DOI: 10.15514/ISPRAS-2025-37(1)-7

Методика поиска уязвимостей в программном обеспечении, написанном на нескольких языках программирования

[1, 2, 3] *Б.А.Позин, ORCID: 0000-0002-0012-2230 <bpozin@ec-leasing.ru>*
[2] П.А. *Бородушкина, ORCID: 0009-0009-8897-9025 <paborodushkina@edu.hse.ru>*
[2] *Д.А. Коротков, ORCID: 0009-0004-3064-8980 <dakorotkov_1@edu.hse.ru>*
[2] *М.А. Федоров, ORCID: 0009-0005-5473-9868 <mafedorov_6@edu.hse.ru>*
[2] *А.Ф. Муратов, ORCID: 0009-0009-5044-616X <afmuratov@edu.hse.ru>*

[1] *Институт системного программирования им. В.П. Иванникова РАН, Россия, 109004, г. Москва, ул. А. Солженицына, д. 25.*

[2] *Научно-исследовательский университет Высшая Школа Экономики Россия, 101978, Мясницкая ул., д.20.*

[3] *ЗАО ЕС-лизинг, Россия, 117405, Москва, Варшавское ш. д.125.*

Аннотация. В связи с необходимостью повышения производительности труда при анализе (разметке) результатов автоматизированного поиска уязвимостей в программах, проведенного с использованием инструментального средства, возникает проблема дефицита на рынке высококвалифицированных аналитиков для проведения разметки результатов. В работе описана разработанная методика для поиска уязвимостей в программном обеспечение (Далее - ПО), написанном на нескольких языках программирования (C, C++, Java, Python, Go). В ходе ее разработки проведен анализ всех автоматически обнаруживаемых детекторов в программах на этих языках и элементов их конструкций, подлежащих анализу. Детекторы упорядочены в соответствии с классификацией регулятора. Применение методики позволяет снизить квалификационные требования к аналитикам, проводящим разметку, и подготовить таких специалистов в разрабатывающих компаниях.

Ключевые слова: уязвимость; статический анализ программ; детектор; разметка; квалификационные требования.

Для цитирования: Позин Б.А., Бородушкина П.А., Коротков Д.А., Федоров М.А., Муратов А.Ф. Методика поиска уязвимостей в ПО, написанном на нескольких языках программирования. Труды ИСП РАН, том 37, вып. 1, 2025 г., стр. 121–132. DOI: 10.15514/ISPRAS–2025–37(1)–7.

Благодарности: Авторы признательны д.ф.-м.н. А.А. Белеванцеву за плодотворное обсуждение работы.

Vulnerability Detection Methodology in Software Written in Several Programming Languages

[1,2,3] *B.A. Pozin, ORCID: 0000-0002-0012-2230 <bpozin@ec-leasing.ru>*
[2] *P.A. Borodushkina, ORCID: 0009-0009-8897-9025 <polinaborod21@gmail.com>*
[2] *D.A. Korotkov, ORCID: 0009-0004-3064-8980 < dakorotkov_1@edu.hse.ru >*
[2] *M.A. Fedorov, ORCID: 0009-0005-5473-9868 <mafedorov_6@edu.hse.ru>*
[2] *A.F. Muratov, ORCID: 0009-0009-5044-616X <afmuratov@eu.hse.ru>*

[1] *Ivannikov Institute for System Programming of the Russian Academy of Sciences, 25, Alexander Solzhenitsyn st., Moscow, 109004, Russia.*

[2] *HSE University, 20, Myasnitskaya Ulitsa, Moscow, 101978, Russia*

[3] *«EC-leasing» Co. 125 Varshavskoye shosse, Moscow, 117405, Russia*

Abstract. Due to the need to increase labor productivity when analyzing (marking up) the results of an automated vulnerability search in programs conducted using SAST (Static Application Security Testing) tool, there is a problem of a shortage in the market of highly qualified analysts to mark up the results. The paper describes a developed technique for finding vulnerabilities in software written in several programming languages (C, C++, Java, Python, Go). During its development, an analysis of all automatically detectable detectors in programs in these languages and elements of their structures to be analyzed was carried out. The detectors are ordered according to the classification of the regulator. The application of the methodology allows reducing the qualification requirements for analysts conducting markup and training such specialists in developing companies.

Keywords: vulnerability; static analysis of programs; detector; markup; qualification requirements.

For citation: Pozin B.A., Borodushkina P.A., Korotkov D.A., Fedorov M.A., Muratov A.F. Vulnerability detection methodology in software written in several programming languages. Trudy ISP RAN/Proc. ISP RAS, vol. 37, issue 1, 2025. pp. 121-132 (in Russian). DOI: 10.15514/ISPRAS-2025-37(1)- 7.

Acknowledgements. Authors are grateful to Dr. A.A. Belevantsev for a fruitful discussion of the work.

1. Введение

В течение последних нескольких лет на рынке стали доступными инструментальные средства статического анализа программ для поиска уязвимостей и недекларированных возможностей. По данным [1] эти средства используются более, чем в 40% проектов. Повышение эффективности статического анализа программ с целью поиска уязвимостей и недекларированных возможностей в этой связи является чрезвычайно актуальным. Издан стандарт ГОСТ Р 71207-2024 [2]. В этом стандарте предусмотрено проведение разметки (анализа) результатов автоматизированного статического анализа с целью формирования начальной базы предупреждений о потенциальных ошибках и последующего обнаружения истинных и ложноположительных результатов [3-5]. В то же время существует проблема дефицита на рынке высококвалифицированных аналитиков для проведения разметки результатов статического анализа [6]. Данная работа посвящена разработке методики проведения разметки программ, написанных на языках C/C++, Java, Python, Go, которая позволит обучать специалистов для ее проведения и соответственно снижать требования к их начальной квалификации, а также трудоемкость и стоимость проведения таких работ.

2. Постановка задачи

С развитием информационных технологий, возникла необходимость в обеспечении безопасности программ от уязвимостей и недекларированных возможностей, начиная с самых ранних этапов разработки и на всём жизненном цикле ПО [2]. Появился запрос на проведение статического анализа программ, при котором осуществляют проверку кода программы на наличие уязвимостей и недекларированных возможностей. Требования к автоматизации статического анализа и последующей валидации его результатов (разметке) установлены в стандарте [2].

В результате проверки программы статическим анализатором им формируется база данных предупреждений об уязвимостях в той или иной программе.

Содержание этой базы доступно для анализа путем отображения средством доступа к базе. Так, в случае использования статического анализатора Svace [4], средством отображения является понятный для пользователя интерфейс Svacer. Svacer позволяет аналитику оставлять заметки в базе и сравнивать результаты разных запусков анализатора.

Результаты запуска статического анализатора для повышения качества рекомендовано подвергать дополнительному анализу. Этот процесс анализа называют разметкой результата прогона статического анализатора, или просто *разметкой*.

Целью разметки является рассмотрение экспертом предупреждений на предмет истинности и выявления ошибок первого рода (ложноположительные) и ошибок второго рода (ложноотрицательные). Из-за особенностей работы статических анализаторов, определенная часть предупреждений могут быть ложноположительными (данное понятие подразумевает под собой то, что анализатор указывает на конкретный участок кода с предупреждением о возможной уязвимости, однако данный участок кода является безопасным). Ложноотрицательные предупреждения – те, которые не обнаружены анализатором, но известно, что ошибочная конструкция в программе присутствует и может реализоваться при работе программы.

Задача состоит в том, чтобы:

- описать процесс работы по выявлению уязвимости программы на языке программирования, приводящий к устранению максимального количества остаточных уязвимостей после выполнения нескольких последовательных запусков анализатора после устранения найденных на предыдущем запуске уязвимостей;
- провести работу по разметке результата запуска статического анализатора с учетом свойств конкретного языка, на котором написана программа, с достижением высоких показателей по обнаружению и классификации выявленных уязвимостей.

Собственно, разметка и проводится для соотнесения найденной статическим анализатором потенциальной уязвимости (далее называемой - детектором) с реальной уязвимостью, соответствующей одному из выбранных регулятором типов – по классификации [12].

Разрабатываемый процесс должен быть унифицирован для большинства языков программирования и должен применяться любыми аналитиками по разметке в составе разрабатывающей организации.

3. Решение

Для такого рода проблемы потребовалось создание решения, которое позволит за достаточно небольшой период времени подготовить на рынке значительное количество экспертов по разметке за счет сокращения длительности их обучения и снижения квалификационного порога для их включения в проекты в данной сфере. Это позволит за относительно короткий период разрабатывающим компаниям создать в своей структуре команды, умеющие выявлять уязвимости в программах, написанных на различных языках программирования.

3.1 Процесс работы по систематическому выявлению уязвимостей

Процесс работы по систематическому выявлению уязвимостей представлен на *Рис. 1*.

В цикле работ по анализу результатов запуска статического анализатора с учетом свойств конкретного языка осуществляется проверка наличия выявленных Svace срабатываний, или потенциальных уязвимостей: детекторов (детектор - конкретный тип уязвимости и степень ее критичности [13]) и маркеров уязвимостей (предупреждение Svace о позиции уязвимости в исходном файле). При их обнаружении последовательно проводят анализ предполагаемых уязвимостей в соответствии с методикой, описанной ниже, выявление и фиксацию срабатываний, интерпретацию истинных срабатываний в терминах классификации ФСТЭК [12] и анализ выявленных уязвимостей. Далее отчет об обнаруженных уязвимостях передают на устранение. После доработки файл вновь передают на анализ статическим анализатором. Если после автоматизированного анализа маркеров уязвимости найти не удалось, процесс завершается. Если маркеры есть, то цикл повторяется.

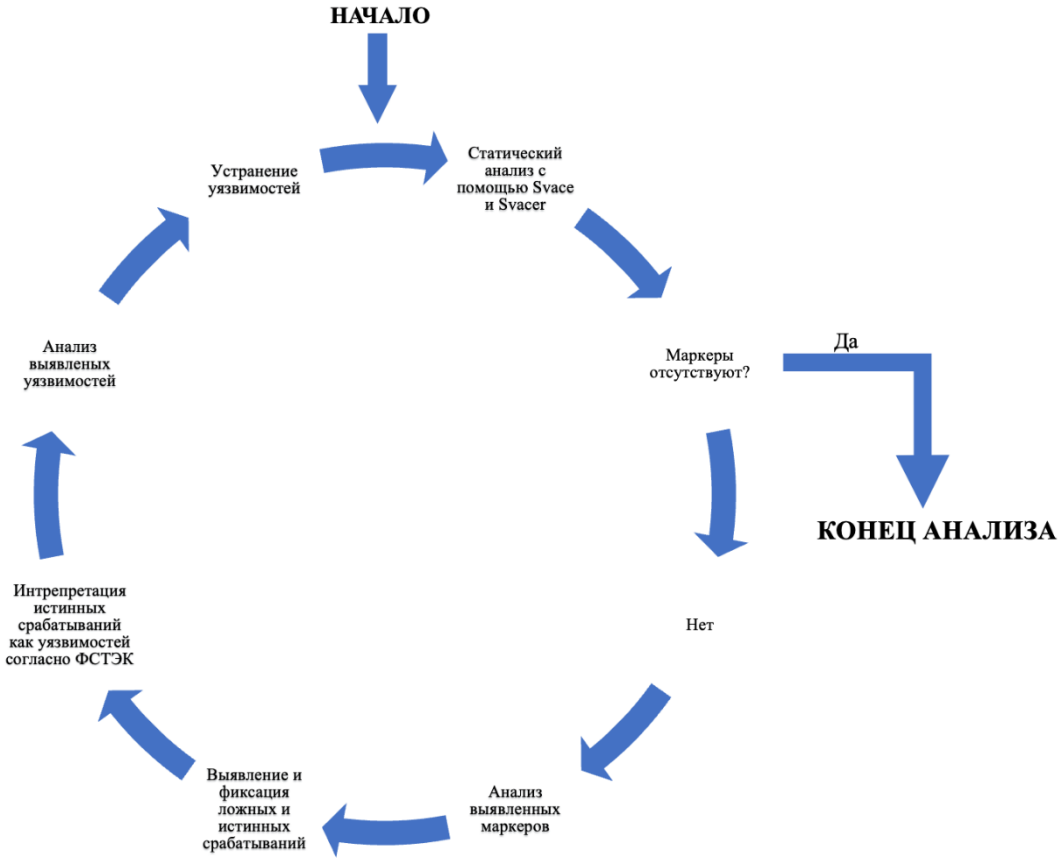

Рис. 1. Схема статического анализа.
Fig. 1. Static Analysis Scheme.

Для работы с результатами анализа используется Svacer, который представляет собой понятный для пользователя интерфейс, обеспечивающий возможность просмотра каталога найденных дефектов, фиксацию результатов верификации аналитиком найденных дефектов как уязвимостей и сравнение результатов анализа после разных запусков Svace.

Следует отметить, что Svace обнаруживает пять степеней критичности уязвимостей: Critical, Major, Normal, Minor, Undefined. С учетом того, что по функциональности и видам анализа

Svace полностью соответствует требованиям стандарта [2], процент истинных срабатываний составляет 60-90% [5]. Проведение разметки позволяет практически достигнуть этих показателей для анализируемых программ.

3.2 Базовые положения для методики разметки программ

Методика разметки программ, разработанная в данной работе, основывается на исследовании потенциально уязвимых конструкций языка программирования, которые обнаруживаются в программах. Для пяти языков программирования исследованы все типы конструкций, обнаруживаемых анализатором Svace как детекторы. Функциональные возможности Svace в этой части приведены в тТабл. 1.

Табл. 1 Функциональные возможности Svace.
Table 1. Svace functionality.

Язык программирования	Всего уязвимостей	Уровни критичности				
		Critical	Major	Normal	Minor	Undefined
C/C++	589	82	165	92	184	66
Go	118	20	30	35	18	15
Java	752	40	301	83	318	10
Python	42	4	7	0	7	24
Всего	150	146	503	210	527	115

В ходе работ над методикой для каждого языка программирования описаны все типы уязвимостей применительно ко всем реализующим их конструкциям каждого языка программирования. Фактически по каждому из пяти языков программирования разработаны справочники по оценке конструкций, в которых описаны уязвимости (детекторы), и правила анализа этих уязвимостей (см. раздел 4). Это описание включено в состав документа «Методика статического анализа для программ на языке». Методика включена в состав автоматизированной системы Центр кибербезопасности [14].

При разработке методики выявления уязвимостей и недекларированных возможностей в программном обеспечении для каждого из языков с применением Svace проведена идентификация всех возможных детекторов как потенциальных уязвимостей согласно классификации, предложенной регулятором [12]. Пример такой идентификации приведен в Табл. 2:

- **Первая группа**: Функции и процедуры, относящиеся к разным прикладным программам и несовместимые между собой (не функционирующие в одной операционной среде) из-за конфликтов, связанных с распределением ресурсов системы.
- **Вторая группа**: Функции, процедуры, изменение определенным образом параметров которых позволяет использовать их для проникновения в операционную среду информационной системы персональных данных (Далее – ИСПДн) и вызова штатных функций операционной системы, выполнения несанкционированного доступа без обнаружения таких изменений операционной системой.
- **Третья группа**: Фрагменты кода программ ("дыры", "люки"), введенные разработчиком, позволяющие обходить процедуры идентификации, аутентификации, проверки целостности и др., предусмотренные в операционной системе.
- **Четвертая группа**: Отсутствие необходимых средств защиты (аутентификации,

проверки целостности, проверки форматов сообщений, блокирования несанкционированно модифицированных функций и т.п.).

- **Пятая группа**: Ошибки в программах (в объявлении переменных, функций и процедур, в кодах программ), которые при определенных условиях приводят к сбоям, в том числе к сбоям функционирования средств и систем защиты информации, к возможности несанкционированного доступа к информации.

Табл. 2 Пример распределения детекторов в Python по типам уязвимостей согласно [12].
Table 2. An example of the distribution of detectors in Python by vulnerability type according to [12].

	Первая группа	Вторая группа	Третья группа	Четвертая группа	Пятая группа
Critical				tainted_array_index tainted_int tainted_ptr tainted_ptr.format_string	
Major		wrong_arguments_order		division_by_zero.ex catch.no_body	invariant_result similar_branches return_in_finally bad_copy_paste
Normal					
Minor	user.malloc_zero_light_demo			division_by_zero.ex.float user.bad_string_demo user.bad_keyword_demo user.bad_string_light_demo user.bad_keyword_light_demo	passed_to_proc_after_release
Un-defined			hardcoded_password	mypy.syntax mypy.var_annotated	accidental_tuple mutable_default_argument mypy.misc mypy.return mypy.operator mypy.attr_defined mypy.valid_type mypy.index mypy.call_overload mypy.arg_type mypy.union_attr mypy.return_value mypy.assignment mypy.list_item mypy.call_arg mypy.override mypy.no_redef mypy.name_defined mypy.str_format mypy.has_type mypy.duplicate_module

3.3 Методика разметки программ

В процессе разметки осуществляется автоматизированный поиск детекторов методами статического анализа, выявление ложноположительных срабатываний в результате работы аналитика по разметке, а также анализ выявленных уязвимостей (правильных срабатываний) на предмет выработки рекомендаций по устранению.

В результате применения методики аналитик по разметке по существу должен ответить на вопрос, являются ли выявленные при статическом анализе детекторы уязвимостями и насколько сильно они сказываются на безопасности написанной программы? Для проведения указанных работ аналитику по разметке необходим доступ (с использованием Svacer) к базе результатов статического анализа и документ «Методика статического анализа для программ на языке».

Основные шаги методики таковы:

1. После получения результатов от Svace и открытия их в веб интерфейсе Svacer приступить к работе. При анализе следует учитывать, что для каждого языка количество детекторов может разниться для разных уровней критичности ПО по сравнению с табл. 1. От аналитика по разметке требуется каждый выявленный детектор соотносить с имеющимися в методике и проверять в соответствии с контекстом программы на наличие ложноположительных выявленных уязвимостей.
2. Для того, чтобы выявить истинные и ложные срабатывания, необходимо:
   - рассмотреть контекст, то есть понять назначение кода и входные данные, которые он обрабатывает;
   - проверить код вручную: проверить наличие признаков ложных срабатываний, таких как использование безопасных методов программирования, правильность обработки ошибок или наличие комментариев, объясняющих отсутствующую уязвимость;
   - оценить критичность детектора;
   - для выявленной уязвимости выработать рекомендации по ее устранению.
3. Аналитику по разметке рекомендуется в ходе работ или по их завершении обсудить результаты анализа с коллегами (аналитиками), разработчиками конструкций, оцененных как уязвимость, а также со специалистами по безопасности, чтобы получить независимое «второе» мнение, прежде всего по поводу влияния выявленных уязвимостей или сомнительных конструкций на безопасность системы, в состав которой включено рассматриваемое ПО. В результате таких консультаций могут быть получены дополнительные рекомендации по написанию тестов, специально предназначенных для проверки отсутствия уязвимости. Могут быть получены и рекомендации по необходимости применить другие инструменты SAST или динамического анализа программ (DAST) для подтверждения результатов. В частности, может быть получена рекомендация использовать инструменты для отслеживания потока данных, чтобы проверить, может ли ошибочная конструкция действительно привести к нарушению безопасности (в неочевидных случаях).
4. Документировать результаты разметки в виде отчета, который направляется в соответствии с технологией работ организации, в которой работает аналитик по разметке.

В состав отчета рекомендуется включить следующие разделы:

- Общая статистика – количество рассмотренных строк кода, выявленное количество детекторов и уязвимостей;

- Подробное описание результатов по каждому детектору: название-описание детектора, маркеры (срабатывания), какие являются истинными, а какие ложноположительными, основания для выработки заключения по маркеру, рекомендации по исправлению для истинных срабатываний.
- Общая статистика и таблица найденных детекторов согласно методике. Возможно использование для этих целей отчетов, выдаваемых Svacer. Рекомендуется передать в составе отчета таблицы, вырабатываемые Svacer до и после разметки.

4. Примеры описания правил анализа детекторов для использования аналитиками по разметке

Методика разметки, включенная в состав системы [14], содержит подробное описание всех типов детекторов, которые представлены в Таблице 1 и обнаруживаются анализатором Svace. По каждому детектору дается описание того, в чем суть выявленной угрозы, поясняет возможные способы исправления уязвимости, но оставляет право принятия решения об исправлении за разработчиком. При этом аналитик по разметке выдает квалифицированные рекомендации по исправлению кода. Наличие такого описания позволяет снизить начальные требования к уровню знания языка программирования аналитиком по разметке. Соответственно после нескольких разметок такой аналитик начинает лучше понимать срабатывания и выявленные дефекты и глубже погружаться в свойства языка, повышать скорость и эффективность достижения высокого качества разметки. Последнее может быть полезно для работников локальных служб информационной безопасности.

На рисунках 2-5 приведены примеры детекторов для программ на языке Java (рис. 2 и 3) и Python (рис. 4 и 5).

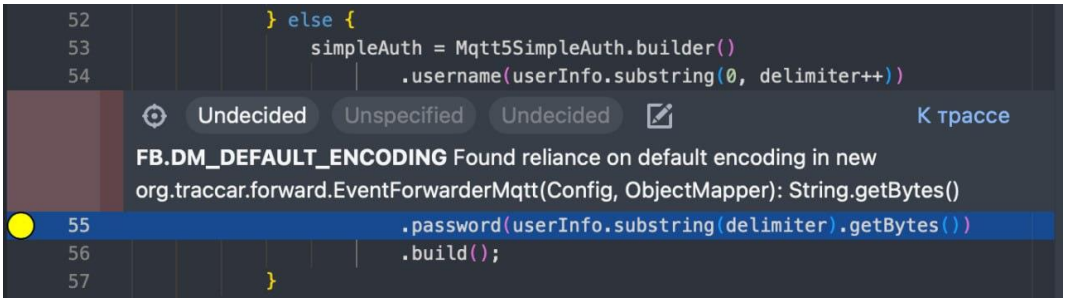

Рис. 2. Детектор FB.DM_DEFAULT_ENCODING в интерфейсе Svacer.
Fig. 2. FB.DM_DEFAULT_ENCODING detector in the Svacer interface.

Детектор "FB.DM_DEFAULT_ENCODING" определяет ошибку – вызов метода, который выполняет преобразование байта в строку (или строку в байт) и предполагает, что кодировка платформы по умолчанию подходит. Это приведет к тому, что поведение приложения будет различаться на разных платформах. Используйте альтернативный API (application programming interface — программный интерфейс приложения) и явно укажите имя набора символов или объект Charset.

Рис. 3. Детектор FB.WA_NOT_IN_LOOP в интерфейсе Svacer.
Fig. 3. FB.WA_NOT_IN_LOOP detector in the Svacer interface.

Метод детектора "FB.WA_NOT_IN_LOOP" содержит вызов процедуры java.lang.Object.wait(), который не находится в цикле. Если монитор используется для нескольких условий, то ожидаемое событие может быть заменено иным, включенным в монитор. При необходимости отработки ожидаемого условия рекомендуется включить java.lang.Object.wait() в цикл для гарантированной отработки требуемой конструкции.

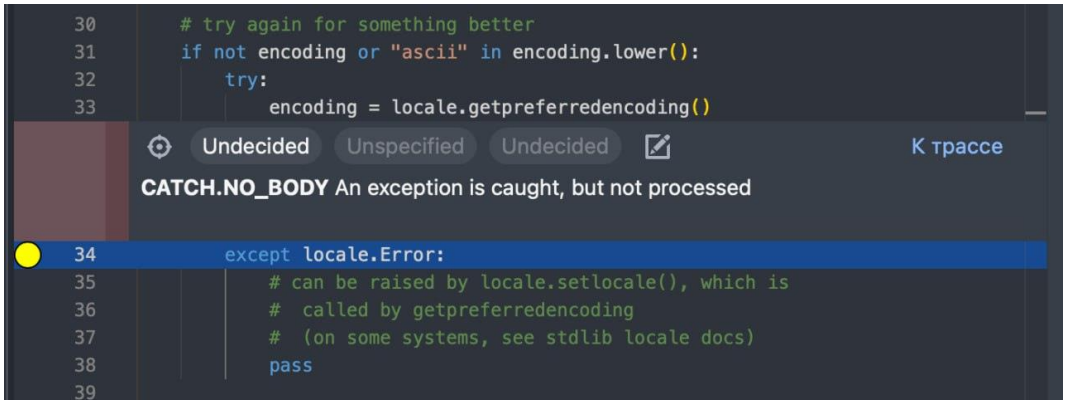

Рис. 4. Детектор CATCH.NO_BODY в интерфейсе Svacer.
Fig. 4. CATCH.NO_BODY detector in the Svacer interface.

Детектор "CATCH.NO_BODY" возникает, когда в блоке try-except исключение перехватывается, но сам блок except остается пустым или не содержит кода для обработки исключения. Может возникнуть ситуация, когда исключение будет перехвачено, но не будет обработано.

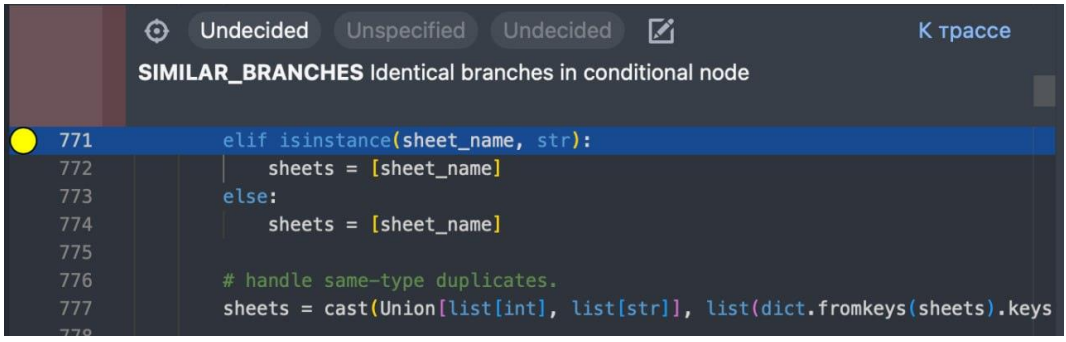

Рис. 5. Детектор SIMILAR_BRANCHES в интерфейсе Svacer.
Fig. 5. SIMILAR_BRANCHES detector in the Svacer interface.

Детектор "SIMILAR_BRANCHES" возникает, когда в условном операторе (if-else) обнаруживаются две идентичные ветви кода, то есть в обеих ветвях выполняется одно и то же действие. Это может указывать на ненужное или избыточное повторение кода, и такая конструкция может быть улучшена или упрощена.

5. Связь с CWE

Разработанная методика имеет потенциал к развитию за счет систематизации детекторов как уязвимостей. На данный момент в методике (и в Svace) поддерживается классификация отечественного регулятора ФСТЭК. С позиций развития методики основной идеей является конвертируемость результатов в различные классификации уязвимостей. Одной из наиболее распространённых и общепринятых в мире является классификация CWE (Common Weakness Enumeration [15] – классификация недостатков безопасности в программном и аппаратном обеспечении. Она представляет собой иерархический словарь, предназначенный для разработчиков и специалистов по обеспечению безопасности ПО [8]. Основная цель CWE – предотвращать возникновение уязвимостей за счёт обучения специалистов наиболее распространённым типам уязвимостей и за счет этого – предотвращение их возникновения в программах.

Совместное использование классификаций - ФСТЭК и CWE, (а впоследствии и CVE) позволит повысить процент выявляемых типов уязвимостей, которые сегодня не поддерживаются при использовании классификации регулятора, но могут быть выявлены при совместном использовании двух и более международных классификаций. Цель использования международных классификаций - конвертируемость результатов в различные метрики оценки качества поиска уязвимостей. В настоящее время ведутся работы по сопоставлению классификаций ФСТЭК и CWE, что позволит улучшить понимание выявляемых уязвимостей, расширить состав выявляемых уязвимостей и упростит работу по их исправлению.

6. Заключение

По итогам проведенных работ получены следующие результаты:

- Разработан процесс выявления уязвимостей аналитиком по разметке с использованием инструментальных средств Svace/Svacer.
- Разработана методика разметки программ с применением документа «Методика статического анализа для программ на языке» – то есть методика поиска уязвимостей и недекларированных возможностей в программах на пяти языках программирования (C/C++, Java, Python, Go). Она ориентирована на помощь экспертам в выявлении неочевидных моментов и в проверке подлинности найденных угроз, при использовании в качестве статического анализатора Svace и снижает квалификационные требования к аналитикам по разметке, одновременно являясь учебником по проведению разметки. Данная методика включена в состав Автоматизированной Системы Центр кибербезопасности.
- Ключевым направлением развития этих работ является создание плагина для продукта Svacer. Плагин позволит для каждого рассматриваемого детектора по запросу аналитика получить:
  - описание сути найденного детектора, предполагаемой уязвимости или НДВ (если она обнаружена),
  - причины, по которой эта конструкция в данном языке и в текущих условиях применения конструкции скорее всего является уязвимостью,
  - в случае возможных нескольких вариантов уязвимости данной конструкции – получить примеры правильных конструкций.

При реализации плагина будет реализована возможность совместного использования классификаций уязвимостей ФСТЭК и CWE.

Список литературы / References

[1]. "Белеванцев А. (ИСП РАН) для форума "Russia DevOps Report - 2023". [Электронный ресурс]. – 2023 – URL: https://russiadevopsreport.ru/. – (Дата обращения: 13.10.2024)

[2]. ГОСТ Р 71207-2024. Защита информации. Разработка безопасного программного обеспечения. Статический анализ программного обеспечения. Общие требования.

[3]. PVS-Studio: Ложноположительные срабатывания статического анализатора кода. [Электронный ресурс]. – 2021 – URL: https://pvs-studio.ru/ru/blog/terms/6461/. – (Дата обращения: 03.11.2024)

[4]. Иванников В.П., Белеванцев А.А., Бородин А.Е., Игнатьев В.Н., Журихин Д.М., Аветисян А.И., Леонов М.И. Статический анализатор Svace для поиска дефектов в исходном коде программ. Труды Института системного программирования РАН. 2014;26(1): с.231-250. / Ivannikov V.P., Belevantsev A.A., Borodin A.E., Ignatiev V.N., Zhurikhin D.M., Avetisyan A.I., Leonov M.I. Svace static analyzer for searching for defects in program source code. Proceedings of the Institute of System Programming of the Russian Academy of Sciences. 2014;26(1): p.231-250. DOI: https://doi.org/10.15514/ISPRAS-2014-26(1)-7.

[5]. Белеванцев А., Аветисян А. Многоуровневый статический анализ для поиска закономерностей ошибок и дефектов в исходном коде. В: Петренко А., Воронков А. (ред.) Перспективы системной информатики. PSI 2017. Конспекты лекций по информатике, том 10742, стр. 28–42. /Belevantsev, A., Avetisyan, A. Multi-level Static Analysis for Finding Error Patterns and Defects in Source Code. В: Petrenko, A., Voronkov, A. (ред.) Perspectives of System Informatics. PSI 2017. Lecture Notes in Computer Science, vol 10742, p. 28–42. Springer, Cham, 2018, DOI:10.1007/978-3-319-74313-4_3.

[6]. Positive Technologies: Positive Technologies: раскрытие уязвимостей и опыт взаимодействия исследователей и вендоров в 2022–2023 годах. [Электронный ресурс]. – 2024 – URL: https://www.ptsecurity.com/ru-ru/research/analytics/vulnerability-disclosure-and-researcher-vendor-interaction-experience-in-2022-2023/. – (Дата обращения: 04.11.2024)

[7]. ГОСТ Р ИСО/МЭК 12207-2010. Информационная технология. Системная и программная инженерия. Процессы жизненного цикла программных средств.

[8]. ГОСТ Р 56939–202X. Защита информации. Разработка безопасного программного обеспечения. Общие требования.

[9]. ГОСТ Р ИСО/МЭК 27034. Информационная технология. Методы и средства обеспечения безопасности. Безопасность приложений. Часть 1. Обзор и общие понятия.

[10]. ГОСТ Р 50922-2006. Защита информации. Основные термины и определения.

[11]. ГОСТ Р 58412-2019. Защита информации. Разработка безопасного программного обеспечения. Угрозы безопасности информации при разработке программного обеспечения.

[12]. КонсультантПлюс: (ФСТЭК) Общая характеристика уязвимостей прикладного программного обеспечения. [Электронный ресурс]. – 2008 – URL: https://www.consultant.ru/document/cons_doc_LAW_99662/2da3bb1b6c61f5acbbefcb29ae7dc99615ce0d05/.– (Дата обращения: 20.10.2024)

[13]. PVS-Studio: Сортировка предупреждений статических анализаторов по приоритету при поиске и исправлении программных ошибок. [Электронный ресурс]. – 2016 – URL: https://habr.com/ru/companies/pvs-studio/articles/305532/. – (Дата обращения: 03.11.2024)

[14]. Б. Позин. Автоматизация и экономика для обеспечения жизненного цикла безопасного ПО., Информационная безопасность, №4, 2024, с.60/ B.Pozin. Automation and Economics for Secure Software Life Cycle Assurance., Information Security, No.4, 2024, p.60.

[15]. Common Weakness Enumeration [Электронный ресурс]. – 2023 – URL: https://cwe.mitre.org/about/new_to_cwe.html. – (Дата обращения: 02.11.2024)

Информация об авторах / Information about authors

Борис Аронович ПОЗИН – доктор технических наук, профессор, главный научный сотрудник Института системного программирования им. В.П. Иванникова РАН, профессор базовой кафедры ЗАО ЕС-лизинг в МИЭМ НИУ ВШЭ, технический директор ЗАО ЕС-лизинг. Сфера научных интересов: программная инженерия, системы обеспечения жизненного цикла доверенного программного обеспечения, автоматизированное тестирование программ.

Boris Aronovich POZIN – Dr. Sci. (Tech.), Professor, Chief Researcher at the Ivannikov Institute for System Programming of the Russian Academy of Sciences, Professor of the Basic Department of CJSC EC-Leasing at the Higher School of Economics, Technical Director of CJSC EC-Leasing. Research interests: software engineering, life cycle ensuring systems for trusted software, automated software testing.

Полина Андреевна БОРОДУШКИНА – студентка третьего курса НИУ ВШЭ Московского институт электроники и математики им. А.Н. Тихонова департамента прикладной математики. Сфера научных интересов: проектирования безопасного ПО с применением SAST, анализ существующих угроз и их классификации на языках программирования C/C++.

Polina Andreevna BORODUSHKINA – third-year student of the National Research University Higher School of Economics, A.N. Tikhonov Moscow Institute of Electronics and Mathematics, Department of Applied Mathematics. A.N. Tikhonov Moscow Institute of Electronics and Mathematics, Department of Applied Mathematics. Her research interests are study of secure software design using SAST, analysis of existing threats and their classification in C/C++ programming languages.

Дмитрий Антонович КОРОТКОВ – студент четвертого курса НИУ ВШЭ Московского институт электроники и математики им. А.Н. Тихонова департамента электронной инженерии. Сфера научных интересов: проектирование безопасного ПО с применением Статического анализа (SAST), верификация результатов статического анализа, анализ существующих угроз и их классификации на языке программирования Java.

Dmitry Antonovich KOROTKOV – fourth-year student of the National Research University Higher School of Economics, A.N. Tikhonov Moscow Institute of Electronics and Mathematics, Department of Electronic Engineering. A.N. Tikhonov Moscow Institute of Electronics and Mathematics, Department of Electronic Engineering. Research interests: design of secure software using Static Analysis (SAST), verification of static analysis results, analysis of existing threats and their classification in Java programming language.

Михаил Александрович ФЕДОРОВ – студент четвертого курса НИУ ВШЭ Московского институт электроники и математики им. А.Н. Тихонова департамента электронной инженерии. Сфера научных интересов: проектирование безопасного ПО с применением Статического (SAST) и Композиционного (SCA) анализов, разработка ПО в сфере фронт-энд, проектирование безопасной архитектуры Операционных систем.

Michael Alexandrovich FEDOROV – fourth-year student of the National Research University Higher School of Economics, A.N. Tikhonov Moscow Institute of Electronics and Mathematics, Department of Electronic Engineering. A.N. Tikhonov Moscow Institute of Electronics and Mathematics, Department of Electronic Engineering. Research interests: design of secure software using Static (SAST) and Compositional (SCA) analyses, front-end software development, design of secure architecture of Operating Systems.

Айнур Фуатович МУРАТОВ – студент третьего курса НИУ ВШЭ Московского институт электроники и математики им. А.Н. Тихонова департамента компьютерной инженерии. Сфера научных интересов: изучение проектирования безопасного ПО с применением SAST, анализ существующих угроз и их классификации на языке программирования Go.

Aynur Fuatovich MURATOV – third year student of the National Research University Higher School of Economics, A.N. Tikhonov Moscow Institute of Electronics and Mathematics, Department of Computer Engineering. A.N. Tikhonov Moscow Institute of Electronics and Mathematics, Department of Computer Engineering. Research interests: studying secure software design using SAST, analyzing existing threats and their classification in Go programming language.