesis makes it possible to reduce the results of the comparison on individual criteria to conclusions about the applicability of each approach to different usage scenarios (corporate security, personal privacy control, antivirus/DPI solutions, parental control), and also to identify areas in which the existing approaches do not provide a sufficient level of control over network connections in both OSs. Generalization is understood as reducing partial assessments to a common comparison structure, and synthesis as the transition from this structure to practical recommendations.

The results are understood as the assessments based on the criteria, the conclusions on the applicability of the approaches, and the final recommendations. The generalization and synthesis procedure is as follows. First, the comparison results are reduced to a comparative table of the approaches (Table 2). Next, differences are classified as systemic (manifested in all four groups of criteria) or local. On this basis, the "usage scenario – recommended approach" matrix is formed (Table 3). The limitations common to both OSs and the limitations of the system of criteria itself are recorded separately. Such a methodology avoids the mixing of characteristics from different levels that is typical of existing studies and ensures that results obtained for both OSs under a single system of criteria are comparable.

Aggregation rule: a difference on a criterion is recorded if the values of at least two approaches differ. A difference between platforms is considered systemic if it is recorded on at least one criterion assessed in each of the four groups, and local if it is confined to individual groups. The conclusion on the applicability of an approach to a scenario is formed on the decisive criteria given in the last column of Table 3 and takes the values "recommended," "recommended conditionally," and "not recommended." Reproducibility is ensured by the fact that each value is tied to a public primary source, and the selection rules, admissible values, and order of aggregation are specified explicitly.

The consistency of the assessments was verified procedurally:

– the values in Table 2 were established by one of the authors from primary sources and then independently verified by another author against the same sources;

– the measure of consistency is the share of matching values, discrepancies in wording were eliminated by reference to the primary source.

Verification of consistency with the involvement of external experts is envisaged in further research.

2. CRITERIA FOR COMPARING NETWORK FIREWALLS IN ANDROID AND iOS

The formation of the system of criteria relies on the results of decomposing the Android and iOS architectures into comparable levels – the kernel, system services/extensions, and the user application. Since a network firewall as an object of analysis combines heterogeneous aspects, ranging from the technical implementation of network connection interception to regulatory requirements for the protection of users' personal data (GDPR, CCPA, LGPD, PIPL, and PDPA), a system of criteria limited to technical parameters alone would not allow a comprehensive and practically applicable assessment of the approaches under consideration.

Therefore, the proposed system combines four mutually complementary groups of criteria, each of which reflects an independent aspect of how a network firewall operates:

– architectural criteria characterize the level and manner of the technical implementation of network firewalling;

– functional criteria characterize the composition and limitations of the extension points available to third-party developers;

– operational criteria characterize the practical limitations of applying the solutions on real mobile devices (performance, power consumption, compatibility, restart resilience);

– security and trust criteria characterize the solutions' resistance to circumvention, their auditability, and their compliance with personal data protection requirements.

The completeness of the system of criteria is justified on two grounds. First, the criteria cover the entire chain of a network firewall's operation:

– interception (interception level, extension points, privileges);

– decision-making (inspection depth, protocol support, decision-making variant, rule granularity);

– maintaining the rule in operation (compatibility, resilience to restarts, performance, power consumption);

– trust in the component (circumvention, auditability, moderation, compliance with data protection requirements).

Second, each of the three decomposition levels (see Fig. 1) is represented by at least one criterion. A formal completeness check was performed using the coverage criterion for the "decomposition level × network firewall function" matrix (three levels × three functions: interception, decision-making, rule enforcement).

Kernel level:

– interception – interception level, required privileges, source code openness;

– decision – inspection depth, protocol support;

– enforcement – performance and latency, resistance to circumvention attempts.

System services/extensions level:

– interception – available extension points, component isolation model, limit on the number of simultaneously active components;

– decision – decision-making variant;

– enforcement – compatibility with other network functions, resilience to restarts.

Application level:

– interception – certification/moderation requirements;

– decision – granularity rule;

– enforcement – transparency/auditability, compliance with personal data protection requirements, power consumption.

Each of the nine combinations is covered by at least one criterion, which is the formal condition of completeness within the adopted decomposition. The system is intended for comparing architectural approaches rather than specific software products, therefore, parameters determined by the implementation (ergonomics, cost, quality of reputation databases) are deliberately left outside its scope.

The check against primary documentation also revealed system functions and operating modes that are not separate approaches but change the values of the criteria:

– in Android – Always-on VPN with blocking of non-tunnel connections, Private DNS, and Device Owner administration through the Device Policy Manager [13], [14], [25];

– in iOS – the tunnel's includeAllNetworks parameter, Always On VPN via MDM, Screen Time/Family Controls, NEDNSProxyProvider, NEDNSSettingsManager, and MDM profiles [7], [8], [12], [16], [28], [29], [30].

They are treated as conditions that change the values of the criteria "required privileges," "resistance to filtering circumvention attempts," and "available extension points" (Table 2), rather than as additional criteria.

To ensure the objectivity and reproducibility of the comparative analysis, a system of criteria grouped into the four groups indicated above is proposed (Table 1). In this study, the criteria in Table 1 are applied qualitatively (descriptively) on the basis of primary technical documentation, while the criteria "performance and latency" and "power consumption" require measurements and are not assessed in Table 2, which is noted as a limitation.

The four groups of criteria considered (see Table 1) are not isolated. They are connected by logical (structural) relationships. As shown by the criteria-based comparison of the Android and iOS architectures, most of them rest on one and the same fundamental architectural difference.

This difference is that Android's open model, which allows both kernel-oriented and application-level solutions, stands opposed to iOS's closed, centrally managed model, within which NetworkExtension provides several specialized provider types (Packet Tunnel, App Proxy, and Filter Data).

Table 1. **System of criteria for the comparative analysis of a network firewall in Android and iOS**

Criterion	Content	Sources. Admissible values
1. Architectural criteria		
Interception level of network connections	OS kernel (Netfilter) / user-space virtual network interface (VpnService, utun) / system extension (NetworkExtension)	[7], [13], [26], [27]. Kernel / user space (virtual interface) / system extension
Required privileges	Presence or absence of a requirement for root access (Android) or for application-level special permissions and an Apple signature (iOS)	[7], [8], [13], [14]. Not required / root or system privileges / special Apple permission (entitlement) for the Network Extension capability / supervised or managed device / Device Owner rights
Isolation model of the network firewalling component	Operation in the application's address space / in a separate system extension process / in the kernel	[7], [13]. Kernel / application process / separate extension process
Limit on the number of simultaneously active components	In Android: – only one VpnService is active at a time; – kernel-oriented Netfilter/eBPF rules can coexist with it only given the appropriate privileges. In iOS: – one provider of each NetworkExtension type per configuration profile is active at a time; – providers of different types (for example, Packet Tunnel and Filter Data) can work together	[8], [13]. One active / one per provider type / unlimited
Available extension points / APIs	The set of official interfaces for a third-party component, comparing VpnService (Android) with the NetworkExtension provider types (Packet Tunnel Provider, App Proxy Provider, Filter Data/Control Provider) and with eBPF interception facilities given the appropriate privileges	[7], [9], [10], [11], [12], [13]. List of official APIs (VpnService, NetworkExtension providers, eBPF)
Source code openness of the component	Presence of an open implementation in AOSP (netd, eBPF programs, auxiliary Netfilter management code) compared with the closed internal implementation of PF and NetworkExtension, where only the client API is open	[15], [26], [27], [31]. Open / closed (according to the availability of public source code)
2. Functional criteria		
Packet inspection depth	Header-level filtering (IP/port/protocol) compared with connection-level (stateful) inspection and with payload/DNS/SNI analysis	[9], [10], [13]. Headers / connection state / payload (DNS, SNI)
Granularity rule	Ability to configure rules at the level of an individual application, domain, IP address, port, or network type (Wi-Fi/cellular)	[8], [9], [10], [13], [15], [17]. Application / domain / IP address / port / network type
Protocol support	Coverage of TCP/UDP, IPv4/IPv6, and modern transport protocols (QUIC/HTTP-3)	[9], [10], [13]. TCP, UDP, IPv4, IPv6, QUIC (UDP datagrams).
Decision-making variant	Static rule lists compared with dynamic/heuristic filtering and integration with external reputation databases	[9], [24]. Static rules / dynamic logic / external databases
3. Operational criteria		
Performance and latency	Impact on throughput and latency resulting from the additional routing of network connections through user space or an extension	Requires measurements (ms, Mbit/s). Not assessed in this study
Power consumption	Additional load on the battery of a mobile device when the intermediary service is continuously active	Requires measurements (mAh, % of charge). Not assessed in this study

Table 1 (continuation)

Criterion	Content	Sources. Admissible values
Compatibility with other network functions	Conflicts with other VPN apps, corporate MDM solutions, and multi-profile support features	[8], [13]. Coexistence with other VPN/MDM components: yes / only for different types / no
Resilience to system restarts/updates	Behavior of the network firewalling component during an OS service restart, an app update, or a change of the communication network	[13], [14], [16], [30]. Restart is automatic (Always-on, system) / managed by the application
4. Security and trust criteria		
Resistance to filtering circumvention attempts	Ability of an application to bypass firewalling (for example, through direct system calls that bypass VpnService or through alternative network interfaces)	[5], [6], [8], [13], [14], [16], [22], [30]. Circumvention possible / possible without an enforced mode / impossible
Transparency to the user/auditability	Availability of network connection logs and exported logs, as well as ability to verify rules independently	[15], [22], [26], [27], [31]. Logs, platform code openness, verifiability
Certification/moderation requirements	The need to pass Apple's review process in order to distribute an extension through the App Store. No comparable barrier for Android apps distributed outside Google Play	[7], [8], [29], [32]. App Store or Google Play moderation / special Apple permission / none
Compliance with personal data protection requirements	Ability to confirm technically that the analyzed network connections are not transmitted to third-party servers, compliance with data minimization principles (GDPR, CCPA, LGPD, PIPL, PDPA, and similar acts)	[5], [6], [9], [22]. Platform restricts data transmission / does not restrict / application audit only

Source: **compiled by the authors**

Thus, the interception level, required privileges, isolation model of the network firewalling component, and the set of available extension points (architectural criteria) directly determine the inspection depth, rule granularity, and protocol support (functional criteria), which in turn give rise to compatibility with other network functions (operational criteria) and to certification/moderation requirements (security and trust criteria). Accordingly, the proposed system of criteria does not merely list individual parameters but formalizes the identified fundamental architectural difference in the form of a comparable and reproducible structure. These relationships are logical in nature. They were derived from the analysis of documentation, and no separate method of establishing causality was applied in the study.

The logical (structural) relationships between the groups of criteria that form this structure are shown in Fig. 2.

This structure of criteria provides a basis for qualitative assessment (the criteria-based comparison method, Section 1) and, at the generalization and synthesis stage (method 4, Section 1), makes it possible to move from partial architectural differences to practical conclusions on the applicability of each approach in the scenarios of corporate protection, personal privacy control, antivirus/DPI solutions, and parental control. The recommendations are presented in the form of a "usage scenario – recommended approach" matrix.

3. RESULTS OF THE COMPARATIVE ANALYSIS OF NETWORK FIREWALLS IN ANDROID AND iOS

In accordance with the system of criteria that has been formed, the comparison of architecturally heterogeneous approaches to implementing a network firewall in Android and iOS must be carried out on common grounds. In practice, each of the four blocks of criteria (see Table 1) is filled in with the values recorded for the three approaches on the basis of the analysis of their architectures and the comparative analysis of developers' primary technical documentation (method 2, Section 1).

Generalizing the Android and iOS architectural solutions considered makes it possible to single out a number of key differences between the approaches to organizing a network firewall, given in Table 2. Table 2 compares the approaches on the criteria that correspond to Table 1 (all criteria whose values can be established from documentation. The criteria "performance and latency" and "power consumption" require measurements and are not assessed), and the last column gives the sources for each value.

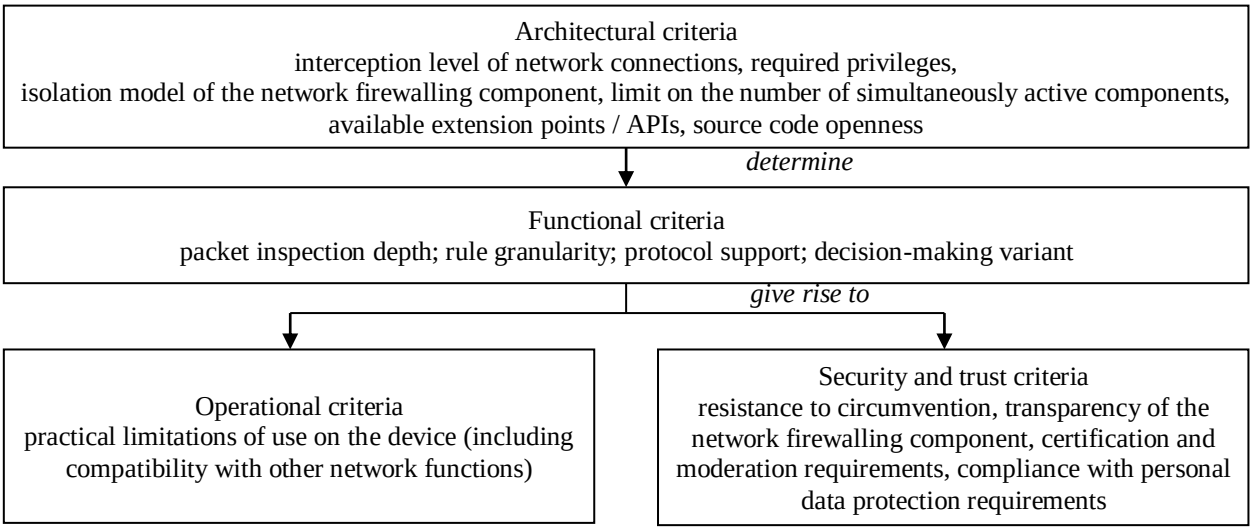


Fig. 2. **Logical (structural) relationships between the groups of criteria for comparing network firewalls in Android and iOS**

Source: compiled by the authors

Table 2. **Comparison of approaches to organizing a network firewall in Android and iOS**

Criterion	Android	iOS	Sources
Filtering model / interception level	Netfilter/iptables rules in the Linux kernel. Since Android 9, they have been partially replaced by eBPF (cgroup filters, bpfloader)	XNU kernel subsystems – the PF packet filter (of BSD origin), the socket content filtering subsystem (content filter), and the NECP network policy subsystem – whose source code is published as part of XNU [31], and the NetworkExtension extensions in user space. NetworkExtension providers rely precisely on the content filter and NECP: – a user-space agent receives a copy of the socket data and makes the “allow/drop” decision; – iOS supports up to two simultaneous content filters [31]. The configuration of these subsystems in iOS builds is not specified in the open sources	[15], [26], [27], [31]
Available extension points / APIs	VpnService (Java/Kotlin API) – without root access is the only official way for third-party apps to intercept all network connections. eBPF/Netfilter – given system privileges	NEPacketTunnelProvider / NEFilterDataProvider / NEAppProxyProvider – isolated system extensions with separate access rights, in addition, NEDNSProxyProvider and NEDNSSettingsManager (DNS level)	[7], [9], [10], [11], [12], [13], [14]
Isolation model of the network firewalling component	VpnService runs in the application’s process within the standard Android isolation model (UID, sandboxing, permission system)	Runs in a separate extension process with its own App Sandbox and restricted access to disk and network	[7], [13]
Granularity rule	VpnService: – rules at the application level (addAllowedApplication / addDisallowedApplication, since API 21), of routes and addresses (addRoute, addAddress), and, within its own tunnel, arbitrary filtering by IP address,	NEFilterDataProvider: – an “allow/block” verdict for each connection flow of any application on the device, with granularity by source application (sourceAppIdentifier), remote host and port (NEFilterSocketFlow), or URL (NEFilterBrowserFlow);	

Table 2 (continuation)

Criterion	Android	iOS	Sources
Granularity rule	– port, and domain (DNS) [13], [17]; – the network type is determined through ConnectivityManager. System per-UID rules (iptables/eBPF) are available only to the netd process with system privileges [15]	– has access to flow data for decision-making but cannot modify it or transmit it externally [9]. Packet Tunnel: route and address rules, per-app only on managed devices; – DNS Proxy – domain query level [8], [10], [12]. Deployment conditions are given in the row "Required privileges."	[8], [9], [10], [12], [13], [15], [17]
Limit on the number of simultaneously active components	Only one VPN/VpnService tunnel is active at a time	One provider of each extension type per configuration profile is active at a time	[8], [13]
Required privileges	System level (netd, root) for iptables/eBPF. VpnService without root access	Root access is not required, but special Apple permissions for the app and the corresponding device mode are required [8]: – Packet Tunnel – without restrictions (per-app – managed device); – App Proxy – managed devices only; – Filter Data, DNS Proxy – supervised devices (exceptions: apps using the Screen Time API on children's devices – iOS 15+, per-app mode on managed devices – iOS 16+). The system PF is configured only by the OS itself	[7], [8], [13], [14]
Accounting of network connections by UID/application (a component of the auditability criterion)	eBPF maps (traffic_uid_stats_map and others), previously, the xt_qtaguid module	Accounting at the level of the connection flow/application via NEFlowMetaData, without access to raw kernel sockets	[9], [15]
Source code openness of the component	netd, the auxiliary iptables management code, and the eBPF programs are part of AOSP and are open to study	The NetworkExtension implementation is closed, and only the client API is open, whereas Apple publishes the XNU source code, including the PF, content filter, and NECP subsystems, as part of Darwin [31]	[15], [31]
Packet inspection depth	VpnService: – the app receives whole IP packets (headers and payload; – TLS content – in encrypted form). Netfilter/eBPF – given privileges: headers and connection state	Packet Tunnel Provider – IP packets. NEFilterDataProvider – flow data for decision-making, without modification. NEDNSProxyProvider: DNS queries	[9], [10], [12], [13], [26], [27]
Protocol support	VpnService – IPv4/IPv6 packets, any protocols over IP, including QUIC/HTTP-3 as UDP datagrams with an encrypted payload	Packet Tunnel Provider – IP packets. NEFilterDataProvider and NEAppProxyProvider – TCP and UDP flows	[9], [10], [11], [13]
Decision-making variant	Arbitrary logic in the app – static lists, heuristics, real-time packet classification [24]	– the decision is made by the Filter Data Provider; – the control provider (NEFilterControlProvider) can update the rules; – the extension runs in a restricted sandbox	[9], [24]
Compatibility with other network functions	Only one VpnService is active: – a VPN client and a network firewall built on it displace each other; – consolidating the functions in a single app is forced	– providers of different types (Packet Tunnel, Filter Data) can be active simultaneously; – two providers of the same type conflict	[8], [13]

Table 2 (continuation)

Criterion	Android	iOS	Sources
Resilience to restarts/updates	Always-on VPN mode: – the system starts the service and restarts it when necessary; – in lockdown mode, connections outside the tunnel are blocked, including during VPN startup or failure	The extension's lifecycle is managed by the system; – on-demand rules allow the tunnel to be restored automatically; – when the includeAllNetworks parameter is enabled (iOS 14+), the system drops network traffic while the tunnel is unavailable [16]; – Always On VPN via MDM keeps the IKEv2 tunnel permanently up [30]. The behavior of connections when a content filter stops cannot be established unambiguously from the open documentation and requires experimental verification	[13], [14], [16], [30]
Resistance to filtering circumvention attempts	VpnService without lockdown: – some connections may pass outside the tunnel (DNS/IPv6 leaks were found in [5], [6]); – Always-on VPN with blocking of non-tunnel connections (set by the user through settings or by the Device Owner through DevicePolicyManager) prevents apps from bypassing the VPN	– Packet Tunnel with the includeAllNetworks parameter (iOS 14+) blocks traffic outside the tunnel without MDM involvement, although some system services may remain outside it [16]; – the content filter and DNS proxy are deployed on supervised devices, and Always On VPN via MDM [8], [30], which corresponds to enforced policy application; – on unmanaged devices without these modes, the user can disable the VPN configuration; – routing of part of the data outside the tunnel was found in iOS VPN apps [22]	[5], [6], [8], [13], [14], [16], [22], [30]
Transparency to the user / auditability	– open-source AOSP and Linux kernel; – the state of the eBPF counters is available through diagnostic tools (dumpsys netd) [15]	– the NetworkExtension implementation is closed; – behavior can be verified through the client API and studies of apps [22]	[15], [22], [26], [27], [31]
Certification / moderation requirements	– there is no mandatory certification for distribution outside Google Play; – Google Play applies moderation, whereas distribution outside the official app store (sideloading) is associated with higher risks [32]	– distribution through the App Store with Apple's review; – special Apple permissions are required to use Network Extension, and for Family Controls, a separate Apple permission for App Store use [8], [29]	[7], [8], [29], [32]
Performance, latency, and power consumption	Not assessed: requires measurements (outside the scope of the study)	Not assessed: requires measurements (outside the scope of the study)	–
Compliance with personal data protection requirements	– a VpnService app sees all packets and has network access, so the platform does not guarantee that no data is transmitted; – in practice, leaks and the transmission of identifiers have been found [5], [6]	– according to the documentation, NEFilterDataProvider is restricted in its access to the network and disk, which reduces the risk of data transmission; – Packet Tunnel and App Proxy have network access and require an audit of the app [22]	[5], [6], [9], [22]
Auxiliary system functions and modes that change the values of the criteria	– Always-on VPN with lockdown; – Private DNS (DNS over TLS, since Android 9) [25];	– includeAllNetworks for Packet Tunnel (since iOS 14) [16]; – Always On VPN via MDM (IKEv2) [30]; – Screen Time / Family Controls (since iOS 15, child authorization);	[7], [8], [12], [13], [14], [16], [25], [28], [29], [30]

Table 2 (continuation)

Criterion	Android	iOS	Sources
	– Device Owner / Device Policy Manager (enforced VPN configuration)	– NEDNSProxyProvider (supervised; – per-app on managed devices since iOS 16); – NEDNSSettingsManager; – MDM Web Content Filter (supervised)	

Source: **compiled by the authors**

From the data in Table 2, it can be concluded that the architectural difference between Android and iOS manifests itself in three components (the kernel, the programming interface, and execution isolation).

Firstly, at the kernel level, Android uses the Netfilter/eBPF subsystem, which is open and documented in the AOSP project, whereas the XNU kernel in iOS is closed to extension by third-party code, and the NetworkExtension implementation is closed to external study, which complicates independent auditing (Apple publishes the XNU source code, including the PF, content filter, and NECP subsystems [31]). A reduction in the attack surface for third-party code is achieved not by closedness as such but by the absence of an extension point at the kernel level and by the isolation of the extension process.

Secondly, at the level of developer APIs, Android provides a third-party app without root access with a single official extension point – VpnService – with no division by functional purpose. iOS provides a centralized NetworkExtension extension system with three specialized provider types (tunneling, proxying, content filtering). This limits the flexibility of Android solutions without root access but simplifies their development. By contrast, iOS requires greater specialization for the chosen provider type. In addition, NetworkExtension provides NEDNSProxyProvider and NEDNSSettingsManager for the DNS level, and the use of NEDNSProxyProvider is restricted to the supervised/managed modes [8], [12].

Thirdly, at the level of execution isolation, the iOS model was designed from the outset with a separate extension process and a restrictive sandbox, whereas in Android the network firewalling code, implemented in particular on the basis of VpnService, uses the standard app isolation model based on UID, sandboxing, and the permission system.

What the two platforms share is the single-interception-point principle within one component class. In Android without root access, only one VpnService is active at a time, therefore, a VPN client and a separate network firewall based on the same API displace each other. In iOS, the restriction applies to the provider type – one active instance of each type per configuration profile. Therefore, a Packet Tunnel Provider and an NEFilterDataProvider can be active simultaneously if their types differ. A conflict arises only between two providers of the same type or between two tunnels. This difference must be taken into account when designing cross-platform solutions. In Android, consolidating VPN and filtering in a single VpnService is forced, whereas in iOS, it is not, provided the functions are separated across different provider types.

The comparison performed confirms the architectural differences recorded during the criteria-based comparison of the architectures and formalized in the system of criteria (see Table 1).

In accordance with the aggregation rule (Section 1), differences between Android and iOS are recorded in each of the four groups, on criteria whose values are established from documentation:

– architectural (interception level, required privileges, source code openness. The limit on the number of simultaneously active components is assigned to this group in accordance with Table 1);

– functional (inspection depth, rule granularity);

– operational (compatibility with other network functions: forced consolidation of VPN and filtering in a single VpnService in Android versus the coexistence of providers of different types in iOS);

– security and trust criteria (auditability: the open source code of AOSP components versus the closed implementation of NetworkExtension;

– moderation requirements (App Store review and special Apple permissions versus the possibility of distribution outside Google Play);

– resistance to circumvention (different mechanisms for enforcing policy)).

Differences in performance and power consumption were not accessed. The conclusion that the differences are systemic is justified only within the criteria that can be accessed from documentation. The results obtained (see Table 2) demonstrate the applicability of the system of

criteria for the structured comparison of approaches but are not experimental confirmation of its effectiveness.

In this way, the gap identified in the problem statement is filled, namely the absence of a reproducible procedure for comparing a network firewall as an independent object of analysis, which led to an asymmetry between the detailed elaboration of the Android network firewall architecture [19], [20], [21] and the fragmentary nature of similar studies for iOS. The iOS network firewalling architecture – the NetworkExtension framework and the XNU kernel subsystems on which it relies (content filter, NECP, PF) – is compared with the Android Netfilter/iptables/eBPF architecture not within a general list of OS security characteristics, as was done in studies [1], [2], [18], but as an independent subject of analysis structured by a single system of criteria.

The criteria were formed from the results of the architecture decomposition, therefore, the coincidence of the identified differences with the criteria is to be expected and cannot be regarded as independent proof that the system works. At the same time, the system is not reduced to confirming differences: it also records similarities (for example, the single-interception-point model is common to both OSs), and some of the criteria (source code openness, moderation, limit on the number of components) were introduced as universal attributes independent of the identified differences. A partial independent verification was carried out on four open-source network firewall implementations (Table 4). A full verification on a representative set of apps, with an assessment of the consistency of external experts, is deferred to further research.

Compared with studies of mobile VPN apps [5], [6], [22], the system of criteria makes it possible to link the shortcomings identified in them to platform properties. The leaks of DNS and IPv6 connections outside the tunnel recorded for Android in [5], [6] and for iOS in [22] are consistent with the fact that, in the VpnService and Packet Tunnel models, the completeness of routing connections into the tunnel depends on the configuration set by the developer (the criterion “resistance to filtering circumvention attempts”), and that enforced blocking of non-tunnel connections is provided in Android only by Always-on VPN mode with lockdown [13], [14], and in iOS by the tunnel’s includeAllNetworks parameter [16] or by the managed supervised/MDM modes, in particular Always On VPN [8], [28], [30]. Unlike [4], where the platforms are compared in terms of app privacy, the differences obtained here are established at the level of the platforms’ architectural solutions, which makes it possible to transfer the conclusions to new apps without re-analyzing each of them.

The identified architectural differences and the limitations common to both OSs form the basis for moving to the stage of generalization and synthesis and for formulating practical recommendations (method 4, Section 1).

The results of the synthesis (method 4) are presented in the form of a “usage scenario – recommended approach” matrix (Table 3). The recommendations are conditional in nature and are valid within the documentation analyzed.

The verification of criterion values on specific implementations was carried out on four open-source network firewalls whose architecture is documented in the projects’ source code and accompanying documentation (Table 4). Projects were selected that represent each of the three approaches and are publicly available for independent study.

None of the values in Table 2 established from documentation contradicts the implementations considered. In particular, the restriction “one active VpnService” and the status of VpnService as the only extension point without root are confirmed by the NetGuard documentation [34], and the possibility of blocking without MDM in iOS is confirmed by the Lockdown Privacy implementation based on Packet Tunnel [36]. No implementations based on NEFilterDataProvider were found among open-source projects, which is consistent with the restriction of this provider to supervised devices (Table 2).

CONCLUSIONS

The scientific novelty of the results obtained is as follows. A methodology for the comparative analysis of Android and iOS network firewalls has been proposed as an ordered set of known methods (systems analysis, analysis of primary technical documentation, criteria-based comparison, generalization and synthesis), supplemented by rules for reducing architecturally heterogeneous approaches to a comparable form (three-level decomposition, the concepts of functionally equivalent levels and of an approach), criteria selection rules, defined data sources, and an order of aggregating results. Unlike existing studies [1], [2], [18], which compare OSs by general security characteristics, the network firewall has been considered as an independent object of analysis.

Table 3. **"Usage scenario – recommended approach" matrix**

Usage scenario	Recommended approach	Decisive criteria and conditions
Corporate protection and MDM-managed environments	Android – kernel-oriented (given privileges) or VpnService with Always-on VPN and lockdown set by the Device Owner. iOS – NetworkExtension (Filter Data, DNS Proxy, App Proxy) on supervised/managed devices, Always On VPN via MDM. Assessment: recommended	Access to system extension points is governed by device policy (Android Enterprise, Apple MDM) [8], [14], [28], [30]. Both OSs provide a comparable level of control
Personal privacy control with detailed rule configuration	Android: – kernel-oriented (root) and VpnService-oriented (without root) approaches – recommended. iOS: – NetworkExtension; Filter Data only on supervised devices – recommended conditionally	– openness of Netfilter/eBPF and the possibility of combining the approaches [15], [26], [27]; – in iOS, NEFilterDataProvider allows only "allow/block" without modifying the flow [8], [9]
Antivirus and DPI solutions	Android: – VpnService-oriented approach with the analysis functions consolidated in a single app – recommended conditionally. iOS: – Packet Tunnel Provider and Filter Data Provider of different types – recommended conditionally (Filter Data – supervised devices)	Limit on the number of simultaneously active components and compatibility with other network functions [8], [13]
Parental control	Android: – VpnService with Always-on VPN and lockdown set by the Device Owner – recommended conditionally; – without lockdown – not recommended. iOS: – Screen Time / Family Controls with child authorization (iOS 15+), a supervised device with Filter Data/DNS Proxy, or Packet Tunnel with includeAllNetworks – recommended conditionally; – on an unmanaged device without these modes – not recommended	The decisive criterion is resistance to circumvention: – it is determined by the availability of an enforced mode, not by the OS as a whole [8], [13], [14], [16], [29], [30]; – distribution outside the official app store increases the risk that the app will be concealed [32]

Source: **compiled by the authors**

Table 4. **Verification of criterion values on open-source network firewall implementations**

Implementation (OS, approach)	Confirmed values of the criteria in Table 1 and Table 2	Source
AFWall+ (Android, kernel-oriented approach)	– iptables rules in the Linux kernel; – root access required; – per-UID (application) granularity and granularity by network type; – works in parallel with VPN apps because it does not occupy VpnService	[33]
NetGuard (Android, VpnService-oriented approach)	The project documentation directly records two criterion values: – the only way to build a network firewall without root is VpnService, and the impossibility of combining it with other VPN apps (one active VpnService); – per-app, domain, and IP-address rules; – open source code	[34]
Rethink DNS + Firewall + VPN (Android, VpnService-oriented approach)	– app filtering, DNS encryption, and a WireGuard tunnel are consolidated in a single VpnService (forced consolidation of functions); – per-app, domain, and IP-address rules; – open source code	[35]
Lockdown Privacy (iOS, NetworkExtension – Packet Tunnel Provider)	– blocking of trackers in all apps on the device without MDM through a Packet Tunnel extension; – decisions at the packet/DNS level, without access to TLS content; – distribution only through the App Store, open source code	[36]

Source: **compiled by the authors**

A four-component system of criteria (architectural, functional, operational, and security and trust) has been formed, with logical (structural) relationships between the groups, which reduces the difference between Android's open architecture and iOS's closed, centrally managed architecture to a comparable assessment structure. The comparison approach has been improved by singling out three approaches to be compared – the kernel-oriented and VpnService-oriented approaches in Android and NetworkExtension in iOS – which avoids the mixing of characteristics from different levels that is inherent in studies [1], [2], [18], [19], [20], [21]. Practical recommendations for choosing architectural solutions for typical scenarios are presented in the form of a "usage scenario – recommended approach" matrix (Table 3). The systematization of known platform characteristics (Table 2) is separated from the methodology – it is a result of applying it. The results are qualitative in nature and are based on the analysis of documentation, but they contain no experimental confirmation.

The comparison against primary documentation showed that the differences between Android and iOS are systemic within the criteria that are assessed without measurements. They were recorded in all four groups of criteria (Table 2) and reduce to a single architectural difference: Android's open model allows kernel-oriented and application-level solutions, whereas iOS's closed, centrally managed model provides typed NetworkExtension providers. Neither approach is preferable in itself, and their difference reflects a different distribution of responsibility between the OS and developers.

The practical significance of this difference is as follows. The openness and flexibility of the Android architecture provide deeper control over network connections, in particular at the kernel level (Netfilter/eBPF), and independent verifiability of the code, however, responsibility for the correctness of the tunneling implementation rests with the developer, as confirmed by the shortcomings of mobile VPN apps identified in [5], [6]. iOS has a higher level of enforced execution isolation, which confines a third-party developer to typed scenarios (tunneling, proxying, content filtering, DNS proxying) and does not allow the NetworkExtension implementation to be verified independently. At the same time, routing of part of the data outside the tunnel has also been recorded in iOS VPN apps [22].

The practical consequence of the architectural difference is as follows. A developer of a cross-platform network firewall cannot transfer the Android model "one VpnService does everything" to iOS without losing provider specialization, and cannot transfer the iOS model "filter and tunnel as different types" to Android without consolidating them in a single VpnService. This feature must be taken into account when designing cross-platform solutions for ensuring the network security of mobile devices running Android and iOS. The recommendations are given in Table 3.

The practical conclusions by usage scenario are summarized in Table 3 and are not repeated here. In general terms, they are as follows. In corporate and MDM-managed environments, both OSs provide a comparable level of control, since access to system extension points is governed by device policy rather than by the distribution channel [8], [14], [28], [30]. For personal solutions with detailed rule configuration, Android has an architectural advantage owing to the openness of Netfilter/eBPF and the possibility of combining the kernel-oriented and VpnService-oriented approaches, whereas NEFilterDataProvider in iOS limits the developer to "allow/block" verdicts on supervised devices [8], [9]. For antivirus and DPI solutions, the limiting factor in Android is the single active VpnService, whereas in iOS providers of different types coexist [8]. For parental control, the decisive criterion is resistance to circumvention, whose value is determined not by the OS as a whole but by the availability of an enforced mode: Always-on VPN with lockdown set by the Device Owner in Android [13], [14], includeAllNetworks, supervised/MDM, or the Screen Time API with child authorization in iOS [8], [16], [29], [30]. The results of [32], obtained for Android apps, confirm the importance of the moderation criterion but do not allow the resistance to circumvention of the two OSs to be compared.

The limitations of the study should be noted. The proposed system of criteria (see Table 1) is predominantly architectural and qualitative: the criterion values were established from primary technical documentation as of May 2026 and have not been confirmed experimentally. The criteria that require measurements (latency, throughput, power consumption) were not assessed, which limits the quantitative ranking of solutions on operational criteria. The platforms' programming interfaces were analyzed first and foremost; specific implementations were used only to verify criterion values on four open-source projects (Table 4), without measuring their behavior. The criteria were formed from the results of decomposition. Therefore, their application to the same approaches

is not independent proof that the system works, and the consistency of the assessments was verified only among the authors, without the involvement of external experts. The results are valid for Android 9–17 and iOS 15-27 according to the documentation as of May 2026 and may change along with the platforms, and functions that have no analogue in the other OS (in particular, the URL filter provider available since iOS 26 [8]) are not included in the system of criteria in accordance with the exclusion rule.

Further research should be directed along four interrelated lines that follow from the limitations identified.

First, it is advisable to assess the latency and power consumption of mobile devices caused by the main approaches – VpnService (Android), NEPacketTunnelProvider (iOS), and the kernel-oriented Netfilter/eBPF approach (Android).

Second, it is advisable to analyze the compliance of the architectures considered with the requirements of GDPR, CCPA, LGPD, PIPL, and PDPA, in particular regarding data minimization and collection limitation.

Since the ability to confirm technically the composition of the collected network metadata is directly determined by rule granularity and by the level of telemetry available in Android and iOS, considered in Sections 2-3, the system of criteria (see Table 1) can serve as a basis for such a compliance analysis.

Third, the development and testing of a cross-platform network firewall prototype that implements the proposed system of criteria as a technical specification will make it possible to verify its practical applicability.

Fourth, it is advisable to verify the reproducibility of the methodology, namely, to carry out an independent application of the system of criteria by several experts, compare the consistency of their assessments, and evaluate the criteria that require measurements on real apps [5], [6], [24].

Conflict of interest: The authors declare that they have no conflict of interest, including financial, personal, authorship-related, or any other kind, that could have influenced the research or the results published in this article

Funding: The research was carried out without financial support

Data availability: All data are available in digital or graphical form in the main text of the manuscript

Use of artificial intelligence (AI) tools: The authors confirm that they did not use AI tools to write this work

REFERENCES

1. Mohamed, I., & Patel, D. “Android vs iOS Security: A Comparative Study”. *12th International Conference on Information Technology — New Generations (ITNG). IEEE*. 2015. p. 725–730. DOI: https://doi.org/10.1109/ITNG.2015.123.

2. 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 (ICETSIS). IEEE*. 2022. DOI: https://doi.org/10.1109/ICETSIS55481.2022.9888847.

3. 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.

4. Kollnig, K., Shuba, A., Binns, R., Van Kleek, M. & Shadbolt, N. “Are iPhones really better for privacy? A comparative study of iOS and Android apps”. *Proceedings on Privacy Enhancing Technologies*. 2022; 2: 6–24. DOI: https://doi.org/10.2478/popets-2022-0033.

5. 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 (NDSS 2026)*. 2026. p. 1–17. DOI: https://doi.org/10.14722/ndss.2026.231573.

6. Ikram, M., Vallina-Rodriguez, N., Seneviratne, S., Kaafar, M. A., & Paxson, V. “An analysis of the privacy and security risks of Android VPN permission-enabled apps”. *Proceedings of the 2016 Internet Measurement Conference (IMC ’16). ACM*. 2016. p. 349–364. DOI: https://doi.org/10.1145/2987443.2987471.

7. “Network Extension”. *Apple Developer Documentation. Apple Inc*. – Available from: https://developer.apple.com/documentation/networkextension. – [Accessed: May 11, 2026].

8. “TN3134: Network Extension provider deployment”. *Apple Inc. Apple Developer Documentation.* – Available from: https://developer.apple.com/documentation/technotes/tn3134-network-extension-provider-deployment. – [Accessed: May, 2026].

9. “NEFilterDataProvider”. *Apple Inc. Apple Developer Documentation*. – Available from: https://developer.apple.com/documentation/networkextension/nefilterdataprovider. – [Accessed: May, 2026].

10. “Packet tunnel provider (NEPacketTunnelProvider)”. *Apple Inc. Apple Developer Documentation*. – Available from: https://developer.apple.com/documentation/networkextension/packet-tunnel-provider. – [Accessed: May, 2026].

11. “App proxy provider (NEAppProxyProvider)”. *Apple Inc. Apple Developer Documentation*. – Available from: https://developer.apple.com/documentation/networkextension/app-proxy-provider. – [Accessed: May, 2026].

12. “DNS proxy provider (NEDNSProxyProvider)”. *Apple Inc. Apple Developer Documentation.* – Available from: https://developer.apple.com/documentation/networkextension/dns-proxy-provider. – [Accessed: May, 2026].

13. “VpnService”. *Android API reference*. – Available from: https://developer.android.com/reference/android/net/VpnService. – [Accessed: May, 2026].

14. “DevicePolicyManager (setAlwaysOnVpnPackage)”. *Android API reference*. – Available from: https://developer.android.com/reference/android/app/admin/DevicePolicyManager. – [Accessed: May, 2026].

15. “Android open source project. eBPF traffic monitoring”. – Available from: https://source.android.com/docs/core/data/ebpf-traffic-monitor. – [Accessed: May, 2026].

16. “includeAllNetworks (NEVPNProtocol)”. *Apple Inc. Apple Developer Documentation.* – Available from: https://developer.apple.com/documentation/networkextension/nevpnprotocol/includeallnetworks. – [Accessed: May, 2026].

17. “VpnService.Builder”. *Android API reference*. – Available from: https://developer.android.com/reference/android/net/VpnService.Builder. – [Accessed: May, 2026].

18. 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*. 2018; 526: 301–313. DOI: https://doi.org/10.1007/978-3-319-92925-5_20.

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

20. 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.

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

22. Wilson, J., McLuskie, D. & Bayne, E. “Investigation into the security and privacy of iOS VPN applications”. *ARES ’20: Proceedings of the 15th International Conference on Availability, Reliability and Security. ACM*. 2020. DOI: https://doi.org/10.1145/3407023.3407029.

23. Mixon-Baca, B., Knockel, J. & Crandall, J.R. “Hidden links: Analyzing secret families of VPN apps”. *Free and Open Communications on the Internet*. 2025; 2: 18–27. – Available from: https://www.petsymposium.org/foci/2025/foci-2025-0008.php. – [Accessed: May, 2026].

24. Shuba, A., Markopoulou, A. & Shafiq, Z. “NoMoAds: Effective and efficient cross-app mobile ad-blocking”. *Proceedings on Privacy Enhancing Technologies.* 2018; 4: 125–140. DOI: https://doi.org/10.1515/popets-2018-0035.

25. Kline, E. & Schwartz, B. “DNS over TLS support in Android P developer preview”. *Android Developers Blog*. 2018. – Available from: https://android-developers.googleblog.com/2018/04/dns-over-tls-support-in-android-p.html. – [Accessed: May, 2026].

26. “The Linux Kernel documentation”. *BPF Documentation*. – Available from: https://docs.kernel.org/bpf. – [Accessed: May, 2026].

27. “Netfilter Project”. *Documentation.* – Available from: https://www.netfilter.org/ documentation. – [Accessed: May, 2026].

28. “Web content filter (Device Management, MDM)”. *Apple Inc. Apple Developer Documentation*. – Available from: https://developer.apple.com/documentation/devicemanagement/webcontentfilter. – [Accessed: May, 2026].

29. “Family controls”. *Apple Inc. Apple Developer Documentation*. – Available from: https://developer.apple.com/documentation/familycontrols. – [Accessed: May, 2026].

30. “IKEv2 device management settings for Apple devices (Always On VPN)”. *Apple Inc. Apple Platform Deployment*. – Available from: https://support.apple.com/guide/deployment/dep4ce9487d. – [Accessed: May, 2026].

31. “XNU kernel source code (Darwin)”. *Apple Inc.* – Available from: https://github.com/apple-oss-distributions/xnu. – [Accessed: May, 2026].

32. 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.

33. “AFWall+ (Android Firewall+): source code and documentation”. *Ukanth.* – Available from: https://github.com/ukanth/afwall. – [Accessed: May, 2026].

34. Bokhorst, M. “NetGuard: a simple way to block access to the internet per application”. *Source code and documentation.* – Available from: https://github.com/M66B/NetGuard – [Accessed: May, 2026].

35. “Rethink DNS + Firewall + VPN for Android: source code and documentation”. *Celzero.* – Available from: https://github.com/celzero/rethink-app. – [Accessed: May, 2026].

36. “Lockdown Privacy (Lockdown iOS): source code and documentation”. *Confirmed Inc*. – Available from: https://github.com/confirmedcode/Lockdown-iOS. – [Accessed: May, 2026].

Received 29.08.2026
Received after revision 22 10.2026
Accepted 30.10.2026

DOI: https://doi.org/10.15276/aait.09.2026.34
UDC 004.056.53:004.451.9

Методологія та система критеріїв порівняльного аналізу мережевих екранів в операційних системах Android та iOS

Задерейко Олександр Владиславович[1)]
ORCID: https://orcid.org/0000-0003-0497-9861; zadereyko@onua.edu.ua. Scopus Author ID: 57196005917
Гура Володимир Ігорович[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
Павленко Альона Юріївна[3)]
ORCID: https://orcid.org/0000-0002-3776-8377; aliona.pavlenko@nuos.edu.ua
[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

Актуальність. Мережеві екрани операційних систем Android та iOS побудовано на архітектурно різнорідних рішеннях, а наявні порівняльні дослідження розглядають мережевий екран лише як один з елементів загальної організації захисту, без відтворюваної процедури його зіставлення як самостійного об’єкта аналізу. **Метою статті** є розроблення відтворюваної методології та системи критеріїв порівняльного аналізу мережевих екранів Android (від дев’ятої до сімнадцятої версії) та iOS (від п’ятнадцятої до двадцять сьомої версії). **Завдання:** проаналізувати наявні дослідження; побудувати методологію як упорядковану послідовність методів; формалізувати систему критеріїв; застосувати її до трьох підходів і сформулювати рекомендації. **Методи:** системний аналіз архітектур (декомпозиція на рівень ядра, рівень системних служб/розширень і рівень користувацького застосунку), порівняльний аналіз первинної технічної документації розробників, зіставний аналіз за визначеними критеріями, узагальнення і синтез. **Наукова новизна** полягає не в окремих методах, які є стандартними, а в їх упорядкованому поєднанні з явними правилами приведення різнорідних архітектур до порівнянного вигляду, відбору критеріїв із визначеними джерелами даних і допустимими значеннями та зведення результатів. **Практична значимість:** матриця

«сценарій застосування – рекомендований підхід» для корпоративного захисту, персонального контролю конфіденційності, антивірусних рішень і батьківського контролю. **Результати.** Розроблено чотири групи критеріїв (архітектурні, функціональні, експлуатаційні, безпеки та довіри) і застосовано їх якісно до ядро-орієнтованого та VpnService-орієнтованого підходів в Android і підходу NetworkExtension в iOS; розбіжності між платформами мають системний характер за критеріями, які оцінюються без вимірювань, і проявляються на рівні ядра, програмного інтерфейсу для розробників та ізоляції виконання. **Висновки.** Дослідження має якісний характер, без вимірювань затримки, пропускної здатності та енергоспоживання; відкриті реалізації використано лише для перевірки значень критеріїв; подальші дослідження охоплюють експериментальну оцінку, зовнішню експертизу і кросплатформний прототип.

Ключові слова: мережевий екран; Android; iOS; методологія порівняння; система критеріїв; NetworkExtension

ABOUT THE AUTHORS



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. Scopus Author ID: 57196005917
Research field: systems analysis, analysis and security of digital systems, artificial intelligence

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 і безпека цифрових систем, 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

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: інженерія програмного забезпечення, мережеві технології, штучний інтелект



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

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



Alona Y. Pavlenko - senior lecturer, Department of information management systems and technologies. National Admiral Makarov National University of Shipbuilding, 3, Tsentralnyi Av, Mykolaiv, 54000, Ukraine
ORCID: https://orcid.org/0000-0002-3776-8377; aliona.pavlenko@nuos.edu.ua
Research field: programming and object-oriented programming

Павленко Альона Юріївна - старший викладач кафедри Інформаційних управляючих систем та технологій. Національний університет кораблебудування імені адмірала Макарова, пр. Центральний 3, Миколаїв, 54000, Україна
Галузь досліджень: програмування та об’єктно-орієнтоване програмування