Расширенные архитектуры раздельного туннелирования для VPN-приложений на базе WireGuard в Windows: Глубокое погружение в Windows Filtering Platform (WFP)

Резюме

В современном ландшафте VPN-приложений критически важным становится обеспечение гибкого и интуитивно понятного управления сетевым трафиком. Традиционные методы раздельного туннелирования, основанные исключительно на IP-адресах, демонстрируют существенные ограничения, особенно в контексте динамических облачных сервисов и постоянно меняющихся пулов IP-адресов. Это создает острую потребность в решениях, способных управлять трафиком на уровне отдельных приложений, что напрямую влияет на удобство использования продукта и его конкурентоспособность на рынке.

Предлагаемое исследование сосредоточено на многокомпонентной архитектуре, основанной на привилегированной вспомогательной службе, использующей Windows Filtering Platform (WFP) и драйвер-выноску режима ядра. Этот подход позволяет реализовать раздельное туннелирование на уровне приложений, преодолевая недостатки IP-ориентированных методов. В отчете подробно анализируются ключевые технические аспекты, такие как арбитраж фильтров WFP, безопасное межпроцессное взаимодействие (IPC) между пользовательским и системным режимами, а также сложные процессы цифровой подписи драйверов. Применение WFP обеспечивает бесшовное взаимодействие с сетевым стеком Windows, высокую производительность и надежность, что делает его оптимальным выбором для создания профессионального VPN-продукта.

1. Введение: Эволюция раздельного туннелирования VPN в Windows

В данном разделе рассматривается фундаментальная проблема, которую призвано решить продвинутое раздельное туннелирование, и подчеркиваются недостатки более простых альтернатив.

1.1. Ограничения традиционного раздельного туннелирования на основе IP-адресов (AllowedIPs WireGuard)

WireGuard зарекомендовал себя как высокопроизводительный, современный и безопасный VPN-протокол, отличающийся простотой и минималистичным подходом.[1] Однако его встроенная функциональность раздельного туннелирования ограничена параметром

AllowedIPs в конфигурационном файле. Этот параметр определяет, какой трафик (на основе IP-адресов или подсетей) должен проходить через VPN-туннель. Например, конфигурация AllowedIPs = 0.0.0.0/0 направляет весь IPv4-трафик через VPN, в то время как AllowedIPs = 192.168.1.0/24 ограничивает туннель только трафиком для указанной локальной сети.[1]

Такой подход, основанный на IP-адресации, является фундаментальным ограничением для современных VPN-приложений, ориентированных на пользователя. Основные недостатки заключаются в следующем:

- **Отсутствие привязки к приложениям:** Невозможно напрямую указать, что трафик от chrome.exe должен идти мимо туннеля, а трафик от outlook.exe — через него. Пользователи мыслят категориями приложений, а не IP-адресов.[1]
- **Динамические IP-адреса:** Многие облачные сервисы и веб-сайты (например, Netflix, Office 365) используют широкий и постоянно меняющийся пул IP-адресов, распределенных через сети доставки контента (CDN). Поддержание актуального списка IP-адресов для исключения таких сервисов становится практически невыполнимой задачей.[1]
- **Сложность управления:** Ручное управление списками IP-адресов и подсетей неудобно для конечного пользователя и подвержено ошибкам.[1]

Внутренние ограничения IP-ориентированного раздельного туннелирования, в частности проблемы, связанные с динамическими IP-адресами и отсутствием контроля на уровне приложений, напрямую обусловливают необходимость

разработки более совершенного, привязанного к приложениям решения. Простота AllowedIPs является преимуществом для базовой настройки VPN, но становится критическим недостатком, когда речь идет об удобстве использования и поведении современных приложений. Пользователи не управляют своим сетевым трафиком по IP-адресам; они взаимодействуют с приложениями. Облачные сервисы усугубляют эту проблему, делая статические списки IP-адресов практически сразу устаревшими. Этот разрыв между ожиданиями пользователей/современной интернет-архитектурой и нативными возможностями WireGuard создает острую необходимость в более сложном решении, работающем на более высоком семантическом уровне. Это не просто техническое предпочтение, а рыночная необходимость для конкурентоспособных VPN-продуктов.

Неспособность легко настраивать раздельное туннелирование для каждого приложения значительно ухудшает пользовательский опыт любого коммерческого VPN-продукта. Пользователи ожидают интуитивно понятного и гранулированного контроля над сетевым поведением своих приложений, а решение, которое вынуждает их вручную управлять IP-адресами, является принципиально неудобным. Это подчеркивает, что выбранное решение на основе WFP является не только техническим улучшением, но и критически важной функцией для конкурентоспособности на рынке и удовлетворенности пользователей. Успех VPN-продукта на потребительском или корпоративном рынке во многом зависит от его удобства использования. Если ключевая функция, такая как раздельное туннелирование, сложна в настройке или приводит к частым отключениям (как это происходило бы при динамических изменениях IP-адресов с AllowedIPs), пользователи быстро откажутся от продукта. Техническая задача раздельного туннелирования на уровне приложений, таким образом, напрямую переходит в бизнес-императив для создания бесшовного и интуитивно понятного пользовательского интерфейса, что, в свою очередь, требует надежного базового технического решения, такого как WFP.

1.2. Необходимость управления трафиком на уровне приложений

Для создания конкурентоспособного VPN-продукта с функцией раздельного туннелирования необходимо реализовать механизм, который работает на уровне приложений, а не IP-адресов.[1] Это требует выхода за рамки стандартных возможностей WireGuard и внедрения более глубокой интеграции с сетевым

стеком операционной системы Windows.[1]

Переход от сетевого уровня (IP) к уровню приложений отражает более широкую, продолжающуюся тенденцию в сетевой безопасности и управлении. В современных вычислительных средах идентификация и контекст приложения становятся все более важными для обеспечения политики, часто превосходя необработанные сетевые адреса. Возможность фильтрации по идентификатору приложения (App ID) в WFP является архитектурным ответом Microsoft на эту развивающуюся потребность в гранулированном, контекстно-ориентированном применении сетевых политик. Интернет эволюционировал от простой сети хостов до сложной экосистемы облачных сервисов и приложений. Управление трафиком исключительно по IP-адресам сродни управлению городом по номерам улиц, не зная, какие здания находятся на этих улицах. Управление на уровне приложений обеспечивает необходимое семантическое понимание. WFP, предлагая такие функции, как FWPM_CONDITION_APP_ID, напрямую обеспечивает этот сдвиг, позволяя сетевым политикам соответствовать тому, как пользователи и администраторы воспринимают свою цифровую среду и управляют ею.

1.3. Архитектурный план: Модель привилегированной вспомогательной службы

Прямое управление сетевым трафиком на уровне всей системы, включая его перехват и перенаправление, требует повышенных привилегий, которых стандартное пользовательское приложение (GUI) не должно иметь по соображениям безопасности. Запуск всего приложения с правами администратора является плохой практикой, так как это увеличивает поверхность атаки и создает риски для системы.[1]

Для решения этой критической проблемы безопасности предлагается многокомпонентная архитектура, основанная на **модели привилегированной вспомогательной службы (Privileged Helper Service)**. Этот подход тщательно разделяет логику приложения на компоненты с разными уровнями привилегий, что является ключевым принципом безопасного проектирования.[1] Предлагаемая архитектура неукоснительно соблюдает принцип наименьших привилегий, фундаментальный принцип безопасной разработки программного обеспечения. Путем изоляции операций с высокими привилегиями в выделенной службе Windows и драйвере режима ядра, поверхность атаки пользовательского

GUI-приложения минимизируется. Это значительно снижает потенциальное воздействие уязвимости в GUI, предотвращая ее эскалацию до полного компрометации системы. Это критически важное архитектурное решение для любого профессионального продукта безопасности. В безопасном проектировании программного обеспечения основным правилом является то, что ни одна часть системы не должна иметь больше разрешений, чем ей абсолютно необходимо. GUI, который обрабатывает пользовательский ввод (общий вектор атаки), никогда не должен запускаться с правами администратора. Если бы это произошло, простая переполнение буфера или ошибка синтаксического анализа могли бы привести к полному захвату системы. Разделяя приложение на GUI с низкими привилегиями, службу с высокими привилегиями и драйвер режима ядра, область поражения каждого компонента ограничивается. Служба и драйвер, хотя и привилегированные, могут быть разработаны с меньшей, более проверяемой кодовой базой для своих критических функций, тем самым повышая общую безопасность системы.

Рекомендуемая архитектура состоит из трех основных компонентов:

1. **Графический интерфейс на C# (GUI):** Компонент, с которым взаимодействует пользователь. Он отвечает за отображение списка запущенных процессов, прием выбора пользователя и отображение статуса VPN-соединения. Это приложение работает с обычными правами пользователя.[1]
2. **Привилегированная служба Windows на C++:** "Мозг" всей системы, работающий как служба Windows с правами учетной записи LocalSystem. Эта служба выполняет все задачи, требующие повышенных привилегий: управление туннелем WireGuard, взаимодействие с сетевым стеком Windows и применение правил фильтрации.[1]
3. **Драйвер-выноска (Callout Driver) WFP на C++ (для наиболее продвинутой стратегии):** Компонент режима ядра, необходимый для реализации наиболее надежного и гибкого метода перенаправления трафика. Он работает на самых глубоких уровнях сетевого стека.[1]

Взаимодействие между GUI с низкими привилегиями и службой с высокими привилегиями осуществляется через защищенный канал межпроцессного взаимодействия (IPC), такой как именованные каналы (Named Pipes). Такая архитектура обеспечивает изоляцию компонентов, повышает безопасность и стабильность всей системы.[1] Архитектурное решение о разделении компонентов по уровням привилегий (GUI с низкими привилегиями, служба с высокими привилегиями) напрямую требует реализации надежного и безопасного

механизма межпроцессного взаимодействия (IPC). Без такого механизма эти отдельные компоненты не смогли бы координировать свои действия, что сделало бы всю систему нефункциональной. После того как принято решение о разделении приложения на несколько процессов, особенно когда эти процессы работают в разных контекстах безопасности (например, пользовательский режим против LocalSystem), они теряют прямой доступ к памяти и ресурсам друг друга. Они больше не могут просто вызывать функции через границы процессов. Поэтому хорошо определенный уровень связи становится необходимым. Вот почему IPC, в частности именованные каналы (как будет подробно описано далее), становится критически важной и не подлежащей обсуждению частью общей архитектуры системы, выступая в качестве безопасного канала через эти границы привилегий.

2. Освоение Windows Filtering Platform (WFP)

В этом разделе рассматриваются тонкости Windows Filtering Platform, объясняются ее основные компоненты и способы их использования для расширенного управления трафиком на уровне приложений.

2.1. Основные компоненты WFP: Base Filtering Engine (BFE), уровни фильтрации, фильтры и драйверы-выноски

Windows Filtering Platform (WFP) — это современный и мощный набор API и системных служб, предоставляющий надежную основу для разработки приложений сетевой фильтрации.[1] Она была разработана для замены устаревших, менее гибких технологий, таких как TDI-фильтры, NDIS-фильтры и Winsock LSP.[1] Примечательно, что встроенный в Windows "Брандмауэр Защитника Windows в режиме повышенной безопасности" (WFAS) сам реализован поверх WFP, что подчеркивает его фундаментальную роль в архитектуре сетевой безопасности Windows.[1]

Ключевые компоненты экосистемы WFP включают:

- **Base Filtering Engine (BFE):** Это служба пользовательского режима (bfe.dll,

работающая в процессе svchost.exe), которая управляет всей системой фильтрации.[1] Наша привилегированная служба на C++ будет активно взаимодействовать с BFE для программного добавления, удаления и изменения правил фильтрации. Основные обязанности BFE включают управление компонентами WFP, хранение конфигураций фильтров и обеспечение модели безопасности WFP. Это критически важная зависимость для многочисленных функций Windows, включая брандмауэр Windows, VPN-подключения и IPsec.[5]

- **Уровни фильтрации (Filtering Layers):** Это предопределенные точки перехвата в сетевом стеке Windows, где могут быть применены фильтры.[1] Для задачи раздельного туннелирования на уровне приложений ключевым является уровень FWPM_LAYER_ALE_AUTH_CONNECT_V4 (и его аналог _V6 для IPv6), который позволяет перехватывать попытки установки исходящих TCP/UDP-соединений.[1] WFP включает примерно 10 уровней фильтрации пользовательского режима и около 50 уровней фильтрации режима ядра.[3] Уровни принудительного применения на уровне приложений (ALE) примечательны тем, что они являются stateful, то есть они сохраняют информацию о соединении на протяжении всего жизненного цикла сетевого потока.[4]
- **Фильтры (Filters):** Это фундаментальные правила в WFP, состоящие из набора условий и указанного действия.[1] Условия могут быть очень гранулированными, например, полный путь к приложению, удаленный порт или конкретный сетевой протокол.[1] Действие определяет судьбу трафика, соответствующего этим условиям: он может быть разрешен (Permit), заблокирован (Block) или передан пользовательскому драйверу-выноске (Callout) для дальнейшей обработки.[1] Фильтрам также присваиваются веса, которые определяют их приоритет в данном подуровне во время процесса арбитража фильтров.[4]
- **Драйверы-выноски (Callout Drivers):** Это драйверы режима ядра, которые расширяют нативные возможности WFP путем реализации пользовательских функций-выносок.[1] Они незаменимы для выполнения глубокого анализа или модификации сетевого трафика, обеспечивая расширенную логику, такую как перенаправление, необходимое для истинного раздельного туннелирования. Выноски поддерживают глубокую инспекцию и модификацию как пакетов, так и потоков.[9]

Архитектура WFP, характеризующаяся ее отдельными уровнями, центральным Base Filtering Engine (BFE) и расширяемым механизмом драйверов-выносок,

представляет собой сложную и очень расширяемую основу для сетевой безопасности. Такая конструкция обеспечивает невероятно точный контроль в различных точках сетевого стека, позволяя сторонним разработчикам интегрировать сложную пользовательскую логику без непомерных усилий по перереализации всего сетевого стека. Тот факт, что собственный брандмауэр Windows с расширенной безопасностью (WFAS) от Microsoft построен непосредственно на WFP, еще раз подтверждает его надежную и авторитетную позицию в экосистеме безопасности Windows. Решение Microsoft унифицировать сетевую фильтрацию под WFP, заменив старые, разрозненные API, было стратегическим шагом. Многоуровневый подход является ключевым: он предоставляет конкретные, четко определенные точки, где сетевой трафик может быть перехвачен и проанализирован. Механизм "драйвера-выноски" является ключевой точкой расширяемости, позволяющей разработчикам внедрять собственный код режима ядра для выполнения действий, выходящих за рамки простых правил разрешения/блокировки, таких как глубокая проверка пакетов или, в нашем случае, перенаправление соединений. Эта архитектурная конструкция делает расширенные функции, такие как раздельное туннелирование на уровне приложений, осуществимыми и надежными, поскольку она использует нативные, высокопроизводительные возможности фильтрации ОС.

Глубокая зависимость основных функций сетевой безопасности Windows (таких как встроенный брандмауэр) от WFP означает, что любое продвинутое сетевое приложение, включая наш VPN-клиент WireGuard, *должно* беспрепятственно интегрироваться с WFP. Попытка обойти или игнорировать WFP неизбежно приведет к серьезным конфликтам с существующими системными политиками и другим программным обеспечением безопасности, что приведет к непредсказуемому поведению, нестабильности сети и принципиально ненадежному продукту. Таким образом, понимание и правильная интеграция с WFP является не просто опцией, а обязательным требованием для обеспечения совместимости, правильного арбитража сетевых правил и общей стабильности системы. Если бы VPN-клиент реализовал свои собственные механизмы сетевой фильтрации вне фреймворка WFP, он работал бы изолированно, не зная и потенциально конфликтуя с правилами, применяемыми брандмауэром Windows, антивирусным программным обеспечением или другими сетевыми утилитами, которые *действительно* используют WFP. Такая "теневая фильтрация" создала бы хаотичную среду, где сетевой трафик мог бы быть неожиданно заблокирован или неправильно маршрутизирован, что привело бы к разочаровывающему пользовательскому опыту и многочисленным проблемам с поддержкой.

Интегрируясь с WFP, наш VPN-клиент становится "добропорядочным гражданином" в сетевой экосистеме Windows, участвуя в установленном процессе арбитража и обеспечивая гармоничное сосуществование с другими системными компонентами.

2.2. Проектирование для сосуществования: Пользовательские поставщики и высокоприоритетные подуровни

Значительная проблема возникает при реализации фильтров WFP: обеспечение их гармоничного сосуществования с другими приложениями сетевой фильтрации, включая сам официальный клиент WireGuard. Например, функция "kill-switch" в официальном клиенте WireGuard обычно реализуется с помощью WFP-фильтров, которые явно блокируют весь трафик, не исходящий от его собственного виртуального интерфейса.[1] Если наш пользовательский фильтр

Permit для конкретного приложения (например, chrome.exe) будет иметь более низкий приоритет обработки, чем блокирующий фильтр WireGuard, трафик все равно будет заблокирован, что сделает наше раздельное туннелирование неэффективным.[1]

Порядок применения фильтров WFP определяется его сложным механизмом арбитража.[1] Трафик сначала проходит через

подуровни в порядке убывания их веса (приоритета). Внутри каждого подуровня фильтры также применяются в порядке убывания их индивидуального веса. Решение Block из подуровня или фильтра с более высоким приоритетом обычно переопределяет решение Permit из подуровня с более низким приоритетом.[1]

Решение: Чтобы гарантировать, что трафик нашего приложения правильно управляется и перенаправляется, необходимо создать собственный пользовательский подуровень с достаточно высоким приоритетом. Это гарантирует, что наши правила будут оценены и применены *до* конфликтующих правил брандмауэра Windows или блокирующих фильтров WireGuard.[1] Необходимость создания пользовательских подуровней с высоким приоритетом выявляет распространенную и часто неочевидную проблему в многоуровневых архитектурах безопасности: эффективное управление приоритетом правил для обеспечения применения конкретных политик приложения поверх более широких

системных или сторонних правил безопасности. Это не просто добавление фильтра, а стратегическое позиционирование этого фильтра в сложной иерархии обработки WFP. Неспособность правильно управлять этим арбитражем может привести к скрытым сбоям политики, когда предполагаемый трафик все равно блокируется. В сложной операционной системе, такой как Windows, несколько продуктов безопасности (антивирусы, другие VPN, брандмауэры, встроенные функции ОС) вносят правила в WFP. Каждый из этих компонентов имеет свой собственный набор фильтров и часто работает в своих собственных подуровнях. Если правила раздельного туннелирования нашего VPN находятся в подуровне с низким приоритетом, они могут быть переопределены правилом "блокировать все" с более высоким приоритетом от другого продукта безопасности, даже если наше правило явно разрешает трафик. Свойство

weight является явным механизмом, который WFP предоставляет для управления этим арбитражем. Для VPN абсолютно критично обеспечить обработку его "разделенных" правил *до* любых правил "блокировать все, если не VPN" от самого клиента WireGuard или других брандмауэров. Это гарантирует, что трафик приложения будет правильно идентифицирован и перенаправлен до того, как он будет перехвачен общим правилом блокировки.

Этапы реализации этой критической настройки включают:

1. **Получение дескриптора BFE:** Получение дескриптора Base Filtering Engine (BFE) с помощью FwpmEngineOpen0.[1] Эта функция открывает сессию для движка фильтрации, что является предварительным условием для всех операций управления WFP.
2. **Начало транзакции:** Инициирование транзакции WFP с помощью FwpmTransactionBegin0.[1] Все последующие изменения объектов WFP (поставщиков, подуровней, фильтров) должны быть инкапсулированы в эту транзакцию. Это обеспечивает их атомарное применение; если какой-либо шаг не удастся, вся транзакция откатывается, оставляя систему в согласованном состоянии.[1]
3. **Определение и добавление поставщика/подуровня:** Определение и добавление нашего пользовательского поставщика с помощью FwpmProviderAdd0 и нашего пользовательского подуровня с помощью FwpmSubLayerAdd0.[1] Поставщик служит для идентификации нашего приложения как источника этих правил, что полезно для диагностики, когда на системе присутствует несколько поставщиков WFP.[17]
4. **Установка веса подуровня:** Параметр subLayer.weight является критически важным. Хотя 0x8000 является обычной отправной точкой, для

окончательного переопределения других фильтров и обеспечения приоритета наших правил может потребоваться более высокое значение, например MAXUINT16.[1]

5. **Завершение транзакции:** Завершение транзакции с помощью FwpmTransactionCommit0.[1] Это действие фиксирует все изменения, внесенные в рамках транзакции, в системе WFP, обеспечивая их постоянство и целостность.

Явное акцентирование на инкапсуляции всех модификаций WFP в атомарные транзакции (FwpmTransactionBegin0, FwpmTransactionCommit0) подчеркивает важнейшую передовую практику для поддержания целостности системы при работе со сложными, совместно используемыми системными конфигурациями. Такой транзакционный подход гарантирует, что либо все изменения применяются успешно, либо ни одно из них, тем самым предотвращая переход системы в частичное, несогласованное или потенциально небезопасное состояние в случае ошибки или сбоя во время обновлений конфигурации. Изменение сетевых фильтров — это чувствительная операция, которая напрямую влияет на сетевое подключение и безопасность системы. Если последовательность добавления или удаления фильтров потерпит неудачу на полпути (например, из-за сбоя или непредвиденной ошибки), система может остаться с поврежденной или несогласованной конфигурацией WFP. Это может привести к проблемам с сетевым подключением, уязвимостям безопасности или непредсказуемому поведению. Транзакции WFP обеспечивают атомарность, критически важное свойство для таких операций: если какая-либо часть транзакции терпит неудачу, весь набор изменений автоматически откатывается, гарантируя, что конфигурация WFP системы остается в известном, согласованном состоянии. Это жизненно важно для стабильности и надежности профессионального VPN-клиента, особенно во время установки, удаления или динамических изменений правил, инициированных пользователем.

2.3. Идентификация приложений: Использование FWPM_CONDITION_APP_ID и стратегии обхода EDR/AV

После создания базовой инфраструктуры WFP (поставщика и высокоприоритетного подуровня) следующим шагом является добавление конкретных фильтров для приложений, выбранных пользователем. Это

достигается путем вызова функции FwpmFilterAdd0.[1]

Ключевые свойства в структуре FWPM_FILTER0 для этой цели включают:

- layerKey: Установлено в FWPM_LAYER_ALE_AUTH_CONNECT_V4 (и _V6 для трафика IPv6) для таргетинга попыток исходящих соединений.[1]
- subLayerKey: Установлено в GUID нашего пользовательского, высокоприоритетного подуровня для обеспечения правильного арбитража.[1]
- action.type: Для перенаправления (Стратегия Б) это должно быть FWP_ACTION_CALLOUT_TERMINATING, что указывает WFP передать управление нашему пользовательскому драйверу-выноске режима ядра.[1]

Наиболее важным условием для идентификации трафика приложений является FWPM_CONDITION_APP_ID. Этот тип условия позволяет фильтровать трафик на основе полного пути к исполняемому файлу (например, C:\Program Files\Google\Chrome\Application\chrome.exe). Однако значением этого условия является не просто текстовый путь, а специальный бинарный блоб.[1] Стандартная функция WFP API

FwpmGetAppIdFromFileName0 предназначена для преобразования пути к файлу в этот необходимый формат блоба App ID.[1]

Проблема совместимости с EDR/AV: Значительная проблема возникает из-за того, что FwpmGetAppIdFromFileName0 внутренне вызывает CreateFileW для доступа к исполняемому файлу. Системы защиты конечных точек (EDR) и антивирусы (AV) часто блокируют доступ к своим собственным процессам (например, для предотвращения их отключения или изменения). Следовательно, попытка вызвать FwpmGetAppIdFromFileName0 для процесса EDR/AV может привести к ошибке "Доступ запрещен", что помешает VPN применять правила раздельного туннелирования к этим критически важным приложениям.

Решение: Чтобы обойти эти защитные механизмы и обеспечить совместимость с различными системными средами, рекомендуется вручную формировать блоб App ID на основе пути к файлу, не вызывая при этом CreateFileW. Этот метод был продемонстрирован в таких проектах, как EDRSilencer, который специально нацелен на обход таких ограничений EDR/AV. Реализация этого ручного формирования App ID имеет решающее значение для надежного и профессионального продукта.

Весь процесс добавления или удаления фильтра для приложения всегда должен быть инкапсулирован в транзакцию WFP для обеспечения целостности и

согласованности конфигурации фильтрации.[1]

Необходимость обхода стандартного API FwpmGetAppIdFromFileName0 для процессов EDR/AV выявляет значительную, часто упускаемую из виду проблему совместимости в экосистеме безопасности Windows. Легитимные приложения, пытающиеся управлять сетевым трафиком на низком уровне, могут непреднамеренно запускать защитные механизмы другого программного обеспечения безопасности, что приводит к ошибкам "Доступ запрещен" и функциональным ограничениям. Это требует разработки и внедрения передовых методов обхода для обеспечения полной совместимости и ожидаемой функциональности в реальных условиях. Хотя WFP предоставляет удобный API (FwpmGetAppIdFromFileName0) для преобразования пути к файлу в App ID, его внутренняя зависимость от CreateFileW создает прямой конфликт с механизмами самозащиты продуктов EDR/AV. Эти средства безопасности предназначены для предотвращения несанкционированного доступа к их собственным двоичным файлам, и попытка VPN запросить их App ID может быть расценена как подозрительное поведение. Для коммерческого VPN это критическое препятствие, поскольку пользователи ожидают возможности раздельного туннелирования *любого* приложения, независимо от того, является ли оно стандартной программой или защищенным агентом EDR/AV. Проект EDRSilencer демонстрирует практическое, хотя и сложное, решение этой проблемы, подчеркивая глубокие технические знания и навыки реверс-инжиниринга, иногда необходимые для достижения полной совместимости системы.

Само существование и практическая необходимость таких инструментов и методов, как те, что используются в EDRSilencer для ручного создания блобов App ID, иллюстрируют непрерывную "гонку вооружений" между защитным программным обеспечением безопасности и приложениями, требующими глубокого, низкоуровневого взаимодействия с операционной системой. Это означает, что разработчики таких сетевых приложений должны не только обладать глубоким пониманием внутренних механизмов ОС, но и оставаться бдительными и адаптируемыми к меняющемуся ландшафту защитных механизмов. Поддержание функциональности продукта в разнообразных и защищенных средах требует постоянных исследований и разработок для противодействия новым методам самозащиты, применяемым поставщиками EDR/AV. Ландшафт кибербезопасности динамичен. Поставщики EDR и AV постоянно улучшают свои возможности самозащиты и обнаружения. Метод обхода, работающий сегодня, может стать неэффективным с будущим обновлением программного обеспечения. Это означает, что разработка

VPN-клиента на основе WFP — это не статический проект, а проект, требующий постоянного мониторинга экосистемы безопасности, активных исследований новых методов защиты и приверженности адаптации решения для поддержания его функциональности. Эти постоянные усилия являются необходимой платой за предоставление надежного и производительного продукта в среде, ориентированной на безопасность.

3. Программное управление туннелями WireGuard

В этом разделе подробно описываются методы управления туннелем WireGuard, подчеркивая преимущества прямого взаимодействия с API для обеспечения бесперебойной работы пользователя.

3.1. Сравнительный анализ: Утилиты командной строки против нативного C API (wireguard-nt)

Для реализации динамического раздельного туннелирования приложению необходим надежный способ управления жизненным циклом и конфигурацией туннеля WireGuard. Существует два основных подхода:

- **Метод 1: Управление через утилиты командной строки (wireguard.exe и wg.exe)**
  - Этот метод предполагает, что приложение будет взаимодействовать с официальным клиентом WireGuard для Windows как с внешним инструментом.
  - **Установка туннеля как службы:** Приложение может программно установить туннель как службу Windows. Для этого необходимо из C++ кода запустить процесс wireguard.exe с аргументом /installtunnelservice и путем к конфигурационному файлу (например, "C:\Program Files\WireGuard\wireguard.exe" /installtunnelservice "C:\path\to\my_tunnel.conf"). Эта команда создает службу Windows с именем вида WireGuard Tunnel$<имя_конфига>.[1]
  - **Управление службой:** После установки службы ею можно управлять стандартными средствами Windows, вызывая из кода утилиты sc.exe или

net.exe для запуска (sc start WireGuardTunnel$my_tunnel) и остановки (sc stop WireGuardTunnel$my_tunnel) туннеля.[1]
    - **Получение статуса и конфигурации:** Для получения информации о состоянии туннеля (публичный ключ, endpoint, время последнего "рукопожатия" (handshake) и статистика трафика) используется утилита wg.exe. Вызов wg show <имя_интерфейса> возвращает текстовую информацию, которую можно парсить для отображения в GUI.[1]
    - **Архитектурный компромисс:** Этот подход прост в реализации, так как не требует интеграции с библиотеками WireGuard и рассматривает его как "черный ящик". Однако он имеет существенные недостатки для профессионального продукта. Любое изменение конфигурации (например, добавление IP-адреса в AllowedIPs) требует остановки службы, модификации текстового .conf файла на диске и перезапуска службы. Это приводит к кратковременному прерыванию VPN-соединения, что негативно сказывается на пользовательском опыте.[1] Такой метод подходит для прототипов или простых утилит, но не для полнофункционального коммерческого VPN-клиента, где требуется бесшовная работа.
- **Метод 2: Управление через нативный C API (wireguard-nt)**
    - Это более современный и предпочтительный метод, использующий нативный C API, предоставляемый проектом wireguard-nt в виде wireguard.dll.[20] Этот подход позволяет управлять туннелем напрямую из кода, без вызова внешних процессов и перезапуска служб.
    - **Загрузка API:** Ядро приложения на C++ должно динамически загружать wireguard.dll с помощью LoadLibraryEx() и получать адреса функций через GetProcAddress().[20] Заголовочный файл wireguard.h, поставляемый с wireguard-nt, содержит все необходимые определения типов данных и прототипы функций.[20] Пример инициализации можно найти в файле example.c в исходном коде проекта.[20]
    - **Ключевые функции API:**
        - WireGuardCreateAdapter: Программно создает виртуальный сетевой адаптер WireGuard, указывая его имя (например, "MyVPN") и тип ("WireGuard").[20]
        - WireGuardSetConfiguration: Применяет конфигурацию к созданному адаптеру. Важнейшее отличие от предыдущего метода заключается в том, что конфигурация передается в виде C-структур (WIREGUARD_INTERFACE, WIREGUARD_PEER и др.), а не через .conf файл. Это позволяет изменять параметры, включая AllowedIPs, "на

лету".[20]
    - WireGuardSetAdapterState: Включает (WIREGUARD_ADAPTER_STATE_UP) или выключает (WIREGUARD_ADAPTER_STATE_DOWN) адаптер.[20]
  - **Преимущество "внутрипроцессного" подхода:** Использование нативного API является ключом к созданию гибкого и отзывчивого приложения. Когда пользователь в GUI выбирает приложение для исключения из туннеля, и система (в рамках Стратегии A) определяет IP-адреса этого приложения, привилегированная служба может немедленно обновить конфигурацию, вызвав WireGuardSetConfiguration с новым набором AllowedIPs. Это происходит без разрыва соединения и каких-либо видимых для пользователя задержек.[1] Данный метод превращает WireGuard из внешней утилиты в интегрированную библиотеку, являющуюся частью приложения. Именно этот подход лежит в основе рекомендуемой архитектуры и обеспечивает необходимую гибкость для реализации продвинутого раздельного туннелирования.

Явный контраст между использованием утилит командной строки и нативного C API для управления туннелем WireGuard прекрасно иллюстрирует общий, критически важный компромисс в разработке. В то время как подход с CLI предлагает простоту в первоначальной реализации, он серьезно ухудшает пользовательский опыт (из-за прерываний соединения) и вводит накладные расходы на производительность (из-за частых перезапусков служб). Нативный API, хотя и требует более сложной интеграции, обеспечивает значительно превосходящий, бесшовный и отзывчивый продукт. Этот выбор напрямую влияет на жизнеспособность продукта на рынке. Для любого коммерческого программного обеспечения пользовательский опыт является основным отличием. VPN, который постоянно отключается и переподключается при изменении правил раздельного туннелирования, принципиально ошибочен с точки зрения удобства использования. Нативный API wireguard-nt напрямую решает эту проблему, позволяя динамически обновлять конфигурацию в памяти без разрыва туннеля. Это означает, что пользователь может добавлять или удалять приложения из раздельного туннелирования, не замечая никаких прерываний в сетевом подключении. Это не просто техническое предпочтение, а стратегическое решение, которое напрямую влияет на удовлетворенность и удержание пользователей.

Явное использование LoadLibraryEx() и GetProcAddress() для загрузки wireguard.dll демонстрирует надежный и гибкий подход к управлению внешними

библиотечными зависимостями. Этот механизм динамической загрузки позволяет легче обновлять основной компонент WireGuard без необходимости полной перекомпиляции или повторного выпуска всего приложения. Это особенно выгодно для компонентов, чувствительных к безопасности, таких как VPN-туннели, где обновления (например, для исправлений безопасности или улучшений производительности) могут быть частыми. Статическая компоновка большой библиотеки, такой как wireguard.dll, может увеличить размер исполняемого файла и затруднить обновление только компонента библиотеки. Динамическая загрузка, с другой стороны, позволяет VPN-клиенту загружать библиотеку WireGuard во время выполнения. Это означает, что если выпущена новая версия wireguard.dll, VPN-клиент потенциально может быть обновлен путем простой замены файла DLL, без необходимости распространения нового исполняемого файла. Эта модульность полезна для обслуживания, исправления ошибок и общей гибкости продукта.

3.2. Реализация динамических обновлений конфигурации (WireGuardSetConfiguration)

Суть динамического управления конфигурацией заключается в функции WireGuardSetConfiguration. Эта функция позволяет привилегированной службе C++ обновлять параметры туннеля WireGuard, включая критически важный список AllowedIPs, без прерывания активного VPN-соединения.[22] В отличие от подхода с командной строкой, который требует модификации текстовых файлов и перезапуска служб,

WireGuardSetConfiguration принимает данные конфигурации в виде структурированных C-типов (WIREGUARD_INTERFACE, WIREGUARD_PEER, WIREGUARD_ALLOWED_IP). Это позволяет выполнять программные обновления в памяти.[22]

Когда пользователь выбирает приложение для раздельного туннелирования, привилегированная служба определяет необходимые изменения IP-адресов (если используется Стратегия А) или подготавливает фильтр WFP (для Стратегии Б). Затем она конструирует обновленные структуры конфигурации WireGuard и вызывает WireGuardSetConfiguration для мгновенного применения этих изменений.[22] Эта возможность бесшовного обновления критически важна для обеспечения профессионального и отзывчивого пользовательского опыта, при

котором изменения в правилах раздельного туннелирования применяются немедленно без каких-либо видимых сетевых сбоев.[1]

Функция WireGuardSetConfiguration является техническим средством, обеспечивающим "бесшовный пользовательский опыт", обещанный раздельным туннелированием на уровне приложений. Ее способность обновлять конфигурацию VPN в реальном времени, без отключения, напрямую приводит к плавному и не прерывающемуся взаимодействию для конечного пользователя. Это ключевое отличие для премиального VPN-продукта. Восприятие пользователем качества VPN сильно зависит от его отзывчивости и стабильности. Если добавление или удаление приложения из раздельного туннелирования вызывает кратковременное отключение, это нарушает рабочие процессы и вызывает разочарование. WireGuardSetConfiguration обходит это, позволяя базовому VPN-туннелю динамически адаптировать свои правила маршрутизации. Эта техническая возможность напрямую связана с превосходным пользовательским опытом, что делает ее краеугольным камнем рекомендуемой архитектуры.

4. Межпроцессное взаимодействие (IPC) и мост пользователь-ядро

В этом разделе подробно описываются механизмы связи, необходимые для многокомпонентной, привилегированной архитектуры, охватывающей пользовательский интерфейс и ядро.

4.1. Пользовательский интерфейс (C#): Перечисление процессов и выбор пользователя

Графический пользовательский интерфейс (GUI), разработанный на C#, служит основным интерфейсом для пользователя для управления раздельным туннелированием. Его основная функция — предоставить интуитивно понятный список запущенных приложений в системе, из которого пользователь может

выбрать те, чей трафик должен обходить VPN-туннель.[1]

Получение информации о процессе:

- Для получения списка всех активных процессов используется метод System.Diagnostics.Process.GetProcesses().[23] Этот метод возвращает массив объектов
  Process, каждый из которых представляет один запущенный процесс.
- Для каждого объекта Process извлекается важная информация для улучшения пользовательского опыта и облегчения фильтрации:
  - ProcessName: process.ProcessName предоставляет удобочитаемое имя.[1]
  - Путь к исполняемому файлу: process.MainModule.FileName является критически важной информацией. Этот полный путь будет передан в службу C++ для настройки фильтров WFP на основе идентификатора приложения.[1]
  - Иконка приложения: System.Drawing.Icon.ExtractAssociatedIcon(process.MainModule.FileName) извлекает иконку приложения, значительно улучшая удобство использования, позволяя пользователю быстрее идентифицировать приложения визуально.[26]

Критический аспект: Совместимость 32-битных и 64-битных процессов:

- Распространенная и серьезная проблема возникает, когда 32-битное C#-приложение пытается получить доступ к свойству process.MainModule 64-битного процесса. Эта операция обычно приводит к исключению System.ComponentModel.Win32Exception с сообщением "A 32 bit process cannot access modules of a 64 bit process".[28] Это исключение может полностью заблокировать работу функции перечисления процессов.
- **Решение:** Чтобы устранить эту проблему, C#-приложение должно быть настроено на работу как 64-битный процесс в 64-битной операционной системе. Это достигается правильной настройкой проекта в Visual Studio:
  1. **Платформа решения (Platform target):** Должна быть установлена в "Any CPU".[1]
  2. **Предпочитать 32-разрядную версию (Prefer 32-bit):** Этот флажок в свойствах проекта должен быть снят.[1]
- Эта конфигурация гарантирует, что на 64-битной системе приложение будет запущено как 64-битный процесс, что даст ему необходимые права для запроса информации у других 64-битных процессов. Кроме того, код должен корректно обрабатывать исключения при доступе к MainModule, так как для некоторых системных процессов (например, "System" или "System Idle

Process") это свойство недоступно в принципе.[28]

Исключение System.ComponentModel.Win32Exception, возникающее, когда 32-битное C#-приложение пытается получить доступ к MainModule 64-битного процесса, является классической, неочевидной ловушкой в разработке под Windows. Эта проблема подчеркивает критическую важность понимания нюансов базовой архитектуры операционной системы (32-битные против 64-битных процессов) и того, как они влияют на доступ к данным между процессами. Это требует специфических настроек сборки и надежной обработки ошибок, демонстрируя, что даже кажущиеся прямолинейными задачи, такие как перечисление процессов, могут скрывать сложные взаимодействия. Разработчик может изначально написать код для перечисления процессов и доступа к MainModule.FileName, не учитывая разрядность участвующих процессов. В 64-битной системе, если C#-приложение скомпилировано как 32-битное (что часто является настройкой по умолчанию или предпочтительной для более широкой совместимости), оно потерпит неудачу при попытке проверить 64-битные процессы. Это тонкая, но критическая ошибка времени выполнения, которую может быть трудно диагностировать без понимания базовой архитектуры процессов Windows. Решение заключается не в самой логике кода, а в конфигурации сборки, что подчеркивает целостный характер надежной разработки программного обеспечения.

Необходимость извлечения и отображения ProcessName, FileName и, особенно, Icon для каждого запущенного приложения является прямым примером того, как требования к пользовательскому опыту (UX) фундаментально определяют конкретные технические реализации. Предоставление визуальных подсказок, таких как иконки, значительно улучшает удобство использования и интуитивность приложения, позволяя пользователям быстро идентифицировать и выбирать программы. Это, в свою очередь, требует надежной обработки ошибок для сценариев, когда информация MainModule (и, следовательно, иконка) может быть недоступна для определенных системных процессов. С чисто функциональной точки зрения, GUI мог бы просто перечислять идентификаторы процессов или имена. Однако для удобного приложения визуальные элементы, такие как иконки, имеют решающее значение для быстрого распознавания и простоты использования. Эта, казалось бы, незначительная функция UX налагает значительное техническое требование: способность надежно извлекать иконки и пути из различных процессов, включая обработку граничных случаев, таких как системные процессы, где MainModule может быть недоступен для прямого запроса. Это демонстрирует, как дизайнерские решения на уровне

пользовательского интерфейса распространяются вниз, влияя на низкоуровневые детали технической реализации и стратегии обработки ошибок.

4.2. Безопасное IPC: Именованные каналы для связи между GUI и привилегированной службой

Поскольку GUI работает с правами обычного пользователя, а основная логика находится в службе C++, работающей с привилегиями LocalSystem, для их взаимодействия необходим надежный и безопасный механизм межпроцессного взаимодействия (IPC).[1]

Анализ вариантов IPC:

- **WCF (Windows Communication Foundation):** Хотя WCF является мощным инструментом, он часто считается избыточно сложным и тяжеловесным для простой, локализованной связи между двумя процессами на одной машине.[1]
- **Файлы, отображаемые в память (Memory-Mapped Files):** Этот метод обеспечивает очень высокую производительность, но требует ручной реализации примитивов синхронизации (таких как мьютексы и семафоры) для предотвращения состояний гонки и обеспечения согласованности данных, что добавляет значительные накладные расходы на разработку и потенциальные ошибки.[1]
- **Именованные каналы (Named Pipes):** Это рекомендуемый выбор. Именованные каналы — это стандартный, хорошо документированный и изначально безопасный механизм для клиент-серверного IPC в средах Windows.[31] Они идеально подходят для модели "GUI-клиент" и "служба-сервер" благодаря встроенным функциям безопасности и надежности.

Схема реализации на Named Pipes:

- **C++ служба (сервер):** В отдельном потоке служба C++ создает сервер именованного канала с помощью CreateNamedPipe. Затем она ожидает подключения клиента (GUI) с помощью ConnectNamedPipe. Для каждого подключения служба создает экземпляры для чтения (ReadFile) и записи (WriteFile).[1]
- **C# GUI (клиент):** C# GUI использует класс System.IO.Pipes.NamedPipeClientStream для подключения к каналу,

созданному службой. После подключения клиент может отправлять сериализованные команды службе (например, в виде строк типа "ADD_APP:C:\path\to\app.exe" или "REMOVE_APP:C:\path\to\app.exe").[31]
- **Двунаправленная связь:** Именованный канал служит двунаправленным каналом связи. Он используется не только для отправки команд от GUI к службе, но и для обратной связи службы с GUI, такой как обновления статуса туннеля, статистика трафика и подтверждения выполнения команд.[1]

4.3. Маршалинг данных между C# и C++ (P/Invoke)

Для некоторых задач, например для получения от C++ ядра списка уже отфильтрованных приложений, может потребоваться прямой вызов функций из C++ DLL. Это делается с помощью механизма Platform Invoke (P/Invoke).[1]

Определение P/Invoke: В C# коде объявляются статические extern методы, помеченные атрибутом ``, где core.dll — имя библиотеки C++.[1]

Сложность маршалинга (преобразования типов): Передача простых типов (int, bool) тривиальна. Однако передача сложных структур данных, таких как список строк, требует особого подхода. Нельзя напрямую передать std::vector<std::string> из C++ в C# List<string>.[1]

Решение заключается в использовании C-совместимого формата данных:

1. **На стороне C++:** Экспортируемая функция должна выделять память под массив указателей на C-строки (char**) и возвращать этот массив вместе с его размером (например, через out-параметр). Пример:
   C++
   ```
   // Пример экспортируемой C++ функции
   extern "C" __declspec(dllexport) void GetFilteredApps(char*** appPaths, int* count);
   extern "C" __declspec(dllexport) void FreeFilteredAppsArray(char** appPaths, int count);
   ```

2. **На стороне C#:** В объявлении DllImport используется out IntPtr или ref IntPtr и атрибут [MarshalAs]. C# код вызывает первую функцию, получает массив строк, использует его, а затем вызывает вторую функцию (FreeFilteredAppsArray) для освобождения памяти, выделенной в C++, чтобы избежать утечек памяти.[35] Пример:
   C#

```
public static extern void GetFilteredApps(out IntPtr appPaths, out int count);
public static extern void FreeFilteredAppsArray(IntPtr appPaths, int count);

// В коде C#
IntPtr pathsPtr;
int appCount;
GetFilteredApps(out pathsPtr, out appCount);
string apps = new string[appCount];
IntPtr ptrs = new IntPtr[appCount];
Marshal.Copy(pathsPtr, ptrs, 0, appCount);
for (int i = 0; i < appCount; i++)
{
    apps[i] = Marshal.PtrToStringAnsi(ptrs[i]);
}
FreeFilteredAppsArray(pathsPtr, appCount); // Обязательно!
```

Этот подход обеспечивает корректную и безопасную передачу списков строк между управляемым кодом C# и нативным кодом C++.[1]

4.4. Перенаправление трафика в режиме ядра: Драйвер-выноска WFP

Хотя фильтры WFP, описанные в предыдущей части, могут разрешать или блокировать трафик приложения, они не управляют его маршрутизацией. Если в конфигурации WireGuard указано AllowedIPs = 0.0.0.0/0, то системная таблица маршрутизации все равно попытается направить разрешенный трафик в туннель. Чтобы по-настоящему разделить трафик и направить его на физический сетевой интерфейс, необходимо перенаправить само соединение, что требует использования драйвера режима ядра.[1]

Обоснование для вмешательства в режим ядра:
Для принудительного направления трафика в обход VPN-туннеля необходимо вмешаться в процесс установки соединения на низком уровне. WFP предоставляет для этого специальные уровни перенаправления, доступ к которым возможен только из режима ядра через драйвер-выноску (callout driver).1
Архитектура решения:

1. Служба C++ добавляет WFP-фильтр для целевого приложения (например,

chrome.exe).[1]
2. В качестве действия для этого фильтра указывается FWP_ACTION_CALLOUT_TERMINATING и GUID нашего драйвера-выноски.[1]
3. Драйвер-выноска регистрирует свою функцию-классификатор (classifyFn) в системе на уровнях FWPM_LAYER_ALE_CONNECT_REDIRECT_V4 и FWPM_LAYER_ALE_CONNECT_REDIRECT_V6.[1]
4. Когда chrome.exe пытается установить новое соединение, WFP находит соответствующий фильтр и вызывает classifyFn нашего драйвера.[1]
5. Внутри classifyFn драйвер изменяет параметры соединения, чтобы оно было установлено через физический сетевой интерфейс, а не через WireGuard.[1]

Реализация драйвера-выноски (classifyFn):
Центральным элементом драйвера является его callback-функция classifyFn. Она получает на вход данные о классифицируемом сетевом событии и должна принять решение о его судьбе. Алгоритм перенаправления в classifyFn включает:

1. **Получение хендла классификации:** Вызывается FwpsAcquireClassifyHandle0, чтобы получить хендл, который будет использоваться в последующих вызовах.[40]
2. **Получение изменяемых данных соединения:** Вызывается FwpsAcquireWritableLayerDataPointer0. Эта функция возвращает указатель на структуру FWPS_CONNECT_REQUEST0, содержащую все параметры исходящего соединения (локальный/удаленный IP и порт, PID процесса и т.д.).[40]
3. **Модификация параметров соединения:** Это ключевой шаг. Драйвер изменяет поля в структуре FWPS_CONNECT_REQUEST0.[1]
    - **Перенаправление на локальный прокси (стандартный паттерн):** Можно изменить remoteAddressAndPort на адрес локального прокси-сервиса (например, 127.0.0.1:8888). В этом случае прокси-сервис, работающий в пользовательском режиме, принимает соединение и уже от своего имени устанавливает новое соединение с исходным адресатом через физический интерфейс.[43]
    - **Прямое перенаправление на интерфейс (более сложный подход):** Можно попытаться манипулировать параметрами соединения, чтобы заставить сетевой стек направить его через конкретный физический интерфейс, LUID которого можно определить заранее с помощью GetAdaptersAddresses.[44] Этот метод сложнее, но позволяет избежать необходимости в промежуточном прокси.
4. **Применение изменений:** Вызывается FwpsApplyModifiedLayerData0, чтобы зафиксировать изменения в сетевом стеке.[40]

5. **Разрешение модифицированного соединения:** Функция classifyFn должна вернуть FWP_ACTION_PERMIT, чтобы разрешить установку уже измененного соединения. В структуре classifyOut устанавливается classifyOut->actionType = FWP_ACTION_PERMIT.[1]
6. **Освобождение хендла:** Вызывается FwpsReleaseClassifyHandle0.[40]

Основная цель — обойти маршрут по умолчанию, который ведет в туннель WireGuard. Простое разрешение (Permit) трафика на уровне ALE_AUTH_CONNECT недостаточно, так как решение о маршрутизации принимается на более низких уровнях стека. Перенаправление соединения на уровне ALE_CONNECT_REDIRECT — это единственный надежный способ гарантировать, что трафик пойдет по нужному пути.[43]

Взаимодействие между службой и драйвером (DeviceIoControl):
Привилегированная служба C++ должна сообщать драйверу, трафик каких приложений необходимо перенаправлять. Это классическая задача коммуникации между пользовательским режимом (user-mode) и режимом ядра (kernel-mode).1

- **Схема реализации на IOCTL:**
    1. **Драйвер:** В своей функции DriverEntry создает объект устройства (IoCreateDevice) и символическую ссылку на него (IoCreateSymbolicLink). Также он регистрирует функцию-обработчик для запросов IRP_MJ_DEVICE_CONTROL.[53]
    2. **Служба C++:** Открывает хендл драйвера с помощью CreateFile, используя имя символической ссылки (например, \\.\MyWfpDriver).[55]
    3. **Служба C++:** Отправляет данные в драйвер (например, список PID процессов для перенаправления) с помощью функции DeviceIoControl, указывая пользовательский код операции (IOCTL).[57]

Усиление безопасности (Hardening): Любые данные, поступающие из пользовательского режима, являются недоверенными. Драйвер обязан тщательно проверять все указатели и буферы, полученные через DeviceIoControl. Если этого не сделать, вредоносное приложение может передать в драйвер указатель на область памяти ядра, что приведет к отказу системы (BSOD) или к уязвимости безопасности, позволяющей повысить привилегии.[1] Для безопасной обработки в обработчике

IRP_MJ_DEVICE_CONTROL необходимо:

- Оборачивать все операции доступа к пользовательским буферам в блоки try/except.[1]
- Использовать функции ProbeForRead и ProbeForWrite для проверки того, что

переданные указатели и размеры буферов указывают на валидные, доступные для чтения/записи области памяти пользовательского режима.[1] Это абсолютное и не подлежащее обсуждению требование безопасности при разработке драйверов.

5. Развертывание, подписание и установка

Создание функционального драйвера — это только половина дела. Для его работы на компьютерах пользователей необходимо решить сложные вопросы цифровой подписи и создать надежный установщик.

5.1. Требования к подписи драйвера: Тестовый и продуктовый режимы

Политика подписи драйверов в Windows строго зависит от версии ОС и состояния Secure Boot.[1]

Этап разработки (тестовая подпись):

- На тестовой машине, где будет проводиться отладка, администратор должен включить тестовый режим подписи командой bcdedit /set testsigning on (требуется перезагрузка).[25]
- В свойствах проекта драйвера в Visual Studio можно установить режим подписи "Test Sign". При сборке проекта будет сгенерирован тестовый сертификат, и .sys файл будет подписан им.[26] Этого достаточно для загрузки драйвера только на машинах с включенным тестовым режимом.

Продуктовый релиз (аттестационная подпись):

- **Неоспоримая реальность:** Начиная с Windows 10 версии 1607, при включенной функции Secure Boot, операционная система не загрузит ни один новый драйвер режима ядра, если он не подписан Microsoft через портал разработчиков оборудования.[1] Самостоятельная подпись драйвера, даже с использованием коммерческого сертификата для подписи кода, недостаточна.[1]
- Попытка установить некорректно подписанный драйвер на обычной

пользовательской машине с включенным Secure Boot приведет либо к блокировке установки, либо к ошибке "Secure Boot Violation" при загрузке системы, что является абсолютно неприемлемым для конечного пользователя.[1]

Процесс получения подписи Microsoft (Attestation Signing):

1. **Приобретение EV-сертификата (Extended Validation):** Необходимо купить EV Code Signing сертификат. Он является обязательным условием для регистрации аккаунта на портале Microsoft.[1]
2. **Регистрация в Windows Hardware Developer Program:** Создать учетную запись на Partner Center (ранее - SysDev).[65]
3. **Упаковка драйвера в CAB-файл:** Все файлы драйвера (.sys, .inf, .cat) упаковываются в один CAB-архив с помощью утилиты makecab.exe.[65]
4. **Подпись CAB-файла:** Созданный CAB-архив подписывается вашим EV-сертификатом.[1]
5. **Отправка на портал:** Подписанный CAB-файл загружается на Partner Center для "аттестационной подписи". Microsoft автоматически проверяет пакет и, в случае успеха, переподписывает его своей подписью. В результате вы получаете .cat файл, подписанный Microsoft.[1]
6. **Результат:** Драйвер с таким каталогом будет доверенным на всех версиях Windows и будет корректно загружаться при включенном Secure Boot.[1]

Аттестационная подпись — единственный путь для публичного распространения драйвера.[1]

5.2. Создание надежного установщика (WiX/NSIS)

Установщик должен корректно развернуть все компоненты приложения: GUI на C#, привилегированную службу на C++ и драйвер режима ядра.[1]

- **WiX Toolset:** Мощный инструмент на основе XML для создания MSI-пакетов. Для установки и управления службой Windows используются элементы <ServiceInstall> и <ServiceControl>.[68]
    - **Установка драйвера с помощью WiX:** Сам по себе установщик Windows (MSI) не поддерживает установку драйверов режима ядра (атрибут Type="kernelDriver" в ServiceInstall не работает). Стандартной практикой является использование Custom Action для запуска утилиты DPInst.exe

(Driver Package Installer) из состава WDK. DPInst.exe корректно обрабатывает INF-файл и устанавливает драйвер в систему.[32]

- **NSIS (Nullsoft Scriptable Install System):** Альтернатива на основе скриптов.
  - Для установки службы можно использовать плагин SimpleSC::Install или вызывать системную утилиту sc.exe create.[1]
  - Для установки драйвера используется команда nsExec::ExecToLog для вызова PnPutil.exe (встроен в Windows) или DPInst.exe с путем к INF-файлу.[71]

Чистое удаление: Установщик должен обеспечивать полное и чистое удаление всех компонентов. При удалении приложения необходимо не только остановить и удалить службу и драйвер, но и корректно удалить все созданные WFP-объекты (фильтры, подуровни, поставщика). Это делается вызовами FwpmFilterDeleteByKey0, FwpmSubLayerDeleteByKey0 и FwpmProviderDeleteByKey0 из кода службы перед ее остановкой.[76] Образцы WFP от Microsoft содержат примеры кода для корректной деинсталляции.[1]

6. Вопросы производительности и перспективы

6.1. Вопросы производительности

- **Сравнение WFP и NDIS:** Хотя NDIS-фильтры работают на более низком уровне сетевого стека, для задач IP-фильтрации на современных версиях Windows (начиная с Vista) WFP является более производительным и эффективным решением.[1] Это связано с тем, что WFP-выноски обрабатывают только релевантный трафик (согласно условиям фильтра) и работают с нативной для современного стека структурой NET_BUFFER_LIST (NBL), избегая затратных преобразований в устаревший формат NDIS_PACKET, необходимых для старых NDIS IM-драйверов.[1] Некоторые тесты показывают, что для чистого захвата всех пакетов WFP может быть медленнее, но для нашей задачи перенаправления, где важен контекст высокого уровня (например, App ID), WFP является единственным подходящим и современным выбором.[1]

- **Накладные расходы:** Переход контекста из пользовательского режима (C++ служба) в режим ядра (драйвер-выноска) и обратно не является бесплатной операцией с точки зрения производительности процессора.[1] Однако для нашей задачи, где решения принимаются на уровне установки соединения (а не для каждого пакета), эти накладные расходы пренебрежимо малы и не окажут заметного влияния на пропускную способность сети.[1]

6.2. Новые технологии: eBPF для Windows

Этот раздел представляет взгляд в будущее, чтобы оценить долгосрочную актуальность выбранной архитектуры. Microsoft активно работает над реализацией eBPF (extended Berkeley Packet Filter) для Windows.[1]

- **eBPF и WFP:** Важно понимать, что eBPF для Windows не заменяет WFP, а строится поверх него.[1] Многие сетевые хуки eBPF, такие как BPF_CGROUP_INET4_CONNECT (используемый для перенаправления соединений), реализованы через существующие уровни WFP, в частности, через уровень перенаправления соединений.[1] Это означает, что WFP остается фундаментальным слоем сетевой фильтрации в Windows.
- **Перспективы:** Инвестиции в разработку решения на базе WFP сегодня являются стратегически верными. Накопленные знания и созданная архитектура будут напрямую применимы и легко адаптируемы для eBPF, когда эта технология станет общедоступной в Windows. Это открывает путь к созданию более кросс-платформенного кода (совместимого между Linux и Windows) в будущем.[1]

7. Заключение и рекомендации

Данный отчет детально проанализировал различные подходы к реализации раздельного туннелирования на уровне приложений для WireGuard VPN-клиента в Windows.

Настоятельно рекомендуется **Стратегия Б (Перенаправление с помощью WFP и драйвера-выноски)** как наиболее надежное, безопасное и функциональное

решение, полностью отвечающее как явным, так и неявным требованиям, предъявляемым к современному VPN-приложению. Хотя этот подход требует значительно больших усилий в разработке и сопряжен с необходимостью прохождения процедуры подписи драйвера Microsoft, он единственный обеспечивает необходимый уровень контроля и бесшовный пользовательский опыт.

Ниже приведена итоговая таблица, сравнивающая рассмотренные методы:

Таблица 1: Сравнение методов реализации раздельного туннелирования

Метод	Гранулярность	Сложность	Накладные расходы на производительность	Уровень безопасности	Препятствия при развертывании
WireGuard AllowedIPs	IP/Подсеть	Низкая	Нативная (отсутствуют)	Высокий	Простое (редактирование.conf)
WFP-фильтрация по App-ID (Block/Permit)	Приложение	Средняя	Низкие	Высокий	Среднее (требуется привилегированная служба)
WFP-перенаправление (драйвер-выноска)	Приложение	Высокая	Очень низкие (на уровне соединения)	Наивысший (при корректной реализации)	Высокое (требуется подпись драйвера Microsoft)

Эта таблица наглядно демонстрирует компромиссы между различными подходами. Она показывает, что значительное увеличение сложности и трудозатрат на развертывание метода с WFP-перенаправлением полностью оправдывается его непревзойденной гибкостью, производительностью и функциональностью, что делает его единственным выбором для создания профессионального и конкурентоспособного продукта.

Рекомендации:

- **Приоритет Стратегии Б:** Сосредоточить усилия на реализации Стратегии Б, признавая ее как наиболее перспективный и надежный путь для обеспечения

высококачественного раздельного туннелирования на уровне приложений.

- **Инвестиции в экспертизу WFP и драйверов:** Учитывая сложность WFP и разработки драйверов режима ядра, необходимо обеспечить наличие или развитие глубокой экспертизы в этих областях.
- **Планирование процесса подписи драйверов:** Заранее включить в план проекта этапы получения EV-сертификата и прохождения аттестационной подписи Microsoft, так как это длительный и обязательный процесс для публичного распространения.
- **Разработка надежного установщика:** Создать комплексный установщик, который будет корректно развертывать все компоненты (GUI, службу, драйвер) и обеспечивать чистое удаление, включая очистку объектов WFP.
- **Постоянный мониторинг:** Поддерживать актуальность знаний о WFP и eBPF, а также о методах обхода защитных решений (EDR/AV), чтобы обеспечивать долгосрочную совместимость и функциональность продукта.
- **Строгое соблюдение принципов безопасности:** При разработке всех компонентов, особенно драйвера и привилегированной службы, неукоснительно следовать принципам наименьших привилегий и безопасной обработки данных из пользовательского режима.

Источники

1. Раздельное туннелирование WireGuard в Windows_.pdf
2. Windows Filtering Platform - Win32 apps | Microsoft Learn, дата последнего обращения: июня 18, 2025, https://learn.microsoft.com/en-us/windows/win32/fwp/windows-filtering-platform-start-page
3. Windows Filtering Platform, engine for local security - CiteSeerX, дата последнего обращения: июня 18, 2025, https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=32ea7c0a9a1c0d1b8107887f574221c75e639ae3
4. What The Filter (WTF) is Going on With Windows Filtering Platform (WFP)? - Zero Networks, дата последнего обращения: июня 18, 2025, https://zeronetworks.com/blog/wtf-is-going-on-with-wfp
5. WFP Architecture - Win32 apps - Learn Microsoft, дата последнего обращения: июня 18, 2025, https://learn.microsoft.com/en-us/windows/win32/fwp/windows-filtering-platform-architecture-overview
6. Base Filtering Engine (BFE) Windows Service Startup Type, Default Configuration, and Information - Smart PC Utilities, дата последнего обращения: июня 18, 2025, https://www.smartpcutilities.com/windows-service.html?name=BFE
7. Access control (Windows Filtering Platform) - Win32 apps | Microsoft Learn, дата последнего обращения: июня 18, 2025,

https://learn.microsoft.com/en-us/windows/win32/fwp/access-control
8. Guided tour inside WinDefender's network inspection driver - Quarkslab's blog, дата последнего обращения: июня 18, 2025, https://blog.quarkslab.com/guided-tour-inside-windefenders-network-inspection-driver.html
9. Introduction to Windows Filtering Platform (WFP) Callout Drivers - Learn Microsoft, дата последнего обращения: июня 18, 2025, https://learn.microsoft.com/en-us/windows-hardware/drivers/network/introduction-to-windows-filtering-platform-callout-drivers
10. Introduction to Windows Filtering Platform (WFP) Callout Drivers - Learn Microsoft, дата последнего обращения: июня 18, 2025, https://learn.microsoft.com/lt-lt/windows-hardware/drivers/network/introduction-to-windows-filtering-platform-callout-drivers
11. nf-fwpmu-fwpmtransactionbegin0.md - MicrosoftDocs/sdk-api - GitHub, дата последнего обращения: июня 18, 2025, https://github.com/MicrosoftDocs/sdk-api/blob/docs/sdk-api-src/content/fwpmu/nf-fwpmu-fwpmtransactionbegin0.md
12. FwpmEngineOpen0 function (fwpmu.h) - Win32 apps | Microsoft Learn, дата последнего обращения: июня 18, 2025, https://learn.microsoft.com/en-us/windows/win32/api/fwpmu/nf-fwpmu-fwpmengineopen0
13. FwpmTransactionCommit0 function (fwpmu.h) - Win32 apps | Microsoft Learn, дата последнего обращения: июня 18, 2025, https://learn.microsoft.com/en-us/windows/win32/api/fwpmu/nf-fwpmu-fwpmtransactioncommit0
14. FwpmTransactionBegin0 function (fwpmu.h) - Win32 apps | Microsoft Learn, дата последнего обращения: июня 18, 2025, https://learn.microsoft.com/en-us/windows/win32/api/fwpmu/nf-fwpmu-fwpmtransactionbegin0
15. c++ - Ошибка E0513 значение типа "const wchar_t *" нельзя присвоить сущности типа "wchar_t *" - Stack Overflow на русском, дата последнего обращения: июня 18, 2025, https://ru.stackoverflow.com/questions/943778/%D0%9E%D1%88%D0%B8%D0%B1%D0%BA%D0%B0-e0513-%D0%B7%D0%BD%D0%B0%D1%87%D0%B5%D0%BD%D0%B8%D0%B5-%D1%82%D0%B8%D0%BF%D0%B0-const-wchar-t-%D0%BD%D0%B5%D0%BB%D1%8C%D0%B7%D1%8F-%D0%BF%D1%80%D0%B8%D1%81%D0%B2%D0%BE%D0%B8%D1%82%D1%8C-%D1%81%D1%83%D1%89%D0%BD%D0%BE%D1%81%D1%82%D0%B8-%D1%82%D0%B8%D0%BF%D0%B0-wch
16. FwpmSubLayerAdd0 function (fwpmu.h) - Win32 apps | Microsoft Learn, дата последнего обращения: июня 18, 2025, https://learn.microsoft.com/en-us/windows/win32/api/fwpmu/nf-fwpmu-fwpmsublayeradd0
17. WFP provider and Sublayer creation - Microsoft Q&A, дата последнего обращения: июня 18, 2025,

https://learn.microsoft.com/en-us/answers/questions/782700/wfp-provider-and-sublayer-creation
18. FwpmFilterAdd0 function (fwpmu.h) - Win32 apps | Microsoft Learn, дата последнего обращения: июня 18, 2025, https://learn.microsoft.com/en-us/windows/win32/api/fwpmu/nf-fwpmu-fwpmfilteradd0
19. FwpmFilterAdd0 failed (-2144206813) - c++ - Stack Overflow, дата последнего обращения: июня 18, 2025, https://stackoverflow.com/questions/58569513/fwpmfilteradd0-failed-2144206813
20. README.md - wireguard-nt - WireGuard implementation for NT kernel, дата последнего обращения: июня 18, 2025, https://git.zx2c4.com/wireguard-nt/tree/README.md?id=30a2817d913460ed8a23388d3da485cf9347afa3
21. WireGuard/wireguard-nt: This repo is a mirror only. Official repository is at https://git.zx2c4.com/wireguard-nt - GitHub, дата последнего обращения: июня 18, 2025, https://github.com/WireGuard/wireguard-nt
22. example.c « example - wireguard-nt - WireGuard implementation for ..., дата последнего обращения: июня 18, 2025, https://git.zx2c4.com/wireguard-nt/tree/example/example.c
23. Process.GetProcesses Method (System.Diagnostics) | Microsoft Learn, дата последнего обращения: июня 18, 2025, https://learn.microsoft.com/en-us/dotnet/api/system.diagnostics.process.getprocesses?view=net-9.0
24. Process.GetCurrentProcess Method (System.Diagnostics) | Microsoft Learn, дата последнего обращения: июня 18, 2025, https://learn.microsoft.com/en-us/dotnet/api/system.diagnostics.process.getcurrentprocess?view=net-9.0
25. Process.GetProcessesByName Method (System.Diagnostics) | Microsoft Learn, дата последнего обращения: июня 18, 2025, https://learn.microsoft.com/en-us/dotnet/api/system.diagnostics.process.getprocessesbyname?view=net-9.0
26. ProcessModule.FileName Property (System.Diagnostics) | Microsoft ..., дата последнего обращения: июня 18, 2025, https://learn.microsoft.com/en-us/dotnet/api/system.diagnostics.processmodule.filename?view=net-9.0
27. Get All Modules Used By Process In C# - C# Corner, дата последнего обращения: июня 18, 2025, https://www.c-sharpcorner.com/UploadFile/87b416/get-all-modules-used-by-process3/
28. win32/desktop-src/FWP/windows-filtering-platform-start-page.md at docs - GitHub, дата последнего обращения: июня 18, 2025, https://github.com/MicrosoftDocs/win32/blob/docs/desktop-src/FWP/windows-filtering-platform-start-page.md
29. win32/desktop-src/FWP/about-windows-filtering-platform.md at docs - GitHub,

дата последнего обращения: июня 18, 2025, https://github.com/MicrosoftDocs/win32/blob/docs/desktop-src/FWP/about-windows-filtering-platform.md
30. Stream Inspection - Windows drivers | Microsoft Learn, дата последнего обращения: июня 18, 2025, https://learn.microsoft.com/en-us/windows-hardware/drivers/network/stream-inspection
31. Using named pipes for interprocess communication in C# | Michael John Peña, дата последнего обращения: июня 18, 2025, https://michaeljohnpena.com/blog/namedpipes/
32. NamedPipeClientStream Class (System.IO.Pipes) | Microsoft Learn, дата последнего обращения: июня 18, 2025, https://learn.microsoft.com/en-us/dotnet/api/system.io.pipes.namedpipeclientstream?view=net-9.0
33. How to: Use Named Pipes for Network Interprocess Communication ..., дата последнего обращения: июня 18, 2025, https://learn.microsoft.com/en-us/dotnet/standard/io/how-to-use-named-pipes-for-network-interprocess-communication
34. Create Windows named pipe in C++ - Stack Overflow, дата последнего обращения: июня 18, 2025, https://stackoverflow.com/questions/26561604/create-windows-named-pipe-in-c
35. c++ - Marshal an array of strings from C# to C code using p/invoke ..., дата последнего обращения: июня 18, 2025, https://stackoverflow.com/questions/13317931/marshal-an-array-of-strings-from-c-sharp-to-c-code-using-p-invoke
36. P/Invoke Tutorial: Passing strings (Part 2) - CodeProject, дата последнего обращения: июня 18, 2025, https://www.codeproject.com/Articles/401922/P-Invoke-Tutorial-Passing-strings-Part
37. How To P/Invoke char* [] in C# - Stack Overflow, дата последнего обращения: июня 18, 2025, https://stackoverflow.com/questions/25137788/how-to-p-invoke-char-in-c-sharp
38. WFP-Traffic-Redirection-Driver/sys/inspect.h at master - GitHub, дата последнего обращения: июня 18, 2025, https://github.com/BOT-Man-JL/WFP-Traffic-Redirection-Driver/blob/master/sys/inspect.h
39. Marshaling Structs - P/INVOKE WITH C# AND C++ TUTORIAL #3 - YouTube, дата последнего обращения: июня 18, 2025, https://www.youtube.com/watch?v=dyVHBemrYdI
40. FwpsAcquireClassifyHandle0 function (fwpsk.h) - Windows drivers ..., дата последнего обращения: июня 18, 2025, https://learn.microsoft.com/en-us/windows-hardware/drivers/ddi/fwpsk/nf-fwpsk-fwpsacquireclassifyhandle0
41. FwpsAcquireWritableLayerDataPointer0 function (fwpsk.h ..., дата последнего

обращения: июня 18, 2025, https://learn.microsoft.com/en-us/windows-hardware/drivers/ddi/fwpsk/nf-fwpsk-fwpsacquirewritablelayerdatapointer0
42. _FWPS_CONNECT_REQUEST0 (fwpsk.h) - Windows drivers | Microsoft Learn, дата последнего обращения: июня 18, 2025, https://learn.microsoft.com/en-us/windows-hardware/drivers/ddi/fwpsk/ns-fwpsk-_fwps_connect_request0
43. Using Bind or Connect Redirection - Windows drivers - Learn Microsoft, дата последнего обращения: июня 18, 2025, https://learn.microsoft.com/en-us/windows-hardware/drivers/network/using-bind-or-connect-redirection
44. test1213145/ms-EDRSilencer: A tool uses Windows ... - GitHub, дата последнего обращения: июня 18, 2025, https://github.com/test1213145/ms-EDRSilencer
45. FwpsApplyModifiedLayerData0 function (fwpsk.h) - Windows drivers ..., дата последнего обращения: июня 18, 2025, https://learn.microsoft.com/en-us/windows-hardware/drivers/ddi/fwpsk/nf-fwpsk-fwpsapplymodifiedlayerdata0
46. Sample Kernel-Mode Drivers - Windows drivers | Microsoft Learn, дата последнего обращения: июня 18, 2025, https://learn.microsoft.com/en-us/windows-hardware/drivers/kernel/sample-kernel-mode-drivers
47. Developing Kernel Drivers with Modern C++ - Pavel Yosifovich - YouTube, дата последнего обращения: июня 18, 2025, https://www.youtube.com/watch?v=AsSMKL5vaXw
48. FwpsReleaseClassifyHandle0 function (fwpsk.h) - Windows drivers ..., дата последнего обращения: июня 18, 2025, https://learn.microsoft.com/en-us/windows-hardware/drivers/ddi/fwpsk/nf-fwpsk-fwpsreleaseclassifyhandle0
49. Modern C++ in the Windows Kernel - Jeremy Hurren, дата последнего обращения: июня 18, 2025, https://lordjeb.com/2013/09/19/modern-c-in-the-windows-kernel/
50. Automated Malware Analysis Report for anyconnect-win-4.9.06037-core-vpn-predeploy-k9.msi - Generated by Joe Sandbox, дата последнего обращения: июня 18, 2025, https://www.joesandbox.com/analysis/382637/0/html
51. win32/desktop-src/WinSock/sio-query-wfp-connection-redirect-context.md at docs - GitHub, дата последнего обращения: июня 18, 2025, https://github.com/MicrosoftDocs/win32/blob/docs/desktop-src/WinSock/sio-query-wfp-connection-redirect-context.md
52. WFP ALE_CONNECT_REDIRECT layer block filter doesn't work - Stack Overflow, дата последнего обращения: июня 18, 2025, https://stackoverflow.com/questions/34986867/wfp-ale-connect-redirect-layer-block-filter-doesnt-work
53. Windows-driver-samples/general/ioctl/wdm/sys/sioctl.c at main - GitHub, дата последнего обращения: июня 18, 2025,

https://github.com/Microsoft/Windows-driver-samples/blob/master/general/ioctl/wdm/sys/sioctl.c
54. Userland/Kernel communication – DeviceIoControl method « Eric ..., дата последнего обращения: июня 18, 2025, https://ericasselin.com/userlandkernel-communication-deviceiocontrol-method
55. Windows-driver-samples/general/ioctl/wdm/exe/testapp.c at main - GitHub, дата последнего обращения: июня 18, 2025, https://github.com/Microsoft/Windows-driver-samples/blob/master/general/ioctl/wdm/exe/testapp.c
56. CreateFileA function (fileapi.h) - Win32 apps | Microsoft Learn, дата последнего обращения: июня 18, 2025, https://learn.microsoft.com/en-us/windows/win32/api/fileapi/nf-fileapi-createfilea
57. Ioctl.cpp - microsoft/Windows-driver-samples - GitHub, дата последнего обращения: июня 18, 2025, https://github.com/microsoft/Windows-driver-samples/blob/main/pos/drivers/barcodescanner/Ioctl.cpp
58. IRP_MJ_DEVICE_CONTROL - Windows drivers | Microsoft Learn, дата последнего обращения: июня 18, 2025, https://learn.microsoft.com/en-us/windows-hardware/drivers/kernel/irp-mj-device-control
59. Sending IRP_MJ_SYSTEM_CONTROL request from user-mode - Stack Overflow, дата последнего обращения: июня 18, 2025, https://stackoverflow.com/questions/35634047/sending-irp-mj-system-control-request-from-user-mode
60. Why ProbeForWrite/ProbeForRead - NTDEV - OSR Developer ..., дата последнего обращения: июня 18, 2025, https://community.osr.com/t/why-probeforwrite-probeforread/52014
61. ProbeForWrite function (wdm.h) - Windows drivers - Learn Microsoft, дата последнего обращения: июня 18, 2025, https://learn.microsoft.com/en-us/windows-hardware/drivers/ddi/wdm/nf-wdm-probeforwrite
62. Driver code signing requirements - Windows drivers | Microsoft Learn, дата последнего обращения: июня 18, 2025, https://learn.microsoft.com/en-us/windows-hardware/drivers/dashboard/code-signing-reqs
63. EV Code Signing Certificates - Sign Windows Drivers & Packages, дата последнего обращения: июня 18, 2025, https://signmycode.com/ev-code-signing
64. EV Code Signing - Overview - GlobalSign Support, дата последнего обращения: июня 18, 2025, https://support.globalsign.com/code-signing/ev-code-signing-overview
65. Attestation sign Windows drivers - Learn Microsoft, дата последнего обращения: июня 18, 2025, https://learn.microsoft.com/en-us/windows-hardware/drivers/dashboard/code-signing-attestation

66. Microsoft Windows Driver Signing With Code Signing Certificates, дата последнего обращения: июня 18, 2025, https://codesigningstore.com/microsoft-windows-driver-signing-with-code-signing-certificates
67. Attestation sign Windows drivers - Windows drivers | Microsoft Learn, дата последнего обращения: июня 18, 2025, https://learn.microsoft.com/th-th/windows-hardware/drivers/dashboard/code-signing-attestation
68. EchoInst: An Example Wix Install of the WDK Echo Driver - CodeProject, дата последнего обращения: июня 18, 2025, https://www.codeproject.com/Articles/573183/EchoInst-An-Example-Wix-Install-of-the-WDK-Echo-Dr
69. How To: Install a Windows service | Docs, дата последнего обращения: июня 18, 2025, https://docs.firegiant.com/wix3/howtos/general/install_windows_service/
70. Drivers Installation With WiX | Apriorit, дата последнего обращения: июня 18, 2025, https://www.apriorit.com/dev-blog/164-driver-installation-with-wix
71. PnPUtil Examples - Windows drivers - Learn Microsoft, дата последнего обращения: июня 18, 2025, https://learn.microsoft.com/en-us/windows-hardware/drivers/devtest/pnputil-examples
72. Driver installation and update - NSIS Wiki - SourceForge, дата последнего обращения: июня 18, 2025, https://nsis.sourceforge.io/Driver_installation_and_update
73. 64 bit - Installing drivers from NSIS installer in x64 system - Stack ..., дата последнего обращения: июня 18, 2025, https://stackoverflow.com/questions/2464843/installing-drivers-from-nsis-installer-in-x64-system
74. PnPUtil - Windows drivers | Microsoft Learn, дата последнего обращения: июня 18, 2025, https://learn.microsoft.com/en-us/windows-hardware/drivers/devtest/pnputil
75. Best practices - NSIS Wiki, дата последнего обращения: июня 18, 2025, https://nsis.sourceforge.io/Best_practices
76. Windows Filtering Platform Sample - Code Samples | Microsoft Learn, дата последнего обращения: июня 18, 2025, https://learn.microsoft.com/en-us/samples/microsoft/windows-driver-samples/windows-filtering-platform-sample/
77. msdn-code-gallery-microsoft/Official Windows Driver Kit Sample ..., дата последнего обращения: июня 18, 2025, https://github.com/microsoftarchive/msdn-code-gallery-microsoft/blob/master/Official%20Windows%20Driver%20Kit%20Sample/Windows%20Driver%20Kit%20(WDK)%208.1%20Samples/%5BC++%5D-windows-driver-kit-81-cpp/WDK%208.1%20C++%20Samples/Windows%20Filtering%20Platform%20Sample/C++/HCK/WFPLogo_WFPSampler.Info
78. FwpmFilterDeleteByKey0 - Windows drivers | Microsoft Learn, дата последнего обращения: июня 18, 2025,

https://learn.microsoft.com/en-us/windows-hardware/drivers/ddi/fwpmk/nf-fwpmk-fwpmfilterdeletebykey0
79. December 2022 - Pavel Yosifovich, дата последнего обращения: июня 18, 2025, https://scorpiosoftware.net/2022/12/
80. Automated Malware Analysis Report for BfeToolWin8.exe - Generated by Joe Sandbox, дата последнего обращения: июня 18, 2025, https://www.joesandbox.com/analysis/691995/0/html
81. Completely Uninstall Dev-C++ on Windows (Quick & Easy Guide) - YouTube, дата последнего обращения: июня 18, 2025, https://www.youtube.com/watch?v=68T1h_q3He0
82. Best Practices (Windows Filtering Platform) - Win32 apps | Microsoft Learn, дата последнего обращения: июня 18, 2025, https://learn.microsoft.com/en-us/windows/win32/fwp/best-practices
83. FwpmSubLayerDeleteByKey0 function (fwpmu.h) - Win32 apps | Microsoft Learn, дата последнего обращения: июня 18, 2025, https://learn.microsoft.com/en-us/windows/win32/api/fwpmu/nf-fwpmu-fwpmsublayerdeletebykey0
84. FwpmFilterDeleteByKey0 function (fwpmu.h) - Win32 apps | Microsoft Learn, дата последнего обращения: июня 18, 2025, https://learn.microsoft.com/en-us/windows/win32/api/fwpmu/nf-fwpmu-fwpmfilterdeletebykey0