УДК 528.71

А. Ф. Миннимухаметова[1]✉, Е. В. Рыжкова[1]

Статический и динамический анализ: два инструмента для идеального кода

[1]Сибирский государственный университет геосистем и технологий,
г. Новосибирск, Российская Федерация
e-mail: minnimuhametovaalina@yandex.ru

Аннотация. В статье приведены результаты исследований методов статического (SAST) и динамического (DAST) анализа кода, их преимущества и ограничения. Актуальность темы связана с ростом уязвимостей в ПО, особенно при использовании сторонних компонентов. Цель работы – показать, что комбинированное применение SAST и DAST повышает безопасность ПО. Для анализа использовались инструменты PVS-Studio (SAST) и OWASP ZAP (DAST). Результаты представлены в сравнительной таблице, где выделены ключевые различия методов. Инструменты PVS-Studio (SAST) и OWASP ZAP (DAST) доказали свою эффективность в выявлении различных классов уязвимостей: от ошибок в исходном коде.В ходе работы был сделан основной вывод, что совместное использование SAST и DAST обеспечивает более надежную защиту ПО. Исследование подчеркивает важность интеграции обоих подходов в процесс разработки.

Ключевые слова: динамический анализ, статический анализ, код

A. F. Minnimuhametova[1]✉, E. V. Ryzhkova[1]

Static and dynamic analysis: two tools for perfect code

[1]Siberian State University of Geosystems and Technologies, Novosibirsk, Russian Federation
e-mail: minnimuhametovaalina@yandex.ru

Abstract. The article presents the results of research on static (SAST) and dynamic (DAST) code analysis methods, their advantages, and limitations. The relevance of this topic is related to the increasing number of vulnerabilities in software, especially when using third-party components. The goal of this work is to demonstrate that the combined use of SAST and DAST enhances software security. The analysis was conducted using the PVS-Studio (SAST) and OWASP ZAP (DAST) tools. The results are presented in a comparative table, highlighting the key differences between the methods. The PVS-Studio (SAST) and OWASP ZAP (DAST) tools have proven to be effective in identifying various classes of vulnerabilities: from errors in the source code. The main conclusion of the work is that the combined use of SAST and DAST provides more reliable software protection. The study highlights the importance of integrating both approaches into the development process.

Keywords: dynamic analysis, static analysis, code

Введение

В современном цифровом мире программное обеспечение стало основой технологического прогресса, управляя критически важными системами – от финансовых операций до медицинского оборудования и промышленных процессов [1]. Однако по мере усложнения программных систем растут и связанные с ними риски безопасности [2].

Актуальные исследования показывают тревожную статистику:

– более 50 % веб-приложений содержат уязвимости высокой и средней степени опасности [3];

– в 91 % случаев внутреннего тестирования злоумышленники успешно получали контроль над доменами и доступ к критическим системам [3].

Серьезным вызовом безопасности стало повсеместное использование сторонних библиотек и компонентов. Заимствованный код часто содержит скрытые уязвимости и недокументированные возможности (НДВ), которые сложнее обнаружить, чем дефекты собственного кода. Особую опасность представляют:

– умышленно внедренные уязвимости в популярных библиотеках [4];

– поддельные пакеты в репозиториях (typosquatting атаки);

– устаревшие версии компонентов с известными уязвимостями.

Для противодействия этим угрозам применяется тестирование безопасности приложений (AST), которое позволяет выявлять уязвимости на ранних этапах разработки и контролировать качество сторонних компонентов [5, 6].

Эффективное AST требует комплексного подхода, сочетающего статический (SAST) и динамический (DAST) анализ [6]:

– SAST выявляет уязвимости в исходном коде до запуска приложения;

– DAST тестирует работающую систему, имитируя действия злоумышленников.

Цель исследования – рассмотреть и сравнить два ключевых подхода к анализу кода – статический (SAST) и динамический (DAST), продемонстрировать их важность в создании безопасного и качественного программного обеспечения, а также показать, как их совместное использование способствует достижению «идеального кода» – надежного, эффективного и свободного от уязвимостей.

В работе будут рассмотрены:

– терминология статического и динамического анализов;

– принципы и инструменты каждого метода;

– типы обнаруживаемых уязвимостей инструментами;

– преимущества комбинированного подхода.

Поставленные задачи:

– провести сравнительный анализ SAST и DAST – оценка преимуществ и ограничений каждого подхода;

– провести практическое тестирование кодов с применением инструментов статического и динамического анализов.

Результаты помогут специалистам осознать, что для разработки безопасного ПО важно применять оба подхода в комплексе.

Методы и материалы

Статический анализ кода (SAST – Static application security testing) – это процесс выявления недочетов, ошибок и потенциальных уязвимостей в исходном коде программ [7].

SAST моделирует работу приложения, анализируя, как исходный код, библиотеки, платформы и компоненты взаимодействуют между собой и как они будут функционировать при запуске. Этот метод также называют тестированием по принципу «белого ящика», поскольку анализ проводится «изнутри» приложения, и о его структуре известно заранее [8].

Динамический анализ кода (DAST – Dynamic Application Security Testing) – это автоматизированное тестирование, которое проводится в реальном времени на запущенном приложении или веб-сервисе для выявления уязвимостей безопасности [9]. Механизмы DAST имитируют атаки на работающее приложение, чтобы обнаружить такие проблемы, как межсайтовый скриптинг (XSS), подделка межсайтовых запросов (CSRF), SQL-инъекции и небезопасные конфигурации.

Поскольку инструменты DAST работают без доступа к исходному коду, этот метод также называют тестированием по принципу «черного ящика» [10].

Традиционные инструменты DAST анализируют только внешнее поведение приложения, например, его пользовательский интерфейс и веб-службы, не затрагивая внутреннюю логику работы [11]. Это ограничивает их способность обнаруживать уязвимости, связанные с внутренними компонентами приложения.

В ходе исследования для анализа безопасности кода были выбраны инструменты статического (SAST) и динамического (DAST) анализа:

– PVS Studio – статический анализатор, выявляющий ошибки на этапе разработки [12];

– OWASP ZAP – инструмент динамического тестирования, имитирующий атаки на работающее приложение [13].

PVS Studio проводит глубокий анализ исходного кода, обнаруживая:

– ошибки памяти: утечки, разыменование нулевых указателей [12];

– логические дефекты: выход за границы массива, некорректные условия [12];

– потенциальные уязвимости: неправильная обработка данных, ошибки синхронизации;

– оптимизационные проблемы: избыточные вычисления, неэффективные алгоритмы.

Преимущество SAST – раннее обнаружение ошибок до запуска программы, что снижает затраты на исправление [11].

OWASP ZAP тестирует приложение в реальном времени, выявляя:

– уязвимости веб-приложений: SQL-инъекции, XSS, CSRF;

– проблемы аутентификации и авторизации;

– небезопасные настройки сервера и API [14].

DAST полезен для проверки поведения системы в условиях, близких к реальным.

Результаты

В ходе работы была построена сравнительная таблица 1 двух методов анализа – статического и динамического.

Таблица 1

Сравнительная таблица SAST и DAST

Критерий	Статический анализ (SAST)	Динамический анализ (DAST)
Принцип работы	Анализ исходного кода без его выполнения	Тестирование работающего приложения в реальном времени
Этап применения	На стадии разработки	На стадии тестирования/эксплуатации
Полный обзор кодовой базы	Анализирует весь исходный код	Проверяет только исполняемые пути
Анализ неисполняемого кода	Находит неиспользуемые функции	Использует только код, который выполняется в тестах
Автоматизация	Легко интегрируются	Требуется подготовка тестовой среды
Типы ошибок	Выявляет синтаксические и логические ошибки	Обнаруживает ошибки времени выполнения
Время выполнения	От нескольких минут до нескольких часов, так как не требует выполнения программы	От нескольких часов до нескольких дней, так как зависит от покрытия тестами
Ложные срабатывания	Инструменты могут ошибочно указывать на несуществующие уязвимости	Исключены

Работа PVS-Studio представлена на рисунке 1, где продемонстрирован пример кода, содержащий две ошибки и показывающий возможности PVS-Studio в анализе потока данных.

Во-первых, код содержит опечатку, из-за которой в переменную «a» дважды подряд записываются адреса двух выделенных буферов памяти. Это приводит к потере одного указателя, и анализатор предупреждает об утечке памяти. Еще одно последствие опечатки – разыменование нулевого указателя внутри функции «use» [15].

Во-вторых, память была выделена с помощью оператора «new», а освобождается с помощью вызова функции «free». Это может привести к утечке ресурсов, также будет некорректный вывод внутренних указателей [15].

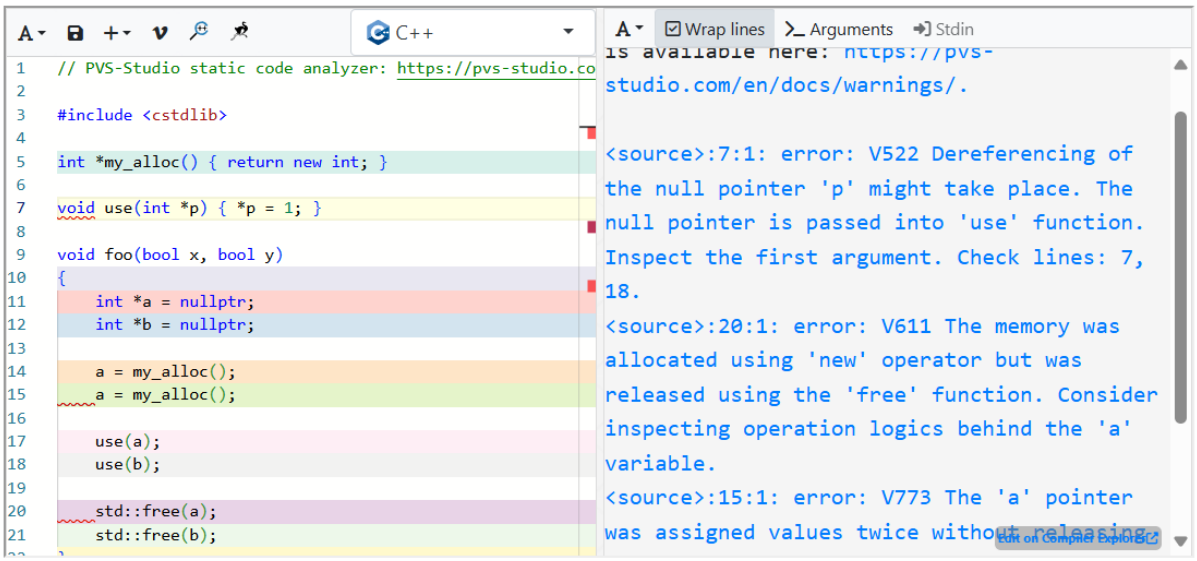

Рис. 1. Результат работы инструмента статического анализа PVS Studio

Результат работы инструмента динамического анализа – Owasp Zap представлен на рис 2.

Изображен фрагмент отчета динамического анализатора OWASP ZAP, демонстрирующий три критичных проблемы безопасности веб-приложения.

Первая ошибка – это утечка SQL-кода. В ходе тестирования обнаружено, что сервер при обработке POST-запроса к /login возвращает фрагмент SQL-запроса: SELECT username, password FROM users WHERE username. Это свидетельствует о двух потенциальных проблемах:

– раскрытие структуры базы данных, что упрощает проведение SQL-инъекций;

– возможная утечка исходного кода через сообщения об ошибках.

Анализатор также указывает на CWE-540 (уязвимость раскрытия конфиденциальной информации в исходном коде) и рекомендует:

– запретить вывод SQL-запросов в ошибках;

– проверить конфигурацию сервера на случайную отдачу исходников.

Вторая ошибка – подробные ошибки сервера. При запросе к тому же эндпоинту сервер возвращает HTTP 500 с полным стеком вызова, что:

– раскрывает внутреннюю структуру приложения;

– может содержать чувствительную информацию (пути к файлам, версии компонентов).

Третья – отсутствие security-заголовков. Отчет отмечает отсутствие критически важных заголовков безопасности.

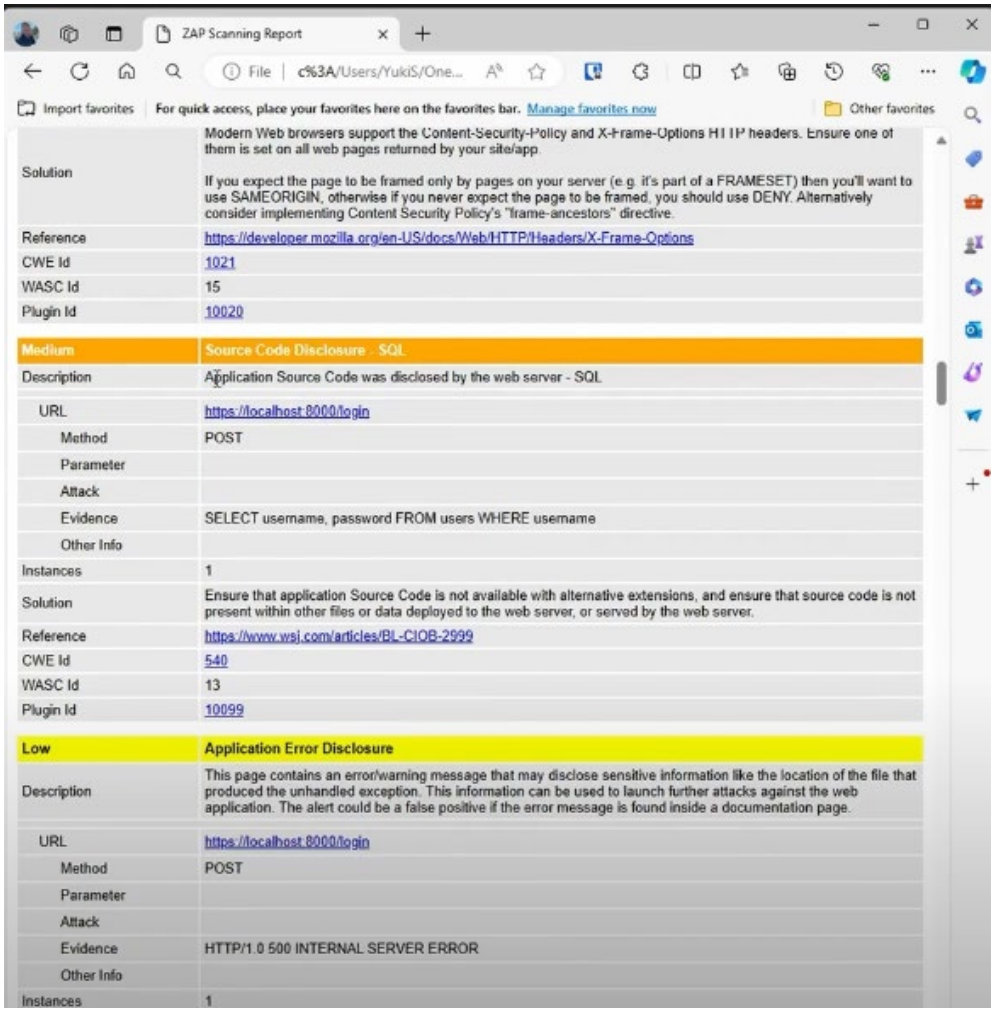

Рис. 2. Отчет инструмента динамического анализа OWASP Zap

Заключение

Проведенное исследование продемонстрировало, что комбинированное применение статического (SAST) и динамического (DAST) анализа значительно повышает уровень безопасности программного обеспечения [17-20]. Инструменты PVS-Studio (SAST) и OWASP ZAP (DAST) доказали свою эффективность в выявлении различных классов уязвимостей: от ошибок в исходном коде.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Касперски К. Технологии анализа защищенности программного обеспечения. – М.: ДМК Пресс, 2019. – 320 с.

2. Бахадуров Б., Баллыев А., Байрамова Дж. Методы тестирования безопасности программного обеспечения: от статического анализа до пентестинга // Вестник науки №10 (79) том 1. Октябрь 2024. С 399-402.

3. Чепелев Д.А., Боровиков А.С. Современные методы статического анализа кода // Программная инженерия. – 2020. – № 11(3). – С. 45-52.

4. Аветисян А. И., Белеванцев А. А., Чукляев И. И. Технологии статического и динамического анализа уязвимостей программного обеспечения // Вопросы кибербезопасности. 2014 № 3 (4).С 20-26.

5. Тестирование программного обеспечения. Фундаментальные концепции менеджмента бизнес-приложений: Пер. с англ./Сэм Канер, Джек Фолк, Енг Кек Нгуен. — К.: Издательство «ДиаСофт», 2001 — 544 с.

6. Аветисян А.И. Современные методы статического и динамического анализа программ для решения приоритетных проблем программной инженерии : автореф, дис. Доктора физ.-мат. наук: 05.13.11/ Аветисян Арютюн Ишханович. - М., 2011 - 36 с

7. Петров К.Л. Интеграция SAST и DAST в DevOps-процессы // Труды конф. "Инфофорум-2023". – М., 2023. – С. 112-118.

8. .PVS-Studio. Статьи по статическому анализу кода. – 2023.

9. Кузнецов М.А. Методы динамического тестирования веб-приложений // Кибербезопасность и программирование. – 2021.

10. Гультяев А.К. Тестирование и отладка программного обеспечения. – СПб.: БХВ-Петербург, 2022. – 416 с.

11. Шмаков Р.В. Методы и инструменты тестирования безопасности ПО. – М.: Солон-Пресс, 2021.

12. PVS-Studio. Документация по статическому анализу кода. – 2023.

13. OWASP ZAP. User Guide. – 2023

14. OWASP. Руководство по тестированию веб-безопасности, 2020г. Практическое пособие

15. PVS-Studio. Online Examples (C, C++)

16. Howard M., Lipner S. The Security Development Lifecycle. – Microsoft Press, 2018. – 256 p.

17. Никитин О.И., Воропаев А.В., Чукляев Е.И. Применение программных технологий анализа поиска уязвимостей в интересах обеспечения защищенности разрабатываемого программного обеспечения // Наукоемкие технологии в космических исследованиях Земли. 2014 №5. С 54-58.

18. Петренко С.А., Курбатов В.А. Безопасность программного обеспечения. – М.: Инфра-М, 2020. – 256 с.

19. Иванов П.С., Смирнова Е.А. Сравнительный анализ SAST и DAST инструментов // Информационная безопасность. – 2022. – Т. 24, № 4. – С. 67-75.

20. ГОСТ Р 56939-2024. Защита информации. Разработка безопасного программного обеспечения. Общие требования