Оригинальная статья / Original article

https://doi.org/10.21869/2223-1536-2026-16-2-167-181

УДК 004.056.52

Сравнительное исследование инструментов статического анализа PHP-кода

Я. В. Огневой[1], А. Г. Спеваков[1] ✉, И. В. Калуцкий[1], П. А. Клименко[2]

[1] Московский политехнический университет
ул. Большая Семёновская, д. 38, г. Москва 107023, Российская Федерация

[2] Курский филиал Финансового университета при Правительстве Российской Федерации
ул. Ломоносова, д. 3, г. Курск 305016, Российская Федерация

✉ e-mail: aspev@yandex.ru

Резюме

Целью исследования *является сравнительный анализ инструментов статического анализа PHP-кода для оценки их эффективности в выявлении ошибок, потенциальных уязвимостей и проблем типизации на ранних этапах разработки веб-приложений. Особое внимание уделяется определению практической применимости рассматриваемых решений в проектах различного масштаба, а также их влиянию на повышение качества программного обеспечения, снижение числа дефектов в коде и минимизацию рисков безопасности.*

Методы. *В исследовании использован метод сравнительного анализа, основанный на тестировании инструментов на наборе типовых сценариев, отражающих распространённые ошибки и уязвимости PHP-приложений. Оценка инструментов проводилась по критериям полноты выявления ошибок, точности диагностики, гибкости настройки правил, удобства интеграции в процесс разработки, производительности и потребления ресурсов. Дополнительно был выполнен анализ документации и возможностей конфигурации. Все эксперименты проводились многократно для обеспечения статистической достоверности результатов.*

Результаты. *В ходе исследования выявлены различия в глубине анализа, строгости проверки типов и механизмах настройки правил. Установлено, что инструменты демонстрируют различную чувствительность к логическим ошибкам, несоответствиям типов и потенциально небезопасным конструкциям. Определены их сильные и слабые стороны в контексте использования в малых и крупных проектах.*

Заключение. *Результаты подтверждают эффективность применения инструментов статического анализа как средства повышения качества и безопасности PHP-кода. Выбор конкретного решения должен основываться на требованиях проекта, уровне строгости анализа и особенностях процесса разработки. Регулярное использование таких инструментов позволяет существенно снизить риск появления дефектов и уязвимостей, повышая надёжность и устойчивость веб-приложений.*

Ключевые слова: *статический анализ кода; безопасность PHP; анализ уязвимостей; качество программного обеспечения; безопасная разработка; автоматизированные инструменты анализа.*

***Конфликт интересов**: Авторы декларируют отсутствие явных и потенциальных конфликтов интересов, связанных с публикацией настоящей статьи.*

Для цитирования: Сравнительное исследование инструментов статического анализа PHP-кода / Я. В. Огневой, А. Г. Спеваков, И. В. Калуцкий, П. А. Клименко // Известия Юго-Западного государственного университета. Серия: Управление, вычислительная техника, информатика. Медицинское приборостроение. 2026. Т. 16, № 2. С. 167–181. https://doi.org/ 10.21869/2223-1536-2026-16-3-167-181.

Поступила в редакцию 08.04.2026 *Подписана в печать 07.05.2026* *Опубликована 30.06.2026*

Comparative study of PHP code static analysis tools

Yaroslav V. Ognevoy[1], Alexander G. Spevakov[1] ✉, Igor V. Kalutskiy[1], Pavel A. Klimenko[2]

[1] Moscow Polytechnic University
38 Bolshaya Semyonovskaya Str., Moscow 107023, Russian Federation

[2] Kursk Branch of the Financial University under the Government of the Russian Federation
3 Lomonosova Str., Kursk 305016, Russian Federation

✉ e-mail: aspev@yandex.ru

Abstract

***The purpose of the research** is a comparative analysis of static PHP code analysis tools to assess their effectiveness in identifying errors, potential vulnerabilities, and typing problems in the early stages of web application development. Particular attention is paid to determining the practical applicability of the solutions under consideration in projects of various scales, as well as their impact on improving software quality, reducing the number of defects in the code and minimizing security risks.*

***Methods.** The study uses a comparative analysis method based on testing tools on a set of typical scenarios reflecting common errors and vulnerabilities of PHP applications. The tools were evaluated according to the criteria of completeness of error detection, accuracy of diagnosis, flexibility of rule configuration, ease of integration into the development process, productivity and resource consumption. Additionally, an analysis of documentation and configuration options was performed. All experiments were carried out repeatedly to ensure the statistical reliability of the results.*

***Results.** The study revealed differences in the depth of analysis, the rigor of type checking, and the mechanisms for configuring rules. It has been found that the tools exhibit varying sensitivity to logical errors, type inconsistencies, and potentially unsafe designs. Their strengths and weaknesses have been identified in the context of use in small and large projects.*

***Conclusion.** The results confirm the effectiveness of using static analysis tools as a means of improving the quality and security of PHP code. The choice of a specific solution should be based on the requirements of the project, the level of rigor of the analysis and the specifics of the development process. Regular use of such tools can significantly reduce the risk of defects and vulnerabilities, increasing the reliability and stability of web applications.*

***Keywords:** static code analysis; PHP security; vulnerability analysis; software quality; secure development; automated analysis tools.*

***Conflict of interest:** The Authors declare the absence of obvious and potential conflicts of interest related to the publication of this article.*

For citation: Ognevoy Y.V., Spevakov A.G., Kalutskiy I.V., Klimenko P.A. Comparative study of PHP code static analysis tools. *Izvestiya Yugo-Zapadnogo gosudarstvennogo universiteta. Serija: Upravlenie, vychislitel'naja tekhnika, informatika. Meditsinskoe priborostroenie = Proceedings of the Southwest State University. Series: Control, Computer Engineering, Information Science. Medical Instruments Engineering.* 2026;16(2):167–181. (In Russ.) https://doi.org/10.21869/2223-1536-2026-16-2-167-181.

Received 08.04.2026 *Accepted 07.05.2026* *Published 30.06.2026*

Введение

PHP остаётся одной из наиболее востребованных платформ для создания веб-приложений. Вместе с ростом сложности проектов увеличивается вероятность появления ошибок, которые могут негативно повлиять на безопасность системы. Наиболее уязвимыми оказываются участки исходного кода, в которых разработчик допускает логические неточности, неверно обрабатывает данные или нарушает принципы безопасного программирования. Поэтому задача своевременного выявления подобных проблем имеет большое значение.

Практика показывает, что проверка качества и безопасности кода может выполняться двумя основными способами – динамическим и статическим анализом. Динамический анализ основывается на исполнении программы, в то время как статический позволяет выявлять потенциальные ошибки без запуска кода, на этапе разработки. Эта особенность делает статический анализ ценным для среды PHP-разработки, где ошибки нередко проявляются во время выполнения и могут оставаться незамеченными [1].

Современные инструменты статического анализа для PHP (такие как PHPStan, Psalm и Phan) предоставляют широкий набор механизмов для выявления типичных ошибок, нарушений типизации, некорректных зависимостей и потенциальных уязвимостей [2]. Несмотря на общую цель, каждый из них использует собственные подходы и имеет различную глубину анализа, что приводит к различиям в полноте обнаружения проблем и количестве ложных предупреждений [3]. Это делает актуальным проведение сравнительного исследования, позволяющего оценить сильные и слабые стороны данных инструментов на основе единых критериев и типичных сценариев ошибок.

Материалы и методы

Верификация программного обеспечения без запуска анализируемых программ

Статический анализ кода позволяет проводить верификацию программного обеспечения без запуска анализируемых программ. Вместо выполнения кода, инструменты статического анализа исследуют исходный код на предмет соответствия заданным правилам, шаблонам уязвимостей и потенциально опасным конструкциям. Это позволяет выявлять ошибки и уязвимости безопасности на ранних этапах разработки [4].

Рассмотрим обобщённый процесс работы инструментов статического анализа (рис. 1).

В данном исследовании рассматриваются три наиболее распространённых средства статического анализа с открытым исходным кодом для PHP. Ниже приведено краткое описание каждого из них.

PHPStan – один из наиболее распространённых статических анализаторов; работает на основе строгой проверки типов и анализа структуры программы. Использует собственную иерархию уровней строгости (от 0 до 9), где каждый уровень добавляет более детализированные проверки. PHPStan помогает обнаруживать ошибки типизации, недостижимый код, некорректные сигнатуры методов, ошибки в работе с массивами и данные, которые потенциально могут привести к логическим уязвимостям. Имеет модульную

архитектуру и поддержку расширений, легко интегрируется в существующие проекты и CI/CD-пайплайны [5].

Psalm – мощный статический анализатор, часто рассматриваемый в роли альтернативы PHPStan. Известен глубоким анализом типов, поддержкой аннотаций и встроенными механизмами проверки кода, помогающими выявлять потенциальные проблемы безопасности [6]. В Psalm реализована система уровней строгости, где каждый уровень включает всё более детализированные проверки: от базовой типизации и поиска ошибок до сложного анализа логики программы и потенциально небезопасных операций. Такая структура позволяет гибко настраивать глубину анализа под конкретный проект и требования к качеству кода [7].

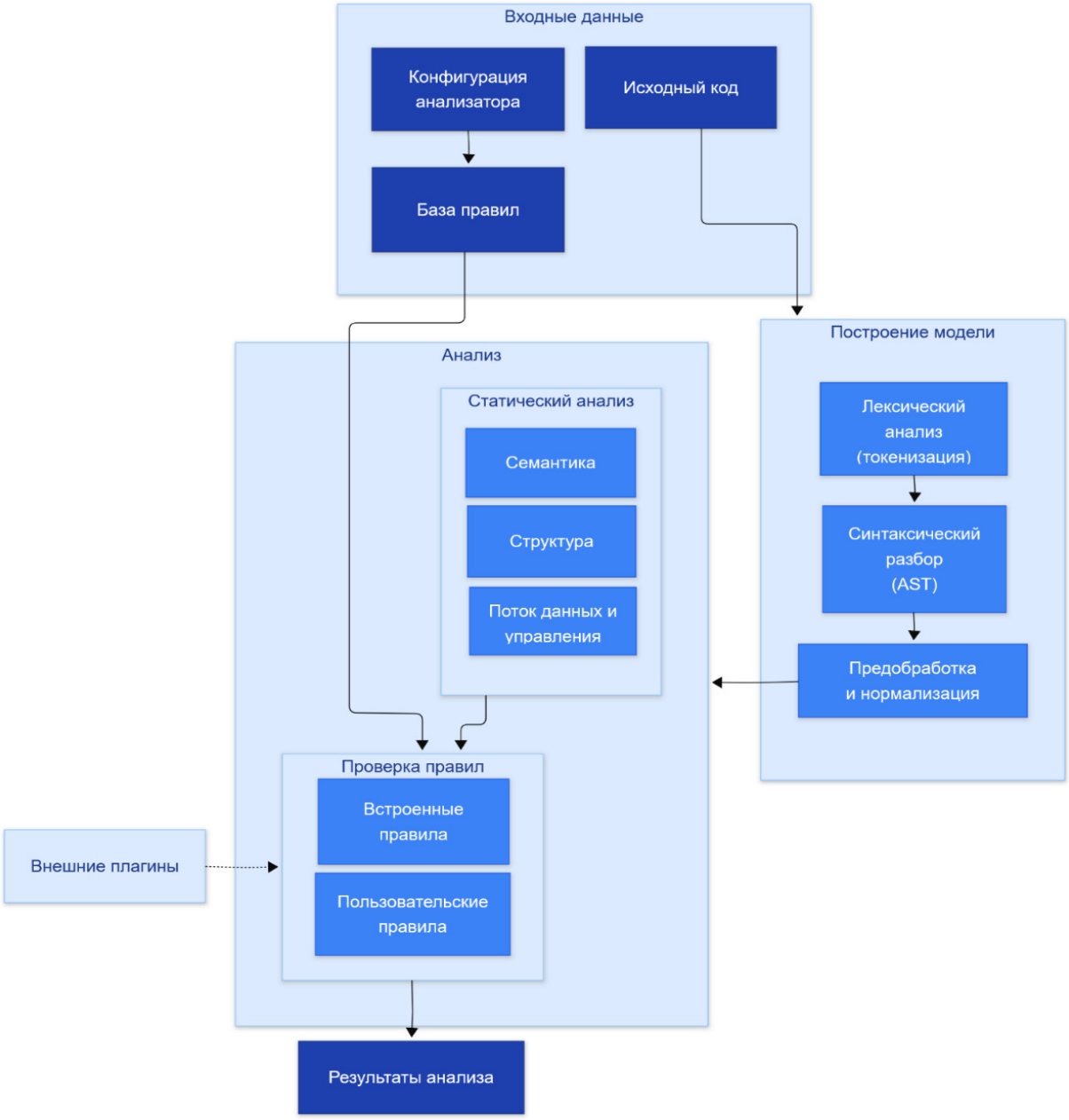

Рис. 1. Этапы статического анализа кода

Fig. 1. The stages of static code analysis

Phan один из первых статических анализаторов, ориентированных на современные версии PHP и систему типов, введённую в PHP 7+. Способен выявлять широкий спектр ошибок: использование неопределённых переменных и методов, несоответствие типов, ошибки в наследовании и типовые нарушения. Phan часто обнаруживает дополнительные логические несоответствия, однако может быть более склонен к ложным предупреждениям по сравнению с PHPStan и Psalm. Поддерживает плагины, позволяющие расширять его возможности, и интеграцию с редакторами кода.

Сравнительный анализ инструментов статического анализа PHP-кода

В данном разделе представлено сравнительное исследование трёх средств статического анализа для PHP: PHPStan, Psalm и Phan. Целью исследования была оценка их возможностей по обнаружению различных категорий уязвимостей и дефектов кода [8], классифицированных согласно Common Weakness Enumeration (CWE) [9].

При настройке анализаторов был применен единый подход с использованием стандартных конфигураций: для PHPStan установлен уровень 5, для Psalm – уровень 3, для Phan использована конфигурация по умолчанию. Данные настройки отражают типичный сценарий использования инструментов в реальных проектах без экстремальных требований к строгости анализа.

Для формирования списка возможностей анализаторов был составлен набор категорий уязвимостей и дефектов. Из обширного списка CWE были отобраны наиболее характерные для экосистемы PHP категории с акцентом на те из них, которые представляют наибольший риск для веб-приложений (рис. 2). Оценка возможностей каждого средства проводилась путём тщательного изучения их официальной документации и репозиториев.

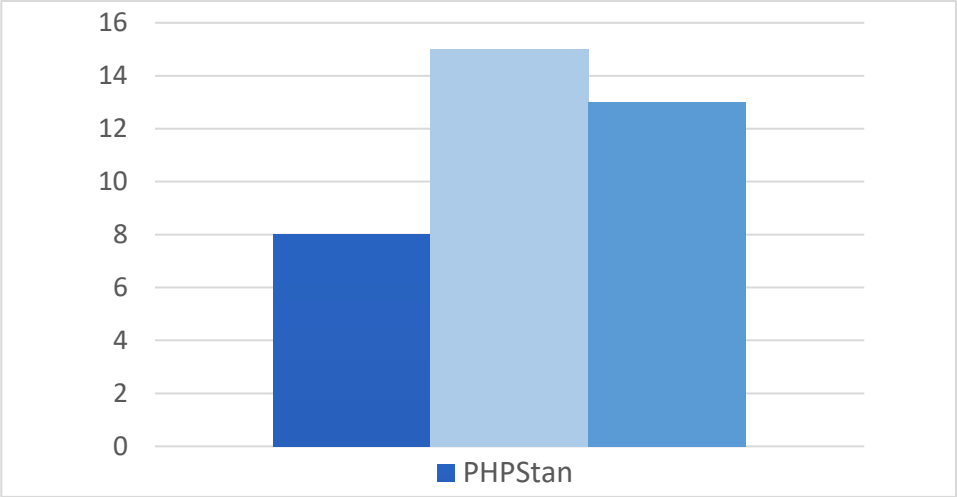

Рис. 2. Сравнительная характеристика инструментов по коэффициенту обнаружения уязвимостей

Fig. 2. Comparative effectiveness of the tools by vulnerability detection rate

Результаты и их обсуждение

Рассмотрим результаты оценки (табл. 1).

Для количественной оценки эффективности был рассчитан коэффициент обнаружения каждого инструмента – отношение количества поддерживаемых категорий уязвимостей к их общему количеству в выбранном наборе. Результаты представлены ниже (табл. 2).

Таблица 1. Сравнительный анализ инструментов статического анализа PHP-кода [10]

Table 1. Comparative analysis of PHP static analysis tools [10]

Категория уязвимости (CWE)	Описание	PHPStan	Psalm	Phan
CWE-78	Инъекция команд ОС	✗	✓*	✓
CWE-79	Межсайтовый скриптинг (XSS)	✗	✓*	✓
CWE-89	SQL-инъекция	✗	✓*	✓
CWE-327	Использование ненадёжного криптоалгоритма	✗	✓*	✗
CWE-369	Divide By Zero (Деление на ноль)	✓	✓	✓
CWE-390	Detection of Error Condition Without Action (Игнорирование ошибок)	✓	✓	✓
CWE-401	Memory Leak (Утечка памяти)	✗	✗	✗
CWE-476	NULL Pointer Dereference (Обращение к нулевому указателю)	✓	✓	✓
CWE-522	Insufficiently Protected Credentials (Недостаточная защита учетных данных)	✗	✓*	✗
CWE-561	Dead Code (Мёртвый код)	✓	✓	✓
CWE-563	Assignment to Variable without Use (Присвоение значения неиспользуемой переменной)	✓	✓	✓
CWE-570	Expression is Always False (Выражение всегда ложно)	✓	✓	✓
CWE-571	Expression is Always True (Выражение всегда истинно)	✓	✓	✓
CWE-591	Sensitive Data Storage in Improperly Locked Memory (Недостаточная блокировка)	✗	✗	✗
CWE-690	Unchecked Return Value (Непроверенное возвращаемое значение)	✓	✓	✓
CWE-772	Missing Release of Resource (Неосвобожденный ресурс)	✗	✓*	✗
CWE-776	XML Injection (XXE)	✗	✗	✓
CWE-835	Infinite Loop (Бесконечный цикл)	✗	✓	✓

Примечание. Знаком «✓» отмечены поддерживаемые категории уязвимостей, а знаком «✗» – неподдерживаемые, знаком «✓*» – категории, требующие дополнительной настройки или установки плагинов для полноценной работы.

Таблица 2. Коэффициент обнаружения уязвимостей

Table 2. Vulnerability detection rate

Инструмент	Количество обнаруженных категорий	Коэффициент обнаружения
PHPStan	8	8/18 (44,4%)
Psalm	15*	15/18 (83,3%)
Phan	13	13/18 (72,2%)

Для практической оценки эффективности анализаторов был использован специально разработанный тестовый PHP-проект, содержащий код с известными уязвимостями [11]. Проект состоит из 22 файлов, в которые было преднамеренно внедрено 75 уязвимостей различных типов, соответствующих рассмотренным ранее категориям CWE. Такой подход позволил создать среду для объективного сравнения возможностей.

На основе анализа этого набора была составлена детальная статистики для каждого инструмента. В таблицах 3–5 представлен полный перечень всех типов предупреждений [12], сгенерированных при анализе тестового проекта, что позволяет оценить характер и специфику обнаруживаемых дефектов [13].

Таблица 3. Детальная статистика предупреждений PHPStan

Table 3. Detailed statistics of PHPStan warnings

Кол.	Описание предупреждения	Идентификатор
8	Unreachable statement - code above always terminates.	deadCode.unreachable
4	Method should return ... but returns ...	return.type
4	Offset ... does not exist on ...	offsetAccess.notFound
3	Comparison operation ">=" between ... is always true.	greaterOrEqual.alwaysTrue
3	Call to method ... on a separate line has no effect.	method.resultUnused
2	Result of && is always false.	booleanAnd.alwaysFalse
2	Result of || is always true.	booleanOr.alwaysTrue
2	Parameter ... of method ... expects ..., ... given.	argument.type
2	Variable ... on left side of ?? always exists and is not nullable.	nullCoalesce.variable
2	Variable ... might not be defined.	variable.undefined
2	Call to function ... with ... will always evaluate to true.	function.alreadyNarrowedType
2	Variable ... in empty() always exists and is not falsy.	empty.variable
2	Strict comparison using !== between ... will always evaluate to true.	notIdentical.alwaysTrue
2	Comparison operation "<" between *NEVER* and ... results in an error.	smaller.invalid
1	Method ... is unused.	method.unused
1	Path in include() ... is not a file or it does not exist.	include.fileNotFound
1	While loop condition is always true.	while.alwaysTrue
1	Cannot call method ... on null.	method.nonObject
1	Comparison operation "<" between ... is always false.	smaller.alwaysFalse
1	Comparison operation ">" between *NEVER* and ... results in an error.	greater.invalid
1	Comparison operation "==" between *NEVER* and ... results in an error.	equal.invalid
1	Comparison operation ">" between ... is always false.	greater.alwaysFalse

Таблица 4. Детальная статистика предупреждений Psalm

Table 4. Detailed statistics of Psalm warnings

Кол.	Описание предупреждения	Идентификатор
10	Condition is always true or false	RedundantCondition
8	Type for variable is never of type	TypeDoesNotContainType
4	The declared return type is incorrect	InvalidReturnType
4	The inferred type does not match the declared return type	InvalidReturnStatement
2	Argument expects string, but int provided	InvalidScalarArgument
2	Cannot access value on empty array variable	EmptyArrayAccess
2	Possibly undefined variable	PossiblyUndefinedVariable
2	Type for variable does not contain null	TypeDoesNotContainNull
2	Cannot access array value on null variable	NullArrayAccess
1	Unsafe shell_exec	ForbiddenCode
1	Cannot find file to include	MissingFile
1	Cannot call method on null value	NullReference
1	Cannot get property on null variable	NullPropertyFetch

Таблица 5. Детальная статистика предупреждений Phan

Table 5. Detailed statistics of Phan warnings

Кол.	Описание предупреждения	Идентификатор
4	Method has no return type and will inconsistently return or not return	PhanPluginInconsistentReturnMethod
3	Returning type does not match declared return type	PhanTypeMismatchReturn
2	Suspicious array access to variable of type null	PhanTypeArraySuspiciousNull
2	Argument type does not match the declared parameter type	PhanTypeMismatchArgument
2	Invalid offset of array type array{}	PhanTypeInvalidDimOffset
2	Variable is possibly undeclared	PhanPossiblyUndeclaredVariable
2	Call to deprecated function	PhanDeprecatedFunctionInternal
1	Missing required file	PhanMissingRequireFile
1	Unreachable statement detected	PhanPluginUnreachableCode
1	Call to method on non-class type null	PhanNonClassMethodCall
1	Returning type does not match declared return type (probably real)	PhanTypeMismatchReturnProbablyReal

Для сравнительной оценки производительности инструментов был разработан PHP-скрипт, осуществляющий замер времени выполнения (рис. 3) и

потребления памяти (рис. 4) при идентичных условиях [14]. Для повышения достоверности результатов каждый анализатор запускался 10 раз, после чего были вычислены средние значения метрик [15]. Скрипт автоматически запускал каждый анализатор и фиксировал метрики производительности.

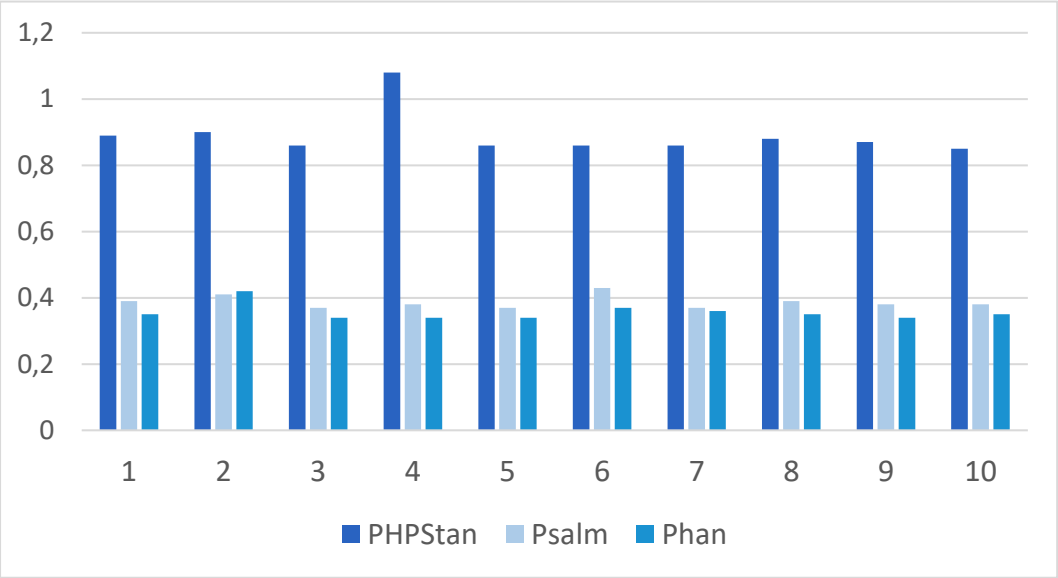

Рис. 3. Сравнение времени выполнения инструментов статического анализа

Fig. 3. Comparison of execution time of static analysis tools

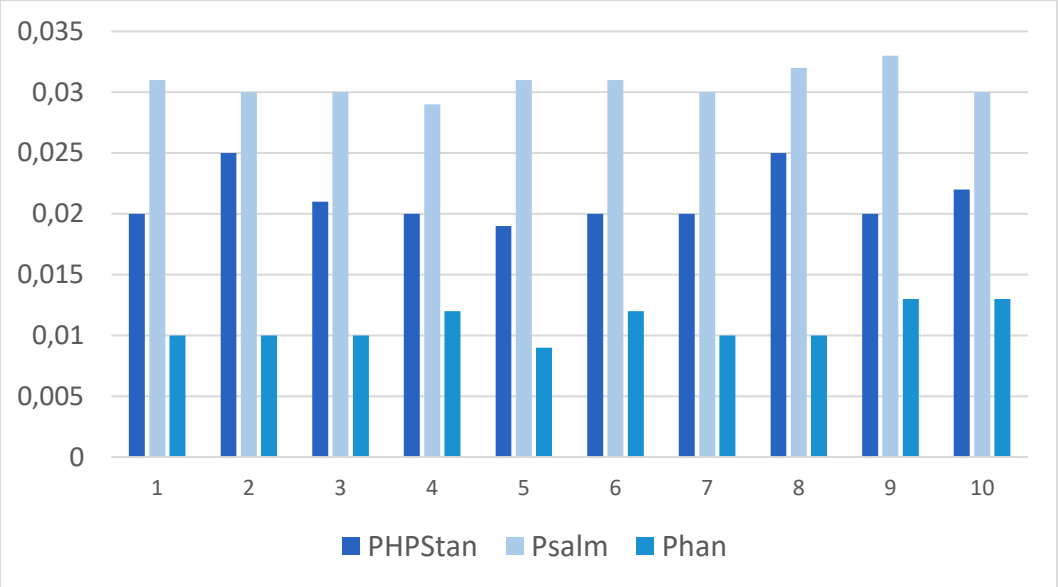

Рис. 4. Сравнение показателя потребления памяти инструментов статического анализа

Fig. 4. Comparison of memory consumption of static analysis tools

Сравнительный анализ инструментов статического анализа PHP-кода [16] позволил выявить различия в их возможностях обнаружения уязвимостей и производительности (табл. 6).

Таблица 6. Сравнение производительности инструментов статического анализа (средние значения за 10 запусков)

Table 6. Performance comparison of static analysis tools (average values over 10 runs)

Инструмент	Время анализа, с	Использование памяти, МВ	Количество файлов	Количество проблем
PHPStan	0,89 ± 0,07	0,021 ± 0,002	22	48
Psalm	0,39 ± 0,02	0,031 ± 0,001	22	40
Phan	0,35 ± 0,02	0,011 ± 0,002	22	21

В части обнаружения уязвимостей результаты (см. табл. 1) характеризуют возможности инструментов по обнаружению различных категорий уязвимостей CWE. Наибольший охват демонстрирует Psalm, который поддерживает 15 из 18 категорий (83,3%). Большую роль в этом играет taint-анализ, позволяющий выявлять такие уязвимости, как инъекции (CWE-78, CWE-89) [17], XSS (CWE-79) [18] и недостаточная защита учетных данных (CWE-522).

Phan занимает второе место с поддержкой 13 категорий (72,2%), демонстрируя сильные стороны в обнаружении специфических, таких как XML-инъекции (CWE-776), что не поддерживается другими средствами.

PHPStan демонстрирует наименьший охват уязвимостей – 8 категорий (44,4%). Преимущество инструмента заключается в анализе типов и логических конструкций, но он не ориентирован на выявление высокоуровневых уязвимостей безопасности без использования расширений.

В части обнаружения дефектов на тестовом наборе PHPStan выявил наибольшее количество проблем (48), проявив себя как эффективное решение для обнаружения логических несоответствий, недостижимого кода [19] и нарушений типизации. Psalm обнаружил 40 проблем, основная часть которых связана с избыточными условиями, несоответствиями типов и потенциально опасными операциями доступа к данным. Phan обнаружил 21 проблему, большая часть которых связана с согласованностью возвращаемых типов и использованием устаревших конструкций.

При оценке производительности наиболее быстрым инструментом оказался Phan (0,35 ± 0,02 с), тогда как PHPStan показал наибольшее время анализа (0,89 ± 0,07 с). По потреблению памяти все инструменты продемонстрировали схожие низкие показатели, однако Phan и PHPStan использовали минимальный объем памяти (0,011 ± ± 0,002 МВ и 0,021 ± 0,002 МВ соответственно).

Анализ показал, что ни одно средство не покрывает все категории уязвимостей, что подтверждает целесообразность их комбинированного использования [20] в процессе разработки для достижения максимального охвата проверок безопасности.

Выводы

Проведённое сравнительное исследование инструментов статического анализа PHP-кода показало, что PHPStan, Psalm и Phan демонстрируют разную эффективность при анализе кодовой базы. Psalm продемонстрировал наилучшие результаты в поиске уязвимостей безопасности. Встроенный taint-анализ эффективно выявляет риски инъекций и межсайтового скриптинга, что важно для веб-приложений, работающих с пользовательскими данными. Phan показал себя как наиболее сбалансированное решение. Он сочетает хорошее покрытие категорий уязвимостей с высокой производительностью, что делает его оптимальным для ежедневного использования в процессе разработки. PHPStan показал лучшие результаты в обнаружении логических ошибок и дефектов кода, таких как недостижимый код и проблемы с типами данных. Имеет строгую систему типов и способен выявлять избыточные проверки, что делает его предпочтительным решением для поддержания читаемости кода.

На практике оптимальным подходом становится комбинирование этих инструментов. Например, можно использовать Phan для ежедневных проверок, Psalm – для аудита безопасности, PHPStan – для глубокого анализа логики и рефакторинга. Такой комплексный подход позволяет выявить максимальное количество проблем на разных уровнях кодовой базы.

Список литературы

1. Проверка кода на уязвимости на всех стадиях разработки / А. В. Скрыпников, В. В. Денисенко, И. А. Высоцкая, И. И. Савченко, К. С. Евтеева // Современные наукоемкие технологии. 2021. № 3. С. 77–81.

2. Методика поиска уязвимостей в программном обеспечении, написанном на нескольких языках программирования / Б. А. Позин [и др.] // Труды Института системного программирования РАН. 2025. Т. 37, № 1. С. 122–132.

3. Кубрин Г. С., Зегжда Д. П. Поиск уязвимостей программного обеспечения с применением ансамбля алгоритмов анализа графового представления кода // Проблемы информационной безопасности. Компьютерные системы. 2023. № 4. С. 148–158.

4. Seara J. P., Serrão C. Automation of system security vulnerabilities detection using open-source software // Electronics. 2024. Vol. 13, N 5. P. 873.

5. Rio A., Fernando Brito e Abreu. PHP code smells in web apps: Evolution, survival and anomalies // Journal of Systems and Software. 2023. Vol. 200. P. 111644.

6. A critical comparison on six static analysis tools: Detection, agreement, and precision / V. Lenarduzzi [et al.] // Journal of Systems and Software. 2023. Vol. 198. P. 111575. https://doi.org/10.48550/arXiv.2101.08832.

7. Benchmarking static analysis for PHP applications security / J. Zhao [et al.] // Entropy. 2025. Vol. 27, N 9. P. 926.

8. Анализ методов предобработки программного кода для повышения эффективности применения больших языковых моделей в задачах обнаружения уязвимостей / В. В. Чаругин [и др.] // Computational nanotechnology. 2025. Vol. 12, N 3. P. 67–79.

9. Schuckert F., Katt B., Langweg H. Insecurity refactoring: Automated injection of vulnerabilities in source code // Computers & Security. 2023. Vol. 128, N 10. P. 103121. https://doi.org/10.1016/j.cose.2023.103121.

10. Analysis of IoT security challenges and its solutions using artificial intelligence / T. Mazhar [et al.] // Brain sciences. 2023. Vol. 13, N 4. P. 683.

11. Benzekki K., Messai M. L. Empowering Cybersecurity Analysis: Unifying CVE, CWE, and CPE through Knowledge Graphs // Computers & Security. 2025. Vol. 160. P. 104726. https://doi.org/10.1016/j.cose.2025.104726.

12. Shaikhelislamov D. S., Drobyshevskiy M. D., Belevantsev A. A. Ensuring Trustworthy Code: Leveraging a Static Analyzer to Identify and Mitigate Defects in Generated Code // Journal of Mathematical Sciences. 2024. Vol. 540. P. 233–251.

13. Пилькевич П. В., Спеваков А. Г., Калуцкий И. В. Модель системы принятия решений на основании мониторинга инцидентов // Труды МАИ. 2025. № 143. С. 1–27.

14. Воротникова Т. Ю. Надежный код: статический анализ программного кода как средство повышения надежности программного обеспечения информационных систем // Информационные технологии в УИС. 2020. № 2. С. 22–27.

15. Сравнительный анализ и выбор статических анализаторов безопасности кода / А. С. Марков, И. С. Антипов, С. С. Арустамян, Н. А. Магакелова // Вопросы кибербезопасности. 2024. № 5 (63). С. 79–88.

16. Alqaradaghi M., Morse G., Kozsik T. Detecting security vulnerabilities with static analysis – A case study // Pollack Periodica. 2022. Vol. 17, N 2. C. 1–7.

17. SQL injection attack detection in network flow data / Ignacio Samuel Crespo-Martínez, Ángel Manuel Guerrero-Higueras, Adrián Campazas Vega, Virginia Riego // Computers & Security. 2023. Vol. 127, N 4. P. 103093.

18. An XSS Attack Detection Model Based on Two-Stage AST Analysis / Qiuhua Wang, Chuangchuang Li, Lifeng Yuan, Dong Wang, Yeru Wang, Yizhi Ren, Weizhi Meng // IEEE Transactions on Dependable and Secure Computing. 2026. Vol. 23, is. 2. P. 4071–4084.

19. Vasileva V. I., Borodin A. E., Volkov A. E. Detection of dead function calls as source code defects through static analysis // Труды Института системного программирования РАН. 2025. Т. 37, № 4. С. 65–78.

20. A large scale analysis of code security in public repositories / Ciprian Oprisa, Dominic Octavian Grigoruţ, Haralambos Mouratidis, Eftychia Lakka // International Journal of Information Security. 2026. Vol. 25, N 1. P. 24. https://doi.org/10.1007/s10207-025-01187-w.

References

1. Skripnikov A.V., Denisenko V.V., Vysotskaya I.A., Savchenko I.I., Evteeva K.S. Checking code for vulnerabilities at all stages of development. *Sovremennye naukoemkie tekhnologii = Modern Science-Intensive Technologies*. 2021;(3): 77-81. (In Russ.)

2. Pozin B.A., et al. A methodology for searching for vulnerabilities in software written in several programming languages. *Trudy Instituta sistemnogo programmirovaniya RAN = Proceedings of the Institute for System Programming of the Russian Academy of Sciences*. 2025;37(1):122–132. (In Russ.)

3. Kubrin G.S., Zegzhda D.P. Search for software vulnerabilities using an ensemble of algorithms for analyzing graph representation of code. *Problemy informatsionnoi bezopasnosti. Komp'yuternye sistemy = Problems of Information Security. Computer Systems*. 2023;(4):148–158. (In Russ.)

4. Seara J.P., Serrão C. Automation of system security vulnerabilities detection using open-source software. *Electronics*. 2024;13(5):873.

5. Rio A., Fernando Brito e Abreu. PHP code smells in web apps: Evolution, survival and anomalies. *Journal of Systems and Software*. 2023;200:111644.

6. Lenarduzzi V., et al. A critical comparison on six static analysis tools: Detection, agreement, and precision. *Journal of Systems and Software*. 2023;198:111575.

7. Zhao J., et al. Benchmarking static analysis for PHP applications security. *Entropy*. 2025;27(9):926.

8. Charugin V.V., et al. Analysis of software code preprocessing methods to improve the efficiency of using large language models in vulnerability detection tasks. *Computational Nanotechnology*. 2025;12(3):67-79. (In Russ.)

9. Schuckert F., Katt B., Langweg H. Insecurity refactoring: Automated injection of vulnerabilities in source code. *Computers & Security*. 2023;128(10):103121. https://doi.org/10.1016/j.cose.2023.103121.

10. Mazhar T., et al. Analysis of IoT security challenges and its solutions using artificial intelligence. *Brain Sciences*. 2023;13(4):683.

11. Benzekki K., Messai M.L. Empowering Cybersecurity Analysis: Unifying CVE, CWE, and CPE through Knowledge Graphs. *Computers & Security*. 2025;160:104726. https://doi.org/10.1016/j.cose.2025.104726.

12. Shaikhelislamov D.S., Drobyshevskiy M.D., Belevantsev A.A. Ensuring Trustworthy Code: Leveraging a Static Analyzer to Identify and Mitigate Defects in Generated Code. *Journal of Mathematical Sciences.* 2024;540:233-251.

13. Pilkevich P.V., Spevakov A.G., Kalutskiy I.V. Model of a Decision-Making System Based on Incident Monitoring. *Trudy MAI = Proceedings of the MAI*. 2025;(143):1-27. (In Russ.)

14. Vorotnikova T.Yu. Reliable Code: Static Analysis of Program Code as a Means of Improving the Reliability of Information Systems Software. *Informatsionnye tekhnologii v UIS = Information Technologies in the UIS*. 2020;(2):22–27. (In Russ.)

15. Markov A.S., et al. Comparative Analysis and Selection of Static Code Security Analyzers. *Voprosy kiberbezopasnosti = Cybersecurity Issues*. 2024;(5):79–88. (In Russ.)

16. Alqaradaghi M., Morse G., Kozsik T. Detecting Security Vulnerabilities with Static Analysis – A Case Study. *Pollack Periodica*. 2022;17(2):1–7.

17. Ignacio Samuel Crespo-Martínez, Ángel Manuel Guerrero-Higueras, Adrián Campazas Vega, Virginia Riego. SQL injection attack detection in network flow data. *Computers & Security*. 2023;127(4):103093.

18. Qiuhua Wang, Chuangchuang Li, Lifeng Yuan, Dong Wang, Yeru Wang, Yizhi Ren, Weizhi Meng. An XSS Attack Detection Model Based on Two-Stage AST Analysis. *IEEE Transactions on Dependable and Secure Computing*. 2026;23(2):4071-4084.

19. Vasileva V.I., Borodin A.E., Volkov A.E. Detection of dead function calls as source code defects through static analysis. *Trudy Instituta sistemnogo programmirovaniya RAN = Proceedings of ISP RAS*. 2025;37(4):65-78.

20. Ciprian Oprisa, Dominic Octavian Grigoruţ, Haralambos Mouratidis, Eftychia Lakka. A large scale analysis of code security in public repositories. *International Journal of Information Security*. 2026;25(1):24. https://doi.org/10.1007/s10207-025-01187-w.

Информация об авторах / Information about the Authors

Огневой Ярослав Владимирович, студент факультета информационных технологий, Московский политехнический университет, г. Москва, Российская Федерация, e-mail: ya.ognevoy@yandex.ru, ORCID: 0009-0009-5291-4561

Yaroslav V. Ognevoy, Student at the Faculty of Information Technologies, Moscow Polytechnic University, Moscow, Russian Federation, e-mail: ya.ognevoy@yandex.ru, ORCID: 0009-0009-5291-4561

Спеваков Александр Геннадьевич, кандидат технических наук, доцент, Московский политехнический университет, г. Москва, Российская Федерация, e-mail: aspev@yandex.ru, ORCID: 0000-0003-3940-9607

Alexander G. Spevakov, Cand. Sci. (Engineering), Associate Professor, Moscow Polytechnic University, Moscow, Russian Federation, e-mail: aspev@yandex.ru, ORCID: 0000-0003-3940-9607

Калуцкий Игорь Владимирович, кандидат технических наук, доцент, Московский политехнический университет,
г. Москва, Российская Федерация,
e-mail: kalutsky_igor@mail.ru,
ORCID: 0000-0003-0575-3055

Клименко Павел Алексеевич, кандидат экономических наук, доцент, Курский филиал Финансового университета при Правительстве Российской Федерации,
г. Курск, Российская Федерация,
e-mail: paaklimenko@fa.ru,
ORCID: 0000-0003-2577-4144

Igor V. Kalutskiy, Cand. Sci. (Engineering), Associate Professor, Moscow Polytechnic University, Moscow, Russian Federation,
e-mail: kalutsky_igor@mail.ru,
ORCID: 0000-0003-0575-3055

Pavel A. Klimenko, Cand. Sci. (Economics), Associate Professor, Kursk Branch of the Financial University under the Government of the Russian Federation, Kursk, Russian Federation,
e-mail: paaklimenko@fa.ru,
ORCID: 0000-0003-2577-4144