**Оглавление**

[Уязвимости программного обеспечения 1](#_Toc39053080)

[Основные виды угроз безопасности приложений по «Топ-10 OWASP - 2017». 2](#_Toc39053081)

[Основные инструменты 6](#_Toc39053082)

[Инструменты SAST и языки программирования 7](#_Toc39053083)

[Различные методы тестирования 9](#_Toc39053084)

[Средства статического анализа кода 10](#_Toc39053085)

[Средства статического анализа кода с открытым исходным кодом 12](#_Toc39053086)

[Коммерческие средства статического анализа кода 30](#_Toc39053087)

[Тестирование средств статического анализа кода 56](#_Toc39053088)

[OWASP Benchmark 56](#_Toc39053089)

[Использование OWASP Benchmark на CentOS-7. 63](#_Toc39053090)

[CWE-89: SQL Injection 65](#_Toc39053091)

[CWE-77: Command Injection 68](#_Toc39053092)

[Литература 72](#_Toc39053093)

## Уязвимости программного обеспечения

Уязвимости программ — ошибки, допущенные программистами на этапе разработки программного обеспечения. Они позволяют злоумышленникам получить незаконный доступ к функциям программы или хранящимся в ней данным. Изъяны могут появиться на любом этапе жизненного цикла, от проектирования до выпуска готового продукта. Существует множество различных видов уязвимостей, но из наиболее опасных выделяют 10, рассмотрим их подробнее:

### Основные виды угроз безопасности приложений по «Топ-10 OWASP - 2017».

Данный список уязвимостей представлен частной компанией OWASP [[1](#Owasp)], целью которой является борьба с уязвимостями программного обеспечения. Все приведенные ниже уязвимости взяты из документа «Топ-10 OWASP - 2017» [[2](#TOP2017)].

*Внедрение* – уязвимости, связанные с отправкой интерпретатору в составе команды или запроса, непроверенных данных, которые могут заставить интерпретатор выполнить непредусмотренные команды или отправить закрытые данные третьим лицам. По документу «Топ-10 OWASP - 2017» данный тип угроз имеет среднюю распространенность, низкие сложности обнаружения и эксплуатации. Для предотвращения внедрения рекомендуется изолировать данные от команд и запросов.

*Недостатки аутентификации* – уязвимости, связанные с некорректной реализацией аутентификаций и управлениями сессиями, которые позволяют злоумышленникам, обнаружив их, используя автоматизированные инструменты, изменять пароли, ключи и перехватывать учетные данные пользователей. По данному документу данный тип угроз имеет средние распространенность и сложность обнаружения, и низкую сложность эксплуатации. Это связано с необходимостью подбора паролей из нескольких сотен тысяч действительных комбинаций имен.

*Разглашение конфиденциальных данных* – уязвимости, связанные с плохой защищенностью критичных финансовых, медицинских или персональных данных, которые злоумышленники могут похитить или заменить. Данных тип атак считается одним из самых распространенных и опасных, при средней сложности обнаружения и эксплуатации. Это связано с ненадежными способами шифрования конфиденциальных данных.

*Внешние сущности XML* – уязвимости, связанные с обработкой внешних сущностей старыми обработчиками XML документов, которые получают доступ к внутренним файлам, могут использоваться для выполнения удаленных запросов с сервера. По документу «Топ-10 OWASP - 2017» данный тип угроз имеет средние сложность эксплуатации и распространенность, при низкой сложности обнаружения. Это связано с тем, что использование данных уязвимостей - напрямую зависит от обработчика XML, поэтому для предотвращения появления небезопасных участков кода рекомендуется использовать инструменты обнаружения уязвимостей или обучать тестировщиков XXE - тестированию.

*Недостатки контроля доступа* – уязвимости, связанные с возможностями взаимодействия пользователей с приложением, которые позволяют злоумышленникам получить несанкционированный доступ к данным пользователей. Данный тип атак имеет средние сложность эксплуатации, сложность обнаружения и распространенность. Это связано с необходимостью осуществлять запрет доступа по умолчанию в тех местах приложения, где это может нанести вред системе.

*Некорректная настройка параметров безопасности* – уязвимости, связанные с неправильной настройкой безопасности, из-за установки стандартных параметров. Эти уязвимости часто пытаются использовать злоумышленники для получения доступа или информации о системе. По рассматриваемому документу данный тип угроз имеет низкую сложности эксплуатации и обслуживания, а также большую распространенность. Эти уязвимости предоставляют огромные возможности злоумышленникам по похищению данных внутри нашего приложения и исправления взаимодействия компонентов.

*Межсайтовое выполнение сценариев (XSS)* – связаны с добавлением непроверенных данных на новую веб-страницу без их проверки или преобразования, что позволяет злоумышленникам выполнять сценарии в браузере жертвы, что позволяет им перехватывать пользовательские сессии, подменять страницы. Данная уязвимость считается второй по распространенности из Топ-10 OWASP и имеет при этом высокие сложность эксплуатации и использования. Для предотвращения (XSS) рекомендуется отделять непроверенные данные от активного контента браузера.

*Небезопасная десериализация* – уязвимость, связанная с ошибками сериализации, с помощью которых злоумышленники могут удаленно выполнять код приложения. По Топ-10 OWASP данная уязвимость имеет высокую сложность эксплуатации, из-за того, что готовые эксплойты редко можно использовать без изменения или доработки, также средние распространенность и сложность обнаружения. Решением данной проблемы будет отключение сериализованных объектов от непроверенных источников.

*Использование компонентов с известными уязвимостями* - эксплуатация уязвимого компонента, такого как библиотеки, фреймворки и программные модули, может привести к потере данных или перехвату контроля над сервером. Чаще всего это связано с устареванием старых компонентов и редкими выходами обновлений текущих мер безопасности.

*Недостатки журналирования и мониторинга* - позволяет злоумышленникам развить атаку, скрыть свое присутствие и проникнуть в другие системы, а также изменить, извлечь или уничтожить данные. Эта проблема лежит в основе почти всех крупных взломов. Злоумышленники проводят атаки, полагаясь на отсутствие контроля и своевременного реагирования на инциденты. По документу «Топ-10 OWASP - 2017» данный тип угроз имеет высокую распространенность и сложность обнаружения, это связано с тем, что чтобы предотвратить данный тип угроз необходимо постоянно регистрировать все ошибки входа, доступа и проверки данных. При этом данная уязвимость имеет среднюю сложность эксплуатации.

Рассмотрим анализ каких языков программирования поддерживают инструменты. Данные результаты будут приведены в виде таблиц:

### Основные инструменты

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Название | Тип | Разработчик | Язык на котором написана | Операционная система | Аппаратная платформа | Последняя версия | Языковая поддержка |
| SonarQube Project | Статический анализатор кода | SonarSource | Java | Кроссплатформенное | Кроссплатформенное | 8.0 (16 октября 2019 года) | CSS, Erlang, Java, Properties, JSON, Puppet |
| OWASP Orizon Project | Статический анализатор кода | Paolo Perego | Java | Кроссплатформенное | Кроссплатформенное | 1.19 (2013-04-22) | Java, C, C++, C#, Perl, XML |
| OWASP LAPSE Project | Статический анализатор кода | Gregory Disney-Leugers | Java | Кроссплатформенное | Кроссплатформенное | LAPSE+ (4/15/2011) | Java |
| OWASP O2 Platform | Статический анализатор кода | Diniz Cruz | C# | Windows | .NET Framework 4.5 | v6.0 (Май 2016 г.) | C# |
| OWASP WAP-Web Application Protection | Статический анализатор кода | Ibéria Medeiros | Java | Linux, Macintosh и Windows | JRE | v2.0.2 ( 02.Out.2015) | PHP |

Таблица 1

Примечание: Данная таблица была составлена по материалам, взятым с официального сайта OWASP.

### Инструменты SAST и языки программирования

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Инструмент SAST | Python | Ruby | Java | C/C++ | Go | XML | JS | PHP | HTML | Perl | SQL | C# | VB | Scala |
| Bandit | + |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Brakeman |  | + |  |  |  |  |  |  |  |  |  |  |  |  |
| Codesake Dawn |  | + |  |  |  |  |  |  |  |  |  |  |  |  |
| Deep Dive |  |  | + |  |  |  |  |  |  |  |  |  |  |  |
| SpotBugs |  |  | + |  |  |  |  |  |  |  |  |  |  |  |
| FindSecBugs |  |  | + |  |  |  |  |  |  |  |  |  |  |  |
| Flawfinder |  |  |  | + |  |  |  |  |  |  |  |  |  |  |
| GolangCI-Lint |  |  |  |  | + |  |  |  |  |  |  |  |  |  |
| Google CodeSearchDiggity |  | + | + |  |  | + | + | + | + |  |  |  |  |  |
| Graudit |  | + | + |  |  |  | + | + |  | + | + |  |  |  |

Таблица 2

Продолжение табл. 2

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Инструмент SAST | Python | Ruby | Java | C/C++ | Go | XML | JS | PHP | HTML | Perl | SQL | C# | VB | Scala |
| LGTM | + |  | + | + |  |  | + |  |  |  |  | + |  |  |
| NET Security Guard |  |  |  |  |  |  |  |  |  |  |  | + |  |  |
| phpcs-security-audit |  |  |  |  |  |  |  | + |  |  |  |  |  |  |
| PMD |  |  | + |  |  |  |  |  |  |  |  |  |  |  |
| PREfast |  |  |  | + |  |  |  |  |  |  |  |  |  |  |
| Progpilot |  |  |  |  |  |  |  | + |  |  |  |  |  |  |

Примечание: Данная таблица была составлена по материалам, взятым с официального сайта OWASP.

Существует несколько способов обнаружения уязвимостей программного обеспечения, рассмотрим наиболее основные:

### Различные методы тестирования

Существует 4 основных метода, которые можно использовать при создании программ тестирования:

* ручные проверки и обзоры;
* моделирование угроз;
* проверка кода;
* тестирование на проникновение.

Рассмотрим каждый из них подробнее:

*Ручные проверки* – это проверки, выполняемые с разработчиком или тестировщиком лично, включающие проверку технологических решений, таких как архитектурные проекты. Они обычно проводятся путем анализа документации или проведения интервью с разработчиками или владельцами системы. Хотя концепция ручных проверок и проверок человеком проста, они могут быть одними из самых мощных и эффективных доступных методов. Спросив кого-то, как что-то работает и почему это было реализовано определенным образом, тестировщик может быстро определить, очевидны ли какие-либо проблемы с безопасностью. Ручные проверки и обзоры являются одним из немногих способов проверить сам процесс жизненного цикла разработки программного обеспечения и убедиться в наличии адекватной политики или навыков.

*Моделирование* – помогает разработчикам систем думать об угрозах безопасности, с которыми могут столкнуться их системы и приложения. Поэтому моделирование угроз можно рассматривать как оценку рисков для приложений. Это позволяет проектировщику разрабатывать стратегии смягчения потенциальных уязвимостей и помогает сосредоточить свои ограниченные ресурсы и внимание на тех частях системы, которые больше всего в этом нуждаются. Рекомендуется, чтобы во всех приложениях была разработана и задокументирована модель угроз. Модели угроз должны создаваться как можно раньше в SDLC (жизненный цикл разработки ПО) и должны пересматриваться по мере развития приложения и его развития. Выходные данные самой модели угроз могут различаться, но обычно представляют собой набор списков и диаграмм.

*Проверка исходного кода* – это процесс ручной проверки исходного кода веб-приложения на наличие проблем безопасности. Многие серьезные уязвимости безопасности не могут быть обнаружены ни с какой другой формой анализа или тестирования. Используя исходный код, тестировщик может точно определить, что происходит (или должно происходить. Анализ исходного кода также может быть эффективным для выявления проблем реализации, таких как места, где проверка входных данных не была выполнена.

*Тестирование на проникновение* – это тестирование, удаленно работающего приложения для обнаружения уязвимостей в безопасности, не зная внутренней работы самого приложения. Как правило, группа по тестированию на проникновение имеет доступ к приложению, как если бы они были пользователями. Тестер действует как злоумышленник и пытается найти и использовать уязвимости. Во многих случаях тестеру будет предоставлена ​​действительная учетная запись в системе. Были разработаны инструменты тестирования на проникновение, которые автоматизируют процесс.

## Средства статического анализа кода

Инструменты анализа исходного кода (Source Code Analysis Tools - SAST), предназначены для анализа исходного кода или скомпилированных версий кода с целью выявления недостатков безопасности.

Некоторые инструменты начинают использоваться в IDE (В процессе разработки). Это необходимо для поиска ошибок программного обеспечения, возникающих на фазе разработки ПО, и организует немедленную обратную связь с разработчиком по вопросам создания кода.

Среди достоинств существующих инструментов можно отметить следующие:

* масштабируемость – то есть инструменты работают на большом количестве программного обеспечения и может запускаться неоднократно;
* используются для нахождения таких ошибок, как переполнение буфера, ошибки SQL-инъекций;
* средства вывода содержат точную информацию о исходных файлах, номерах строк, содержащих ошибки.

Также можно отметить следующие недостатки:

* трудно обнаружить проблемы аутентификации, проблемы контроля доступа, небезопасного использование криптографии;
* обнаруживают относительно небольшой процент недостатков безопасности приложений;
* большое количество ложных срабатываний;
* часто не удается найти проблемы конфигурации, так как они не представлены в коде;
* испытывает трудности с анализом кода, который не может быть скомпилирован. Аналитики часто не могут скомпилировать код, потому что у них нет нужных библиотек, всех инструкций по компиляции или всего кода.

Поэтому можно выделить следующие критерии выбора инструмента:

* поддерживаемые языки программирования;
* типы выявляемых уязвимостей;
* оценка OWASP Benchmark [[3](#Benchmark)];
* список обрабатываемых библиотек;
* требования готового набора исходного кода;
* сложность настройки, использования;
* возможность непрерывной и автоматической работы;
* стоимость лицензии.

### Средства статического анализа кода с открытым исходным кодом

#### Bandit

Bandit [[4](#Bandit)] – это средство анализа исходного кода Python, использующее модуль ast [[5](#ast)] из стандартной библиотеки Python. Исходный код Bandit доступен для свободного скачивания с GitHub, на котором имеет рейтинг 1875 и является основой еще для 181 проекта. Последние исправления вносились в исходный код 19 октября 2019 г.

Модуль ast в средстве Bandit используется для преобразования исходного кода в абстрактное синтаксическое дерево. Средство Bandit позволяет пользователю определять различные типы проверок, которые будут выполняться при обходе этих узлов. По завершении анализа создается отчет, в котором перечислены проблемы безопасности, выявленные в проанализированном исходном коде.

##### Конфигурирование

Рассмотрим основные моменты, связанные с конфигурированием средства Bandit [[6](#BanditConf)]. Файл конфигурации Bandit может выбрать определенные тестовые плагины для запуска и переопределить конфигурации этих тестов по умолчанию. Пример конфигурации может выглядеть следующим образом:

# список включенных тестов:

tests: ['B201', 'B301']

# список пропущенных тестов:

skips: ['B101', 'B601']

Если вам требуется несколько наборов тестов для конкретных задач, следует создать несколько конфигурационных файлов.

Пропуск тестов

Конфигурация средства Bandit может содержать дополнительные списки идентификаторов, которые могут включать или исключать тесты. Эти списки эквивалентны использованию -t и -s в командной строке. Если даны только списки тестов, то Bandit будет включать тесты из списка, исключая все другие. Если заданы только пропуски некоторых тестов, то бандит включит все тесты, отсутствующие в списке. Если оба заданы, то бандит будет включать только тесты из списка подключаемых, а затем удалит те, которые необходимо пропустить из этого набора.

Генерация конфигурации

Bandit поставляется с инструментом bandit-config-generator, который генерирует файл конфигурации. Сгенерированная конфигурация будет включать блоки конфигурации по умолчанию для всех обнаруженных плагинов для тестирования и черного списка. Эти данные затем могут быть удалены или отредактированы по мере необходимости, для получения минимального списка конфигурации, по желанию пользователя. Генератор конфигурации поддерживает параметры командной строки: -t и -s для указания списка идентификаторов тестов, которые должны быть включены или исключены соответственно.

Настройка тестовых плагинов

Файл конфигурации Bandit написан на YAML [[7](#Yaml)], а опции для каждого теста плагина представлены в разделе, название которого соответствует методу теста. Например, если тестовый плагин называется «try\_except\_pass», его раздел конфигурации может выглядеть следующим образом:

try\_except\_pass:

  check\_typed\_exception: True

##### Плагины для анализа

Bandit поддерживает 34 теста для обнаружения различных проблем безопасности в коде Python [[8](#BanditTest)]. Эти тесты создаются как плагины и есть возможность создать новые для расширения функциональности данного средства.

Наиболее популярные плагины:

B101 – assert\_used:

Этот плагин проверяет использование ключевого слова Python - assert. Было обнаружено, что некоторые проекты используют это слово для реализации ограничений интерфейса. Однако assert удаляется при компиляции в оптимизированный байт-код (python -o производит файлы \* .pyo).

B107 ­­­­– hardcoded\_password\_default:

Использование паролей встроенных в код программы, увеличивает вероятность подбора пароля. Этот плагин ищет все определения функций, которые задают строковый литерал по умолчанию для некоторого аргумента. Плагин проверяет, что аргумент не похож на пароль.

B505 – weak\_cryptographic\_key:

С увеличением вычислительной мощности - увеличивается способность взламывать шифры с короткой длиной ключа. Рекомендуемый размер длины ключа для алгоритмов RSA и DSA составляет 2048 и выше. 1024 бита и ниже теперь считаются легкими. Рекомендуется, чтобы длина ключа ЕС была 224 и выше. Этот плагин проверяет надежность ключа и выводит сообщение об ошибке, в случае нарушения уровня безопасности.

B605 – start\_process\_with\_a\_shell:

Данный плагин позволяет выявлять запуск подпроцесса с помощью командной оболочки. Такой подход небезопасен, так как он уязвим для атак внедрения оболочки.

##### Генерация отчета

Bandit поддерживает различные средства для вывода проблем безопасности в коде Python. Они создаются как плагины, и новые могут быть созданы пользователями для расширения функциональности.

#### Brakeman

Brakeman [[9](#Brakeman)] – это средство анализа исходного кода Ruby on Rails. Исходный код Brakeman доступен для свободного скачивания с GitHub, на котором имеет рейтинг 5592 и является основой еще для 587 проектов. Последние исправления вносились в исходный код 25 ноября 2019 г.

Средство Brakeman позволяет пользователю без лишних настроек проверить на уязвимость код своего приложения. По завершении анализа создается отчет, в котором перечислены проблемы безопасности, выявленные в исходном коде.

##### Конфигурирование

Рассмотрим основные моменты, связанные с конфигурированием средства Brakeman. Опции Brakeman можно хранить и читать из файлов YAML. Чтобы упростить процесс записи файла конфигурации, опция -C выведет текущие установленные параметры. Параметры, передаваемые в командной строке, имеют приоритет над файлами конфигурации. Расположение конфигурации по умолчанию: ./config/brakeman.yml, ~ / .brakeman / config.yml и /etc/brakeman/config.yml

Опция -c может использоваться для указания файла конфигурации.

##### Плагины для анализа

Brakeman поддерживает 3 плагина, делающие работу с сервисом и вывод результатов более удобными.

1. ALE (Asynchronous Lint Engine) [[10](#ALe)] – плагин, обеспечивающий проверку синтаксиса и семантических ошибок в NeoVim 0.2.0+ [[11](#NeoVim)] и Vim 8 [[12](#Vim8)] во время редактирования текстовых файлов, и действует как клиент протокола Vim Language Server. ALE использует функции и таймеры управления заданиями NeoVim и Vim 8 для запуска линтеров содержимого текстовых буферов и возврата ошибок при изменении текста в Vim. Это позволяет отображать предупреждения и ошибки в файлах, редактируемых в Vim, до того, как файлы были сохранены обратно в файловую систему. ALE поддерживает различные функций языкового протокола:

* Перейти к определению (: ALEGoToDefinition);
* Поиск ссылок (: ALEFindReferences);
* Hover - информация (: ALEHover);
* Поиск символа (: ALESymbolSearch).

1. pronto-brakeman [[13](#Pronto)] – сопоставляет Brakeman Confidence [[14](#ConfidenceBrakeman)] с уровнями серьезности сообщений, генерируемых Pronto [[15](#Pronto)]. Карты высокой достоверности соответствуют фатальным ошибкам, карты средней достоверности – предупреждениям, а карты низкой достоверности – информации,
2. overcommit [[16](#Overcommit)] – это инструмент для управления и настройки Git Hooks [[17](#GitHooks)].

##### Генерация отчета

Все выходные данные Brakeman, кроме отчетов, отправляются в stderr, что упрощает перенаправление stdout в файл и получение отчета. По умолчанию Brakeman будет возвращать ненулевой код выхода, если обнаружены какие-либо предупреждения безопасности или обнаружены ошибки сканирования. Есть возможность настраивать вывод с помощью команд.

#### GolangCI-Lint

GolangCI-Lint [[18](#GolandCILint)] – является агрегатором линтеров. Линтер – статический анализатор кода. Проверяет синтаксис и корректность кода, не запуская компиляцию или выполнение в случае нахождения ошибки.

Исходный код GolangCI-Lint доступен для свободного скачивания с GitHub, на котором имеет рейтинг 4140 и является основой еще для 340 проектов. Последние исправления вносились в исходный код 13 сентября 2019 г.

GolangCI-Lint легко интегрируется и имеет малое количество ложных срабатываний. Поддерживает модули Go. По завершении анализа создается отчет, в котором перечислены проблемы безопасности, выявленные в проанализированном исходном коде.

##### Конфигурирование

Рассмотрим основные моменты, связанные с конфигурированием средства GolangCI-Lint [[19](#GolandConfig)].

Файл конфигурации имеет более низкий приоритет, чем параметры командной строки. GolangCI-Lint ищет файлы конфигурации по следующим путям из текущего рабочего каталога: .golangci.yml, .golangci.toml, .golangci.json

GolangCI-Lint также ищет конфигурационные файлы во всех каталогах от первого проанализированного до корня. Чтобы увидеть, какой файл конфигурации используется и откуда он был получен, необходимо запустить golangci-lint с параметром -v.

Параметры конфигурации внутри файла идентичны параметрам командной строки. Можно настроить параметры определенных линтеров только в файле конфигурации.

##### Плагины для анализа

GolangCI-Lint поддерживает плагин - SublimeLinter-contrib-golang-cilint [[20](#GolandCILintPlugins)], предоставляющий интерфейс для golangci-lint. Он будет использоваться с файлами с синтаксисом «Go».

##### Генерация отчета

GolangCI-Lint поддерживает различные средства для вывода отчета, содержащего уровень опасности уязвимостей в коде, в удобной для пользователя форме.

#### LGTM

LGTM [[21](#LGTM)] – бесплатная служба статического анализа, которая автоматически отслеживает фиксации общедоступного кода в: Bitbucket Cloud, GitHub или GitLab. Поддерживает C / C ++, C #, COBOL (в бета-версии), Java, JavaScript / TypeScript, Python. Исходный код LGTM доступен для свободного скачивания с GitHub, с рейтингом 463 и является базовым еще для 148 проектов. Последние исправления вносились в исходный код 13 декабря 2019 г.

LGTM – это платформа для анализа кода, которая автоматически проверяет его на наличие реальных CVE (идентификаторы общих уязвимостей и подверженностей воздействиям) и уязвимостей. Сервис ранжирует наиболее релевантные результаты, чтобы показать только наиболее важные оповещения.

##### Конфигурирование

Рассмотрим основные моменты, связанные с конфигурированием средства LGTM [[22](#LGTMConf)].

Анализ проектов и их результаты в LGTM можно настроить, добавив файл lgtm.yml в свой репозиторий. Этот файл может использоваться для того чтобы:

* показать или скрыть результаты конкретных запросов;
* классифицировать файлы и, используя эту классификацию, сообщить LGTM, в каких файлах необходимо увидеть результаты;
* настроить процесс извлечения кода в соответствии с потребностями.

Закончив настройку анализа для своего проекта, можно включить автоматический просмотр кода запросов извлечения (список хост-систем репозитория) или использовать плагин для загрузки предупреждений.

Кроме того, если LGTM не обнаружил язык в проекте или не смог создать код на этом языке, возможно повторить попытку анализа в LGTM.

##### Плагины для анализа

Плагины LGTM IDE позволяют легко просматривать и исправлять существующие проблемы при редактировании кода. Чтобы загружать оповещения из LGTM и исправлять их непосредственно в IDE, используются плагины LGTM.

Для написания и выполнения запросов QL локально поддерживаются плагины следующих IDE:

* Eclipse;
* IntelliJ;
* Visual Studio.

А для приведенных далее IDE для просмотра результатов в рабочей области используются плагины QL:

* Eclipse;
* Visual Studio.

##### Генерация отчета

Необработанные данные, полученные в результате анализа кода, обрабатываются, чтобы определить, какие результаты показывать. Существует возможность заблокировать вывод следующей информации в отчете:

* все оповещения, помеченные комментариями к подавлению или аннотациями @SuppressWarnings (только Java);
* все оповещения, идентифицированные в исходных файлах, имеющие один или несколько тегов;
* любые оповещения о запросах, которые скрыты файлом конфигурации проекта LGTM.

Можно изменить поведение по умолчанию, отредактировав соответствующий файл исходного кода или файл конфигурации проекта lgtm.yml.

Остальные результаты отображаются.

#### PMD

PMD [[23](#PMD)] – это анализатор исходного кода, который находит общие ошибки программирования, такие как неиспользуемые переменные, пустые блоки catch, создание ненужных объектов. Исходный код PMD доступен для свободного скачивания с GitHub, на котором имеет рейтинг 2617 и является основой еще для 905 проектов. Последние исправления вносились в исходный код 30 ноября 2019 г.

PMD включает в себя CPD, детектор копирования-вставки. CPD находит дублированный код в C/C ++, C #, Dart, Fortran, Go, Groovy, Java, JavaScript, JSP, Kotlin, Lua, Matlab, Objective-C, Perl, PHP, PLSQL, Python, Ruby, Salesforce.com Apex, Scala, Swift и Visualforce. Разберемся подробнее с основными моменты, связанные с конфигурированием средства PMD [[24](#PMDConfig)].

##### Конфигурирование.

Правила сообщения можно изменить, указав нужный атрибут сообщения в элементе правила. Это заменит предыдущее значение и изменит сообщение, которое правило напечатает в отчете.

Точно так же приоритет правила может быть изменен через вложенный элемент. Используя приоритет, можно деактивировать некоторые правила на основе минимального порогового значения приоритета (устанавливается с помощью параметра -min CLI). Приоритет – это целое число в диапазоне от 1 до 5, где 1 является наивысшим приоритетом.

Следующее правило понижает приоритет EmptyCatchBlock до 5:

<rule ref="category/java/errorprone.xml/EmptyCatchBlock"

message="Empty catch blocks should be avoided" >

<priority>5</priority>

</rule>

Использование параметров CLI -min 4 приведет к игнорированию правила.

Свойства правил позволяют легко настроить поведение правил непосредственно из XML. Они бывают нескольких типов, которые соответствуют типу их значений. Например, NPathComplexity объявляет свойство «reportLevel» с целочисленным типом значения, которое соответствует порогу, выше которого будет сообщаться о методе. Если вы считаете, что значение по умолчанию 200 слишком велико, вы можете уменьшить его, например, до. 150 следующим образом:

<rule ref="category/java/design.xml/NPathComplexity">

<properties>

<property name="reportLevel">

<value>150</value>

</property>

</properties>

</rule>

##### Плагины для анализа

PMD можно интегрировать с помощью некоторых инструментов непрерывной интеграции:

* Hudson Plugin;
* Maven PMD Plugin.

##### Генерация отчета

PMD поддерживает различные средства для вывода ошибок и уязвимых мест в коде. Они используются для облегчения поиска необходимых параметров анализа. В отчете будут описаны основные ошибки безопасности кода приложения.

#### Puma Scan

Puma Scan [[25](#PUMA)] – это статический анализатор исходного кода .NET C #, который работает как плагин IDE для Visual Studio и через MSBuild в конвейерах CI. Исходный код Puma Scan доступен для свободного скачивания с GitHub, имеющий рейтинг 331 и являющийся основой для 65 других проектов. Последние изменения вносились в исходный код 8 августа 2019 г.

В Visual Studio уязвимости немедленно отображаются в среде разработки в виде предупреждений компилятора, предотвращая попадание ошибок в ваши приложения. Puma Scan также интегрируется в сборку для обеспечения анализа безопасности во время компиляции.

Рассмотрим основные моменты, связанные с конфигурированием средства Puma Scan [[26](#PUMAConfig)].

##### Конфигурирование

Конфигурация позволяет настраивать параметры сканера по умолчанию, параметры правил, нестандартные источники, пользовательские методы очистки и подавлять ложные срабатывания. Команды могут начать с настройки глобальных настроек с помощью End User Edition. Глобальные параметры конфигурации находятся в файле Settings.json в каталоге данных приложения роуминга PumaSecurity.

При первом анализе End User Edition создает файл настроек для конкретного приложения (.pumafile) в хранилище кода со значениями по умолчанию, определенными в файле глобальной конфигурации. Файл .pumafile находится в репозитории исходного кода, чтобы обеспечить историческое отслеживание всех изменений в файле и должен находиться в том же каталоге, что и анализируемый файл решения (.sln), или в корне репозитория исходного кода.

В Server и Azure DevOps используется файл .pumafile, сохраненный в системе управления версиями, вместе с исходным кодом приложения для настройки Puma Scan в конвейере сборки.

При первом анализе отдельные приложения будут иметь глобальные настройки по умолчанию, скопированные в новый .pumafile в репозитории исходного кода. Этот файл должен обеспечивать контроль исходного кода, чтобы обеспечить согласованность конфигурации Puma Scan между рабочими станциями разработчика, локальными серверами сборки и конвейерами Azure DevOps.

##### Плагины для анализа

Puma Scan – имеет плагин MSBuild дающий возможность ему обрабатываться в Jenkins [[27](#Jenkins)] для сообщения об уязвимостях.

##### Генерация отчета

Во время написания кода Puma Scan работает в фоновом режиме в среде Visual Studio, выделяя и подчеркивая ошибки и опасные участки кода. Имеет 3 уровня угроз безопасности, и во время сканирования распределяет участки кода по зонам уязвимостей.

#### Pyre

Pyre [[28](#Pyre)] – средство для производительной проверки типов Python 3. Исходный код Puma Scan доступен для свободного скачивания с GitHub с рейтингом 3163 и используется в 167 проектах. Последние исправления вносились в исходный код 12 декабря 2019 г.

Pyre также может запускать статический анализ для выявления потенциальных проблем безопасности. Эти проблемы безопасности идентифицируются с помощью так называемого Taint Analysis [[29](#TaintAnalysis)]. Pysa работает, отслеживая потоки помеченных (tainted) данных от того места, где они формируются до места, где они небезопасно обрабатываются (приемники).

##### Конфигурирование

Pysa использует два типа файлов для конфигурации: один файл taint.config и неограниченное количество файлов с расширением .pysa. Файл taint.config представляет собой документ JSON, в котором хранятся определения источников, приемников, функций и правил. Файлы .pysa – это файлы-заглушки, в которых код аннотирован с помощью источников, приемников и компонентов, определенных в файле taint.config. Примеры этих файлов можно найти в репозитории Pyre.

Эти файлы находятся в каталоге, настроенном с помощью taint\_models\_path в файле .pyre\_configuration. Любой файл .pysa, найденный в этой папке, будет проанализирован Pysa, и заглушки будут использованы во время анализа.

##### Плагины для анализа

Pyre содержит плагин ide\_plugins [[30](#PyrePlugins)] позволяющий просматривать ошибки Pyre для любого рабочего пространства, содержащего .pyre\_configuration.

##### Генерация отчета

Pyre поддерживает различные средства для вывода проблем безопасности в коде. После первого использования средства есть возможность внести корректировки в конфигурации Pyre, чтобы в последующем отчете, сформированном инстурментом, были более точные или пропущены не интересующие пользователя сведения.

#### RIPS

RIPS [[31](#Rips)] – средство SAST, специализированное для Java и PHP, которое обнаруживает неизвестные уязвимости безопасности и проблемы качества кода. Исходный код RIPS доступен для свободного скачивания с GitHub, на котором имеет рейтинг 5 и является основой еще для 1 проектa. Последние исправления вносились в исходный код 9 декабря 2019 г.

RIPS позволяет просматривать и применять предложения замены мест в коде, которые повышают безопасность приложения в целом.

##### Конфигурирование

RIPS имеет конфигурации, которые помогают максимально эффективно использовать анализ кода. Эти конфигурации зависят от логики и среды приложения, а также от ожиданий результатов сканирования Доступны следующие параметры конфигурации:

'base\_uri' (required, default: http://localhost:8080): API URL

'timeout' (optional, default: 100): Timeout of request in seconds

'connect\_timeout' (optional, default: 10): Number of seconds to wait while trying to connect to server

'oauth2' (optional): OAuth2 configuration, see OAuth2 Config

##### Плагины для анализа

RIPS имеет плагины, которые добавляют новый интерфейс, увеличивают глубину анализа, предоставляют новые возможности сканирования:

* Integrate Security Testing with GitHub Actions [[32](#IntegrateSecurity)] – используется в качестве шлюза безопасности для автоматической проверки сборок приложений на наличие уязвимостей безопасности и проблем с качеством кода;
* Scaling Security Testing to Large Teams [[33](#ScalingSecurity)] – предоставляет сканирование с помощью центра обработки данных, увеличивает глубину анализа с помощью улучшенного анализа зависимостей и инфраструктуры, добавляет улучшенный пользовательский интерфейс с новыми уведомлениями, статистикой активности пользователей и проверкой работоспособности RIPS;
* BigTree [[34](#BigTree)] – небольшая система управления контентом, которая не зависит от многих платформ и содержит новый интрефейс и анализатор.

##### Генерация отчета

После анализа RIPS генерирует отчет о найденных уязвимостях и угрозах кода. Предоставляет полную информацию о типе обнаруженных угроз, времени затраченного на анализ кода. Параметры отчета можно настраивать как до анализа, так и после.

#### SpotBugs

SpotBugs [[35](#SpotBugs)] – статический анализатор исходного кода Java. Исходный код SpotBugs доступен для свободного скачивания с GitHub, имеет рейтинг 1608 и используется еще в 250 проектах. Последние исправления вносились в исходный код 26 ноября 2019 г.

SpotBugs – это программа для поиска ошибок в программах Java. Он ищет экземпляры «шаблонов ошибок» - экземпляров кода, которые могут быть ошибками.

##### Конфигурирование

Файлы фильтра могут использоваться для включения или исключения отчетов об ошибках для определенных классов и методов.

Файл фильтра представляет собой XML-документ с элементом FindBugsFilter верхнего уровня, который имеет некоторое количество элементов Match в качестве дочерних. Каждый элемент Match представляет предикат, который применяется к сгенерированным экземплярам ошибок. Обычно фильтр используется для исключения ошибок. Например:

$ spotbugs -textui -exclude myExcludeFilter.xml myApp.jar

Тем не менее, фильтр также может быть использован для выбора экземпляров ошибок для конкретного отчета:

$ spotbugs -textui -include myIncludeFilter.xml myApp.jar

Элементы соответствия содержат дочерние элементы, которые являются конъюнктами предиката. Другими словами, каждый из наследников должен быть истинным, чтобы предикат был истинным.

##### Плагины для анализа

SpotBugs имеет несколько плагинов [[36](#SpotBugsPlugins)] для работы и содержит возможность создать новые для расширения функциональности:

* SpotBugs Eclipse – позволяет использовать SpotBugs в Eclipse IDE [[37](#Eclipse)];
* SpotBugs Maven – позволяет интегрировать SpotBugs в проект Maven [[38](#Maven)].
* SpotBugs Gradle – позволяет интегрировать SpotBugs в скрипт сборки для Gradle [[39](#Gradle)].

##### Генерация отчета

При создании отчета результаты анализа имеют две основные цели. Во-первых, они позволяют информировать SpotBugs о значении методов в приложении, чтобы оно могло давать более точные результаты или меньше ложных предупреждений. Во-вторых, они позволяют настроить точность выполненного анализа. Снижение точности анализа может сэкономить память и время анализа за счет отсутствия некоторых реальных ошибок или создания большего количества ложных предупреждений. После завершения выводит список с подтверждениями или опровержениями каких-либо уязвимостей в коде.

#### VisualCodeGrepper

VisualCodeGrepper (VCG) [[40](#VCG)] – инструмент для автоматического анализа безопасности кода, который обрабатывает C / C ++, Java, C #, VB и PL / SQL. Исходный код VisualCodeGrepper доступен для свободного скачивания с SourceForge, на котором имеет рейтинг 4.5/5. Последние исправления вносились в исходный код 10 декабря 2019 г.

##### Конфигурирование

Файлы конфигурации существуют для каждого из шести языков, которые сканирует VCG. Они обеспечивают дополнительный уровень сканирования для улучшения встроенного сложного сканирования.

Содержимое файлов конфигурации состоит из списка функций или фрагментов кода для сканирования, а также связанного описания, которое будет отображаться в результатах. Описание включает необязательную настройку опасности в квадратных скобках и отделяется от функции символом «=>» в следующем формате:

имя функции {=>} {{N}} {описание}

(где N - уровень опасности от 1 (критический) до 3 (средний) (или, необязательно, 0 для «нормального»))

##### Плагины для анализа

VisualCodeGrepper имеет возможность внедрения новых пользовательских плагинов для расширения функциональности данного средства.

##### Генерация отчета

Результаты записываются на панель результатов в том порядке, в котором они были расположены. Результаты имеют следующий формат:

Серьезность: проблема с кодом

Номер строки - имя файла

Описание

{фрагмент кода}

Для ясности заголовок выпуска имеет следующие цветовые коды:

* критический – пурпурный;
* высокий – красный;
* средний – оранжевый:
* стандартный – желтый;
* низкий – серо-синий;
* потенциальная проблема – зеленый;
* подозрительный комментарий, указывающий на неработающий код – темно-синий.

Результаты также записываются в сводную таблицу в сокращенной форме. Эти результаты могут быть сохранены как текст ASCII или как XML.

Помимо средств с открытым исходным кодом также существуют коммерческие инструменты для поиска уязвимостей:

### Коммерческие средства статического анализа кода

#### BlueClosure BC Detect

BlueClosure BC Detect (BlueClosure) [[41](#BlueClosure)] – это средство анализа кода JavaScript, исполняемого на стороне клиента. Помогает анализировать и автоматически обнаруживать проблемы межсайтового скриптинга на основе DOM [[42](#DOM)] благодаря гибридному IAST Engine (интерактивное тестирование безопасности приложений) вместе с модулем Smart Fuzzer [[43](#SmartFuzzer)]. Межсайтовый скриптинг (XSS) – это уязвимость, которая заключается во внедрении кода, исполняемого на стороне клиента (JavaScript) в веб-страницу, которую просматривают другие пользователи.

Технология BlueClosure позволяет автоматически сканировать весь сайт и использует движок JavaScript браузера для понимания кода.

##### Возможности интегрирования с другими продуктами

BlueClosure может анализировать любую кодовую базу, написанную с помощью JavaScript-фреймворков, таких как Angular.js, jQuery, Meteor.js, React.js и многих других.

##### Конфигурация

Пользователь может получить доступ к более подробной информации, щелкнув ссылку в поле уязвимости, открыв еще два поля.

В первом поле описывается история уязвимости, затем контролируемое пользователем значение, если уязвимость является опасной, далее могут ли данные быть закодированы / не закодированы, и, щелкнув «Показать операции», вы можете получить доступ к списку операций JavaScript, которые привели к уязвимости.

Второе – «Уязвимый код», который покажет пользователю часть вредоносного кода, который был выполнен с использованием уязвимости.

##### Генерация отчета

Генератор отчетов позволяет легко обнаружить уязвимости в коде. При открытии следующей страницы (http://www.domxss.com/domxss/01\_Basics/00\_simple\_noHead.html) с помощью BCDetect пользователю предлагается всплывающее окно, показывающее потенциальную уязвимость в коде JavaScript страницы. Чтобы лучше исследовать уязвимость, мы можем просто щелкнуть соответствующее поле, и под ним появится подробное окно «истории». Это окно будет содержать всю информацию, которая привела к обнаружению самой уязвимости.

BlueClosure указывает основную информацию об уязвимостях, такую ​​как типология, источник и приемник, которые привели к выполнению вредоносного кода.

#### Codacy

Codacy [[44](#Codacy)] – инструмент статического анализа кода, который позволяет разработчикам автоматически выявлять и решать проблемы безопасности, дублирования, сложности, нарушений стилей и снижения охвата при каждом запросе на фиксацию. Aнализирует 28 различных языков программирования, включая Python, Java, JS, Ruby, Go, SQL, C # и Scala.

##### Возможности интегрирования с другими продуктами

Интегрируется с такими инструментами, как Brakeman, Bandit, FindBugs [[45](#FindBugs)], в облаке с GitHub, BitBucket [[46](#BitBucker)] и Gitlab [[47](#GitLab)]. Codacy запускает анализ для каждого коммита хранилища, анализирует прошлые коммиты, чтобы выявить всю тенденцию изменений. Для каждого коммита видна эволюция качества: количество созданных / удаленных проблем.

##### Конфигурация

Чтобы запустить инструмент, файл конфигурации /src/.codacy.json связывается с файлами, которые вы должны проанализировать, а также с шаблонами, которые вы должны проверить.

Файлы для анализа находятся в /src, что означает, что, когда они указаны в конфигурации, пути относятся к /src. Пример файла конфигурации:

.codacy.json

files: файлы для анализа (их путь относительно /src)

tools: набор инструментов

name: уникальный идентификатор инструмента

patterns: массив шаблонов, которые должны быть проверены

patternId: уникальный идентификатор шаблона

parameters: параметры шаблона

name: уникальный идентификатор параметра

value: значение, которое будет использоваться в качестве значения параметра

{

  "files": ["foo / bar / baz.js", "foo2 / bar / baz.php"],

  "инструменты":[

    {

      "Название": "jshint",

      "узоры": [

        {

          "PatternId": "latedef",

          "параметры": [

            {

              "Название": "latedef",

              «Значение»: «вары»

            }

          ]

        }

      ]

    }

  ]

}

Если /src/.codacy.json не существует или какое-либо его содержимое (файлы или шаблоны) недоступно, вам следует вызвать инструмент для всех файлов из /src (файлы должны рекурсивно искать все папки в /src) и сверьте их с шаблонами по умолчанию. При получении /src/.codacy.json вы должны запускать инструмент только для подмножества файлов и для шаблонов вашего инструмента.

##### Генерация отчета

В сгенерированном отчете могут присутствовать следующие типы оповещений: Error, Warning, Info.

Codacy также может распределяет найденные уязвимости по категорям:

ErrorProne, CodeStyle, UnusedCode, Security, Compatibility, Performance, Documentation.

#### PT Application Inspector

PT AF [[48](#PTAF)] – объединяет статическое тестирование безопасности приложений (SAST), динамическое тестирование безопасности приложений (DAST), интерактивное тестирование (IAST), анализ состава программного обеспечения (SCA), а также сопоставление отпечатков пальцев и шаблонов. Имеет возможность генерировать тестовые запросы (эксплойты) для проверки обнаруженных уязвимостей во время анализа SAST и функции автоматической проверки уязвимостей, фильтрации, инкрементного сканирования и создания интерактивной диаграммы потока данных (DFD) для каждой уязвимости.

PT Application Inspector [[49](#PTAplication)] – инструмент PT AF, который анализирует исходный код или готовое приложение, комбинируя статические, динамические и интерактивные методы (SAST, DAST и IAST). По итогам анализа инструмент выдает данные о номере строки и типе уязвимости и генерирует эксплойты – безопасные, но эффективные тестовые запросы, которые помогают подтвердить или опровергнуть наличие уязвимости. Благодаря модулю динамического анализа эти запросы запускаются на любом тестовом стенде, в том числе в автоматическом режиме.

Особенностью PT Application Inspector является автоматическая выгрузка результатов анализа в межсетевой экран уровня веб-приложений PT Application Firewall, который применяет правила для блокировки атак на уязвимое приложение на время исправления кода.

##### Возможности интегрирования с другими продуктами

Интегрируется с наиболее популярными системами отслеживания ошибок, CI / CD и системами контроля версий (Jira [[50](#Jira)], Jenkins, TeamCity [[51](#TeamCity)]). Автоматического экспортирует отчеты об уязвимостях в PT Application Firewall. Имеет возможность интеграции с продуктами Positive Technologies, включая MaxPatrol [[52](#MaxPatrol)].

Поддерживаемые языки: Java, C#, PHP, JavaScript, Objective C, VB.Net, PL / SQL, T-SQL.

##### Конфигурации

PT Application Inspector адаптируется к логике работы приложений благодаря возможности создавать собственные шаблоны для поиска. Заложив бизнес-логику приложения в систему, можно выявлять недостатки функционирования приложения, а также закладки, оставленные в коде разработчиками или хакерами.

##### Генерация отчета

Генерация эксплойта дает точную картинку возможного риска обнаруживает уязвимости и программные закладки (бэкдоры) по шаблону путем выявления похожей логики или схожего синтаксиса.

#### CAST AIP

CAST Application Intelligence Platform (AIP) [[53](#CastAIP)] – это инструмент измерения и улучшения качества прикладного программного обеспечения для мультиплатформенных, многоязычных приложений. CAST AIP хранит большой объем данных о приложениях и предоставляет результаты через интерфейсы, называемые CAST Application Analytics Dashboard [[54](#CastAAD)] (CAST AAD) и CAST Application Engineering Dashboard – CAST AED [[55](#CastAED)]. Поддерживает множество языков локализации, включая русский.

CAST AIP обеспечивает поддержку широкого спектра правил безопасности, установленных ведущими отраслевыми исследованиями и стандартами по уязвимостям безопасности, таких как OWASP Топ 10.

Данное программное обеспечение поддерживает архитектуру безопасности, предоставляя информацию о структуре приложения, позволяя определять пользовательские правила с помощью средства проверки архитектуры, а также отслеживать и контролировать эти правила в последующей разработке приложения.

##### Возможности интегрирования с другими продуктами

Интегрируется со следующим программным обеспечением с открытым исходным кодом: Apache Commons, jquery, lxml и многими другими.

Поддерживаемые языки: C/C++, .NET, ASP, Visual Basic, Universal Analyzer/Universal Importer, PL/SQL, T-SQL.

##### Конфигурации

CAST Application Engineering Dashboard имеет интерфейс на основе «плитки» или «панели», то есть представление интерфейса в виде нескольких прямоугольников с информацией, который обеспечивает доступ к данным. Эти плитки имеют настройки по умолчанию, но их можно изменить и настроить в соответствии с вашими требованиями. Например:

* цвет плитки можно изменить;
* позиции плитки могут быть изменены;
* содержимое плитки может быть изменено;
* размер плитки можно изменить;
* новые плитки могут быть добавлены.

Представление плитки по умолчанию определено в следующем файле:

% CATALINA\_HOME% \ WebApps \ CAST-AED \ инженерия \ Resources \ ced.json

Чтобы изменить презентацию, отредактируйте файл .json с помощью текстового редактора. CAST рекомендует: Создать копии файлов .json по умолчанию, которые можно использовать в качестве замены, если во время редактирования возникает ошибка, затем сначала изменить существующие параметры, такие как цвет и заголовок, а также переместить плитки, прежде чем приступить к крупномасштабным и более сложным изменениям.

Нет необходимости перезагружать сервер приложений или само приложение, изменения сохраняются сразу после сохранения файла. Но CAST рекомендует очистить кэш браузера, чтобы изменения стали видимыми. Пример конфигурации плитки, которая может быть изменена:

{

"type": "MockTile",

"color": "",

"parameters": {},

"col": 1,

"row": 5,

"sizex": 6,

"sizey": 1,

"max-sizex": 8,

"max-sizey": 6,

"title": "Additional Information",

"id": "c928"}

Также, изменяя конфигурации, есть возможность настроить режим аутентификации, изменяя файл users.properties. Отключение пользователя без удаления его из файла users.properties. Чтобы отключить пользователя, измените параметр enabled на disabled:

jhu=mypassword,disabled,NO\_ROLE

##### Генерация отчета

Генератор отчетов CAST является автономным решением для автоматической генерации отчетов на основе данных, генерируемых CAST AIP. Решение предоставляет возможность подготовить и автоматизировать оценки для каждой новой версии приложения, проанализированной с помощью CAST AIP. Документы основаны на шаблонах Microsoft Office и могут быть изменены для подготовки конкретного шаблона в соответствии с конкретным вариантом использования или в соответствии с форматом компании. После генерации полученный документ может быть дополнительно адаптирован при необходимости. Он взаимодействует с данными, хранящимися в схеме CAST Dashboard для создания различных отчетов Word и PowerPoint из предопределенных или пользовательских шаблонов.

#### CodeSonar

CodeSonar [[56](#CodeSonar)] – инструмент, который обнаруживает дефекты, с набором комплексных средств проверки, которые включают анализ параллелизма и анализ испорченных данных. Этот инструмент обеспечивает глубокий статический анализ, обнаруживая ошибки в использовании статической памяти, неправильного управления ресурсами и дефектов параллелизма.

Исследуя как исходный код, так и двоичные файлы, CodeSonar позволяет анализировать готовые приложения, позволяя контролировать цепочку поставок программного обеспечения и устранять наиболее дорогостоящие и трудно обнаруживаемые дефекты в начале SDLC (жизненный цикл разработки ПО). Может выполнять анализ всей программы на 10M+ строк кода. После того, как начальный базовый анализ был выполнен, функция инкрементального анализа CodeSonar позволяет быстро анализировать ежедневные изменения в кодовой базе. Анализ может выполняться параллельно.

##### Возможности интегрирования с другими продуктами

Поддерживаемые компиляторы: Apple xcode, GCC, Intel C/C++, MS Visual Studio и многие другие.

Поддерживаемые языки: C, C++, C#, Java, Binaries.

##### Конфигурации

Так как CodeSonar действует во время компиляции, ему нужно указать несколько параметров, например путь к классам для проекта. Для этого должен быть указан правильный тип BUILDTYPE. Поскольку необходимо выполнить компиляцию, правильный тип компилятора должен быть настроен для типа сборки.

'C' => {

...

'BUILDTYPE' => [

{'name' => 'Make', 'compiler' => ['Gcc']},

],

...

},

'CPP' => {

...

'BUILDTYPE' => [

{'name' => 'VCProj', 'compiler' => ['VC']},

],

...

}

CodeSonar имеет возможность обнаруживать проблемы разных классов. Все эти классы перечислены в CodeSonar установке в следующем файле: CodeSonar/codesonar/doc/WarningClasses/WarningClasses.html. Пользователь может выбрать любой набор классов, о которых следует сообщать TICS [[57](#TICS)]. Это делается путем редактирования RULES.txt и в IMPL.txt файлов в cfg каталоге на TICS FileServer. Пользователь также может свободно выбирать уровень серьезности, категорию и синопсис в RULES.txt. «Классы предупреждений» являются строками в столбце «Класс предупреждений» в WarningClasses. Краткий пример файла IMPL.txt:

CS # 1 CodeSonar Accept\_on\_socket\_in\_wrong\_state

CS # 2 CodeSonar Bind\_on\_socket\_in\_wrong\_state

CS # 3 CodeSonar Buffer\_Overrun

CS # 4 CodeSonar Buffer\_Underrun

CS # 5 CodeSonar Cast\_Alters\_Value

CS # 6 CodeSonar Connect\_on\_socket\_in\_wrong\_state

CS # 7 CodeSonar Dangerous\_Function\_Cast

CS # 8 CodeSonar Division\_By\_Zero

Стандартная CodeSonar конфигурация правила доступна в виде IMPL.txt, RULES.txt.

##### Генерация отчета

Отчет генерируется в удобной для пользователя форме в одном из следующих форматов: HTML, XML, Text.

#### Contrast Assess

Contrast Assess [[58](#ContrastAcess)] (Contrast Security) – инструмент анализа безопасности приложения, который выполняет защиту кода без фактического статического анализа. Contrast проводит интерактивное тестирование безопасности приложений (IAST), сопоставляя код выполнения и анализ данных и предоставляет результаты, фактически не полагаясь на статический анализ. Обеспечивает анализ в реальном времени и постоянную видимость, поэтому есть возможность обнаруживать и устранять уязвимости практически без ложных срабатываний. Идеально подходит для Agile (гибкая методология разработки), DevOps (методология активного взаимодействия специалистов по разработке со специалистами по информационно-технологическому обслуживанию) и DevSecOps (автоматизация основных задачь безопасности, внедряя контроль этих процессов на раннем этапе DevOps).

Contrast Assess разработан для совместной работы с разработчиками при написании и тестировании веб-сайтов. Команды DevOps могут использовать Contrast со своими стандартными инструментами обмена сообщениями и сборки.

##### Возможности интегрирования с другими продуктами

Contrast Assess автоматически предоставляет анализ безопасности во время автоматизирования тестов. Данное ПО совиещается с инструментами непрерывной интеграции (CI), включая Jenkins, Maven, Gradle и Bamboo [[59](#Bamboo)]. Когда число уязвимостей указанной серьезности превышает пороговое значение, выбранное в интерфейсе, Contrast может дать команду инструменту CI не выполнить сборку.

##### Конфигурации

Данное ПО поддерживает конфигурацию на основе YAML для агента Java. Это позволяет сохранять конфигурацию на диске, которую можно переопределить с помощью переменных среды или аргументов командной строки. Хотя все агенты Contrast имеют одинаковое форматирование свойств в файлах конфигурации YAML, каждый агент должен использовать указанный файл.

Значения конфигурации используют следующий порядок приоритета (где 1 - самый высокий):

1. корпоративное правило (например, переопределенные лицензии с истекшим сроком действия assess.enable),
2. значение системного свойства,
3. значение переменной среды,
4. значение файла конфигурации YAML,
5. контрастность пользовательского интерфейса,
6. значение по умолчанию,
7. путь загрузки;

Можно установить путь к файлу конфигурации YAML, используя переменную среды CONTRAST\_CONFIG\_PATH или системное свойство Java contrast.config.path. Если переменная среды и системное свойство не заданы, агент ищет файл конфигурации YAML в расположении по умолчанию в Windows:

%ProgramData%\Contrast\java\contrast\_security.yaml

В файловых системах Unix / Linux:

/etc/contrast/java/contrast\_security.yaml

Также можно установить все следующие свойства YAML в качестве системных свойств. Для этого нужно извлечь ключ системного свойства из YAML, соединив каждый узел с "." пока не достигнете нижнего свойства.

Пример: Если необходимо переопределить URL-адрес для контрастного пользовательского интерфейса, о котором сообщает агент, можно установить его командой -Dcontrast.api.url=https://exampele.com/Contrast как системное свойство.

api:

url: https://example.com/Contrast

Есть возможность установить свойство enable : true для включения агента или false – для выключения агента.

Рассмотрим список возможных изменений конфигураций связанных с API:

url : установите URL для контрастного интерфейса.

Пример: https://app.contrastsecurity.com/Contrast.

api\_key : установите ключ API, необходимый для взаимодействия с пользовательским интерфейсом Contrast.

Список необходимых ключей:

service\_key : необходимый для связи с контрастным интерфейсом. Он используется для расчета заголовка авторизации.

user\_name : установите имя пользователя, используемое для связи с контрастным интерфейсом. Используется для расчета заголовка авторизации.

Рассмотрим список конфигураций необходимый для настройки прокси:

* enable : добавленное значение свойства, определяет, должен ли агент взаимодействовать с интерфейсом Contrast через прокси. Если значение свойства отсутствует, наличие действительного прокси-хоста и порта определяет включенное состояние. Значения параметров true или false;
* host : Для установки прокси-хоста. Он должен быть установлен с портом и схемой. Пример: localhost;
* port : устанавливает порт прокси. Должно быть установлено с хостом и схемой. Пример:1234;
* scheme : установить схему прокси, должна быть установлена с хостом и портом. Пример: http или https;
* URL : установить в качестве альтернативы для scheme://host:port. Он имеет приоритет над другими настройками, если они указаны; однако будет выдано сообщение об ошибке, если заданы как URI, так и отдельные свойства;
* user : установка прокси-пользователя;
* pass : пароль прокси;
* auth\_type : для установки типа аутентификации прокси. Вариантами значений являются: NTLM, Digest и Basic.

##### Генерация отчета

Данное ПО предоставляет множество информации, включая:

* точная идентификация уязвимости, которая поставляется с подробной информацией о коде и решениях для ее устранения
* общая оценка (информация о том, как работает приложение в целом);
* активность посещаемых URL;
* метрики использования приложения;
* результаты оценки библиотеки с открытым исходным кодом.

Существует два типа предупреждения, доступных в Contrast Assess.

Когда инструмент для анализа обнаруживает недостатки кода во время навигации по вашему приложению, и указывает, где была обнаружена уязвимость и как использовался код. Уязвимости представлены и классифицированы с серьезностью, чтобы помочь расставить приоритеты для следующих шагов.

Страница уязвимостей в интерфейсе Contrast позволяет искать, сортировать и анализировать уязвимости каждого приложения. Данное ПО предоставляет рекомендации по определению способов устранения и предотвращения атаки. Помимо электронной почты, Contrast может отправлять уведомления и оповещения через популярные каналы обмена сообщениями и связи, включая Slack.

#### Fortify

Fortify SCA [[60](#Fortify)] – автоматический статический анализ кода, помогающий находить и устранять уязвимости системы безопасности в режиме реального времени с помощью функции проверки орфографии решений Security Assistant в Eclipse или Visual Studio IDE. Охватывает 800 категорий уязвимостей для SAST, в соответствии со стандартами OWASP Top 10, CWE/SANS Top 25, DISA STIG. Существует возможность запускать сканирование локально с платформы Fortify или через свою среду IDE и конвейер CI/CD. У пользователя есть возможность создавать фильтры и выпускать шаблоны для представлений.

Audit Assistant сокращает время, затрачиваемое на ручной аудит, удаляя до 90% ложных срабатываний. Audit Workbench обеспечивает расширенный анализ и автоматическую приоритизацию.

С помощью фильтров SmartView есть возможность устранять проблемы в наиболее ключевых местах, которые отображаются как источник возникновения проблем с точки зрения потока данных.

##### Конфигурации

У пользователя есть возможность изменить готовые конфигурации данного ПО в файле fortify.properties, либо добавить новые из списка возможных, рассмотрим основные и наиболее полезные из них:

audit.ui.DisableAddingFolders – если установлено значение true, отключает функцию добавления папки.

audit.ui.DisableBugtrackers – если установлено значение true, отключает интеграцию с трекером ошибок.

AuthenticationKey – указывает каталог, в котором хранится токен аутентификации клиента Центра безопасности Micro Focus Fortify. По умолчанию:

${com.fortify.WorkingDirectory} / config / tools

Debug – если установлено значение true, инструменты Fortify Static Code Analyzer запускаются в режиме отладки.

DisableDescriptionXML – если установлено значение true, отключает экранирование XML в описаниях проблем.

WorkingDirectory – задает рабочий каталог, содержащий все пользовательские настройки и рабочие файлы для компонентов Fortify Static Code Analyzer. По умолчанию:

Windows:

${win32.LocalAppdata}/Fortify

Unix:

${user.home}/.fortify

##### Возможности интегрирования с другими продуктами

Fortify позволяет автоматизировать функции безопасности в среде CI/CD с помощью интерфейсов RESTful API с поддержкой Swagger, репозитория GitHub, плагинов для Bamboo, VSTS [[61](#VSTS)] и Jenkins, а также интеграции с инструментами анализа компонентов с открытым исходным кодом. Fortify SCA встраивается в существующие среды разработки через скрипты, плагины и инструменты с графическим интерфейсом для того, чтобы разработчики могли легко и быстро приступить к работе.

##### Генерация отчета

Можно создать файл для фильтрации определенных экземпляров уязвимостей, правил и категорий уязвимостей при запуске команды sourceanalyzer. Необходимо указать файл с помощью опции -filter analysis. Fortify рекомендует использовать файлы фильтров, только если опытным пользователям.

Файл фильтра – это текстовый файл, который можно создать в любом текстовом редакторе. Файл работает как черный список, в котором указаны только те элементы фильтра, которые не нужны при поиске. Каждый элемент фильтра находится на отдельной строке в файле фильтра. Возможно указать следующие типы фильтров:

* категория;
* ID экземпляра;
* ID правила.

Фильтры применяются в разное время в процессе анализа в зависимости от типа фильтра. Fortify Static Code Analyzer применяет фильтры категории и идентификатора правила на этапе инициализации до начала любого сканирования, тогда как фильтр идентификатора экземпляра применяется после этапа анализа.

Чтобы применить фильтрацию в файле test\_filter.txt, необходимо выполнить следующую команду:

sourceanalyzer -b eightball -scan -filter test\_filter.txt

#### Julia

Julia [[62](#Julia)] – это статический анализатор, предназначенный для приложений Java, Android и C #. Он опирается на формальную теорию абстрактной интерпретации, которая позволяет полностью автоматизировать семантический анализ программ. Юлия анализирует байт-код, что позволяет также анализировать код библиотеки.

Данный анализатор кода вычисляет набор методов и конструкторов, которые должны вызываться пользователем анализируемой программы. Они называются точками входа. Это означает, что статический анализ выполняется так, как если бы точки входа были единственными методами, которые могут быть вызваны пользователем кода. Конечно, если точки входа вызывают другие методы, эти другие методы также анализируются и так далее. Этот рекурсивный процесс называется извлечением достижимой части анализируемого приложения.

##### Конфигурации

У пользователя есть возможность изменить готовые конфигурации данного ПО, например, указать точку входа в приложение, с которого начинается анализ.

Имя: entryPoint

Значение по умолчанию: ALL\_ENTRIES

Список возможных значений:

* ALL\_ENTRIES: открытые и защищенные методы и конструкторы являются точками входа;
* ALL\_METHODS: открытые, защищенные и закрытые методы и конструкторы являются точками входа;
* LIBRARY: все открытые и защищенные методы и конструкторы являются точками входа, и предположим, что не финальные классы могут быть переопределены в будущем;
* ONLY\_EXPLICIT\_ENTRIES: Учитывайте только точки входа, помеченные как @EntryPoint;
* ONLY\_STANDARD\_ENTRIES: Учитывайте только точки входа по умолчанию, такие как основные методы, обработчики;

Также пользователь может изменить тип анализируемой системы в параметре framework, установив таким образом «рамки анализа»:

Рассмотрим часть списка возможных значений:

* androidAPI1: использует рамки анализа для Android API 1;
* androidAPI10: использует рамки анализа для Android API 10;
* androidAPI11: использует рамки анализа для Android API 11;
* java1: использует рамки анализа для Java 1.1;
* java10: использует рамки анализа для Java 1.10;
* java11: использует рамки анализа для Java 1.11;
* java2: использует рамки анализа для Java 1.2;
* java3: использует рамки анализа для Java 1.3.

И кроме того, существует возможность изменить опции режима обработки библиотеки в поле libraryProcessingMode, имеющем значение по умолчанию: TYPES, то есть разрешить анализатору отказаться от реализации библиотечных методов и использовать только иерархию библиотечных классов во время анализа. Это снижает точность анализа и может иногда сделать его несостоятельным, но сокращает время анализа.

Возможные значения:

* TYPES: игнорировать код неисполняемых библиотек;
* TYPES\_AND\_CODE: включить код всех классов во время анализа.

##### Возможности интегрирования с другими продуктами

Существует целый ряд платформ, поддерживаемых анализатором Julia, рассмотрим часть из них и особенности работы с ними:

Статический анализатор будет переводить аннотации Apache-CXF [[63](#ApacheCXF)] в свои собственные, отслеживая в основном новые точки входа для анализа и ненадежные данные, поступающие из Интернета, то есть механизм анализа будет реагировать соответствующим образом:

* если элемент помечен org.apache.cxf.jaxrs.ext.Nullable, Джулия считает, что он также помечен: com.juliasoft.julia.checkers.nullness.Nullable;
* если элемент помечен org.apache.cxf.jaxrs.ext.PATCH, Джулия считает, что он также помечен: com.juliasoft.julia.extraction.EntryPoint;
* если элемент помечен org.apache.cxf.jaxrs.ext.multipart.Multipart, Джулия считает, что он также помечен: com.juliasoft.julia.checkers.flows.UntrustedUserInput.

Julia будет переводить аннотации FindBugs в свои собственные, преобразовывая информацию для анализа FindBugs в соответствующую информацию Julia.

Анализатор статического кода будет рассматривать в качестве точек входа все методы servlet и анализировать также код, который будет выполняться в течение жизненного цикла servlet. Обязательные библиотеки: javax.servlet-api.jar. Эта спецификация автоматически применяется, когда:

* рамки анализа содержат слово java(без учета регистра);
* класс, имя которого равно, javax.servlet.http.HttpServlet является супертипом хотя бы одного класса приложения.

В рамках Framework Unity, Julia будет рассматривать в качестве отправных точек для анализа стандартные методы MonoBehaviour подклассов, что приведет к анализу кода, который будет выполняться при графическом или физическом обновлении.

Эта спецификация автоматически применяется, когда:

* рамки анализа содержат слово dotnet (без учета регистра);
* класс, имя которого равно, UnityEngine.MonoBehaviour является супертипом хотя бы одного класса приложения.

##### Генерация отчета

Рассмотрим список предупреждений, которые могут генерироваться анализатором Julia. Каждое предупреждение сопровождается различными классификациями: некоторые относятся к Юлии (категории и степени серьезности), а другие – к другим стандартам (например, CWE ID). Любое из них может иметь несколько классификаций одного типа, в зависимости от контекста, в котором выдается предупреждение. Например, возвращение null-значения вместо массива, как правило, является проблемой стиля, но если это null-значение разыменовывается где-то в достижимом коде, это становится настоящей ошибкой.

* AbsOfRandomWarning: абсолютное значение случайного числа может быть отрицательным. Категория: Bug, Строгость: Minor, CWE ID: 682;
* ApproximateEWarning: Приблизительное значение E используется вместо константы в библиотеках. Категория: Bug, Строгость: Major, CWE ID: 197;
* AuthenticationSetToAnonymousWarning: Анонимная аутентификация LDAP ставит под угрозу безопасность. Категория: Bug, Строгость: Major, CWE ID: 287.

И множество других, полный список можно увидеть на официальном сайте данного продукта.

#### Klocwork

Klocwork Code Review [[64](#Klocwork)] – инструмент статического анализа кода, который помогает разрабатывать безопасный и надежный код.

Code Review может использоваться для:

* проверки кода либо до, либо после фиксации в SCM, либо в IDE (Visual Studio, Eclipse, IntelliJ IDEA [[65](#IntelliJ_IDEA)]) или из командной строки;
* для приглашения рецензентов на обязательной или необязательной основе;
* для создания пользовательских каналов, для наблюдения за пользователями, командами или компонентами.

Klocwork интегрируется со сборкой, собирая всю необходимую информацию, чтобы обеспечить централизованное представление всего потока кода. Он хранит эту информацию в структурированном журнале сборки – текстовом файле, который мы называем спецификацией сборки. Спецификация сборки является основой анализа Klocwork. Она требуется как на уровне интеграции, так и на локальном уровне сборки. Для интеграционной сборки создание спецификации сборки является первой важной задачей после установки.

Для некоторых сред интеграция сборки происходит автоматически; для других нужно создать спецификацию сборки вручную. Важно, чтобы спецификация сборки была точным и современным представлением. Если спецификация сборки является неточной, анализ Klocwork также будет неточным.

Команда kwinject используется для захвата настроек сборки почти для всех проектов C / C. Для других сред распределенной сборки, включая распределенный анализ Klocwork, используется kwwrap plus kwinject.

##### Конфигурации

Рассмотрим альтернативные методы сборки для Visual Studio. Если не выходит создать спецификацию сборки для проекта Visual Studio C / C ++ с помощью kwinject, есть возможность использовать альтернативный метод сборки. kwinject создает наиболее точную спецификацию сборки и является рекомендуемым методом интеграции сборки для всех проектов C / C ++, включая проекты Visual Studio, созданные из командной строки (с помощью devenv, msdev, vcbuild или msbuild). Однако если не удается создать спецификацию сборки для проекта Visual Studio C / C ++ с помощью kwinject, существуют альтернативные методы интеграции сборки.

Чтобы создать спецификацию сборки для проекта с файлом .dsp, .dsw, .vcproj или .sln, необходимо запустить kwvcprojparser.

Если проект Visual Studio содержит несколько платформ, необходимо указать полную конфигурацию, включая платформу. Например:

kwvcprojparser project.vcproj -c "Debug | Itanium"

Чтобы создать спецификацию сборки для нескольких файлов .dsp или .vcproj необходимо:

В Visual Studio создать новый файл рабочей области (решения) и добавить в него существующие проекты (файлы .dsp или .vcproj).

Убедиться в правильности настроек, создав проект Visual Studio. Это гарантирует, что проект правильно настроен, а любые файлы доступны инструментам Klocwork.

При необходимости определить зависимости между проектами. И использовать команду kwvcprojparser для генерации спецификации сборки.

Настройка C / C ++ или Java-анализа позволяет повысить точность анализа Klocwork путем обнаружения более реальных дефектов или уменьшения количества ложных срабатываний.

Добавление условно скомпилированного кода в исходный код – очень знакомый способ работы с различными средами. Макрос \_\_KLOCWORK\_\_, который всегда определяется компилятором Klocwork, позволяет определенному коду быть активным только во время работы компилятора и в других ситуациях игнорируется. Добавление альтернативного определения макроса типа assert в блок \_\_KLOCWORK\_\_ может быть незаметным и простым в управлении.

Например, предположим, что есть следующий код:

MY\_ASSERT(a!= NULL);

use(a->mm);

Если Klocwork неправильно обрабатывает производственное определение MY\_ASSERT, так что он не понимает условия прерывания, он может сообщить о ложном NPD в строке 2. Можно уменьшить вероятность этого, предоставив Klocwork стандартное определение:

#ifdef \_\_KLOCWORK\_\_

# define MY\_ASSERT(x) do { if (!(x)) abort(); }while(0)

#else

# define MY\_ASSERT(x) ... production, advanced definition of assert ...

#endif

##### Возможности интегрирования с другими продуктами

Плагин Klocwork Desktop для Visual Studio, Eclipse и IntelliJ IDEA помогает обнаруживать и устранять проблемы перед регистрацией. Плагин Klocwork Desktop для Visual Studio поддерживает C / C ++, C # и смешанные проекты и решения, Klocwork Desktop для Eclipse поддерживает C / C ++ и Java, Klocwork Desktop для IntelliJ IDEA поддерживает Java.

Klocwork Desktop предлагает графический интерфейс, который можно использовать в дополнение к любым IDE или текстовым редакторам, которые не поддерживаются в виде подключаемого модуля IDE.

##### Генерация отчета

Отчеты Klocwork предоставляют инструмент, который позволяет увидеть разбивку текущих состояний для всех обзоров кода. Их также можно использовать, чтобы отслеживать прогресс отдельного пользователя или команды. Они могут создаваться со следующими объектами:

* Action: позволяет увидеть количество элементов действий, назначенных для проверок кода;
* Review: позволяет видеть состояния активных обзоров кода;
* Revision: позволяет увидеть количество ревизий в любом конкретном файле;
* Comment: позволяет увидеть количество комментариев, сделанных к обзорам кода.

#### Parasoft Test

Parasoft Test [[66](#Parasoft)] – статический анализатор кода приложений, который использует несколько типов анализа для охвата большего числа уязвимостей.

Анализ на основе шаблонов обнаруживает конструкции в исходном коде на основе стандартов программирования, таких как CWE и OWASP, которые, приводят к дефектам программного обеспечения. Статический анализ на основе шаблонов помогает гарантировать, что разработчики следуют передовым методам кодирования, модульного тестирования, а также политике развития организации.

Анализ дублирования кода C / C ++ testcan проверяет наличие дублирующего кода, чтобы помочь пользователям улучшить дизайн приложения и снизить затраты на обслуживание. Во время анализа код разбирается на более мелкие языковые элементы (токены). Токены анализируются в соответствии с набором правил, которые определяют, что следует считать дублирующим кодом. Существует два типа правил для анализа токенов:

Простые правила поиска одинарных дубликатов токенов, например строковых литералов

Сложные правила поиска нескольких дубликатов токенов, например, дубликаты методов или операторов

Анализ метрики – тест C / C ++ может вычислить несколько метрик кода которые могут помочь вам понять потенциальные слабые места в коде, таких как сложность кода, связь между объектами или отсутствие единства.

Анализ потока – это тип технологии статического анализа, который использует несколько методов, включая моделирование путей выполнения приложения, для определения путей, которые могут вызвать дефекты времени выполнения. Обнаруженные дефекты включают использование неинициализированной памяти, разыменование нулевого указателя, деление на ноль, утечки памяти и ресурсов. Поскольку этот анализ включает в себя выявление и отслеживание сложных путей, он выявляет ошибки, уклоняются от статического анализа кода и модульного тестирования.

Способность Flow Analysis – выявлять ошибки без выполнения кода особенно полезна для пользователей с унаследованными базами кода и встроенным кодом (где обнаружение таких ошибок во время выполнения неэффективно или невозможно).

##### Конфигурации

Каждый тестовый прогон из продукта семейства Parasoft Test – выполняется ли он в графическом интерфейсе или из интерфейса командной строки – основан на тестовой конфигурации, которая определяет тестовый сценарий и устанавливает все связанные параметры теста (например, для статического анализа, выполнения теста сканирование кода, обнаружение ошибок во время выполнения). Чтобы изменить способ проведения теста, необходимо изменить параметры конфигурации теста, которую планируется использовать. Например, чтобы изменить правила, проверенные во время статического анализа, необходимо изменять настройки в связанной конфигурации теста статического анализа.

Каждый тестовый продукт Parasoft предоставляет встроенные тестовые конфигурации, основанные на различных популярных тестовых сценариях. Однако, поскольку проекты по разработке и приоритеты команды различаются, часто используются пользовательские конфигурации тестов.

Тестовые конфигурации по умолчанию, которые находятся в категории «Встроенные», не могут быть изменены. Способ создания настраиваемой тестовой конфигурации – это скопировать встроенную тестовую конфигурацию в пользовательскую категорию, а затем изменить скопированную тестовую конфигурацию в соответствии с предпочтениями и средой. Кроме того, есть возможность создать новую тестовую конфигурацию, а затем изменить ее при необходимости.

Чтобы создать пользовательскую конфигурацию теста необходимо:

* Открыть панель «Конфигурации теста», выбрав «Parasoft» -> « Test configurations».
* выполните одно из следующих: действий: если нужно основать пользовательскую конфигурацию теста на встроенной конфигурации теста или на доступной конфигурации Team Test необходимо – щелкнуть правой кнопкой мыши эту конфигурацию теста и выбрать «Duplicate». Если требуется создать собственную тестовую конфигурацию с нуля – нажать New.

##### Возможности интегрирования с другими продуктами

Плагин Parasoft Findings для Jenkins позволяет визуализировать статический анализ и результаты тестирования в Jenkins. Он преобразует XML-отчеты, сгенерированные продуктами Parasoft, в графики тенденций и позволяет удобно просматривать детали или легко переходить к документации правил. Плагин можно использовать с заданиями Freestyle, Maven и Pipeline.

Также Parasoft Findings позволяет визуализировать результаты тестов, сгенерированных продуктами Parasoft в Bamboo. Он включает в себя две задачи, которые преобразуют XML-отчеты Parasoft в графики, чтобы можно было удобно просматривать подробности теста. Еще одной функцией расширения Parasoft Findings является сбор результатов тестов из XML-файлов отчетов Parasoft и отображение количества пройденных / неудачных тестов из результатов Visual Studio Team Services. Результаты также могут отображаться в виде трендовых данных на графике для простой визуализации.

##### Генерация отчета

Когда анализ завершен, результаты могут быть рассмотрены:

* на консоли - итоговый результат выводится на консоль вывода;
* в файле XML - результаты всегда сохраняются в файле XML;
* в HTML (по умолчанию), PDF или в специальном отчете о расширении;
* на DTP – если публикация результатов в DTP включена.

## Тестирование средств статического анализа кода

### OWASP Benchmark

Для тестирования работоспособности статических анализаторов кода компания OWASP разработала OWASP Benchmark Project [[67](#BenchProj)]. Это набор тестов Java, разработанный для оценки точности, охвата и скорости автоматизированных программных средств обнаружения уязвимостей, его можно скачать с официального репозитория OWASP Benchmark на Github [[68](#Benchmark_Git)], на котором он имеет рейтинг 234 и стал основой еще для 143 других проектов.

OWASP Benchmark – это полностью работающее веб-приложение с открытым исходным кодом, которое содержит тысячи пригодных для использования тестовых случаев, каждый из которых сопоставлен с определенными CWE, которые можно анализировать с помощью любого типа инструмента тестирования безопасности приложений (AST), включая SAST, DAST, и инструменты IAST.

Тест OWASP ориентирован только на Java. Он состоит из 2740 тестовых случаев, хранящихся в одном каталоге с именем «testcode». Ожидаемые результаты описаны в файле CSV. Для каждого теста это:

* на какой тип уязвимости она нацелена;
* ожидается ли проблема;
* идентификатор CWE, связанный с контрольным примером.

Тест охватывает 11 типов уязвимостей, которые сгруппированы в 2 набора:

Уязвимости инъекций (6):

* SQL-инъекция: 504 тестовых случая;
* обход пути: 268 тестовых случаев;
* инъекция LDAP: 59 тестовых случаев;
* инъекция команды: 251 контрольных примеров;
* XPath инъекции: 35 тестовых случаев;
* межсайтовый скрипт: 455 тестовых случаев.

Неинъекционные уязвимости (5):

* криптография: 246 тестовых случаев;
* хеширование: 236 тестовых случаев;
* Secure Cookie: 67 тестовых случаев;
* граница доверия: 126 тестов;
* слабое случайное число: 493 тестовых случая.

Каждый тестовый пример – это сервлет Java EE, интерфейс, реализация которого расширяет функциональные возможности сервера. BenchmarkTest00001 в версии 1.0 Benchmark был тестом внедрения LDAP, протокол, использующий TCP/IP, с метаданными в прилагаемом файле BenchmarkTest00001.xml.

BenchmarkTest00001.java в OWASP Benchmark 1.0 считывает все значения cookie, ищет cookie с именем «foo» и использует это значение при выполнении запроса LDAP.

На диаграмме sourceCode – программы OWASP Benchmark для отображения результатов тестирования, представлены общие результаты для проверенных инструментов. Оценка для каждого инструмента – это общий истинный положительный показатель (TPR – вертикальная ось см. Рисунок 1.) по всем категориям испытаний, за вычетом общего ложного положительного показателя (FPR – горизонтальная ось). Таким образом, чем выше и левее инструментальные баллы, тем выше точность.

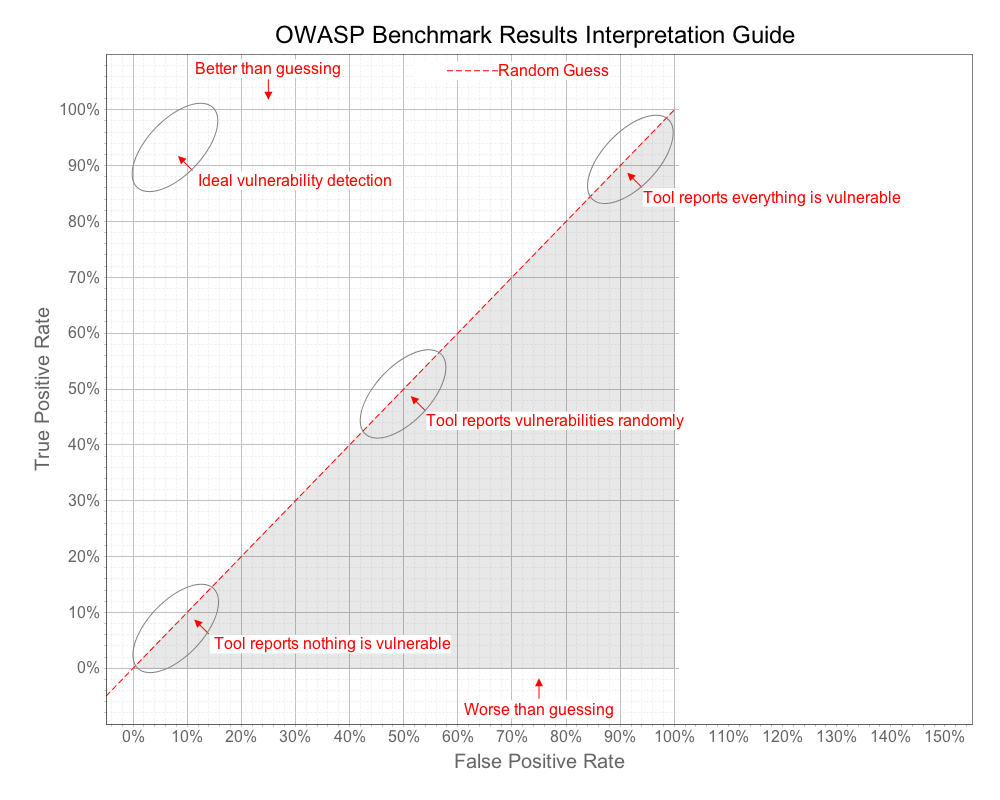


Рисунок : Исследование эффективности инструмента

Существует несколько основных параметров, на основании которых производится вычисление точности инструментов для последующего сравнения. В первую очередь необходимо вычислить следующие показатели: True Positive (TP) – количество тестов, в которых инструмент правильно определил реальную уязвимость, True Negative (TN) – количество тестов, в которых инструмент правильно игнорирует ложную тревогу, False Negative (FN) – количество тестов, в которых инструмент не смог обнаружить реальную уязвимость этот параметр также известен в теории вероятности как ошибка второго рода, False Positive (FP) – число тестов, в которых инструмент неверно находит несуществующие уязвимости, этот параметр также встречается в теории вероятности и более известен как ошибка первого рода. Из-за возможности ложных срабатываний не удаётся полностью автоматизировать борьбу со многими видами угроз. Как правило, вероятность ложного срабатывания коррелирует с вероятностью пропуска события (ошибки второго рода). То есть: чем более чувствительна система, тем больше опасных событий она находит и, следовательно, предотвращает. Но при повышении чувствительности неизбежно вырастает и вероятность ложных срабатываний. Поэтому чересчур чувствительно настроенная система защиты может выродиться в свою противоположность и привести к тому, что вред от неё будет превышать пользу.

Благодаря найденным значениям, появляется возможность вычислить параметры, с помощью которых scorecard и оценивает эффективность инструмента, рассмотрим их подробнее:

* True Positive Rate (TPR) = (TP / (TP + FN)) \* 100%. – процент точности, с которой инструмент правильно сообщает о реальных уязвимостях.
* False Positive Rate (FPR) = (FP / (FP + TN)) \* 100% – показывает процент неверных срабатываний инструмента к общему числу ложных тревог.

В статье Bazhenov.me [[69](#Bazhenov)] подробно описаны эти понятия: чувствительность (TPR) и выпадение (FPR) являются метриками, которые используются при оценке большей части алгоритмов обнаружения уязвимостей. Чувствительность системы – это процент найденных инструментом тестов, содержащих уязвимость, относительно всех тестов в выборке, формула для вычисления: TPR = (TP / (TP + FN)) \* 100%. Выпадение – это процент найденных инструментом тестов, в которых неверно определена уязвимость, относительно числа всех ложных тревог, формула для вычисления: FPR = (FP / (FP + TN)) \* 100%.

По этим двух характеристикам вычисляется последняя величина, необходимая для определения уровня эффективности инструмента, называемая Score и равная нормированному расстоянию от диагональной линии зависимости TPR и FPR до ROC – кривой. ROC-кривая получается следующим образом: для каждого последующего значения в тестах рассчитываются значения TPR и FPR и строится график зависимости: по оси Y откладывается TPR, по оси X – FPR. Подробный график см. Рисунок 2 (кривая; оранжевым показан идеальный алгоритм, фиолетовым — типичный, а синим — худший): Идеальный алгоритм проходит через точку (0;1), худший — прямая TPR = FPR.

Таким образом, Score = (TPR – FPR) – oбщая эффективность инструмента. Точка идеального инструмента должна быть в верхнем левом углу см. Рисунок 1 (находит все и не выдает ложных сигналов).

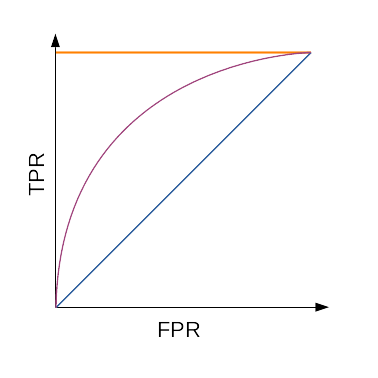


Рисунок 2: ROC-кривая

Рассмотрим работу некоторых инструментов с тестами Benchmark:

Benchmark включает в себя бесплатный инструмент PMD. Чтобы применить его к существующим тестам необходимо запустить скрипт: ./script/runPMD.sh. И отчет будет сгенерирован в файле /results в формате xml. В отчете содержится указание общие ошибки, такие как неиспользуемые переменные, пустые блоки catch, создание ненужных объектов, как и уверяли разработчики данного продукта. Отчет хранится в виде XML - файла в папке results.

Время работы на заданных тестах 14.394 секунд. Без указания точного номера строки не всегда понятно, где именно находится ошибка в коде. Кроме того, PMD способен обнаруживать только общие ошибки и не дает никаких предупреждений по поводу большинства проблем безопасности из OWASP TOP 10.

Аналогично, запустив скрипт: ./script/runSpotBugs.sh, можно проверить работоспособность статического анализатора кода SpotBugs. Время обработки тестов 1 минута 33 секунды. Отчет анализатора будет находится в папке /results в формате xml. SpotBugs, в отличие от PMD обнаруживает реальные угрозы безопасности в исходном коде Benchmark отчет хранится в файле “results”.

SpotBugs в отчете дает точное наименование ожидаемой угрозы безопасности, а также точное указание функции и строк кода, в которых содержится уязвимость, помимо угроз безопасности этот инструмент также дает советы по исправлению некоторых общих ошибок.

Полученные результаты по всем инструментам OWASP Benchmark представляет в виде графика в файле benchmark\_comparison.png, где видно по каждому инструменту, насколько успешно тест находил разного рода уязвимости и какое количество ложных срабатываний было, в процентном соотношении. По результатам этого графика: PMD – 0% обнаруженных угроз и FindBugs с плагином FindSecBugs – 39%. Также Benchmark создает графики сравнения всех инструментов по каждой уязвимости из OWASP TOP 10 отдельно.

Для создания подобного графика необходимо загрузить отчет выбранного инструмента в папку к остальным результатам и запустить генератор Benchmark Scorecard.

Анализатор кода VSG при обработке тестов Benchmark производит анализ на протяжении трех минут и в результате выводит окно с возможными уязвимостями, по которому легко определить, где именно в коде располагаются уязвимости, также отчет можно сформировать в формате xml, html. Но Benchmark Scorecard не может обработать выходной файл VSG.

### Использование OWASP Benchmark на CentOS-7.

Скачать OWASP Benchmark можно с официальной страницы OWASP на GitHub. Перед установкой необходимо убедиться, что установлена необходимая версия Java, начиная с 1.10, скачать ее с «http://www.oracle.com/technetwork/java/javase/downloads» после скачивания необходимо будет прописать пути в переменные окружения, чтобы запуск мог производиться из терминала. Для проверки версии Java необходимо использовать команду «java -version». Аналогично поступаем с Maven, необходима версия выше 3.1. Скачать можно с официального сайта Maven, после скачивания также необходимо прописать пути в переменные окружения, команда «mvn -v» - для проверки версии. После этого, чтобы скачать и собрать все вводим в терминал следующие команды:

$ git clone https://github.com/OWASP/benchmark

$ cd benchmark

$ mvn compile (Компилируем его)

$ runBenchmark.sh (Запускаем)

Затем необходимо перейти по адресу: https://localhost:8443/benchmark/, чтобы перейти на его домашнюю страницу. Он использует самозаверяющий сертификат SSL, поэтому вы получите предупреждение системы безопасности, когда попадете на домашнюю страницу. Рассмотрим подробнее процесс работы данного приложения:

После запуска команды runBenchmark.sh идет процесс компиляции и запуска, который, в случае если программа завершается успешно, говорит, что Benchmark готов к работе. Запуск приложения вызывает создание локального сервера c файлом «LDAPServer.java», где будет происходить перехват событий при обращении к серверу и производится регистрация трех пользователей:

* с именем «foo», паролем «MrFooPa$$word» и адресом «AddressForFoo #345».
* с именем «Ms Bar», паролем «barM$B4dPass» и адресом «The streetz 4 Ms bar».
* с именем «Mr Unknown», паролем «YouwontGue$$» и адресом «Whe home is #678».

Именно их данные и будут использоваться в тестах Benchmark. После этого происходит инициализация базы данных benchmark.db, находящейся по адресу: «comp/env/jdbc/BenchmarkDB», файлом «DatabaseHelper.java», где создается несколько таблиц USERS, SCORE, EMPLOYEE, CERTIFICATE. И помещают в них следующие параметры:

executeSQLCommand("INSERT INTO USERS (username, password) VALUES('User01', 'P455w0rd')");

executeSQLCommand("INSERT INTO USERS (username, password) VALUES('User02', 'B3nchM3rk')");

executeSQLCommand("INSERT INTO USERS (username, password) VALUES('User03', 'a$c11')");

executeSQLCommand("INSERT INTO USERS (username, password) VALUES('foo', 'bar')");

executeSQLCommand("INSERT INTO SCORE (nick, score) VALUES('User03', 155)");

executeSQLCommand("INSERT INTO SCORE (nick, score) VALUES('foo', 40)");

executeSQLCommand("INSERT INTO EMPLOYEE (first\_name, last\_name, salary) VALUES('foo', 'bar', 100)");

На этих параметрах и будет основан поиск уязвимостей с помощью тестов. Также в данном файле происходит обработка ошибок, связанных с базой данных и вывод сообщений о работе с ней. После этого на локальном хосте будет доступна страница с выбором возможных уязвимостей и их эксплуатацией. В файле «DataBaseServer.java» осуществляется управление обращением к базе данных, для извлечения необходимых параметров.

Рассмотрим подробнее внутреннюю структуру и работу тестов Benchmark по каждому типу уязвимостей отдельно:

### CWE-89: SQL Injection

Атака SQL-инъекции состоит из вставки или «внедрения» либо частичного, либо полного SQL-запроса посредством ввода данных или передачи их от клиента (браузера) в веб-приложение. Успешная атака с использованием SQL-инъекции может считывать конфиденциальные данные из базы данных, изменять данные базы данных (вставлять / обновлять / удалять), выполнять административные операции с базой данных (например, отключать СУБД), восстанавливать содержимое файла, существующего на сервере.

Первый шаг в тесте на эту уязвимость – понять, когда приложение взаимодействует с сервером БД, для получения доступа к данным. Типичные примеры случаев, когда приложение должно общаться с БД, включают:

* формы аутентификации: когда аутентификация выполняется с использованием веб-формы, то есть учетные данные пользователя проверяются по базе данных, которая содержит все имена пользователей и пароли;
* поисковые системы: строка, представленная пользователем, может использоваться в запросе SQL, который извлекает все соответствующие записи из базы данных;
* сайты электронной коммерции: продукты и их характеристики, могут храниться в базе данных.

Рассмотрим тест BenchmarkTest00008, он состоит из двух основных файлов BenchmarkTest00008.html, где расположена графическая часть с формой, через которую формируется запрос к базе данных, и внутреннего обработчика запросов, который находится в файле BenchmarkTest00008.java.

В html-файле располагается тег «input» с именем «BenchmarkTest00008» через него обеспечивается связь с базой данных, другие теги со вводом данных «input» и «textarea», расположенные выше, необходимы для заполнения страницы и не содержат уязвимостей, это также необходимо, чтобы лучше можно было проверить инструмент на ложные срабатывания, такие поля встречаются в каждом тесте Benchmark. После ввода данных в тег с именем text и нажатия на кнопку, производится вызов функции взаимодействия с формой «submitForm», через которую POST-запрос AJAX будет отправлен с заголовком BenchmarkTest00008 и значением, записанным в теге. После чего происходит ожидание отклика сервера и вывод соответствующего сообщения либо с ошибкой, либо с ответом. Взаимодействия осуществляются динамически с помощью объекта «XMLHttpRequest» из библиотеки jQuery [[70](#jquery)].

После этого происходит обработка отправленного сообщения в файле java: формируется строка sql-запроса:

String sql = "{call " + param + "}";

Где «param» – это пришедшее сообщение. После этого запрос направляется в базу данных и специальным методом в файле DataBaseHelper.java происходит вывод ответа на запрос из базы данных. Данные, извлеченные из базы, будут выведены в тег «div» с идентификатором «code», в случае неудачной связи с базой данных будет выведено сообщение об ошибке.

Также рассмотрим тест BenchmarkTest00018, он аналогично предыдущему состоит из двух файлов BenchmarkTest00018.html, где расположена графическая часть и внутреннего обработчика запросов BenchmarkTest00018.java.

Отправка сообщений базе данных осуществляется после нажатия на кнопку, созданную тегом input с именем «BenchmarkTest00018», другие теги со вводом данных «input» и «textarea», расположенные выше, необходимы для заполнения страницы и не содержат уязвимостей. После ввода данных в тег с именем text и нажатия на кнопку взаимодействия с формой, со значением value, POST-запрос AJAX будет отправлен с заголовком «BenchmarkTest00018» и значением, записанным в теге. После чего происходит ожидание отклика сервера и вывод соответствующего сообщения либо с ошибкой, либо с ответом. Взаимодействия осуществляются динамически с помощью объекта «XMLHttpRequest».

После этого происходит обработка отправленного сообщения в файле java: формируется строка sql-запроса:

"INSERT INTO users (username, password) VALUES ('foo','"+ param + "')";

Где «param» – это пришедшее сообщение. Заметим, что, в отличие от предыдущего теста, происходит экранирование вставляемого значения скобками и кавычками, что усложняет возможность внедрения SQL-инъекции. После этого запрос на добавление пользователя с именем «foo» и паролем param направляется в базу данных, в случае неудачной связи с базой данных будет выведено сообщение об ошибке с соответствующим кодом.

Тест BenchmarkTest00025 состоит из двух файлов BenchmarkTest00025.html, BenchmarkTest00025.java.

В html-файле располагается тег «select», где предполагается выбрать тип автомобиля и отправить запрос на сервер нажатием кнопки. После чего происходит ожидание ответа сервера и вывод сообщения либо с ошибкой, либо с ответом на запрос пользователя. Взаимодействия также осуществляются с помощью объекта XMLHttpRequest.

После нажатия кнопки происходит обработка отправленного сообщения в файле java: проверяется пустой ли запрос и если нет, то будет выполнен sql-запрос для базы данных:

"SELECT TOP 1 userid from USERS where USERNAME='foo' and PASSWORD='" + param + "'";

Где «param» – это пришедшее сообщение. На этот раз связь с базой данных осуществляется с помощью пакета JDBC из Springframework [[71](#Springfarmework)]. Заметим, что, также, как и в предыдущем запросе происходит экранирование вставляемого значения кавычками, но на этот раз без добавления скобок. После этого запрос на вывод одного идентификатора пользователя с именем «foo» и паролем «param» отправляется в базу данных и после этого уже с помощью специального метода в файле DataBaseHelper.java происходит вывод ответа на запрос из базы данных.

### CWE-77: Command Injection

Внедрение команд в запрос операционной системы – это атака, целью которой является выполнение определенных команд в операционной системе хоста через уязвимое приложение. Атаки с использованием команд возможны, когда приложение передает небезопасные пользовательские данные (формы, файлы cookie, заголовки HTTP) в системную оболочку. В этой атаке предоставленные злоумышленником команды операционной системы обычно выполняются с привилегиями уязвимого приложения. Атаки с использованием команд происходят в основном из-за недостаточной проверки ввода.

Рассмотрим тест BenchmarkTest00006, он состоит из двух основных частей: BenchmarkTest00006.html, где расположена графическая часть, через которую и будут отправляться сообщения для нашей операционной системы и обработчика, который находится в файле BenchmarkTest00006.java.

В html-файле располагается тег «input» с именем «BenchmarkTest00006» и значением по умолчанию «ECHOOO», через этот тег происходит отправка сообщений для обработчика в java-файле, другие теги со вводом данных «input» и «textarea», расположенные выше, необходимы для проверки на ложные срабатывания и не содержат уязвимостей. Взаимодействия будет осуществляться аналогично предыдущим с помощью объекта «XMLHttpRequest» из библиотеки jQuery.

После нажатия кнопки и успешной отправки серверу сообщения с заголовком «BenchmarkTest00006», происходит его обработка в файле java. Создается массив для последовательного формирования запроса, сохраняется в переменную название операционной системы, после чего в зависимости от того Windows это или нет, будут сформированы следующие команды:

if (osName.indexOf("Windows") != -1) {

argList.add("cmd.exe");

argList.add("/c");

} else {

argList.add("sh");

argList.add("-c");

}

После этого входные параметры, отправленные из формы, также добавляются в список для будущего запроса. Для отправки запроса операционной системе будет создан объект «ProcessBuilder», который используется для создания процессов операционной системы. Объект данного класса получает массив аргументов будущей команды и с помощью метода «start», создает новый процесс, выполняющий введенную команду, таким образом операционной системой, выполняется любая команда, переданная нашей программе. После этого, идентификатор процесса отправляется обработчику выполненного запроса в файле Utils.java, в котором происходит отправка сообщения с результатом о выполненной команде.

Также рассмотрим тест BenchmarkTest00007, он аналогично предыдущему состоит из двух основных частей: BenchmarkTest00007.html, с графической частью, и внутренней части для отправки запросов в операционную систему, которая находится в файле BenchmarkTest00007.java.

В html-файле располагается тег «input» с именем «BenchmarkTest00007 «и значением по умолчанию «FOO=echo Injection», через него и происходит отправка сообщений для обработчика в java-файле, по нажатии на кнопку «Login» происходит вызов из js-скрипта функции «submitForm», в которой взаимодействие с сервером будет осуществляться с помощью объекта «XMLHttpRequest».

Затем, после успешной отправки серверу сообщения с заголовком «BenchmarkTest00007», происходит инициализация запроса операционной системе в BenchmarkTest00007.java. В котором первоначально, с помощью метода «getInsecureOSCommandString» из файла Utils.java, создается строка, содержащая абсолютный путь из корневых каталогов до скрипта «insecureCmd», принимающего строку и печатающего ее в обратном порядке слов. После этого, происходит инициализация объекта «Runtime». Во время исполнения приложению Java сопоставляется экземпляр класса «Runtime». Этот объект позволяет взаимодействовать с окружением, в котором запущена Java-программа. Получить его можно с помощью статического метода «Runtime.getRuntime». Затем вызовом метода «exec» в отдельном процессе запускается скрипт «insecureCmd». с параметром в виде переданной строки. и возвращаемый идентификатор процесса отправляется обработчику запроса в файле Utils.java, в котором генерируется сообщение с результатом о выполненной команде для вывода на экран.

Также для примера, рассмотрим тест BenchmarkTest00077, который имеет такую же файловую структуру, как и рассмотренные ранее тесты. В html-файле, с названием BenchmarkTest00077.html располагается кнопка, после нажатия на которую, куки «BenchmarkTest00077», установленный сервером в данном тесте отправляется в обработчик в файле BenchmarkTest00077.java.

Рассмотрим обработку этих данных подробнее:

Первоначально происходит сбор и проверка на существование куки, если данных куки не обнаружено, в строку будущего запроса сохраняется «noCookieValueSupplied», в противном случае в эту строку сохраняются полученный куки в кодировке UTF-8. После чего происходит инициализация строки «ABC», по 3 символу которой в «case-блоке» в переменную bar сохраняется куки. После этого, в зависимости от операционной системы, в список будущего запроса сохраняются команды:

String osName = System.getProperty("os.name");

if (osName.indexOf("Windows") != -1) {

argList.add("cmd.exe");

argList.add("/c");

} else {

argList.add("sh");

argList.add("-c");

}

argList.add("echo " + bar);

Затем создается объект класса ProcessBuilder, который и отправит сформированный из списка команд запрос к операционной системе:

ProcessBuilder pb = new ProcessBuilder(argList);

try {

Process p = pb.start();

org.owasp.benchmark.helpers.Utils.printOSCommandResults(p, response);}

В случае успеха на html-странице будет выведен ответ с помощью обработчика в файле Utils.java, а в случае неудачи сгенерировано исключение и отображено сообщение об ошибке.

## Литература

1. OWASP. [Электронный ресурс] // Owasp.org, 2019. URL: [https://www.owasp.org](https://www.owasp.org/index.php/Benchmark) (дата обращения: 11.10.2019).
2. OWASP Top 2017 [Электронный ресурс] // Owasp.org, 2019. URL: https://owasp.org/www-pdf-archive/OWASP\_Top\_10-2017-ru.pdf (дата обращения: 11.10.2019).
3. Benchmark is a free and open test suite designed to evaluate the speed, coverage, and accuracy of automated software vulnerability detection tools and services. [Электронный ресурс] // Owasp.org, 2019. URL: <https://www.owasp.org/index.php/Benchmark> (дата обращения: 11.10.2019).
4. Bandit is a tool designed to find common security issues in Python code. [Электронный ресурс] // GitHub.com, 2019. URL: https://github.com/PyCQA/bandit (дата обращения: 11.10.2019).
5. The ast – module helps Python applications to process trees of the Python abstract syntax grammar. [Электронный ресурс] // Docs.python.org, 2019. URL: https://docs.python.org/2/library/ast.html (дата обращения: 11.10.2019).
6. Configuration Bandit. [Электронный ресурс] // Bandit.readthedocs.io, 2019. URL: https://bandit.readthedocs.io/en/latest/config.html (дата обращения: 11.10.2019).
7. YAML. [Электронный ресурс] // yaml.org, 2019. URL: https://yaml.org/ (дата обращения: 11.10.2019).
8. Bandit Test Plugins. [Электронный ресурс] // Bandit.readthedocs.io, 2019. URL: https://bandit.readthedocs.io/en/latest/plugins/index.html (дата обращения: 11.10.2019).
9. Brakeman is an open source vulnerability scanner specifically designed for Ruby on Rails applications. [Электронный ресурс] // GitHub.com, 2019. URL: https://github.com/presidentbeef/brakeman (дата обращения: 09.12.2019).
10. ALE. [Электронный ресурс] // GitHub.com, 2019. URL: https://github.com/dense-analysis/ale (дата обращения: 12.12.2019).
11. Hyperextensible is Vim-based text editor. [Электронный ресурс] // Neovim.io, 2019. URL: https://neovim.io/ (дата обращения: 12.12.2019).
12. Vim 8. [Электронный ресурс] // Vim.org, 2019. URL: https://www.vim.org/vim-8.1-released.php (дата обращения: 12.12.2019).
13. Pronto-brakeman. [Электронный ресурс] // GitHub.com, 2019. URL: https://github.com/prontolabs/pronto-brakeman (дата обращения: 12.12.2019).
14. Brakeman Confidence. [Электронный ресурс] // Brakemanscanner.org, 2019. URL: https://brakemanscanner.org/docs/confidence/ (дата обращения: 12.12.2019).
15. Pronto. [Электронный ресурс] // GitHub.com, 2019. URL: https://github.com/prontolabs/pronto (дата обращения: 12.12.2019).
16. Overcommit. [Электронный ресурс] // GitHub.com, 2019. URL: https://github.com/sds/overcommit (дата обращения: 12.12.2019).
17. Git Hooks. [Электронный ресурс] // GitHub.com, 2019. URL: https://git-scm.com/book/en/v2/Customizing-Git-Git-Hooks (дата обращения: 12.12.2019).
18. GolangCI-Lint is a Go Linters aggregator. [Электронный ресурс] // GitHub.com, 2019. URL: https://github.com/golangci/golangci-lint/blob/master/README.md (дата обращения: 12.12.2019).
19. GolangCI-Lint Configuration. [Электронный ресурс] // GitHub.com, 2019. URL: https://github.com/golangci/golangci-lint (дата обращения: 12.12.2019).
20. GolangCI-Lint Plugins. [Электронный ресурс] // GitHub.com, 2019. URL: https://github.com/alecthomas/SublimeLinter-contrib-golang-cilint (дата обращения: 12.12.2019).
21. LGTM is a free for open source static analysis service that automatically monitors commits to publicly accessible code. [Электронный ресурс] // GitHub.com, 2019. URL: https://github.com/Semmle (дата обращения: 14.12.2019).
22. LGTM Configuration. [Электронный ресурс] // GitHub.com, 2019. URL: https://lgtm.com/help/lgtm/configuring-lgtm-analysis-project (дата обращения: 14.12.2019).
23. PMD is a source code analyzer. [Электронный ресурс] // GitHub.com, 2019. URL: https://github.com/pmd/pmd (дата обращения: 14.12.2019).
24. PMD Configuration. [Электронный ресурс] // pmd.github.io, 2019. URL: https://pmd.github.io/latest/pmd\_userdocs\_configuring\_rules.html (дата обращения: 14.12.2019).
25. Puma Scan is a .NET C# open source static source code analyzer that runs as an IDE plugin for Visual Studio and via MSBuild in CI pipelines. [Электронный ресурс] // GitHub.com, 2019. URL: https://github.com/pumasecurity/puma-scan (дата обращения: 14.12.2019).
26. Puma Scan Configuration. [Электронный ресурс] // pumascan.com, 2019. URL: https://pumascan.com/configuration/ (дата обращения: 14.12.2019).
27. Jenkins. [Электронный ресурс] // Jenkins.io, 2019. URL: https://jenkins.io/ (дата обращения: 14.12.2019).
28. Pyre is a performant type-checker for Python 3. [Электронный ресурс] // GitHub.com, 2019. URL: https://github.com/facebook/pyre-check (дата обращения: 14.12.2019).
29. Taint Analysis is a popular method which consists to check which variables can be modified by the user input. [Электронный ресурс] // Mathnet.ru, 2019. URL: http://www.mathnet.ru/php/archive.phtml?wshow=paper&jrnid=tisp&paperid=431&option\_lang=rus (дата обращения: 14.12.2019).
30. Pyre Plugins. [Электронный ресурс] // GitHub.com, 2019. URL: https://github.com/facebook/pyre-check/tree/master/ide\_plugins/vscode (дата обращения: 14.12.2019).
31. RIPS Code Analysis is a SAST solution specialized for Java and PHP that detects unknown security vulnerabilities and code quality issues. [Электронный ресурс] // GitHub.com, 2019. URL: https://github.com/facebook/pyre-check (дата обращения: 14.12.2019).
32. Integrate Security Testing with GitHub Actions. [Электронный ресурс] // Blog.ripstech.com, 2019. URL: https://blog.ripstech.com/2019/integrate-security-testing-with-github-actions (дата обращения: 14.12.2019).
33. Scaling Security Testing to Large Teams. [Электронный ресурс] // Blog.ripstech.com, 2019. URL: https://blog.ripstech.com/2019/rips-3.3-data-center-edition-and-java-library-analysis (дата обращения: 14.12.2019).
34. BigTree. [Электронный ресурс] // Blog.ripstech.com, 2019. URL: https://blog.ripstech.com/2019/sql-injection-big-tree (дата обращения: 14.12.2019).
35. SpotBugs is the active fork replacement for FindBugs. [Электронный ресурс] // GitHub.com, 2019. URL: https://github.com/spotbugs/spotbugs (дата обращения: 14.12.2019).
36. SpotBugs Plugins. [Электронный ресурс] // Spotbugs.readthedocs.io, 2019. URL: https://spotbugs.readthedocs.io/en/latest/maven.html (дата обращения: 14.12.2019).
37. Eclipse. [Электронный ресурс] // Eclipse.org, 2019. URL: https://www.eclipse.org/ide (дата обращения: 14.12.2019).
38. Maven. [Электронный ресурс] // Maven.apache.org, 2019. URL: https://maven.apache.org/ (дата обращения: 14.12.2019).
39. Gradle. [Электронный ресурс] // Gradle.org, 2019. URL: https://gradle.org (дата обращения: 14.12.2019).
40. VisualCodeGrepper is scans C/C++, C#, VB, PHP, Java, and PL/SQL for security issues and for comments which may indicate defective code. [Электронный ресурс] // SourceForge.net, 2019. URL: https://sourceforge.net/projects/visualcodegrepp (дата обращения: 14.12.2019).
41. BlueClosure BC Detect is analyzes client-side JavaScript. [Электронный ресурс] // Blueclosure.com, 2020. URL: https://www.blueclosure.com/product/bc-detect (дата обращения: 7.02.2020).
42. DOM is platform and language-independent programming interface that allows programs and scripts to access the contents of HTML, XHTML and XML documents. [Электронный ресурс] // Wikipedia.org, 2020. URL: https://ru.wikipedia.org/wiki/Document\_Object\_Model (дата обращения: 7.02.2020).
43. Smart Fuzzer is module whose search is driven by a mix of static and dynamic analysis. [Электронный ресурс] // Researchgate.net, 2020. URL: https://www.researchgate.net/publication/4261951\_A\_Smart\_Fuzzer\_for\_x86\_Executables (дата обращения: 7.02.2020).
44. Codacy offers security patterns for languages such as Python, Ruby, Scala, Java, JavaScript. [Электронный ресурс] // Codacy.com, 2020. URL: https://www.codacy.com (дата обращения: 7.02.2020).
45. FindBugs is a program which uses static analysis to look for bugs in Java code. [Электронный ресурс] // Findbugs.sourceforge.net, 2020. URL: http://findbugs.sourceforge.net/ (дата обращения: 7.02.2020).
46. BitBucket is web service for hosting projects and their joint development. [Электронный ресурс] // Bitbucket.org, 2020. URL: https://bitbucket.org/ (дата обращения: 7.02.2020).
47. Gitlab is open source DevOps lifecycle web tool introducing code repository management system for Git from its own wiki. [Электронный ресурс] // Gitlab.com, 2020. URL: https://about.gitlab.com/pricing/ (дата обращения: 7.02.2020).
48. PT AF is web application firewall. [Электронный ресурс] // PTsecurity.com, 2020. URL: https://www.ptsecurity.com/ru-ru/products/af/ (дата обращения: 11.02.2020).
49. PT Application Inspector is the source code analyzer. [Электронный ресурс] // PTsecurity.com, 2020. URL: https://www.ptsecurity.com/ru-ru/products/isim-free-view (дата обращения: 11.02.2020).
50. Jira is development tool for agile teams. [Электронный ресурс] // Atlassian.com, 2020. URL: https://www.atlassian.com/ru/software/jira (дата обращения: 11.02.2020).
51. TeamCity is server software from JetBrains. [Электронный ресурс] // Jetbrains.com, 2020. URL: https://www.jetbrains.com/ru-ru/teamcity (дата обращения: 11.02.2020).
52. MaxPatrol 8 gives an unbiased picture of the state of protection at the system, department, node, and application levels. [Электронный ресурс] // PTsecurity.com, 2020. URL: https://www.ptsecurity.com/ru-ru/products/mp8 (дата обращения: 11.02.2020).
53. CAST AIP. [Электронный ресурс] // Castsoftware.com, 2020. URL: https://www.castsoftware.com/products/application-intelligence-platform (дата обращения: 21.02.2020).
54. CAST Application Analytics Dashboard. [Электронный ресурс] // Stimulsoft.com, 2020. URL: https://www.stimulsoft.com/en/products/dashboards (дата обращения: 21.02.2020).
55. CAST AED. [Электронный ресурс] // Xebialabs.com, 2020. URL: https://xebialabs.com/technology/cast-application-engineering-dashboard (дата обращения: 21.02.2020).
56. CodeSonar. [Электронный ресурс] // Grammatech.com, 2020. URL: https://www.grammatech.com/products/codesonar (дата обращения: 23.02.2020).
57. TICS. [Электронный ресурс] // Tiobe.com, 2020. URL: https://portal.tiobe.com/9.1/docs/install/install\_05\_fileserver.html (дата обращения: 23.02.2020).
58. Contrast Assess. [Электронный ресурс] // Contrastsecurity.com, 2020. URL: https://docs.contrastsecurity.com/installation-javaconfig.html (дата обращения: 24.02.2020).
59. Bamboo. [Электронный ресурс] // Atlassian.com, 2020. URL: https://www.atlassian.com/ru/software/bamboo (дата обращения: 24.02.2020).
60. Fortify SCA. [Электронный ресурс] // Microfocus.com, 2020. URL: https://www.microfocus.com/en-us/products/static-code-analysis-sast/overview (дата обращения: 24.02.2020).
61. Visual Studio Team System (VSTS) is a set of tools from Microsoft for the development of software applications. [Электронный ресурс] // Microsoft.com, 2020. URL: https://azure.microsoft.com/en-us/services/devops/ (дата обращения: 24.02.2020).
62. Julia. [Электронный ресурс] // Juliasoft.com, 2020. URL: https://juliasoft.com/company-js/ (дата обращения: 24.02.2020).
63. Apache-CXF. [Электронный ресурс] // Mvnrepository.com, 2020. URL: https://ru.wikipedia.org/wiki/Apache\_CXF (дата обращения: 24.02.2020).
64. Klocwork is static analysis code. [Электронный ресурс] // Roguewave.com, 2020. URL: https://docs.roguewave.com/ (дата обращения: 26.02.2020).
65. IntelliJ IDEA. [Электронный ресурс] // Jetbrains.com, 2020. URL: https://www.jetbrains.com/ru-ru/idea/ (дата обращения: 26.02.2020).
66. Parasoft Test. [Электронный ресурс] // Parasoft.com, 2020. URL: https://www.parasoft.com/products (дата обращения: 26.02.2020).
67. OWASP Benchmark Project. [Электронный ресурс] // Owasp.org, 2020. URL: https://owasp.org/www-project-benchmark/ (дата обращения: 19.03.2020).
68. OWASP Benchmark on git. [Электронный ресурс] // Github.com, 2020. URL: https://github.com/OWASP/Benchmark (дата обращения: 26.03.2020).
69. Bazhenov.me [Электронный ресурс] // Bazhenov.me, 2020. URL: http://bazhenov.me/blog/2012/07/21/classification-performance-evaluation.html (дата обращения: 23.04.2020).
70. jQuery. [Электронный ресурс] // jQuery.com, 2020. URL: https://jquery.com/download/ (дата обращения: 27.04.2020).
71. Springframework [Электронный ресурс] // Spring.io, 2020. URL: https://spring.io/projects/spring-framework (дата обращения: 29.04.2020).