СЕКЦИЯ «ЗАЩИТА ИНФОРМАЦИИ»

УДК 004.4:004.056.5

ПРОГРАММНЫЙ КОМПЛЕКС СТАТИЧЕСКОГО АНАЛИЗА УЯЗВИМОСТЕЙ ИСХОДНОГО КОДА НА JAVASCRIPT И TYPESCRIPT

Батуро А.И., студент гр.461401

Белорусский государственный университет информатики и радиоэлектроники[1]
г. Минск, Республика Беларусь

Белоусова Е.С. – канд. техн. наук, доцент

Аннотация. Рассмотрена разработка сервиса статического анализа безопасности для JavaScript и TypeScript на основе парсера tree-sitter, правил по абстрактному синтаксическому дереву и внутрифайлового анализа загрязнения данных с ограниченной межпроцедурностью и пакетной передачей файлов через REST API. Результаты сопоставляются с таксономией CWE; поддерживаются форматы JSON, SARIF 2.1.0 и асинхронный режим выполнения. Описаны архитектура комплекса, состав правил, особенности анализа потоков данных и программный интерфейс. Приведены количественные результаты сопоставления с инструментом Semgrep на внутреннем каталоге синтетических примеров и на внешнем корпусе semgrep-rules/javascript, пояснена методика трёх контуров оценки, сформулированы ограничения и направления развития.

Ключевые слова. статический анализ, JavaScript, TypeScript, абстрактное синтаксическое дерево, анализ загрязнения данных, CWE, SARIF, Semgrep, REST API, оценка качества, безопасность приложений.

Введение. Серверная и клиентская логика современных информационных систем всё чаще реализуется на языках JavaScript и TypeScript: это касается как веб-приложений, так и микросервисов на платформе Node.js. Такой стек ускоряет разработку, но сохраняет фундаментальные риски, связанные с обработкой данных, поступающих из сети: инъекции в языки запросов к базам данных и в командную оболочку, межсайтовый скриптинг, обход ограничений при доступе к файлам, небезопасная десериализация, использование слабых криптографических примитивов и ряд других классов ошибок, систематизированных в перечнях OWASP и таксономии CWE [1, 2].

Обнаружение подобных дефектов на этапе разработки возможно средствами статического анализа, не требующего исполнения программы на реальных входных данных. На практике применяются как обобщённые графовые представления кода с последующим поиском опасных путей [3], так и более лёгкие по вычислительной стоимости схемы, основанные на сопоставлении шаблонов с абстрактным синтаксическим деревом и на ограниченном анализе потоков данных. Важно согласовать точность и полноту анализа с требованиями к времени отклика при интеграции в конвейер непрерывной интеграции и с необходимостью обмена результатами с системами управления дефектами; для последнего широко используется стандарт SARIF [6].

Цель работы – разработать и описать программный комплекс, который для кода на JavaScript и TypeScript выполняет сочетание правил на абстрактном синтаксическом дереве и анализа загрязнения данных с явно заданными границами межпроцедурного охвата, предоставляет доступ по протоколу HTTP в форматах JSON и SARIF и сопоставим по результатам с распространённым открытым анализатором Semgrep [7] в рамках воспроизводимой методики.

Задачи исследования:

1. Обосновать выбор архитектуры на основе инкрементально поддерживаемого парсера и модульных правил.
2. Описать модель источников и стоков для анализа загрязнения и правила сопоставления с типовыми фреймворками.
3. Привести состав программного интерфейса и сценариев развёртывания.
4. Сформулировать методику оценки на внутреннем каталоге и внешнем корпусе;
5. Интерпретировать полученные численные результаты и ограничения комплекса.

Практическая значимость заключается в возможности встраивания сервиса в учебные лаборатории по безопасности программного обеспечения, в опытные конвейеры проверки исходного кода и в сравнительные исследования средств анализа без изменения исходного кода целевого проекта – достаточно HTTP-запроса с текстом файла или пакета файлов.

Архитектура и состав модулей. Программный комплекс реализован как сетевой сервис. Подсистема программного интерфейса построена на фреймворке FastAPI и обеспечивает валидацию входных данных, маршрутизацию запросов и согласование типов ответа. Ядро оркестрации принимает решение о языке входного фрагмента, последовательно или параллельно по политике конфигурации запускает разбор, набор правил и при включённой опции – анализ загрязнения данных, агрегирует находки, вычисляет статистику и при необходимости формирует журнал SARIF. Парсеры JavaScript и TypeScript опираются на библиотеку tree-sitter [5], что даёт устойчивость к синтаксису

современных конструкций языка и возможность точного привязки диагностик к позициям в исходном тексте.

Реестр правил организован по идентификаторам вида JS-*-NNN и покрывает, в частности, инъекции SQL и NoSQL, межсайтовый скриптинг (включая React dangerouslySetInnerHTML), инъекции команд, обход пути, загрязнение прототипа, жёстко зашитые секреты, небезопасную десериализацию, слабую криптографию, открытые перенаправления, обращения к внешним URL по данным пользователя (SSRF), уязвимости к регулярным выражениям с катастрофическим откатом (ReDoS). Каждая находка при наличии локальной базы может быть обогащена ссылкой на идентификатор CWE.

Модуль анализа загрязнения данных реализует сопоставление объявленных источников с объявленными стоками на графе, выводимом из абстрактного синтаксического дерева, с учётом вызовов функций в пределах одного файла и при пакетном запросе – с объединением индекса функций по всем переданным путям. Для асинхронной обработки длительных запросов предусмотрена постановка задачи в очередь и опрос состояния по идентификатору сканирования с использованием брокера сообщений и исполнителей фоновых задач; синхронный режим ограничен по размеру входных данных для предсказуемого времени отклика.

Конвейер обработки фрагмента исходного кода представлен на рисунке 1, связь модулей каталога исходного кода комплекса – на рисунке 2.

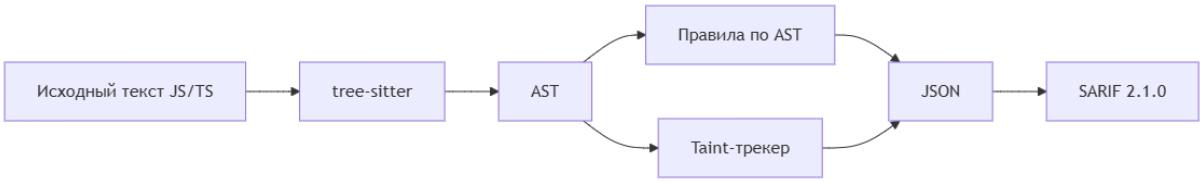

Рисунок 1 – Конвейер обработки исходного кода

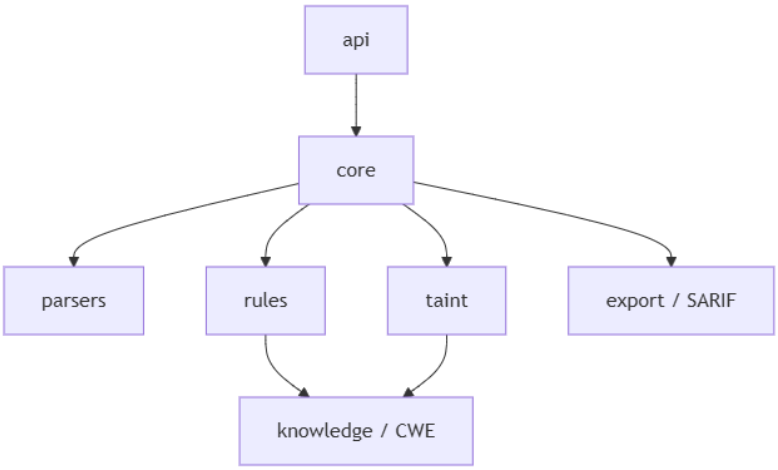

Рисунок 2 – Структура модулей программного комплекса (src/analyzer/)

Программный интерфейс и сценарии использования. Основные конечные точки HTTP следующие: синхронный анализ одного фрагмента исходного кода с телом запроса в формате JSON и опциональным выбором ответа в виде журнала SARIF 2.1.0; синхронный анализ пакета файлов с передачей отображения «путь – текст» для построения сквозного индекса вызываемых функций при анализе загрязнения; асинхронный анализ с возвратом идентификатора и последующим опросом ресурса результатов; перечисление доступных правил; проверка работоспособности сервиса. В ответе синхронного режима помимо списка находок и статистики передаётся перечень идентификаторов правил, при выполнении которых возникло исключение, чтобы оставшиеся правила всё же, отработали.

Типовой сценарий для разработчика – отправка содержимого файла с указанием псевдопути и флага включения анализа загрязнения. Для приложений, разбитых на несколько модулей, предпочтителен пакетный запрос: в него включают как точку входа (например, маршруты), так и файлы с определениями вспомогательных функций, иначе вызовы, разрешённые только по графу импорта, не будут связаны с телами функций в другом файле. Развёртывание возможно, как локально (веб-сервер приложения), так и в контейнерах с выносом брокера и хранилища состояния асинхронных задач.

Методы статического анализа. Разбор исходного текста в абстрактное синтаксическое дерево выполняется парсером tree-sitter; далее правила обходят узлы дерева и срабатывают при выполнении логических условий, характерных для класса уязвимости. Например, для инъекции SQL учитывается конкатенация или шаблонная подстановка пользовательских данных в строку запроса. Такой подход даёт понятные сообщения с привязкой к строке исходного кода и умеренную вычислительную сложность на файл.

Анализ загрязнения данных дополняет правила: он отслеживает распространение «загрязнения» от источников к стокам. В качестве источников используются соглашения об именовании и структуре объектов запроса, характерные для распространённых веб-фреймворков (в стиле Express, Koa, Fastify), а также ряд глобальных объектов среды исполнения. Стоки включают вызовы методов выполнения запросов к базам данных (в том числе сырого SQL в ORM), функции порождения процессов и выполнения строк кода, операции файловой подсистемы, перенаправления ответа и другие конфигурируемые шаблоны. Цепочка необязательного доступа при сопоставлении с шаблонами нормализуется к виду с обычной точкой, чтобы не пропускать эквивалентные вхождения.

Для вызовов вида выполнения запроса с отдельной строкой шаблона и массивом параметров загрязнение, попавшее только в массив параметров, трактуется как привязка к placeholder-ам, а не как встраивание в текст SQL, что снижает число ложных срабатываний на параметризованных API. Подстановки в шаблонных литералах анализируются по узлам подстановки; распространение через вспомогательные функции в одном файле и через возвращаемое значение функции учитывается в пределах ограниченной глубины вложенности вызовов. При передаче нескольких файлов индекс имён функций объединяется, что позволяет связать вызов в одном файле с определением в другом, если оба файла явно включены в запрос.

Результаты сопоставления с инструментом Semgrep и методика оценки. В качестве воспроизводимого эталона сравнения выбран инструмент Semgrep, получивший широкое распространение как в промышленной практике, так и в исследовательской среде благодаря обширным открытым наборам правил [7]. Поскольку однозначная атрибуция корректности срабатывания на уровне отдельной строки кода невозможна без независимой экспертной разметки уязвимостей, в настоящей работе применяется методика, основанная на трёх взаимодополняющих контурах оценки:

1 Внутренний каталог небольших синтетических фрагментов с метками «уязвимый / безопасный» и ожидаемым идентификатором правила; по нему вычисляются истинно положительные, истинно отрицательные, ложноотрицательные и ложноположительные исходы относительно меток каталога (сценарий eval.cli run). Каталог предназначен для регрессионного контроля при изменении кода анализатора, а не как репрезентативная выборка промышленных проектов.

2 Внутренний каталог небольших синтетических фрагментов с параллельным запуском Semgrep и сопоставлением строк находок с учётом файла отображения идентификаторов правил Semgrep на идентификаторы анализатора (eval.cli compare). Совпадения по строкам отражают согласованность при выбранной политике эквивалентности, а не эталонную истину.

3 Обход дерева файлов внешнего корпуса (в примере ниже – каталог javascript из репозитория semgrep-rules), единичный прогон Semgrep по конфигурации из того же дерева и построчное сопоставление с находками анализатора при политике B, когда учитываются все срабатывания обоих инструментов на общем множестве файлов (corpus_paper_eval).

Схема трёх контуров представлена на рисунке 3.

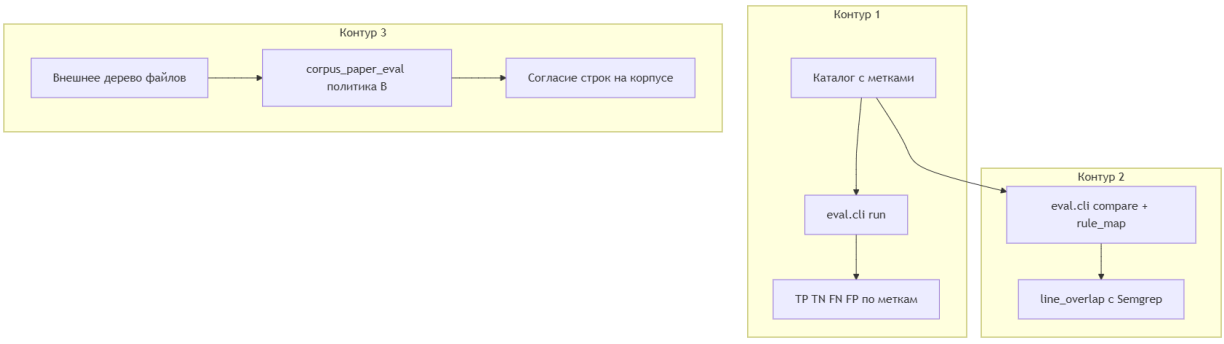

Рисунок 3 – Три контура оценки относительно каталога и корпуса

Опорные значения показателей, зафиксированные в документации репозитория для конфигурации Semgrep 1.156.0 с набором правил semgrep-rules (коммит 28db38e) и коммитом анализатора b59ef6c, приведены ниже. Воспроизведение результатов в последующих работах

требует явного указания версий используемых инструментов, идентификаторов коммитов наборов правил и полной командной строки запуска. Результаты для внутреннего каталога (87 тестовых случаев без метки SKIP, сценарий eval.cli compare) представлены в таблице 1.

Таблица 1 – Результаты по внутреннему каталогу

Показатель	Значение
Исходы по меткам каталога: TP / TN / FN / FP	45 / 33 / 2 / 7
Суммарное число находок: анализатор / Semgrep	76 / 27
Число совпавших пар строк (line_overlap)	16
Суммарно только анализатор / только Semgrep	31 / 7
Кейсы с хотя бы одной строкой Semgrep в карте правил	22
Кейсы с не менее чем одной совпавшей парой строк	15

Метрики TP, TN, FN и FP вычислены относительно меток внутреннего синтетического каталога, а не относительно независимой экспертной разметки. Превышение суммарного числа диагностических сообщений анализатора над числом сообщений Semgrep обусловлено двумя факторами: во-первых, одна строка исходного кода может одновременно активировать правило на дереве разбора и порождать путь загрязнения, что фиксируется как два отдельных срабатывания; во-вторых, наборы правил двух инструментов различаются по охвату классов уязвимостей. Величина пересечения по строкам определяется файлом отображения rule_map.yaml (equivalence_rules) и лишена интерпретационного смысла без предварительного согласования идентификаторов правил Semgrep с соответствующими правилами анализатора. Два зафиксированных ложноотрицательных исхода (FN) – для случаев NOSQL-HARD-VULN-01 и SSRF-HARD-VULN-01 – определяют приоритетные направления доработки: расширение соответствующих правил и уточнение модели стоков.

Результаты для внешнего корпуса (каталог semgrep-rules/javascript, 174 файла, политика B, сценарий corpus_paper_eval) представлены в таблице 2.

Таблица 2 – Результаты по внешнему корпусу

Показатель	Значение
Всего находок Semgrep / анализатора	868 / 219
Совпавшие пары строк	116
Только анализатор / только Semgrep	76 / 613

На корпусе Semgrep даёт существенно большее число срабатываний, поскольку конфигурация включает широкий набор примеров и тестов правил; анализатор нацелен на ограниченный перечень классов CWE-ориентированных правил с иной семантикой. При этом 116 совпадений по строкам показывают пересечение поведения на конкретных местах кода; 76 строк с сообщениями только анализатора и 613 – только Semgrep подчёркивают необходимость раздельной интерпретации и, при желании, комбинирования инструментов в конвейере, а не замены одного другим.

Ограничения. Поддерживаются только языки JavaScript и TypeScript. Анализ статический: не моделируется полное поведение программы при исполнении, не учитываются динамически вычисляемые имена свойств и сложные сценарии рефлексии без явных шаблонов в коде.

Граф зависимостей по import и require не строится автоматически: для разрешения вызова функции, определённой в другом модуле, соответствующий файл должен быть включён в пакетный запрос. Глубина межпроцедурного разбора в анализе загрязнения ограничена константой MAX_INTERPROC_DEPTH = 5. При переносе загрязнения через возвращаемое значение используется первое встреченное в теле функции выражение return. Краткая форма метода в объектном литерале не используется как единственное тело вызываемой функции без отдельного объявления в индексе функций.

Действуют ограничения на размер тела синхронного запроса, на число файлов и на суммарный объём текста в пакете; при задании бюджета времени анализ может быть прерван, а этап анализа загрязнения – пропущен, что должно учитываться при сравнении результатов.

Отдельные правила могут завершиться исключением во время выполнения; они перечисляются в поле failed_rules, остальные правила продолжают работу. Внутренний каталог не заменяет независимый набор промышленных уязвимостей с экспертной разметкой.

Заключение. В работе описан программный комплекс статического анализа уязвимостей для JavaScript и TypeScript, сочетающий правила на абстрактном синтаксическом дереве и анализ загрязнения данных с явно оговорёнными границами, стандартизированные интерфейсы JSON и SARIF и воспроизводимую методику сопоставления с Semgrep на каталоге и на внешнем дереве файлов. По внутреннему каталогу достигнут высокий уровень истинно отрицательных исходов на безопасных фрагментах при умеренном числе ложных срабатываний и двух оставшихся ложно

отрицательных исходах на сложных сценариях NoSQL и SSRF. На корпусе semgrep-rules/javascript инструменты демонстрируют различную плотность сообщений и частичное пересечение по строкам, что ожидаемо при разной полноте набора правил.

Направления дальнейшей работы включают построение или использование графа модулей для автоматического включения зависимостей, расширение правил и стоков для NoSQL и SSRF, уточнение моделей санитизации данных и накопление размеченного корпуса промышленного кода для оценки полноты, и точности независимо от эталона Semgrep.

Список использованных источников:

1. OWASP Top Ten [Электронный ресурс]. – Режим доступа: https://owasp.org/www-project-top-ten/ (дата обращения: 29.03.2026).

2. Common Weakness Enumeration (CWE) [Электронный ресурс]. – Режим доступа: https://cwe.mitre.org/ (дата обращения: 29.03.2026).

3. Yamaguchi F. et al. Modeling and Discovering Vulnerabilities with Code Property Graphs / F. Yamaguchi et al. // Proc. IEEE Symposium on Security and Privacy. – 2014.

4. Livshits B. et al. In Defense of Soundiness: A Manifesto / B. Livshits et al. // Commun. ACM. – 2015. – Т. 58. – № 2.

5. Tree-sitter [Электронный ресурс]. – Режим доступа: https://tree-sitter.github.io/tree-sitter/ (дата обращения: 29.03.2026).

6. SARIF Version 2.1.0 [Электронный ресурс]. – Режим доступа: https://docs.oasis-open.org/sarif/sarif/v2.1.0/ (дата обращения: 29.03.2026).

7. Semgrep: оверлейные правила для статического анализа [Электронный ресурс]. – Режим доступа: https://semgrep.dev/ (дата обращения: 29.03.2026).

8. FastAPI [Электронный ресурс]. – Режим доступа: https://fastapi.tiangolo.com/ (дата обращения: 29.03.2026).

UDC 004.4:004.056.5

SOFTWARE COMPLEX FOR STATIC VULNERABILITY ANALYSIS OF JAVASCRIPT AND TYPESCRIPT SOURCE CODE

Baturo A.I., student gr.461401

Belarusian State University of Informatics and Radioelectronics, Minsk, Republic of Belarus

Belousova E.S. – PhD in Technical Sciences, Associate Professor

Annotation. A static security analysis service for JavaScript and TypeScript is presented, based on the tree-sitter parser, abstract syntax tree rules, and intra-file taint analysis with limited interprocedural scope. Files are transferred in batches via REST API. Findings are mapped to the CWE taxonomy; JSON, SARIF 2.1.0 formats, and asynchronous execution are supported. The system architecture, rule set, data flow analysis features, and API are described. Quantitative comparison with Semgrep is provided on an internal synthetic catalog and the external semgrep-rules/javascript corpus.

Keywords. static analysis, JavaScript, TypeScript, abstract syntax tree, taint analysis, CWE, SARIF, Semgrep, REST API, quality assessment, application security.