SOLAR

SOLAR APPSCREENER 3.15

WHITE PAPER

МОСКВА, 2025

W: RT-SOLAR.RU

СОДЕРЖАНИЕ

1. ПРОБЛЕМАТИКА

1.1. НЕОБХОДИМОСТЬ АНАЛИЗА КОДА ПРИЛОЖЕНИЙ

Развитие удаленных сервисов

Проблемы, связанные с уязвимостями кода приложений и наличием недекларированных возможностей (НДВ), были известны давно, но длительное время не попадали в фокус внимания служб информационной безопасности (ИБ). Приложения чаще всего работали в локальной сети компании и не были доступны внешним пользователям, поэтому офицеры безопасности занимались более очевидными проблемами: защитой периметра, управлением доступом, утечками конфиденциальной информации, защитой конечных точек от вредоносного ПО и т. д. Но по мере развития удаленно доступных сервисов ситуация изменилась:

- стали популярнее онлайн-сервисы и приложения, которыми может воспользоваться любой потенциальный клиент компании;
- существенно возросла доля бизнес-систем, круглосуточно доступных сотрудникам за пределами периметра компании;
- внешний периметр стал размытым и единственным рубежом защиты оказалось само ПО;
- с ростом квалификации и опыта киберпреступников значительно выросли риски эксплуатации уязвимостей и НДВ в коде приложений.

Это привело к тому, что сегодня от наличия уязвимостей и НДВ в коде приложений напрямую зависит работоспособность ИТ-систем, конфиденциальность важной информации и сохранность денег организации и ее клиентов. Это актуально как для онлайн-сервисов и приложений, которые ориентированы на массовую аудиторию, требуют непрерывности работы и позволяют совершать финансовые операции в реальном времени, так и для критических систем, работоспособность которых может быть нарушена внутренним нарушителем (АСУ ТП, внутренние системы).

Ускорение процесса разработки

Проблему уязвимостей в приложениях усугубляет переход на более быстрые методы разработки и публикации кода, связанный с повсеместным ужесточением конкуренции. К таким методам относятся непрерывная интеграция (CI, Continuous Integration), подразумевающая слияние рабочих копий кода в основную ветвь разработки несколько раз в день, и непрерывная доставка (CD, Continuous Delivery), нацеленная на сборку, тестирование и поставку ПО в максимально быстром режиме. В таких условиях на вдумчивый анализ кода просто нет времени – главное вовремя сдать свою часть проекта. В результате приложение может содержать НДВ для более быстрой отладки или экстренного устранения ошибок (например, для сбора логов или данных пользователей), и многочисленные уязвимости, появившиеся из-за невнимательности или спешки.

Унаследованное ПО

Еще одна проблема – широкое использование унаследованных информационных систем. Часто это устаревшие системы с неактуальной документацией, которые сильно изменились со времени создания и чьих разработчиков давно нет на рынке или в организации. Такие системы невозможно обновить для устранения уязвимостей, но зачастую они критически важны для деятельности организации, поэтому от них нельзя отказаться или быстро поменять на более совершенный аналог.

Исследование унаследованных систем на уязвимости и НДВ часто осложняется отсутствием исходных кодов из-за частой смены подрядчиков и разработчиков. Вместе с глубокими и общедоступными исследованиями известных уязвимостей это существенно повышает риски

проникновения в ИТ-инфраструктуру организации, вмешательства в работу унаследованной системы вплоть до ее остановки и похищения критических данных (например, финансовой отчетности и персональных данных сотрудников, и контрагентов).

Уязвимости – основа 90% успешных кибератак

По данным отчета Solar JSOC[1], 90% веб-приложений содержат уязвимости, позволяющие проводить атаки, а 88% приложений включают как минимум одну критическую уязвимость, поэтому наличие уязвимостей в коде может представлять серьезную угрозу и привести к реализации кибератак на компанию.

Активность злоумышленников также не стоит на месте – например, за последнее время заметно выросло число атак, связанных с внедрением вредоносного кода в open-source-компоненты, которые используются в как минимум 70% современных приложений[2]. Злоумышленник заражает open-source-библиотеки, которые могут использоваться для разработки ПО жертвы и под видом легитимного софта попадают в периметр компании.

Такие атаки могут привести к серьезным последствиям – от юридических проблем и утечек конфиденциальных данных до больших денежных и репутационных потерь.

1.2. КАК СНИЗИТЬ РИСКИ ЭКСПЛУАТАЦИИ УЯЗВИМОСТЕЙ КОДА

Для предотвращения или минимизации инцидентов, связанных с эксплуатацией уязвимостей и НДВ в коде приложений, необходимо:

- регулярно анализировать безопасность кода приложений, разрабатываемых внутри компании или подрядчиками;
- максимально быстро устранять обнаруженные уязвимости, в том числе компенсационными мерами, например, оперативно перенастраивая межсетевой экран уровня приложений (Web Application Firewall, WAF);
- контролировать корректировку кода разработчиками с целью устранения уязвимостей и НДВ;
- следить за безопасностью сторонних компонентов, которые вы используете при разработке кода.

Чтобы своевременно обнаруживать уязвимости и НДВ в коде приложений до их официального выпуска в масштабах средних и больших организаций, необходимо внедрить цикл безопасной разработки ПО (Secure SDLC, Secure Software Development Life Cycle).

Подобный подход нашел понимание и поддержку на государственном и отраслевом уровнях:

- В соответствии с Методическими рекомендациями по созданию ведомственных и корпоративных центров государственной системы обнаружения, предупреждения и ликвидации последствий компьютерных атак на информационные ресурсы Российской Федерации (ГосСОПКА) № 149/2/7-200 от 27.12.2017 должно осуществляться выявление известных уязвимостей ПО информационных ресурсов путем анализа состава установленного ПО и обновлений безопасности с применением автоматизированных средств анализа защищенности (системное сканирование, исследование с использованием привилегированных учетных записей и (или) программных агентов), а также других средств защиты информации.

[1] Отчет Solar JSOC «Ключевые уязвимости информационных систем российских компаний 2021–2022 гг.».
[2] Linux Foundation. A Summary of Census II: Open Source Software Application Libraries the World Depends On.

- С 1 июля 2017 года вступил в силу ГОСТ Р 56939-2016 «Защита информации. Разработка безопасного программного обеспечения. Общие требования», утвержденный ФСТЭК России. Новый стандарт обязывает контролировать отсутствие угроз НДВ в ПО («функциональных возможностей средств вычислительной техники, не описанных или не соответствующих описанным в документации, при использовании которых возможно нарушение конфиденциальности, доступности или целостности обрабатываемой информации») и обнаруживать недостатки в защищенности.
- В соответствии с требованиями приказов ФСТЭК № 17, 21 и 31 необходимо анализировать возможные уязвимости используемых информационных систем, в том числе применяемых средств защиты информации, применять меры по их устранению, и подтверждать, что в информационной системе отсутствуют уязвимости, содержащиеся в банке данных угроз безопасности информации ФСТЭК России, а также в иных источниках, или их использование (эксплуатация) нарушителем невозможно.
- Согласно Стандарту Банка России в области информационной безопасности (СТО БР ИББС) в соответствии с требованиями к анализу функционирования системы обеспечения ИБ процедуры анализа функционирования системы обеспечения ИБ, использующие в том числе данные об уязвимостях ИБ должны быть определены и должны выполняться, регистрироваться и контролироваться.

1.3. ТЕХНОЛОГИИ АНАЛИЗА БЕЗОПАСНОСТИ ПРИЛОЖЕНИЙ

Специалисты международной исследовательской и консалтинговой компании Gartner, специализирующейся на ИТ-рынке, считают анализ приложений на уязвимости и НДВ одной из главных технологий, обеспечивающих Secure SDLC и безопасность приложений.

Gartner выделяет 5 основных метода анализа кода:

- **Статический анализ кода** (SAST, Static Application Security Testing) – анализ исходного кода приложения без его реального исполнения (метод «белого ящика»). Максимально подходит для интеграции процесса тестирования кода в разработку приложения для организации Secure SDLC с применением методов CI и CD.
- **Динамический анализ кода** (DAST, Dynamic Application Security Testing) – анализ исполняемых программ на реальном или виртуальном процессоре (метод «черного ящика»). Подразумевает анализ уже выпущенных и работающих приложений и широко применяется в командах, практикующих каскадный метод разработки ПО, а также теми, кто не может получить исходный код приложения.
- **Интерактивный анализ кода** (IAST, Interactive Application Security Testing) – улучшенная версия DAST, отличающаяся от него помещением агента в анализируемое приложение. Это отличие позволяет отслеживать выполнение кода, к которому привел внешний вызов, а также запускать части приложения изнутри. Например, обычно DAST не может найти запись критических данных в логе, так как такая уязвимость никак не скажется на поведении интерфейса, а IAST – может, так как он отслеживает перемещение данных внутри приложения. Можно сказать, что IAST — это метод «серого ящика». Наблюдение ведется изнутри приложения, но без анализа исходного кода, то есть часть уязвимостей и НДВ может быть не обнаружена.
- **Анализ кода мобильных приложений** (mAST, Mobile Application Security Testing) – анализ кода мобильных приложений с учетом специфики мобильных платформ, прежде всего Google Android и Apple iOS.
- **Анализ состава программного обеспечения** (SCA, Software Composition Analysis) – анализ сторонних компонентов, используемых в кодовой базе приложения. SCA

позволяет гарантировать, что компоненты, которые использовались в разрабатываемых приложениях, соответствуют основным стандартам безопасности и не представляют риска для бизнеса.

Компания «Солар» разработала собственное комплексное решение Solar appScreener для проверки приложений на безопасность, которое содержит модули SAST, DAST и OSA. Далее расскажем подробнее о каждом модуле.

2. КРАТКОЕ ОПИСАНИЕ

2.1. НАЗНАЧЕНИЕ

Solar appScreener – комплексное решение для контроля безопасности ПО. Его возможности позволяют эффективно выявлять уязвимости и НДВ с помощью различных методов анализа (SAST, DAST, SCA, SCS) и интегрироваться с другими системами для встраивания в цикл безопасной разработки.

Solar appScreener поддерживает **статический анализ приложений**, написанных на 36 языках программирования или скомпилированных в одном из 10 различных форматов исполняемых файлов, в том числе для Google Android, Apple iOS и Apple macOS. Одна из отличительных особенностей статического анализа Solar appScreener – возможность анализа не только исходного кода, но и исполняемых файлов (бинарного кода).

Проверка безопасности мобильных приложений может осуществляться простым копированием в меню анализатора ссылки на приложение в Google Play или App Store, что является полноценным mAST-анализом по методологии Gartner.

Помимо модуля SAST, Solar appScreener включает модуль **динамического анализа**, благодаря которому можно анализировать веб-приложения на наличие уязвимостей, имитируя вредоносные внешние атаки и используя распространенные уязвимости. Также решение Solar appScreener позволяет коррелировать результаты статического и динамического анализа, благодаря чему можно динамически подтверждать уязвимости, найденные с помощью статического метода.

Модуль анализа сторонних компонентов OSA в Solar appScreener сочетает в себе:

- **Анализ SCA** позволяет проводить анализ состава программного обеспечения, что дает возможность проверять как прямые, так и транзитивные внешние компоненты (заимствованный код) на наличие в них уязвимостей и НДВ, получать рекомендации по исправлению (замене) и обнаруживать устаревшие компоненты.
- **SCS-модуль** позволяет проводить анализ supply chain и оценку лицензионных рисков, а также формирует оценку уровня доверия к используемым внешним компонентам на основе 8 конкретных метрик.
- **Анализ лицензионных рисков** отслеживает лицензионные политики при использовании open source-компонентов. Позволяет подмечать особенности лицензирования сторонних компонентов, а также выдает критичность использования той или иной библиотеки.
- **Комбинированный анализ SAST/OSA** позволяет подтверждать использование уязвимых функций сторонних компонентов в проекте, снижать количество ложных срабатываний и показывает, откуда именно она появилась в коде.

2.2. ПОДДЕРЖИВАЕМЫЕ ЯЗЫКИ ПРОГРАММИРОВАНИЯ И ФОРМАТЫ

Статический анализ

- Поддерживаемые языки программирования: ABAP, Apex, ASP.NET, COBOL, C#, C/C++, Objective-C, Dart, Delphi, Go, Groovy, HTML5, Java, Java for Android, JavaScript, JSP, LotusScript, Kotlin, Pascal, Perl, PHP, PL/SQL, T/SQL, Python, Ruby, Rust, Scala, Solidity, Swift, TypeScript, VBA, VB.NET, VBScript, Visual Basic 6.0, Vyper, 1C.
  - Возможна загрузка проектов в архивах форматов 7Z, ZIP, EAR/AAR, RAR, TAR.BZ2, TAR.GZ, TAR, CPIO.
- Поддерживаемые форматы исполняемых файлов для проведения статического анализа:
  - Java/Scala: JAR/WAR/EAR/AAR;

- C/C++: DLL/EXE;
- Android: APK;
- Apple iOS: IPA;
- Apple macOS: APP.

Динамический анализ

Динамический анализ проводится по URL веб-приложения.

Анализ сторонних компонентов

- Исходный код, на языках программирования из списка, содержащий манифест/манифесты из списка:
  - Python: requirements.txt, Pipfile, setup.py, poetry.lock, requirements.pip,
  - Java: pom.xml, build.gradle, gradle.kts, gradle.lockfile
  - Kotlin: pom.xml, build.gradle, gradle.kts, gradle.lockfile,
  - Groovy: pom.xml, build.gradle, gradle.kts, gradle.lockfile
  - Scala: pom.xml, build.gradle, gradle.kts, gradle.lockfile
  - JavaScript: package.json
  - TypeScript: package.json
  - Vue: package.json
  - PHP: composer.json
  - Go: go.mod, go.sum
  - C#: .(sln|csproj|vbproj)
  - Swift: Podfile, Package.swift
  - Rust: Cargo.(lock|toml)
  - Erlang: rebar.config
  - Ruby: Gemfile(.lock), .gemspec
  - C++: conanfile.txt, conan.lock, CMakeLists.txt
  - Dart: pubspec.lock, pubspec.yaml
- Поддерживаемые языки программирования для проведения комбинированного анализа SAST/OSA: JavaScript/TypeScript, Python, C# и Java
- Поддерживаемые форматы файлов для проведения анализа состава ПО:
  - Архив с исходным кодом в формате ZIP
  - Ссылка на репозиторий с исходным кодом (Gitlab, Github, Bitbucket);
  - SBOM-файл проекта (собранный с помощью CycloneDX).
- Поддерживаемые форматы файлов для проведения анализа цепочек поставок, лицензионных рисков и построения дерева транзитивных зависимостей:

- SBOM-файл проекта (собранный с помощью CycloneDX).

2.3. ОБЛАСТИ ПРИМЕНЕНИЯ

Solar appScreener незаменим, если компании нужно:

- предоставлять онлайн-сервисы внешним пользователям: интернет-банк, личный кабинет пользователя, онлайн-продажа товаров и услуг, сервисы мобильной коммерции и т. д.;
- проверять безопасность приложений на наличие уязвимостей и НДВ, оставленных разработчиками, без возможности доступа к исходным кодам;
- быть уверенными в безопасности кода, в том числе заимствованных компонентов, и понимать, где в этой библиотеке найдены уязвимости;
- выполнять требования стандартов СТО БР ИББС, ФСТЭК России, PCI DSS и ГОСТ Р 56939-2016 в части анализа кода приложения;
- усилить авторитет и влияние службы ИБ на внешних или внутренних разработчиков;
- корректно и в кратчайшие сроки настроить системы защиты и мониторинга онлайн-сервисов.

2.4. ИНТЕРФЕЙС

Графический интерфейс Solar appScreener в первую очередь рассчитан на службу ИБ, а не на разработчиков. В его основе – облегченная логика взаимодействия с пользователем, не требующая глубоких технических знаний для интерпретации результатов анализа. Интерфейс Solar appScreener отличается простотой и удобством, а сам процесс анализа максимально автоматизирован, что позволяет проверять код приложения в два клика.

В Solar appScreener реализована удобная навигация по проектам и результатам анализа, более наглядная и подробная статистическая информация о проектах и дополнительные фильтры для проектов.

В руководство пользователя включен глоссарий, где собраны специфичные термины, используемые в Solar appScreener, а также общие понятия отрасли ИБ.

В решении доступны русский и английский языки интерфейса. Предпочитаемый язык можно поменять прямо во время работы.

Также с Solar appScreener можно работать через командную строку (CLI).

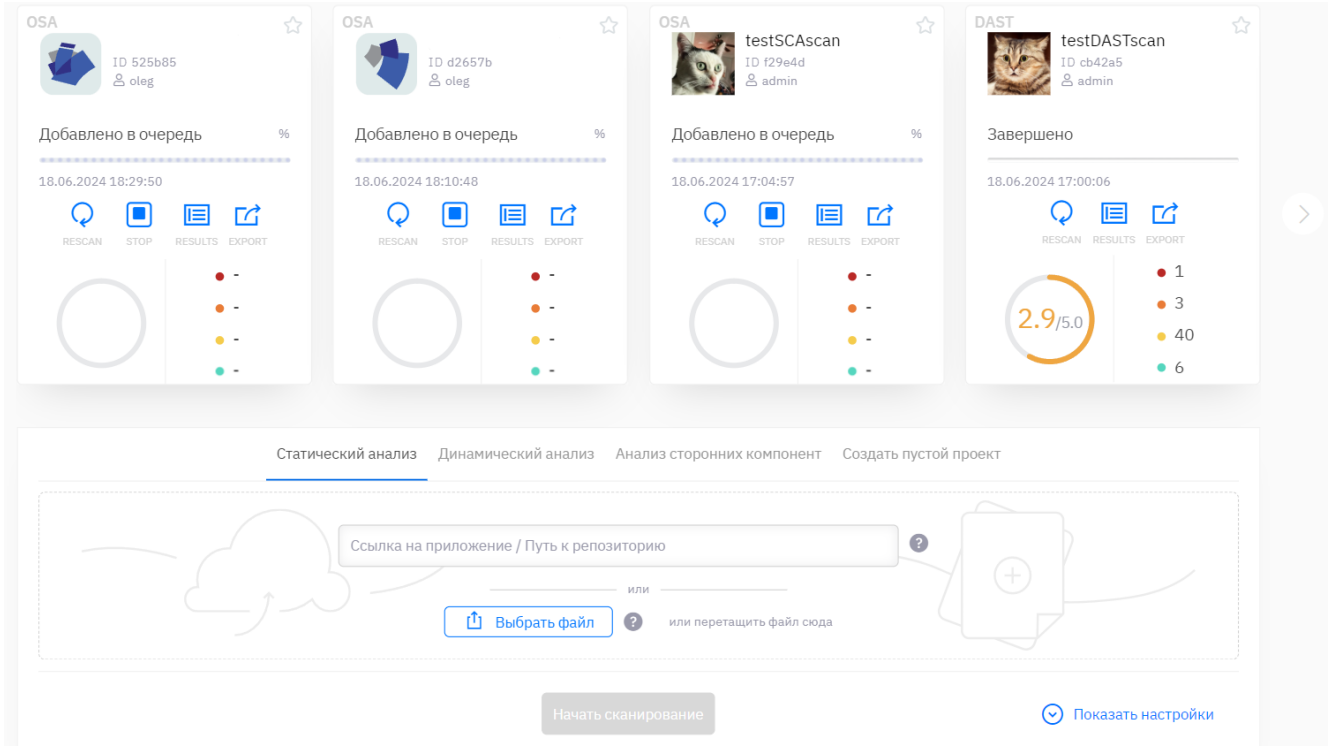


Рисунок 1. Главное окно интерфейса

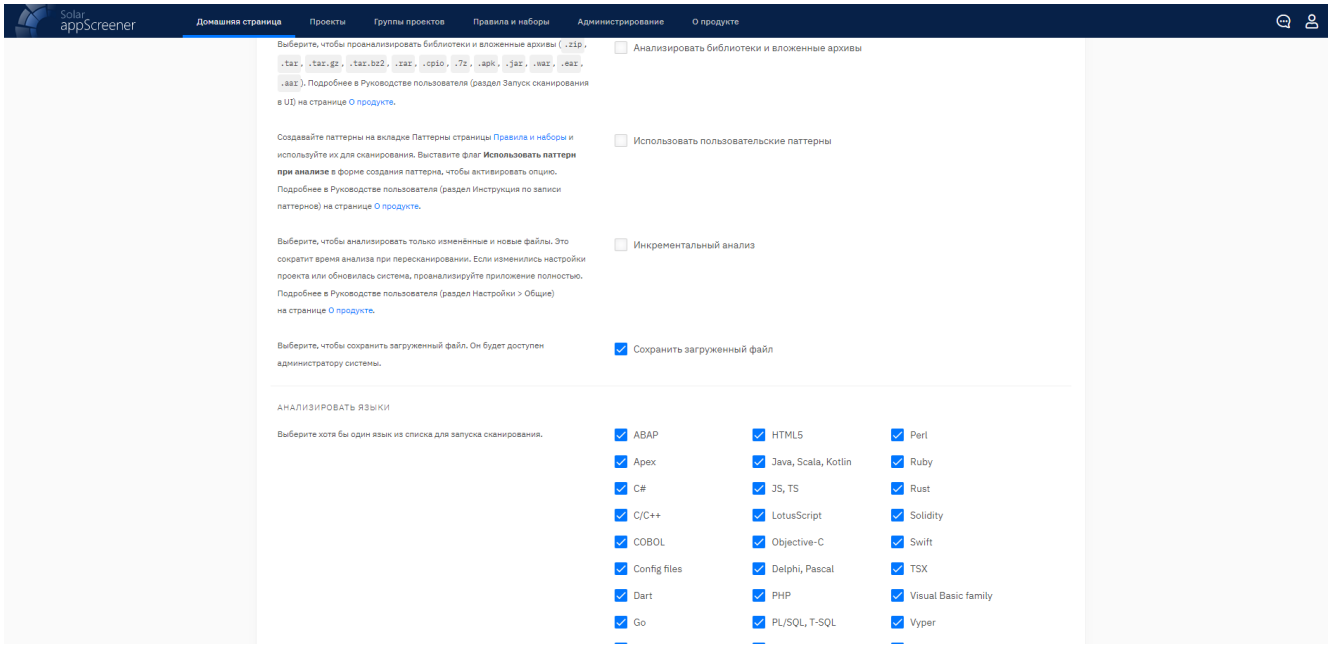


Рисунок 2. Настройки сканирования для статического анализа

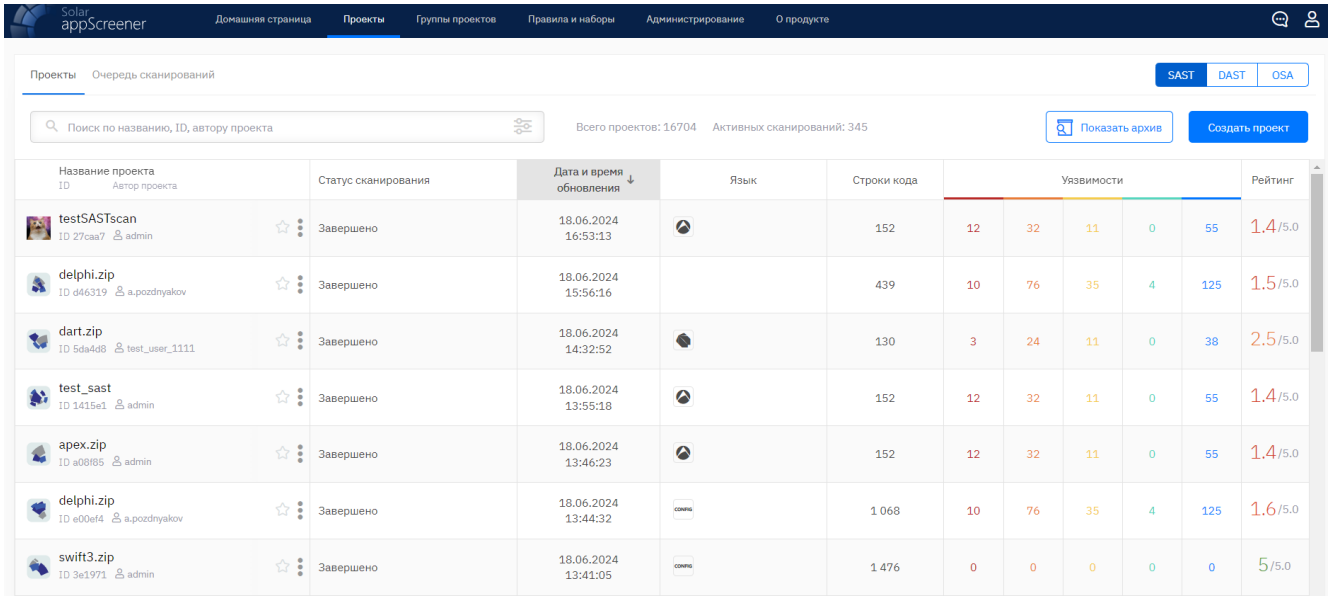


Рисунок 3. Список проектов

2.5. ЛИЦЕНЗИРОВАНИЕ И ПОСТАВКА

Solar appScreener можно развернуть как на собственных вычислительных мощностях организации, так и пользоваться им как сервисом, доступным из облака «Солар» (SaaS). В случае размещения на собственном сервере организации лицензирование осуществляется по количеству пользователей с доступом к системе или по количеству параллельных потоков анализа. В случае SaaS оплачивается количество произведенных проверок кода.

Для небольших вендоров и компаний, использующих заказные разработки, модель SaaS наиболее оптимальна: у них потребность в проверке кода приложений возникает лишь периодически. Таким компаниям достаточно приобрести лицензии на необходимое число проверок кода, через веб-интерфейс загрузить код в облако и дождаться окончания анализа. Передача данных между веб-интерфейсом и облаком осуществляется по защищенному протоколу HTTPS.

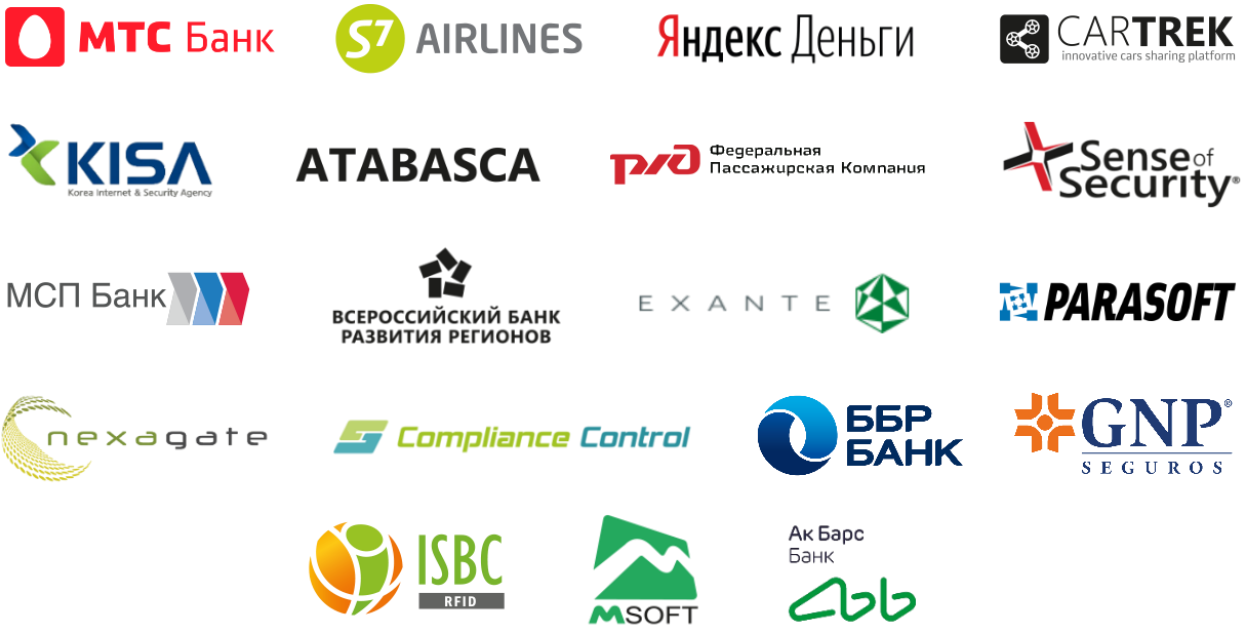
2.6. СООТВЕТСТВИЕ ТРЕБОВАНИЯМ РЕГУЛЯТОРОВ

Solar appScreener разработан в России с применением собственных запатентованных технологий, внесен в Единый реестр отечественного ПО (№ 6119), сертифицирован ФСТЭК России на соответствие «Требованиям по безопасности информации, устанавливающие уровни доверия к средствам технической защиты информации и средствам обеспечения безопасности информационных технологий» (ФСТЭК России, 2020) – по 4 уровню доверия и ТУ (сертификат соответствия № 4007).

Solar appScreener – отличный выбор для компаний, стремящихся выполнять требования различных стандартов безопасности. С его помощью можно сгенерировать отчет, сверстанный в соответствии с классификацией уязвимостей по версии PCI DSS, OWASP Top 10, OWASP Mobile Top 10, БДУ ФСТЭК России, ОУД4, HIPAA, OWASP ASVS, OWASP MASVS или CWE/SANS Top 25, что упрощает обеспечение соответствия требованиям регуляторов.

2.7. КЛИЕНТЫ

Среди заказчиков Solar appScreener:



3. ВОЗМОЖНОСТИ

- **Статический анализ исходного кода**

  Solar appScreener может анализировать исходный код, написанный на 36 языках программирования, включая как популярные (Java, Scala, PHP, C#, Swift, Ruby и др.), так и специализированные (ABAP, 1C, Solidity, PL/SQL и др.) или устаревшие (Delphi, COBOL, Visual Basic 6.0, VBA).

- **Статический анализ исполняемых файлов**

  Технологии декомпиляции и деобфускации бинарного кода приложений позволяют использовать Solar appScreener для анализа исполняемых файлов, в том числе для Google Android, Apple iOS и Apple macOS. Для проверки мобильных приложений достаточно скопировать в анализатор ссылку на соответствующую страницу в Google Play или App Store. Результаты анализа отображаются на основе восстановленного исходного кода.

- **Динамический анализ веб-приложений**

  Solar appScreener может проводить динамическое сканирование веб-приложений. Для этого достаточно указать URL веб-приложения. Модуль DAST выявит уязвимости и НДВ тестируемого проекта.

- **Анализ состава программного обеспечения**

  Технология SCA в Solar appScreener позволяет проводить анализ состава программного обеспечения на наличие в нем уязвимостей и закладок, а также строить дерево транзитивных зависимостей – то есть показывает все зависимости проверяемого компонента и уязвимости в них.

- **Анализ лицензионных рисков стороннего ПО**

  Модуль анализа лицензионных рисков позволяет контролировать лицензионные политики для оценки юридических рисков использования open-source-кода.

- **Анализ цепочки поставок ПО**

  Технология анализа безопасности цепочки поставок (SCS) определяет по 8 метрикам, насколько безопасно использовать библиотеку, а также выдает ее рейтинг на основе этих метрик. Технология SCS в Solar appScreener необходим для обеспечения безопасности на всех этапах пути, по которому ПО попадает в организацию - от момента его создания или покупки обновлений до этапа использования.

- **Выявление уязвимостей**

  Уязвимости выявляются на основе правил поиска после завершения всех процедур анализа и работы собственной технологии для снижения ложных срабатываний Fuzzy Logic Engine.

- **Выявление недекларированных возможностей**

  НДВ выявляются по наличию одной из характерных базовых конструкций: хардкодных учетных записей, скрытой сетевой активности, временных бомб и т. д. Наличие базовых конструкций НДВ может свидетельствовать о присутствии более сложной составной закладки.

- **Проверка унаследованного и заказного ПО методом статического анализа**

Реализованные в Solar appScreener технологии SCA и анализа бинарного кода позволяют проверять на уязвимости и НДВ унаследованные приложения и заказные разработки, в том числе использующие сторонние компоненты (готовые коды из интернета, модули, библиотеки).

- **Корреляция результатов статического и динамического анализа**

  Результаты, полученные по итогам статического и динамического анализа, можно сопоставить между собой, благодаря чему в рамках одного проекта пользователю доступна проверка с полным покрытием кода и указанием уязвимостей, подтвержденных с помощью динамического анализа.

- **Комбинированный анализ SAST/OSA**

  Технология комбинированного анализа SAST/OSA позволяет построить трассу вызовов уязвимых функций в прямых и транзитивных сторонних компонентах, чтобы определить, действительно ли уязвимая функция в библиотеке исполняется в коде или это ложное срабатывание.

- **Построение дерева зависимостей проекта**

  Технология OSA позволяет строить граф сторонних компонентов, используемых в проекте. Граф строится как на основе прямых, так и транзитивных зависимостей и позволяет визуализировать все зависимости библиотеки.

- **Сравнение результатов проверок**

  С помощью Solar appScreener можно сравнивать результаты проведенных тестирований и строить различные графики, что позволяет удобно отслеживать динамику устранения или появления уязвимостей и НДВ, в том числе по группам проектов. При этом учитываются изменения, характерные для процесса написания кода, а в рамках одного проекта отслеживаются сами уязвимости и НДВ, что дает возможность контролировать ход их устранения.

- **Построение отчетов**

  Отчеты по уязвимостям и НДВ формируются автоматически, а их содержание выбирает пользователь. Результаты представляются в интерфейсе Solar appScreener либо выгружаются в формате PDF/DOCX/SARIF/JSON/CSV. Доступна выгрузка в соответствии с классификацией уязвимостей по версии PCI DSS, OWASP Top 10, OWASP Mobile Top 10, БДУ ФСТЭК России, ОУД4, HIPAA, OWASP ASVS, OWASP MASVS или CWE/SANS Top 25, а также гибкая выгрузка отчетных данных через JSON.

- **Разграничение прав разработчиков**

  Для повышения уровня информационной безопасности можно разграничить права доступа разработчиков к Solar appScreener. Поддержка Microsoft Active Directory позволяет упростить управление правами доступа при большом количестве разработчиков.

- **Подготовка рекомендаций для разработчиков**

  Разработчики заинтересованы сдавать проекты максимально быстро и с минимальными замечаниями. Именно поэтому рекомендации для разработчиков содержат описания уязвимостей и НДВ, ссылки на содержащие их участки кода, а также конкретные советы по его изменению.

- **Подготовка рекомендаций для офицеров безопасности**

Офицерам безопасности необходима максимально полная информация о найденных уязвимостях и НДВ. Рекомендации для них содержат детальные описания обнаруженных уязвимостей и НДВ, включая способы их реализации, а также рекомендации по настройке WAF от Imperva, ModSecurity, F5 или Positive Technologies WAF.

- **Работа с системами отслеживания ошибок**

  В базовую версию Solar appScreener входит интеграция с Atlassian Jira и ТУРБО Трекинг. Это позволяет заводить задачи по устранению найденных уязвимостей непосредственно из интерфейса Solar appScreener и отслеживать ход их выполнения. При необходимости можно реализовать поддержку любой другой системы отслеживания ошибок.

- **Интеграция в процесс разработки**

  Solar appScreener можно связать с:

  - репозиториями Git, Subversion и Azure;
  - VCS хостингами GitLab, GitHub, Bitbucket;
  - средствами разработки Eclipse, Microsoft Visual Studio, IntelliJ IDEA;
  - средствами сборки Xcode, CMake, Microsoft Visual Studio, GNU Make, GNU Autotools, Gradle, sbt, Maven;
  - с платформой непрерывного анализа и измерения качества кода SonarQube;
  - серверами CI/CD Jenkins, Azure DevOps Server и TeamCity;
  - системой оркестрации DefectDojo.

  Это позволяет встроить его в процесс разработки и реализовать полноценный Secure SDLC. С помощью открытого API и CLT доступна интеграция с другими системами и сервисами.

4. ПРИНЦИП РАБОТЫ

Solar appScreener состоит из двух основных частей:

- Системы анализа, проводящей статический анализ исходных кодов или бинарных файлов, динамический анализ веб-приложений и анализ сторонних компонентов.
- Системы отчетности, предоставляющей рекомендации по устранению уязвимостей и НДВ, а также настройке WAF.

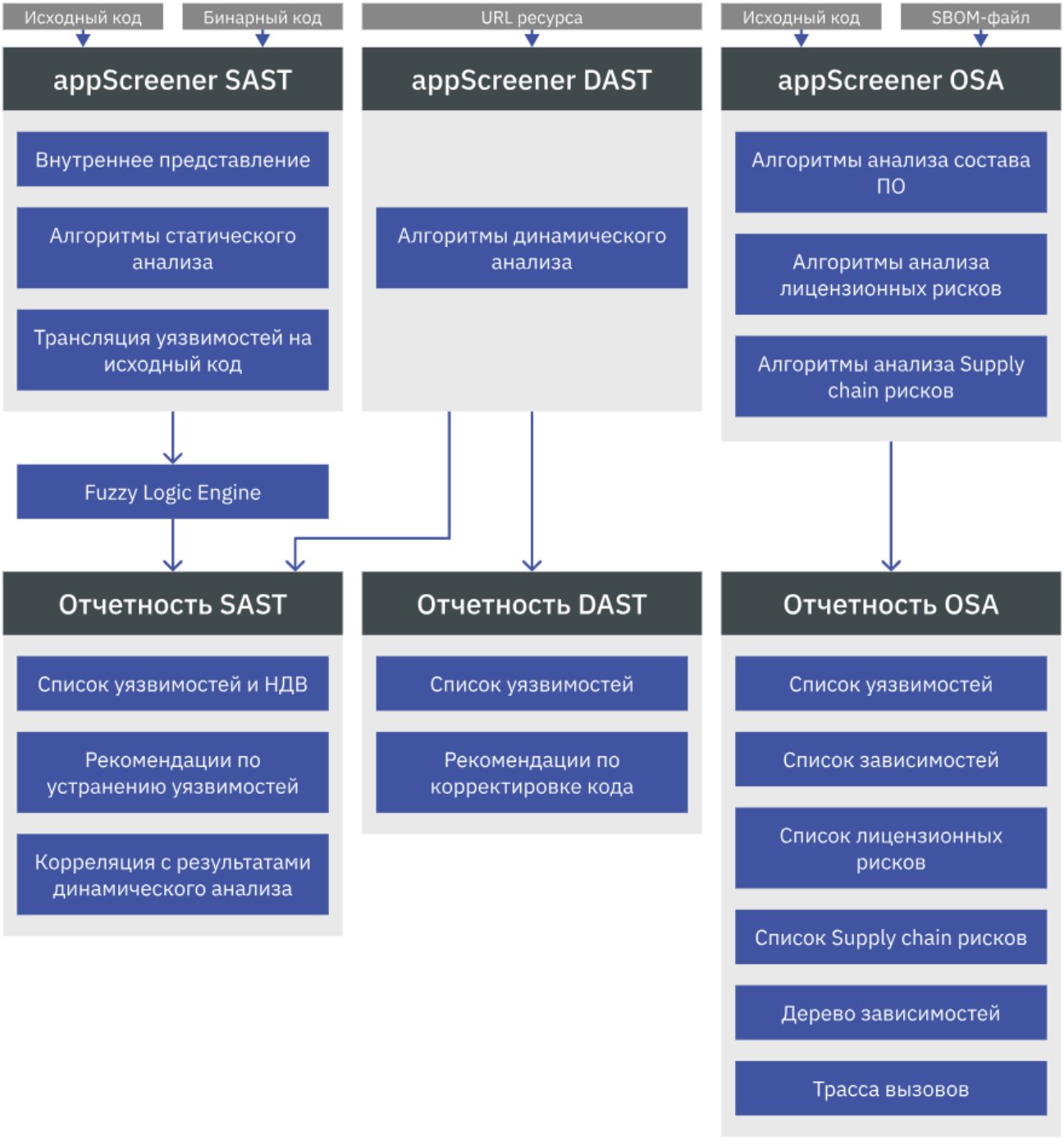


Рисунок 4. Общая схема работы Solar appScreener

4.1. СИСТЕМА АНАЛИЗА КОДА

4.1.1. Технологии статического анализа

Solar appScreener включает в себя следующие алгоритмы статического анализа: лексический, синтаксический, семантический анализы, taint-анализ, распространение констант, распространение типов, анализ синонимов и анализ графов потока управления.

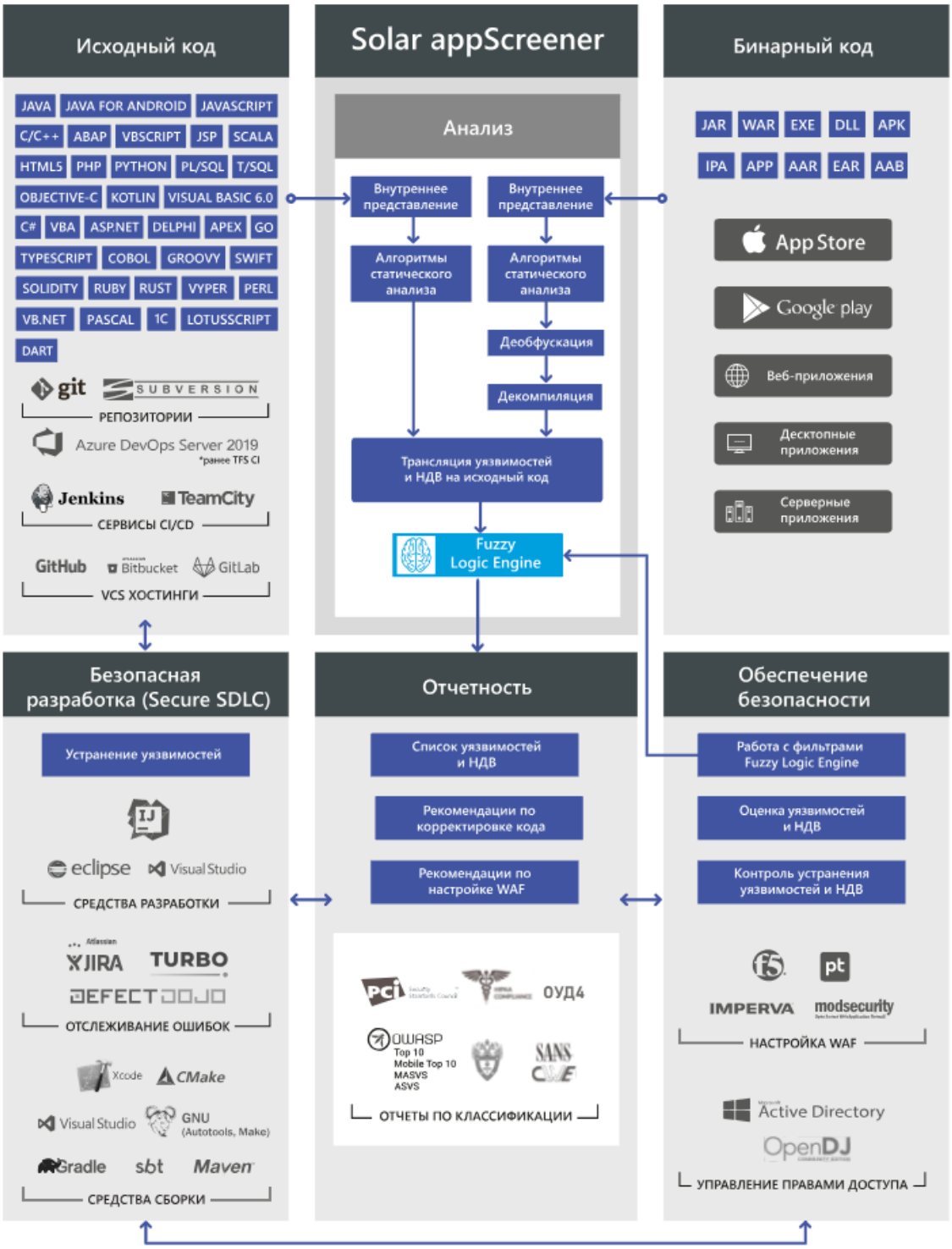


Рисунок 6. Общая схема работы модуля статического анализа

В схеме работы статического анализатора можно выделить три основных шага:

1. Построение промежуточного представления (оно же внутреннее представление или модель кода).
2. Применение алгоритмов статического анализа, в результате работы которых модель кода дополняется новой информацией.
3. Применение правил поиска уязвимостей к дополненной модели кода.

Могут использоваться разные модели кода:

- исходный текст программы;
- поток лексем;
- дерево разбора (AST, Abstract Syntax Tree);
- трехадресный код;
- граф потока управления;
- стандартный или собственный байт-код и т. д.

Лексический, синтаксический и семантический анализ применяются для построения внутреннего представления, чаще всего – дерева разбора (AST, Abstract Syntax Tree).

Лексический анализ разбивает текст программы на лексемы – минимальные смысловые элементы. На выходе получается поток лексем.

Синтаксический анализ проверяет, что поток лексем верен с точки зрения грамматики языка программирования.

Семантический анализ проверяет выполнение более сложных условий, например, соответствие типов данных в инструкциях присваивания.

Получившееся в результате дерево разбора можно использовать как внутреннее представление, или получить из него другие модели, например, перевести его в трехадресный код, из которого строится граф потока управления.

Граф потока управления – основная модель для алгоритмов статического анализа. В качестве внутреннего представления можно использовать и сам исходный код, но тогда не получится достичь необходимого качества анализа. При бинарном анализе (статическом анализе двоичного или исполняемого кода) также строится модель, но здесь уже используются практики обратной разработки: декомпиляции, деобфускации, обратной трансляции. В результате можно получить те же модели, что и из исходного кода. Иногда промежуточным представлением может служить сам бинарный код.

Анализ потока данных – основной алгоритм статического анализа. Он нужен, чтобы определить в каждой точке программы информацию о данных, которые могут храниться в переменных. Например, тип переменной; ее константное значение; какие еще переменные указывают на эти данные.

Задача анализа потока данных зависит от того, какую информацию нужно определить.

Пример:

Таблица 1. Зависимость задачи анализа потока данных от определяемой информации

Что нужно определить?	Задача анализа потока данных
1. Является ли выражение константой 2. Значение константы	Распространение констант
Тип переменной	Распространение типов
Какие переменные указывают на определенную область памяти (хранят одни и те же данные)	Анализ синонимов

Решение задач анализа потока данных

Все эти задачи также используются в теории построения компиляторов. В этой теории описаны решения задач внутрипроцедурного анализа потока данных, когда данные необходимо отслеживать в рамках одной процедуры, функции или метода. Существуют алгоритмы, с помощью которых при соблюдении определенных условий можно решать такие задачи за полиномиальное время. Решения опираются на теорию алгебраических решеток и другие элементы математических теорий.

Но на практике условия теорем не соблюдаются, а главное – необходимо решать задачи межпроцедурного анализа потока данных, так как редко уязвимость полностью локализуется в одной функции. Межпроцедурный анализ потока данных – это экспоненциальная по сложности задача, из-за чего анализатору необходимо проводить ряд оптимизаций и допущений.

Из-за сложности задач анализа потока данных рождаются основные особенности серьезных статических анализаторов:

- длительное время анализа;
- большой объем потребляемых ресурсов;
- ложные срабатывания.

Однако без решения задач межпроцедурного анализа потока данных невозможно находить важнейшие уязвимости.

Taint-анализ – в рамках этого типа анализа отслеживаются метки, которые в некоторых точках программы присваиваются данным. Задача taint-анализа – ключевая для информационной безопасности. Именно с его помощью обнаруживаются уязвимости, связанные с утечкой конфиденциальных данных (запись пароля в журналы событий, небезопасная передача данных) и внедрением данных:

- внедрения в SQL;
- межсайтовый скриптинг;
- открытые перенаправления;
- подделка файлового пути и т. д.

В результате применения указанных выше алгоритмов промежуточное представление дополняется информацией, необходимой для поиска уязвимостей. Правила поиска уязвимостей формулируются в терминах модели кода и описывают, какие признаки в итоговом промежуточном представлении могут говорить о наличии уязвимости.

Пример: необходимо обнаружить уязвимости типа «Внедрение в SQL» (SQL Injection) – когда непроверенные данные от пользователя попадают в методы работы с базой данных.

Для этого необходимо:

1. Определить, что данные поступили от пользователя и добавить им метку taint.
2. Распространить метку по всей анализируемой программе с помощью taint-анализа, учитывая, что данные могут быть валидированы и метка может исчезнуть на одном из путей исполнения.
3. Применить правило поиска уязвимости, которое будет говорить, что к наличию уязвимости приводит вызов определенного метода с параметром, у которого есть пометка «taint».

Как можно видеть из примера выше, помимо глубины алгоритмов важной частью статического анализатора является конфигурация и база правил: описание, какие конструкции в коде порождают данные от пользователя, какие конструкции валидируют такие данны и для каких конструкций критически важно использование этих данных.

Также в системе доступна настройка параметров анализа, в том числе инкрементальный анализ, при котором проверяется только изменившийся код, исключение уязвимостей и файлов, межпроцедурный анализ.

Найденные уязвимости и НДВ графически подсвечиваются непосредственно в коде проанализированного приложения, в том числе если они были в виде исполняемых файлов (файл debug info для этого не нужен).

Для устранения найденных уязвимостей и НДВ нужно не только их обнаружить, но и грамотно описать правила, при которых они работают или закрываются. Solar appScreener дает детальные рекомендации по устранению уязвимостей и НДВ с описанием способов их эксплуатации, а также рекомендации по настройке WAF. База правил поиска уязвимостей и НДВ Solar appScreener постоянно пополняется разработчиками анализатора по итогам проведенных исследований.

В результате анализа формируется отчет, который можно составить в соответствии с классификацией уязвимостей по версии PCI DSS, OWASP Top 10, OWASP Mobile Top 10, БДУ ФСТЭК России, ОУД4, HIPAA, OWASP ASVS, OWASP MASVS или CWE/SANS Top 25. В рамках отчета можно получить рекомендации по настройке WAF.

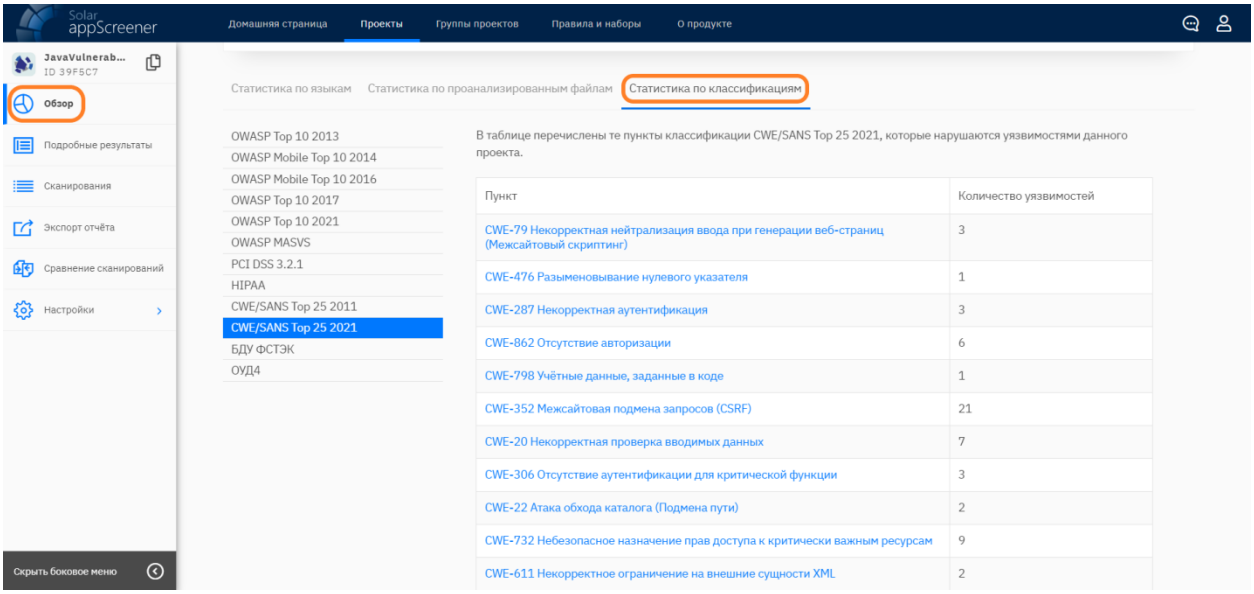


Рисунок 7. Статистика по классификациям безопасности для SAST

4.1.2. Технологии деобфускации и декомпиляции для бинарного анализа модулем статического анализа

Статический анализ исполняемых файлов становится возможным после применения запатентованной технологии реверс-инжиниринга (декомпиляции). Она позволяет с высокой точностью восстанавливать исходный код из бинарного кода, даже если его пытались обфусцировать, то есть запутать. В результате строится внутреннее представление, которое анализируется так же, как и в случае с исходными кодами. При этом анализ идет на низком уровне, а исходный код восстанавливается для трансляции на него найденных уязвимостей и НДВ, чтобы наглядно показать их пользователю Solar appScreener.

Для анализа исполняемых файлов достаточно загрузить код в Solar appScreener, нажать кнопку «Сканировать» и дождаться окончания работы анализатора. Если речь идет о приложениях для Google Android и Apple iOS, то для их анализа достаточно указать ссылку на соответствующий магазин приложений.

Технологии анализа исполняемых файлов позволяют применять SAST, даже когда разработка закончена и нет возможности проанализировать исходный код проекта. Кроме того, можно исследовать код сторонних компонентов, использованных при создании приложения (например, код свободно распространяемых библиотек).

Обратите внимание: для декомпиляции стороннего кода необходимо получить согласие правообладателя этого кода. Если это невозможно, то модуль декомпиляции необходимо отключать. В этом случае будет доступен только перечень уязвимостей и НДВ исполняемого файла без их трансляции на исходный код.

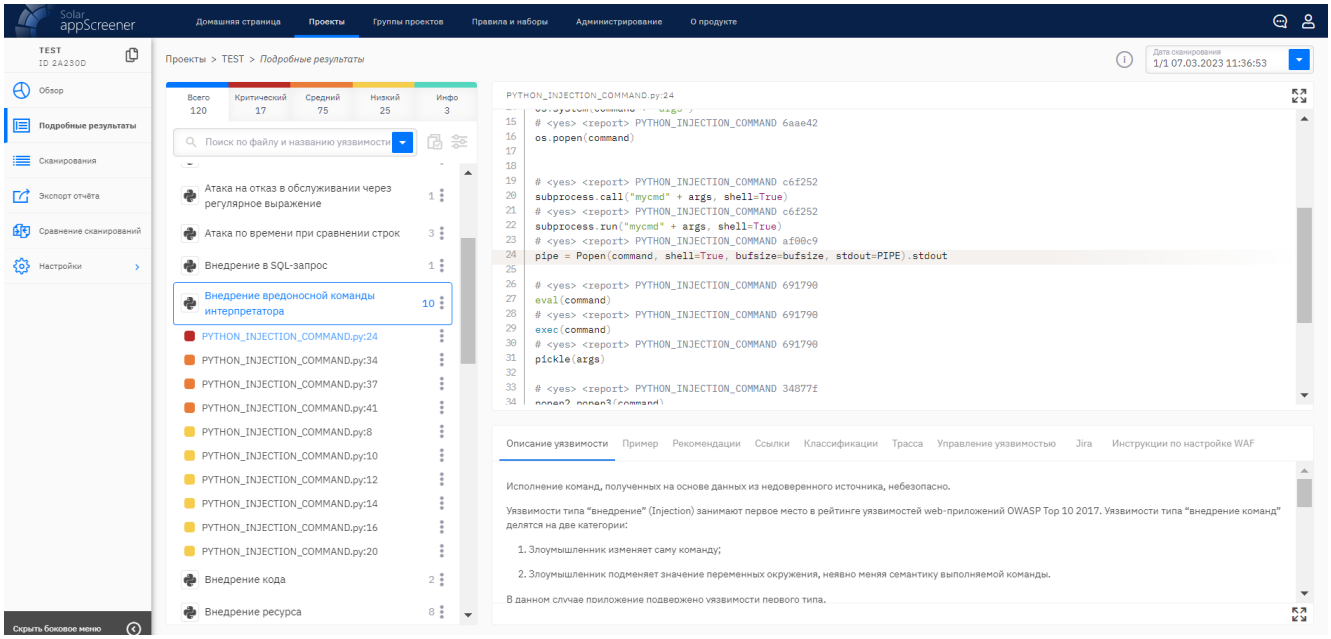


Рисунок 8. Описание найденных уязвимостей статическим анализом

4.1.3. **Технологии динамического анализа**

Модуль динамического анализа является сканером веб-приложений. Он принимает на вход URL веб-приложения, которое необходимо проанализировать, а также его аутентификационные данные. Затем анализатор начинает собирать информацию о страницах, доступных с указанного URL, с помощью специальных поисковых подмодулей - краулеров. В следствие этого образуется дерево URL, собранное краулерами.

Затем динамический анализатор начинает передавать на вход приложения заведомо неправильные или случайные данные, воспроизводя различные типы атак. Найденные в ходе атаки уязвимости сохраняются в системе для дальнейшего отображения.

Solar appScreener включает в себя следующие виды динамического анализа: традиционный, AJAX-веб-сканер, автоматический сканер, пассивный сканер и Fuzzer. Схема работы динамического анализа представлена на рисунке 9.

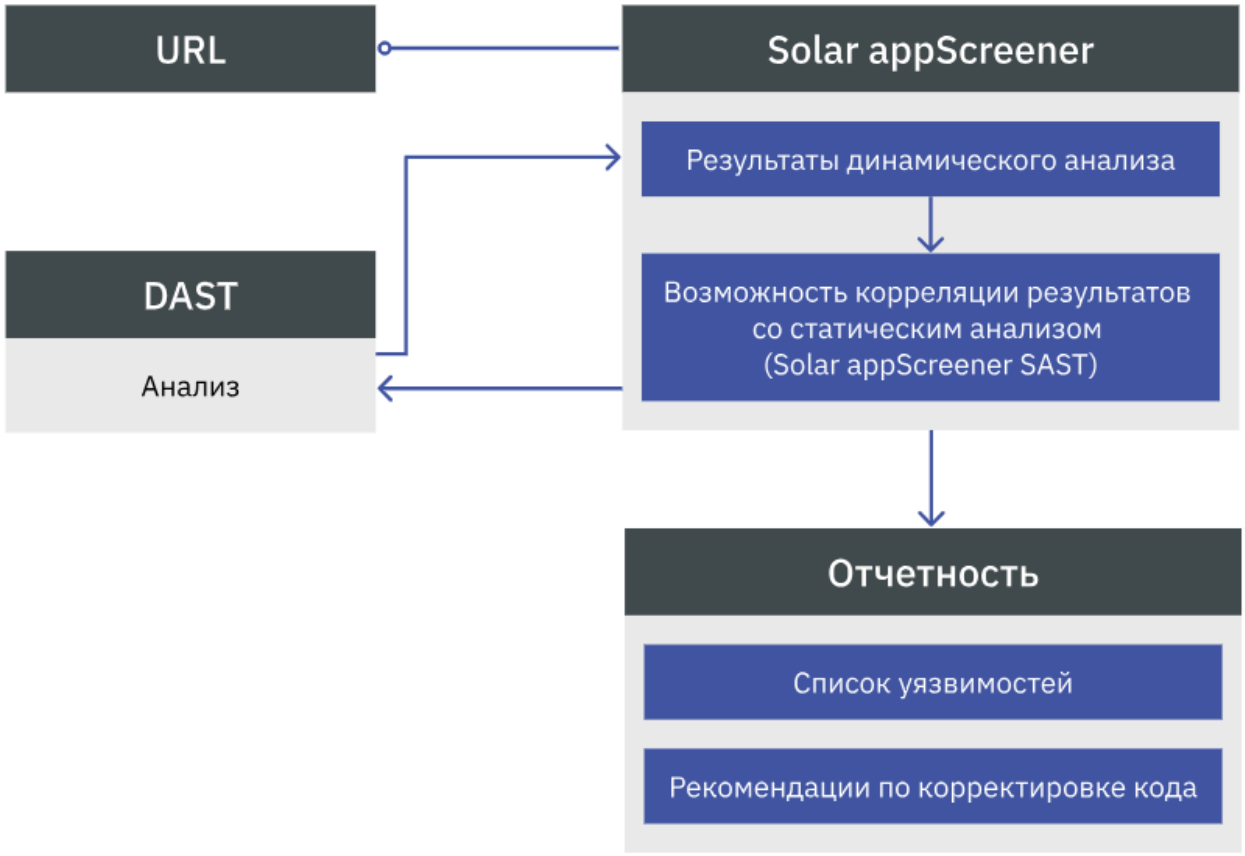


Рисунок 9. Общая схема работы модуля динамического анализа

AJAX-запросы – это технология обращения к серверу без перезагрузки страницы. Аббревиатура расшифровывается как Asynchronous Javascript and XML. AJAX использует два метода работы с веб-страницей: изменение веб-страницы без ее перезагрузки и динамическое обращение к серверу. Второй может осуществляться несколькими способами, в частности, XML Http Request, или техникой скрытого фрейма. В первую очередь AJAX полезен для форм и кнопок, связанных с элементарными действиями: добавить в корзину, подписаться и т. п. В настоящее время в порядке вещей, что такие действия на сайтах осуществляются без перезагрузки страницы.

Инструменты фаззинга тестируют приложение на неожиданный вход, чтобы выяснить, приведет ли это к странным или непредсказуемым результатам и ошибкам. Данная функция может быть полезна для работы с элементами страницы с известными входными данными, например с полями для ввода сумм или Ф. И. О. Если программист был внимателен во время написания кода, большинство из этих недостатков бизнес-логики были им предвидены и предотвращены. Однако практически невозможно предусмотреть каждую потенциальную проблему, из-за чего могут возникнуть уязвимости.

Выбор режима атаки позволяет выбрать один из трех режимов атак в один клик в зависимости от целей сканирования.

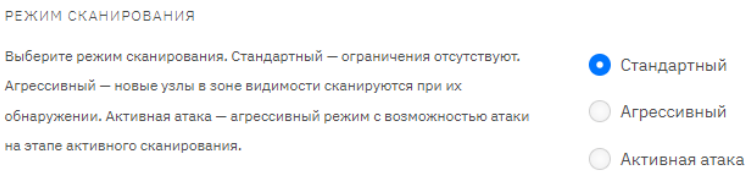


Рисунок 10. Выбор режима атаки

Обратите внимание: для динамического анализа необходимо получить согласие правообладателя ресурса. Также динамический анализ необходимо проводить на клоне рабочего ресурса во избежание поломки. Если это невозможно, то модуль динамического анализа необходимо отключать.

В модуле DAST Solar appScreener реализовано несколько вариантов авторизации на атакуемых веб-приложениях:

- Логин/пароль – анализатор будет использовать указанные логин и пароль на стандартных полях для авторизации.
- Токен аутентификации – анализатор вставит в заголовки запросов указанный bearer-токен.
- Заголовки – анализатор вставит в заголовки запросов все заголовки, указанные в Json-форме.
- Форма авторизации – анализатор вставит логин и пароль в конкретные поля веб-приложения, которые указал пользователь.
- NTLM – анализатор авторизуется на указанном сервере по протоколу NTLM.

Если приложение было просканировано статическим и динамическим способом, то пользователь может провести корреляцию результатов этих методов. Это позволит приоритизировать обработку уязвимостей, которые были найдены обоими видами анализа.

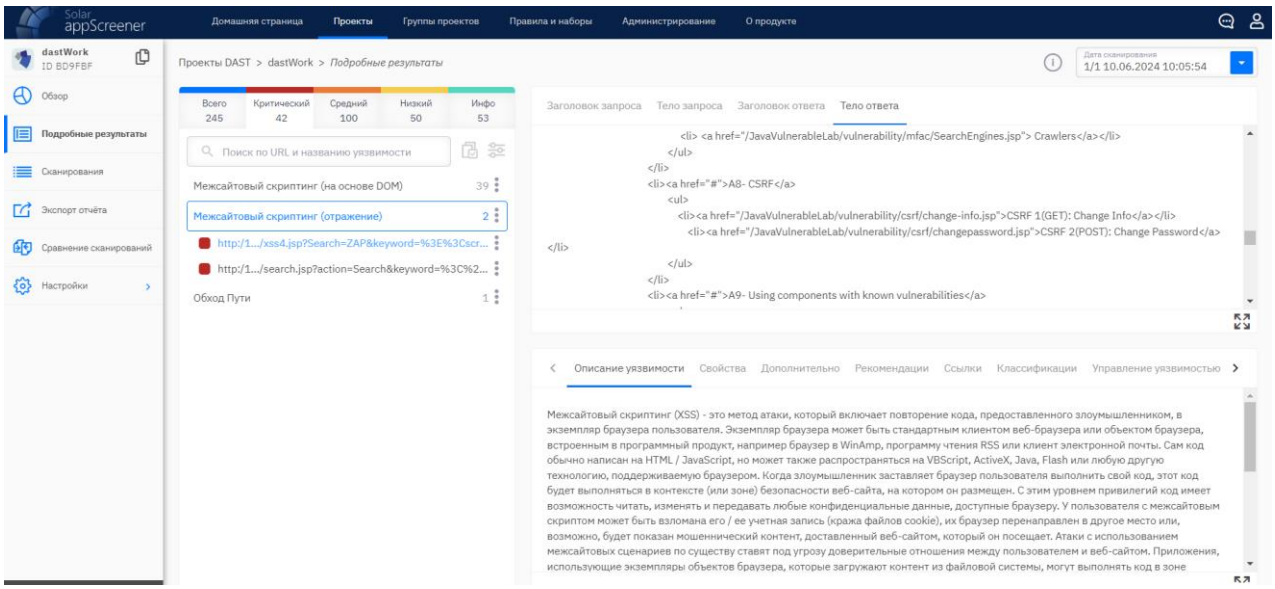


Рисунок 11. Результаты динамического анализа

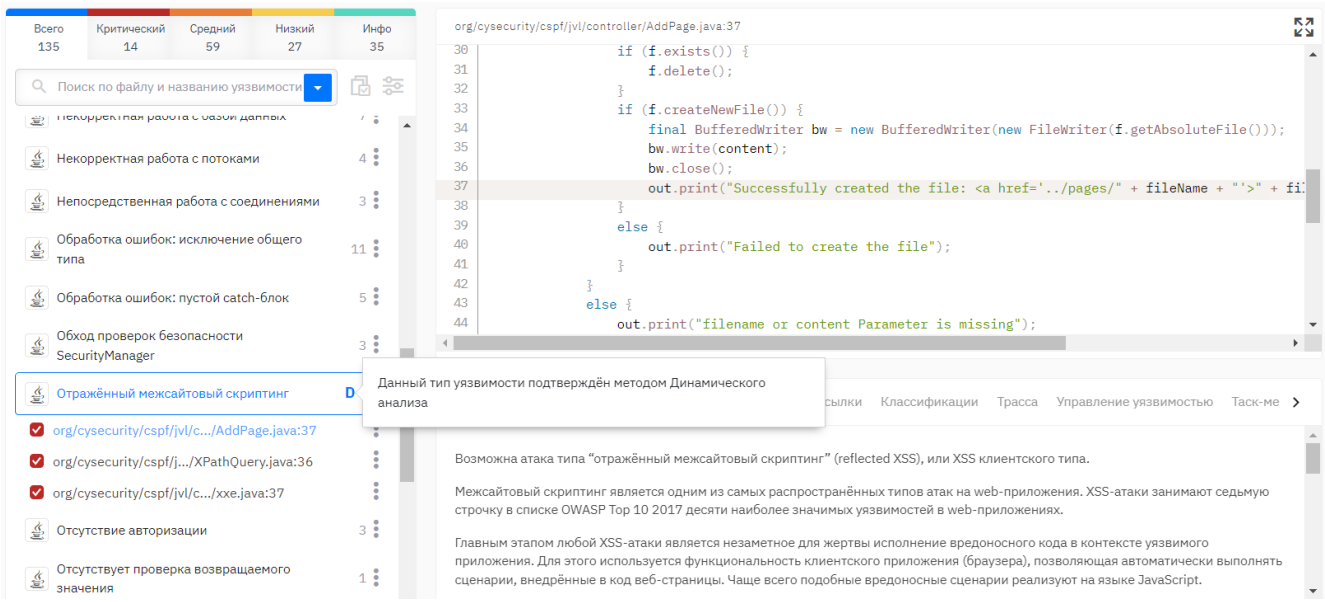


Рисунок 12. Корреляция результатов статического и динамического анализа

4.1.4. Технологии анализа сторонних компонентов (OSA)

Модуль OSA получает на вход SBOM-файл. SBOM может быть как загружен в систему клиентом, так и быть собран системой автоматически из исходного кода приложения с помощью встроенного универсального сборщика SBOM.

Затем OSA собирает информацию о компонентах на основе SBOM-файла, например определяет названия, версии и транзитивные зависимости, используемых в проекте компонентов. После этого запускается анализ всех компонентов с помощью технологий SCA, SCS или анализа лицензионных рисков.

Модуль OSA Solar appScreener позволяет проводить анализ как прямых, так и транзитивных зависимостей, дает описание найденных уязвимостей и предоставляет рекомендации по замене компонентов. Также для всех методов анализа SCA система строит граф сторонних зависимостей проекта, на котором визуализированы как прямые, так и транзитивные зависимости.

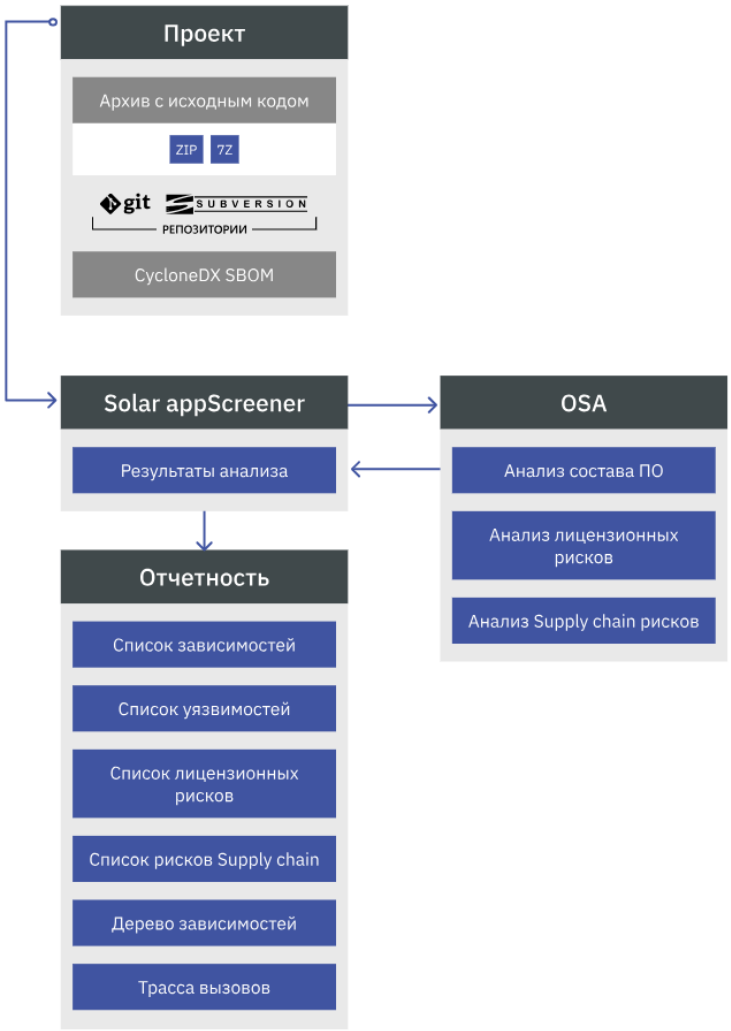


Рисунок 13. Общая схема работы модуля анализа состава сторонних компонентов

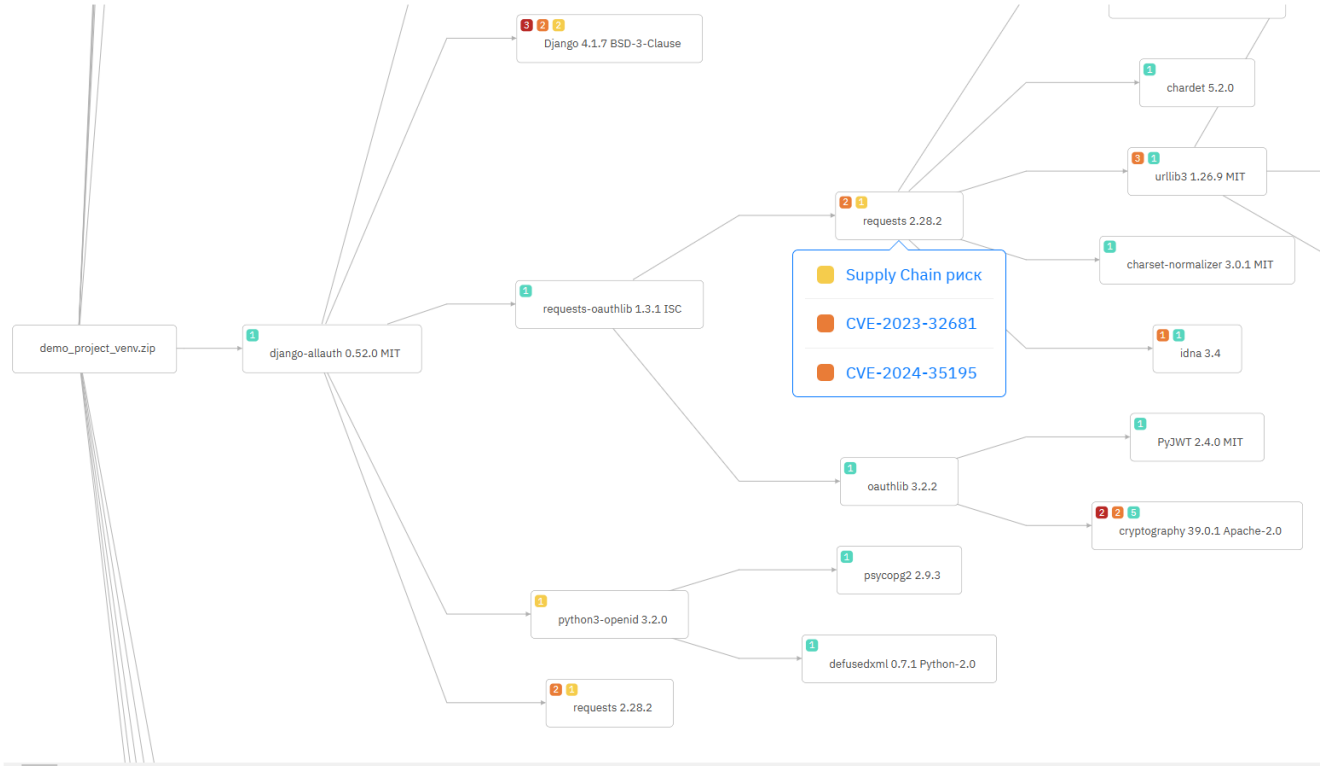


Рисунок 14. Граф транзитивных зависимостей проекта

4.1.4.1. Технология анализа состава ПО (SCA)

Технология **анализа состава ПО (SCA)** проверяет компоненты по базе поиска уязвимостей Solar appScreener. База включает в себя как сторонние фиды (GHSA, Google OSV, Gitlab, БДУ ФСТЭК, EPSS), так и собственную базу уязвимых сторонних компонентов, в которую входят в том числе компоненты, связанные с Ukrainian Malware.

Технология SCA в Solar appScreener позволяет находить компоненты, в которых был обнаружен sabotage open source, проводить known vulnerability analysis, outdated component analysis (поиск устаревших компонентов).

4.1.4.2. Технология анализа лицензионных рисков

Анализ лицензионных рисков проводится способом, аналогичным SCA, то есть компоненты, найденные в SBOM, проверяются по базам лицензий сторонних компонентов, и среди них выявляются лицензии с ограничениями, которые могут быть критичны для пользователя. Это позволяет избежать юридических проблем для пользователей, связанных с лицензиями используемых сторонних библиотек.

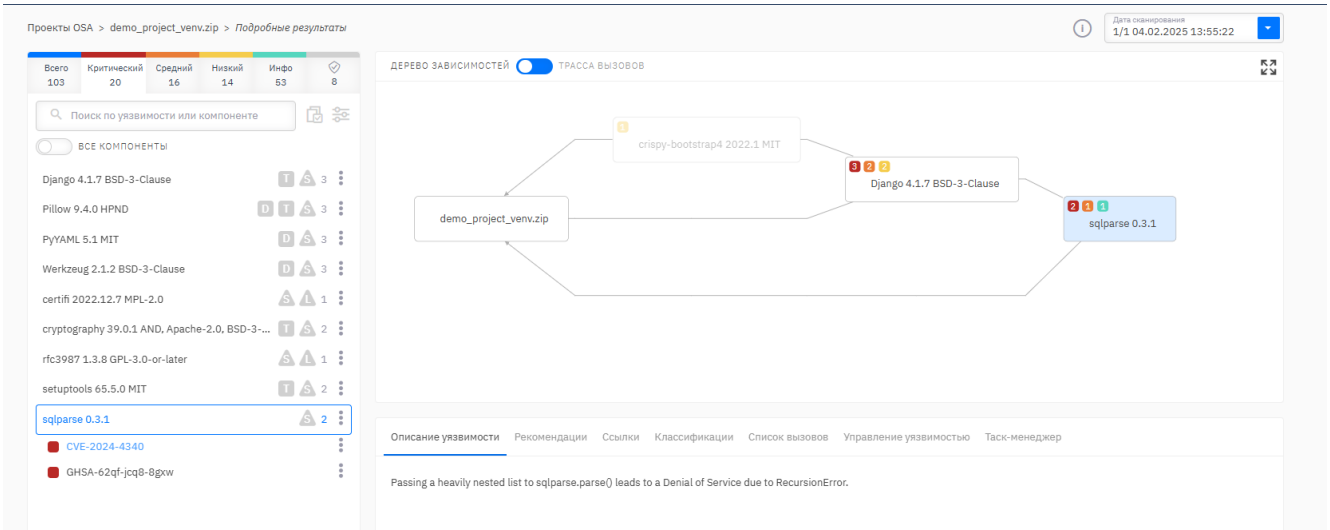


Рисунок 15. Результаты анализа сторонних компонентов

4.1.4.3. Комбинированный анализ SAST и OSA

Комбинированный анализ SAST и OSA нужен для сокращения времени на обработку уязвимых зависимостей. Это позволяет исключить недостижимые уязвимости, тем самым сокращая их общее количество.

Процесс анализа SAST и OSA начинается с поиска известных уязвимостей среди компонентов проекта (OSA), затем включается модуль статического анализа (SAST), который определяет конкретные уязвимые импорты и вызовы функций сторонних компонентов. В результате анализа строится граф вызовов импортов и уязвимых функций зависимостей, используемых в проекте, что показывает достижимость каждой уязвимости.

Комбинированный анализ SAST и OSA поддерживает следующие языки:

- JS/TS, Python: анализируются как директивные, так и транзитивные уязвимые зависимости.
- Java, C#: анализируются только директивные уязвимые зависимости.

Для корректной работы модуля необходимо выполнить следующие подготовительные шаги:

- **JS/TS, Python**: Для определения достижимости уязвимостей в директивных и транзитивных зависимостях вместе с основным кодом проекта нужно передать также папку с кодом сторонних зависимостей.
  По умолчанию для Python это папка venv, для JS/TS – node_modules. Название папки по умолчанию можно изменить в "Общих настройках" при старте сканирования.
- **Java**: Для определения достижимости уязвимостей в директивных зависимостях также требуется папка с зависимостями проекта, в частности jar-файлы. Для этого можно использовать Maven-плагин dependency:copy-dependencies, который копирует все зависимости проекта в указанный каталог.
  По умолчанию для Java это папка .m2/repository/.
  Название папки по умолчанию можно изменить в "Общих настройках" при старте сканирования.
- **C#:** Для определения достижимости уязвимостей в директивных зависимостях достаточно загрузить на анализ только основной код проекта.

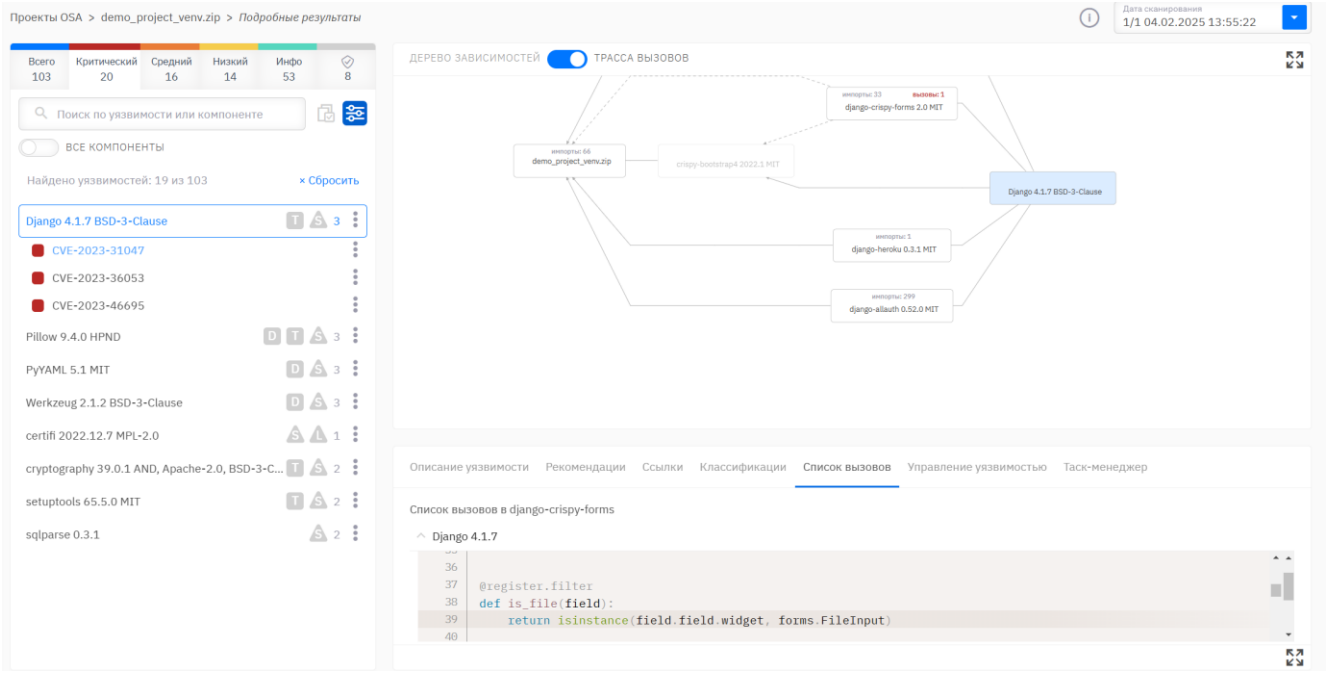


Рисунок 16. Результаты комбинированного анализ SAST/OSA

4.1.4.4. Технологии анализа цепочки поставок (SCS)

Supply Chain Security (SCS) в Solar appScreener позволяет проводить анализ компонентов безопасности на всех этапах пути, по которому ПО попадает в организацию, от момента их создания или покупки до этапа использования.

Технология анализа безопасности цепочки поставок (SCS) позволяет принимать решение о безопасности компонента по 8 метрикам, таким как:

- популярность используемого компонента
- оценка авторского состава
- оценка реакции сообщества на проблемы в безопасности компонента
- оценка заинтересованности авторов в безопасности
- дата выхода компонентов
- версия первого релиза компонента
- количество созданных проектов у разработчика
- процент код-ревью каждого пул реквеста двумя и более рецензентами.

Модуль анализа цепочки поставок Solar appScreener умеет выявлять известные тактики злоумышленников для атаки на цепочку поставок: Dependency Confusion, Starjacking, Typosquatting, MavenGate, а также позволяет находить и определять ненадежных контрибьютеров.

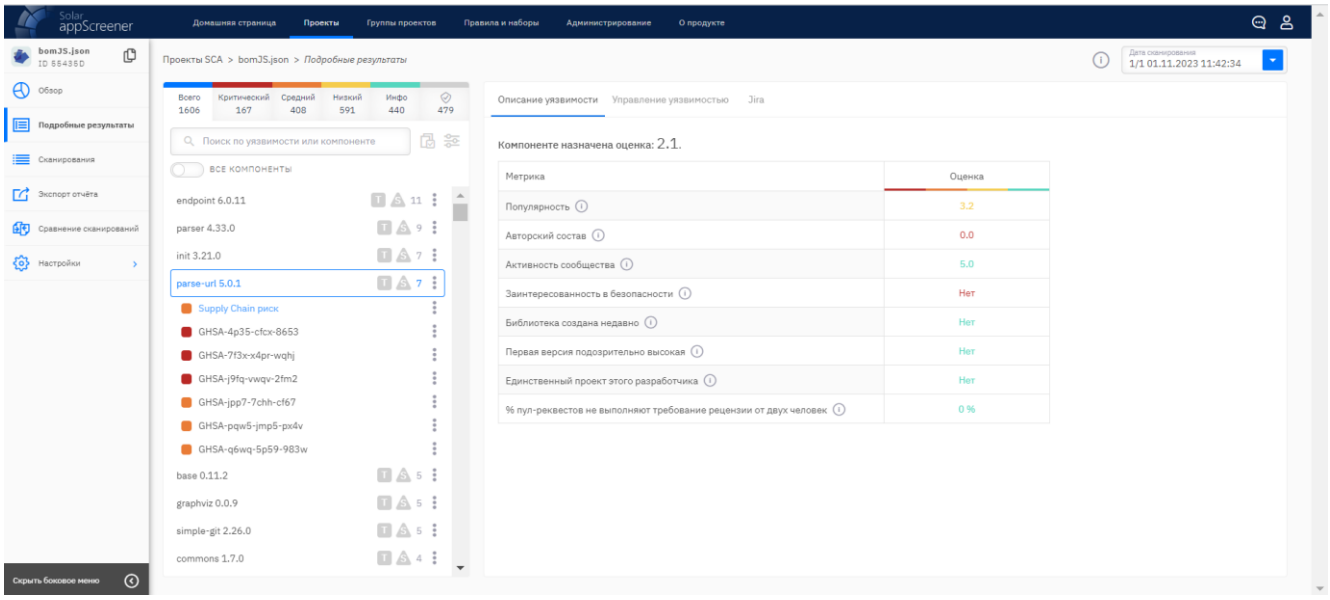


Рисунок 17. Результаты анализа цепочки поставок

4.1.5. Фильтр ложных срабатываний Fuzzy Logic Engine

Для минимизации количества ложных срабатываний и пропуска уязвимостей и НДВ в коде в Solar appScreener реализована уникальная технология Fuzzy Logic Engine. Технология использует математический аппарат нечеткой логики и является технологическим ноу-хау компании «Солар». Параметры работы фильтров определяются базой знаний, которая постоянно пополняется по результатам проведенных исследований.

Математическая теория нечетких множеств (fuzzy sets) и нечеткая логика (fuzzy logic) – это обобщения классической теории множеств и формальной логики. Основной причиной появления новой теории стало наличие нечетких и приближенных рассуждений при описании человеком процессов, систем или объектов.

Количество ложных срабатываний и пропусков уязвимостей – одна из ключевых характеристик анализатора кода, поэтому развитие и совершенствование Fuzzy Logic Engine является важным приоритетом развития Solar appScreener.

В Solar appScreener есть возможность настройки фильтров Fuzzy Logic Engine, чтобы еще больше снизить число ложных срабатываний и пропусков уязвимостей и НДВ.

Fuzzy Logic Engine реализован для статического анализа и комбинированного анализа SAST и OSA.

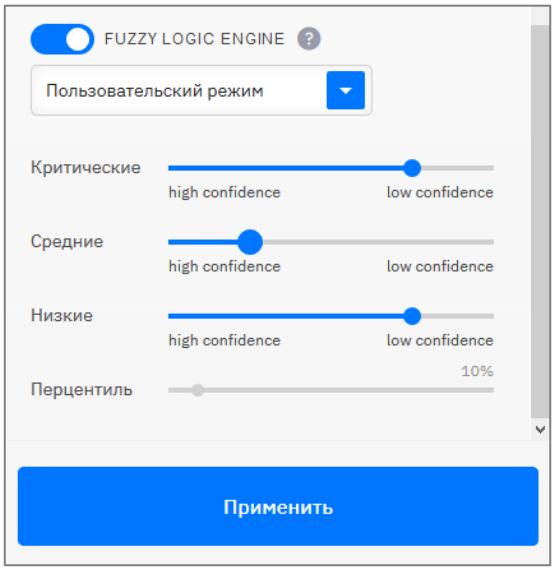


Рисунок 18. Fuzzy Logic Engine. Отображение части/процента наиболее критических уязвимостей

4.2. СИСТЕМА ОТЧЕТНОСТИ

Система отчетности Solar appScreener позволяет:

- Графически подсвечивать найденные уязвимости и НДВ непосредственно в коде проанализированного приложения, даже если анализировался исполняемый файл.

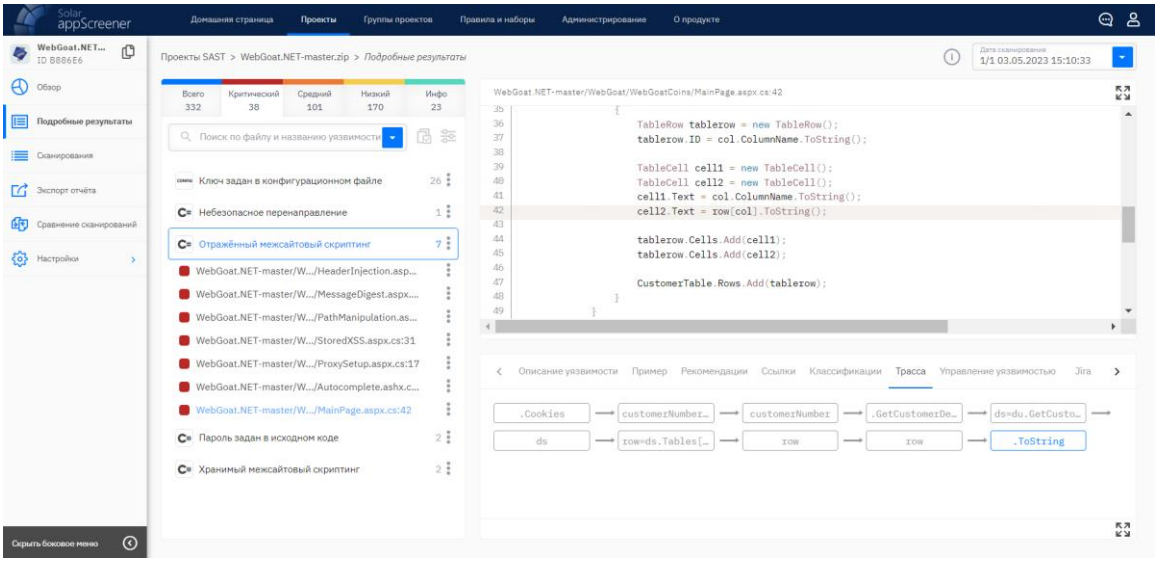


Рисунок 19. Описание реализации уязвимости

- Сравнивать результаты проведенных тестирований в рамках одного проекта или разных групп проектов для отслеживания динамики устранения или появления уязвимостей. При этом учитываются изменения, характерные для процесса написания кода.

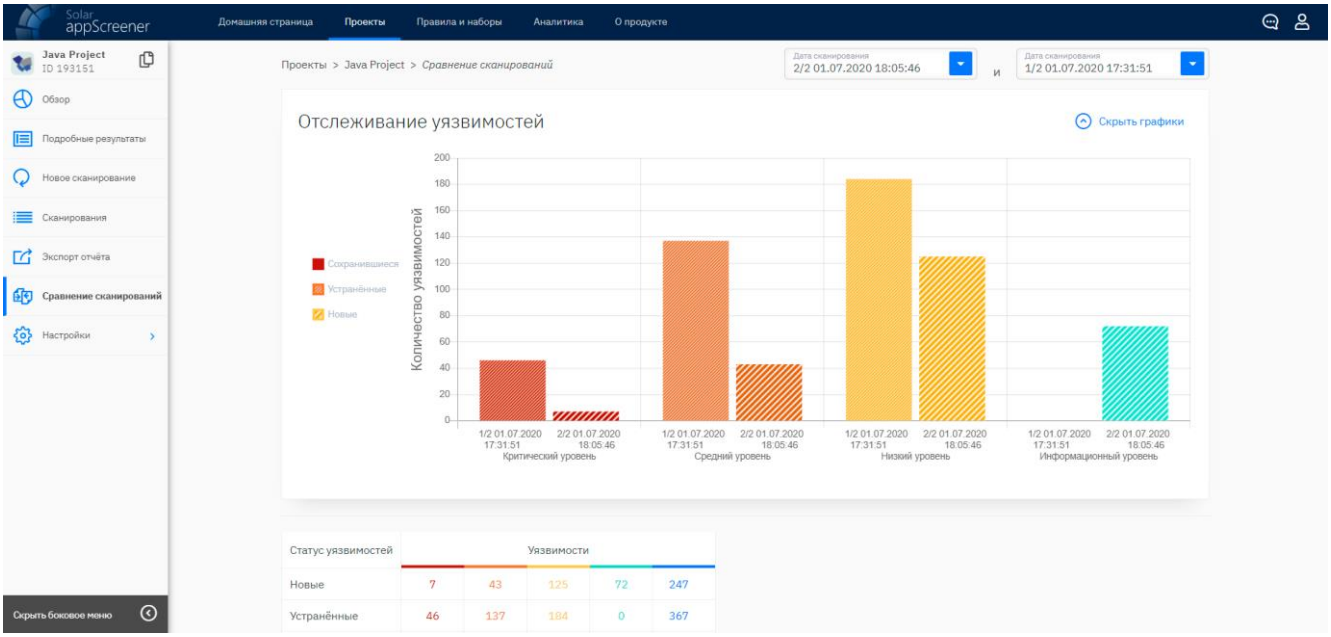


Рисунок 20. Сравнение результатов тестирования

- Получать рекомендации в формате как для службы ИБ, так и команды разработки:
  - Отчетность для разработки. Включает детальные описания уязвимостей и НДВ, ссылки на содержащие их участки, а также рекомендации по их устранению путем внесения изменений в код.
  - Отчетность для ИБ. Включает подробные рекомендации по устранению уязвимостей и НДВ с описанием способов их эксплуатации, а также по настройке WAF для блокировки возможности эксплуатации уязвимостей в приложениях во время исправления кода.

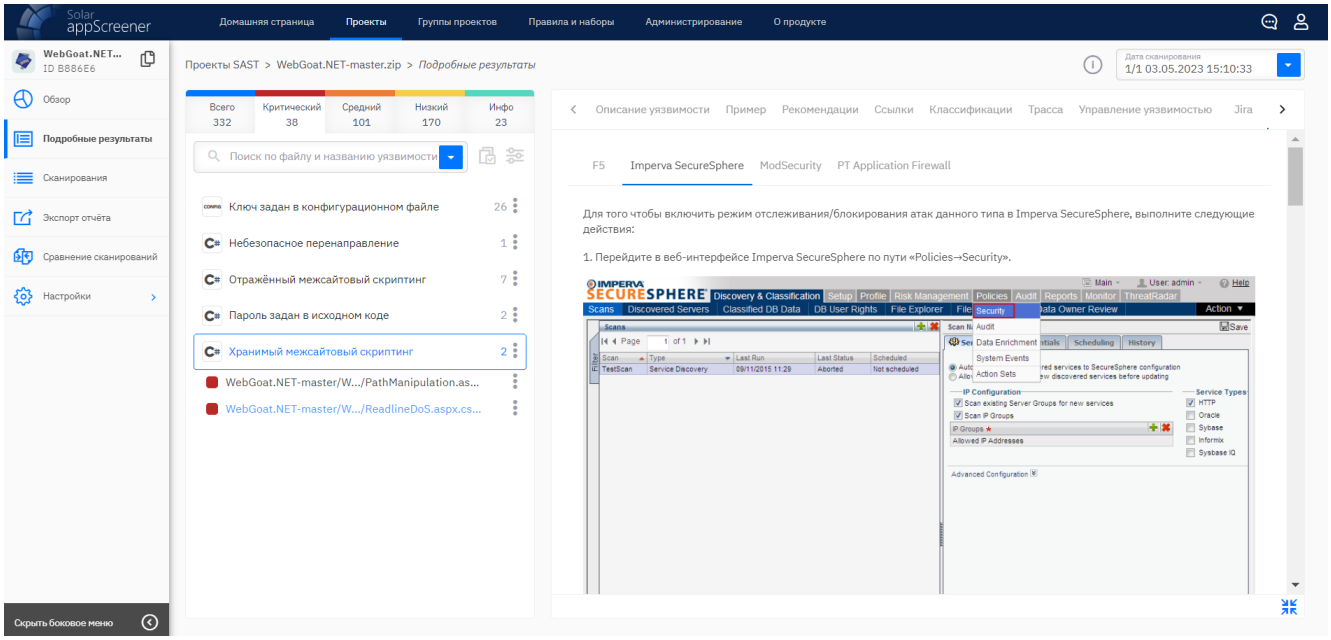


Рисунок 21. Рекомендации по настройке WAF

- Выгружать отчеты в формате PDF/DOCX/JSON/CSV, в том числе сверстанные в соответствии с классификацией уязвимостей по версии PCI DSS, OWASP Top 10 2017/2021, OWASP Mobile Top 10 2016, БДУ ФСТЭК России, ОУД4, HIPAA, OWASP ASVS, OWASP MASVS или CWE/SANS Top 25 2021.
- Получить итоги корреляции результатов статического и динамического анализа в отчете и веб-интерфейсе.
- Получить список уязвимых сторонних компонентов.
- Получить информацию об уязвимостях, найденных в сторонних компонентах.
- Получать уведомления с обновлением статуса проверки по электронной почте.
- Получать отчеты по электронной почте.

4.3. ВОЗМОЖНОСТИ ИНТЕГРАЦИИ

Solar appScreener обладает широкими возможностями по интеграции с репозиториями, системами отслеживания ошибок, интегрированными средствами разработки и сервисами CI/CD.

- ИНТЕГРАЦИЯ С РЕПОЗИТОРИЯМИ РАЗРАБОТКИ GIT, SUBVERSION И AZURE

  Код для анализа загружается напрямую из репозитория, избавляя от необходимости каждый раз загружать файлы с исходным кодом.

- ИНТЕГРАЦИЯ С VCS ХОСТИНГАМИ GITLAB, GITHUB, BITBUCKET

  Можно настроить автоматическое сканирование по расписанию с помощью механизма webhook для получения информации в режиме реального времени. Также поддерживаются push- и tag-события.

- ИНТЕГРАЦИЯ С СИСТЕМАМИ ОТСЛЕЖИВАНИЯ ОШИБОК

  В базовой версии Solar appScreener поддерживает интеграцию с Atlassian Jira и Турбо Трекинг, но при необходимости можно интегрировать любую другую систему отслеживания ошибок. Такая интеграция позволяет офицеру безопасности напрямую заводить задачи по устранению найденных уязвимостей и НДВ и отслеживать ход их выполнения. Например, он может открыть кейс для команды разработки по внесению изменений в код или для системных администраторов для внесения соответствующих правил в WAF.

- ИНТЕГРАЦИЯ В ПРОЦЕССЫ CI/CD И SDLC
  - Поддержка интегрированных средств разработки Eclipse, Microsoft Visual Studio и IntelliJ IDEA.
  - Поддержка средств сборки Xcode, CMake, Microsoft Visual Studio, GNU Make, GNU Autotools, Gradle, sbt, Maven.
  - Поддержка серверов непрерывной интеграции и доставки Jenkins, Azure DevOps Server 2019 и TeamCity.

- Поддержка интеграции с платформой непрерывного анализа и измерения качества кода SonarQube.

- ИНТЕГРАЦИЯ С DEFECT DOJO

  Интеграция Solar appScreener и ASOC DefectDojo позволяет пользователям удобно встраиваться в процессы безопасной разработки.

- ОТКРЫТЫЙ ВСТРОЕННЫЙ API и CLT

  В Solar appScreener встроен открытый API. Он включает в себя JSON API и интерфейс командной строки и предоставляет широкие возможности по дополнительной интеграции и автоматизации.

5. ПРЕИМУЩЕСТВА

- СТАТИЧЕСКИЙ АНАЛИЗ БИНАРНОГО КОДА

  Уникальные технологии декомпиляции и деобфускации кода исполняемых файлов позволяют проверять приложения даже при отсутствии исходных кодов, например, унаследованные приложения или заказные разработки, в том числе для Google Android и Apple iOS.

- ПОДДЕРЖКА 36 ЯЗЫКОВ ПРОГРАММИРОВАНИЯ

  Большое число поддерживаемых языков позволяет анализировать статическим методом почти все приложения, в том числе созданные на ABAP (для SAP), COBOL или Solidity (язык для смарт-контрактов для платформы Ethereum). Язык приложения определяется автоматически. Возможен анализ приложений, написанных на нескольких языках.

- 10+ МЕТОДОВ СТАТИЧЕСКОГО АНАЛИЗА КОДА

  Для анализа приложений в Solar appScreener возможно совместное использование более 10 методов анализа, в том числе анализа потока выполнения и taint-анализа, что позволяет максимизировать выявление уязвимостей и НДВ в коде приложения.

- ПОСТРОЕНИЕ ДЕРЕВА СТОРОННИХ ЗАВИСИМОСТЕЙ ПРОЕКТА

  OSA в Solar appScreener позволяет удобно визуализировать дерево прямых и транзитивных зависимостей, используемых в проекте, с помощью интерактивного графа.

- КОРРЕЛЯЦИЯ РЕЗУЛЬТАТОВ СТАТИЧЕСКОГО И ДИНАМИЧЕСКОГО АНАЛИЗА

  Благодаря поддержке и статического, и динамического анализа можно с помощью DAST подтверждать уязвимости, найденные статическим методом, и устранять их у первую очередь.

- КОМБИНИРОВАННЫЙ АНАЛИЗ SAST/OSA

  Позволяет строить трассу вызовов уязвимых функций из сторонних библиотек, чтобы определять, действительно ли уязвимая функция в библиотеке исполняется в коде.

- ВОЗМОЖНОСТЬ ПРОВЕДЕНИЯ SAST, DAST, OSA В ОДНОМ ИНТЕРФЕЙСЕ

  В пределах одной системы можно проводить анализ кода с помощью SAST, DAST и OSA. Результаты всех сканирований доступны в едином веб-интерфейсе Solar appScreener.

- ПОДРОБНЫЕ РЕКОМЕНДАЦИИ

  Результаты анализа кода приложений предоставляются разработчикам и службе ИБ в формате конкретных рекомендаций по устранению уязвимостей и НДВ. Так, рекомендации по настройке WAF позволяют блокировать уязвимости и НДВ на время исправления кода приложения.

- ПРАВИЛА ПОИСКА ОТ ЭКСПЕРТОВ КИБЕРБЕЗОПАСНОСТИ

  В подготовке правил поиска уязвимостей и НДВ для Solar appScreener участвуют высококвалифицированные специалисты «Солар», что гарантирует высокое качество и актуальность правил. Базы уязвимостей и НДВ можно обновлять как вручную, так и в автоматическом режиме.

- БЫСТРЫЙ ЗАПУСК

  Проверка запускается в несколько кликов и не требует долгой предварительной настройки. Для анализа приложений для Android и Apple iOS достаточно указать ссылку на них в магазинах приложений Google Play и App Store.

- СОВРЕМЕННЫЙ ИНТЕРФЕЙС

  Удобный и современный интерфейс Solar appScreener обеспечивает быструю работу и наглядный результат анализа уязвимостей и НДВ. Сравнение результатов тестирования и работа с интегрированными системами доступны непосредственно из интерфейса решения.

- НЕ ТРЕБУЕТ ОПЫТА РАЗРАБОТКИ

  Solar appScreener в первую очередь рассчитан на службу ИБ. Интерфейс отличается простотой и удобством, а сам анализ максимально автоматизирован. В результате с анализатором может работать офицер безопасности, не имеющий опыта разработки ПО.

- ШИРОКИЙ ОХВАТ И ВЫСОКАЯ СКОРОСТЬ РАБОТЫ

  Статический анализ кода приложений охватывает наибольшее число возможных уязвимостей и НДВ, а также отличается высокой скоростью работы. Для завершения анализа не нужно ждать часы и дни – на обычное приложение достаточно 30-40 минут.

- НИЗКИЙ ПРОЦЕНТ ЛОЖНЫХ СРАБАТЫВАНИЙ

  Для минимизации количества ложных срабатываний и пропущенных уязвимостей и НДВ в коде в Solar appScreener используется технология Fuzzy Logic Engine, которая задействует математический аппарат нечеткой логики и является технологическим ноу-хау компании «Солар».

- УДОБНАЯ ПОСТАВКА

  За счет поставки в docker и возможности управления агентами сканирования, систему можно масштабировать как горизонтально, так и вертикально, устанавливать как на один сервер, так и на несколько.

- ЛЕГКАЯ ИНТЕГРАЦИЯ В SDLC

Solar appScreener легко встраивается в процесс разработки и обеспечивает Secure SDLC. Это возможно благодаря простой интеграции решения с:

- репозиториями Git, Subversion, Azure;
- VCS-хостингами GitLab, GitHub, Bitbucket;
- средствами разработки Eclipse, Microsoft Visual Studio, IntelliJ IDEA;
- средствами сборки Xcode, CMake, Visual Studio, GNU Make, GNU Autotools, Gradle, sbt, Maven;
- платформой непрерывного анализа и измерения качества кода SonarQube;
- серверами CI/CD Jenkins, Azure DevOps Server и TeamCity;
- системами отслеживания ошибок Atlassian Jira и ТУРБО Трекинг.

- ON-PREMISE И SAAS

  В отличие от многих конкурирующих решений, Solar appScreener можно развернуть как на собственных вычислительных мощностях организации, так и использовать его как услугу из облака «Солар» по модели SaaS.

- ОТЕЧЕСТВЕННОЕ ПО

  Solar appScreener разработан в России и внесен в Единый реестр отечественного ПО (№ 6119), что позволяет использовать его для импортозамещения зарубежных аналогов. Над документацией и интерфейсом продукта работали русскоязычные специалисты. Цены представлены в рублях и не зависят от курса валют.

- СЕРТИФИКАТ ФСТЭК РОССИИ

  Solar appScreener сертифицирован ФСТЭК России как программное средство контроля защищенности и соответствует требованиям руководящего документа «Защита от несанкционированного доступа к информации. Часть 1» (Гостехкомиссия России, 1999) по 4-му уровню контроля отсутствия НДВ (сертификат соответствия № 4007).

6. ПРИМЕРЫ ПРИМЕНЕНИЯ

ОПЕРАТИВНАЯ БЛОКИРОВКА УЯЗВИМОСТЕЙ

При приемке новой системы дистанционного банковского обслуживания (ДБО) от департамента разработки служба безопасности банка проанализировала код на уязвимости статическим и динамическим методом. Проверка выявила критические уязвимости, которые позволяли получить в системе ДБО привилегии администратора. Для валидации уязвимостей использовался фильтр ложных срабатываний Fuzzy Logic Engine и проводилась корреляция результатов статического и динамического анализа. На устранение уязвимостей требовалось 3,5 месяца. Крайне сжатые сроки внедрения ДБО существенно влияли на бизнес-показатели, и банк решил развернуть систему, одновременно нейтрализовав возможность использования уязвимостей с помощью WAF. Solar appScreener предоставил детальные рекомендации по настройке WAF, пока разработчики занимались устранением найденных уязвимостей в коде.

ВЫЯВЛЕНИЕ УЯЗВИМОСТЕЙ В СТОРОННИХ КОМПОНЕНТАХ ПО

В ходе первого этапа пилотного проекта по внедрению Solar appScreener финансовая организация решила проверить исходные коды своего бизнес-приложения. Объем файла насчитывал 30 000 строк кода, уязвимостей в нем нашлось немного. На втором этапе проекта с помощью Solar appScreener проверили готовое приложение в виде исполняемых файлов. Бинарный анализ показал реальный объем исходных кодов – более 500 000 строк – и несколько сотен уязвимостей. Выяснилось, что для экономии времени разработки большую часть кода приложения составляли сторонние компоненты: свободно распространяемое ПО, готовые коды из интернета, модули и библиотеки.

ОПРЕДЕЛЕНИЕ РИСКА ИСПОЛЬЗОВАНИЯ СТОРОННИХ БИБЛИОТЕК

Специалисты информационной безопасности компании должны давать разрешение на использование сторонних компонентов для разработчиков. До внедрения Solar appScreener им приходилось проводить ручное изучение каждого компонента или использовать статический анализ. С внедрением SCS Solar appScreener этот процесс стал заметно быстрее. Теперь специалисты ИБ проводят сканирование SBOM-файлов и получают общий рейтинг доверия к компонентам. На основе этого рейтинга принимается решение о запрете или разрешении использования компонента в разработке.

ПРОВЕРКА УНАСЛЕДОВАННОЙ СИСТЕМЫ

В компании более 10 лет использовали унаследованную торговую систему. Анализ исполняемых файлов системы при помощи Solar appScreener выявил функциональность скрытой отправки данных на внешний сервер, после чего его заблокировал межсетевой экран.

КОНТРОЛЬ РАЗРАБОТЧИКОВ

Проанализировав при помощи Solar appScreener выложенное в Google Play мобильное приложение, сотрудники отдела ИБ компании нашли уязвимости, отсутствовавшие в исходном коде, представленном для анализа разработчиками. В ходе расследования выяснилось, что разработчики умышленно предоставляли на анализ урезанную версию кода, чтобы избежать его исправления, которое мог потребовать отдел ИБ компании С, и не потерять премии из-за

задержек релизов. Разработчики обфусцировали скомпилированный код приложения и считали, что об этом никто не узнает – по их мнению, отдел ИБ при всем желании не смог бы восстановить код.

7. СИСТЕМНЫЕ ТРЕБОВАНИЯ

ДЛЯ ОСНОВНОГО СЕРВЕРА СИСТЕМЫ

Системные требования для модулей анализа исходного и бинарного кода языков C/C++, Objective-C и Swift указаны отдельно. Исходный код C/C++ и Objective-C анализируется на операционной системе, на которой возможна успешная сборка кода. Ниже представлены требования для установки на одном узле основного сервера с включенными модулями анализа SAST, DAST, OSA. Требования для установки модулей по отдельности можно найти в инструкции по установке системы.

- Операционная система: Ubuntu 20.04, Ubuntu 22.04, Ubuntu 24.04, Debian 12, CentOS 7, AlmaLinux 8, Red Hat Enterprise Linux 8, Red Hat Enterprice Linux 9, OS Astra SE 1.7.3, RedOS 7, RedOS 8.
- Оперативная память: 64 ГБ.
- Процессор: 12 ядер, 2,5 ГГц.
- Запоминающее устройство: 750 ГБ SSD/SAS (при увеличении количества сканирований требуемый объем увеличивается).
- Права администратора.
- Наличие и доступность пакетов базовых репозиториев для операционной системы.

ДЛЯ МОДУЛЯ АНАЛИЗА ИСХОДНЫХ КОДОВ OBJECTIVE-C

- Операционная система: Apple macOS Mojave и выше.
- Операционная система macOS bigsur x64 и выше + инструмент bear версии от 3.1.2 (возможно установить с помощью утилиты homebrew) при использовании функциональности «Межмодульный анализ»;
- Оперативная память: 16 ГБ.
- Процессор: 4 ядра, 2,2 ГГц.
- Запоминающее устройство: 100 ГБ свободного пространства.
- Отсутствие установленных Java-машин и переменных среды %JAVA_HOME%.
- Сетевое соединение с основным сервером установки.
- Наличие установленных Xcode, clang, cmake, cocoapods.
- Выход в интернет, а также наличие установленных git и/или svn, при необходимости загрузки библиотек из внешних источников.
- Подготовленное окружение, в котором успешно собирается анализируемый проект.

ДЛЯ МОДУЛЯ АНАЛИЗА ИСХОДНЫХ КОДОВ C++ ДЛЯ MICROSOFT WINDOWS

- Операционная система:
  - Windows 10 64-bit: Pro, Enterprise (Build 16299 or later);
  - Windows 10 64-bit Home (version 1903 or higher);
  - Windows 11 64-bit;

  - Windows Server 2019;
  - Windows Server 2022;
- Оперативная память: 16 ГБ.
- Процессор: 4 ядра, 2,2 ГГц.
- Запоминающее устройство: 100 ГБ свободного пространства.
- Установленная Visual Studio или Microsoft Build Tools той же версии, на которой происходит сборка проекта.
- Отсутствие установленных Java-машин и переменных среды %JAVA_HOME%.
- Окружение, в котором успешно собирается проект для анализа.
- Права администратора.

С более подробной информацией можно ознакомиться в инструкции по установке.

8. КОНТАКТНАЯ ИНФОРМАЦИЯ

ТЕЛЕФОНЫ:

+7 (499) 755-07-70 – продажи и общие вопросы

+7 (499) 755-02-20 – техническая поддержка

E-MAIL:

presale.appscreener@rt-solar.ru – продажи и вопросы по сервису

solar@rt-solar.ru – общие вопросы

support.appscreener@rt-solar.ru – техническая поддержка

АДРЕСА:

125009, Москва, Никитский пер., 7, стр. 1

127015, Москва, Вятская ул., 35/4, БЦ «Вятка», 1-й подъезд