(19) Евразийское патентное ведомство

(11) 044799 (13) B1

(12) ОПИСАНИЕ ИЗОБРЕТЕНИЯ К ЕВРАЗИЙСКОМУ ПАТЕНТУ

(45) Дата публикации и выдачи патента
2023.09.29

(21) Номер заявки
202293403

(22) Дата подачи заявки
2022.12.20

(51) Int. Cl. ***G06F 21/57*** (2013.01)
G06N 20/00 (2019.01)

(54) СПОСОБ И СИСТЕМА ВЫЯВЛЕНИЯ ЭКСПЛУАТИРУЕМЫХ УЯЗВИМОСТЕЙ В ПРОГРАММНОМ КОДЕ

(31) **2022106207**
(32) **2022.03.10**
(33) **RU**
(43) **2023.09.28**

(71)(73) Заявитель и патентовладелец:
ПУБЛИЧНОЕ АКЦИОНЕРНОЕ ОБЩЕСТВО "СБЕРБАНК РОССИИ" (ПАО СБЕРБАНК) (RU)

(72) Изобретатель:
Максимова Анна Андреевна,
Гончаренко Лейла Халидовна,
Бачевский Артем Евгеньевич,
Гуртова Кристина Сергеевна,
Умеренко Григорий Сергеевич,
Анистратенко Михаил Артурович
(RU)

(74) Представитель:
Герасин Б.В. (RU)

(56) US-A1-20110296371
US-A1-20160196433
US-B1-9754112
US-A1-20190258803
RU-C2-2755675

(57) Изобретение в общем относится к области вычислительной техники, а в частности к автоматизированному способу и системе выявления эксплуатируемых уязвимостей в программном коде с помощью алгоритмов машинного обучения. Техническим результатом, достигающимся при решении данной проблемы, является повышение скорости и точности выявления эксплуатируемых уязвимостей в программном коде. Указанный технический результат достигается благодаря осуществлению компьютерно-реализуемого способа выявления эксплуатируемых уязвимостей в программном коде, выполняемого с помощью по меньшей мере одного процессора и содержащего этапы, на которых: получают данные об уязвимостях в программном коде; выполняют построение вектора атаки на уязвимость, где каждый элемент вектора атаки указывает на имя элемента в исходном коде и содержит его координату; получают на основе предыдущего этапа данные, содержащие по меньше мере файл исходного кода и тип обнаруженной уязвимости; преобразуют исходный код, содержащий уязвимость в дерево абстрактного синтаксиса (AST), в котором внутренние вершины сопоставлены с операторами языка программирования, а листья с соответствующими операндами; осуществляют поиск элемента вектора атаки по координатам и имени среди вершин AST, содержащий по меньшей мере информацию о типе вершины; формируют путь в AST между найденными элементами вектора атаки; формируют упорядоченную последовательность, которая представляет собой типы каждого элемента пути; формируют векторное представление сформированной последовательности как количества всех возможных типов узлов AST внутри нее; осуществляют обработку полученных данных с помощью модели машинного обучения (МО), обученной на векторных представлениях данных об уязвимостях, в ходе которой осуществляется классификация уязвимостей по степени эксплуатации, и выявляют эксплуатируемые уязвимости в программном коде на основе классификации на предыдущем этапе.

044799 B1

044799 B1

Область техники

Изобретение в общем относится к области вычислительной техники, а в частности к автоматизированному способу и системе выявления эксплуатируемых уязвимостей в программном коде с помощью алгоритмов машинного обучения.

Уровень техники

С развитием информационных технологий IT-решения ("Information technology" - информационные технологии) стали оказывать существенное влияние на все сферы и отрасли жизнедеятельности. В настоящее время различные компании и организации активно внедряют и используют в своей структуре IT-решения.

Разработка программного обеспечения для крупных финансовых организаций (например, банков) всегда трудоемкий и кропотливый труд. Кроме того, при разработке программного продукта необходимо учесть все риски возникновения уязвимостей в программном коде. Для данных проверок привлекаются эксперты по кибербезопасности, которые вручную проверяют наличие уязвимостей в программном коде в разрабатываемом продукте, что многократно увеличивает время проверки, а также не исключает человеческого фактора.

Из уровня техники известен патент US 8631384 B2 "Creating a test progression plan", патентообладатель: IBM, опубликовано: 01.12.2011. В данном патенте описывается автоматизированный процесс составления планов тестирования программных продуктов. Известное изобретение обеспечивает автоматическое создание плана выполнения теста программного обеспечения путем вычисления для каждой единицы периода тестирования x усилий по выполнению тестовых блоков ATTx и усилий по завершению выполнения тестового блока CCx. В вычислении вводятся три переменные, характеризующие стратегию тестирования: эффективность, которая представляет эффективность группы тестирования, коэффициент плотности дефектов и значение коэффициента проверки. Выбирая стратегию тестирования, менеджер тестов определяет значения трех переменных, которые влияют на план развития. Во время выполнения теста кумулятивная кривая "попытка" значений ATTx и кумулятивная кривая "завершение" значений CCx позволяют менеджеру тестирования сравнить уже предпринятые усилия с ожидаемыми усилиями, предпринятыми для испытательных блоков, которые были предприняты и для испытательных единиц, которые были закончены, то есть, когда дефекты, найденные в коде, были исправлены.

Недостатком известных решений в данной области техники является отсутствие возможности автоматизированного выявления эксплуатируемых уязвимостей в программном коде.

Раскрытие изобретения

В заявленном изобретении предлагается новый подход к выявлению эксплуатируемых уязвимостей в программном коде. В данном изобретении используется алгоритм машинного обучения, который позволяет автоматизировать процесс проверки программного кода и значительно ускорить процесс выявления эксплуатируемых уязвимостей.

Таким образом, решается техническая проблема автоматизированного выявления эксплуатируемых уязвимостей в программном коде.

Техническим результатом, достигающимся при решении данной проблемы, является повышение скорости и точности выявления эксплуатируемых уязвимостей в программном коде.

Указанный технический результат достигается благодаря осуществлению компьютерно-реализуемого способа выявления эксплуатируемых уязвимостей в программном коде, выполняемого с помощью по меньшей мере одного процессора и содержащего этапы, на которых:

получают данные об уязвимостях в программном коде;

выполняют построение вектора атаки на уязвимость, где каждый элемент вектора атаки указывает на имя элемента в исходном коде и содержит его координату;

получают на основе предыдущего этапа данные, содержащие по меньше мере файл исходного кода и тип обнаруженной уязвимости;

преобразуют исходный код, содержащий уязвимость в дерево абстрактного синтаксиса (AST), в котором внутренние вершины сопоставлены с операторами языка программирования, а листья с соответствующими операндами;

осуществляют поиск элемента вектора атаки по координатам и имени среди вершин AST, содержащий по меньшей мере информацию о типе вершины;

формируют путь в AST между найденными элементами вектора атаки;

формируют упорядоченную последовательность, которая представляет собой типы каждого элемента пути;

формируют векторное представление сформированной последовательности как количества всех возможных типов узлов AST внутри нее;

осуществляют обработку полученных данных с помощью модели машинного обучения (МО), обученной на векторных представлениях данных об уязвимостях, в ходе которой осуществляется классификация уязвимостей по степени эксплуатации, и

выявляют эксплуатируемые уязвимости в программном коде на основе классификации на предыдущем этапе.

В одном из частных вариантов реализации способа при формировании векторного представления сформированной последовательности, кроме всех возможных типов AST, используются количества N-грамм типов узлов AST.

Кроме того, заявленный технический результат достигается за счет системы выявления эксплуатируемых уязвимостей в программном коде, содержащей:

по меньшей мере один процессор;

по меньшей мере одну память, соединенную с процессором, которая содержит машиночитаемые инструкции, которые при их выполнении по меньшей мере одним процессором обеспечивают выполнение заявленного способа.

Краткое описание чертежей

Признаки и преимущества настоящего изобретения станут очевидными из приводимого ниже подробного описания изобретения и прилагаемых чертежей.

Фиг. 1 иллюстрирует ROC-кривую (кривую ошибок) для классификатора Absolute Path Traversal (обход абсолютного пути) уязвимости, основанного на алгоритме случайного леса.

Фиг. 2 иллюстрирует ROC-кривую (кривую ошибок) для классификатора Improper Resource Shutdown or Release (неправильное отключение или освобождение ресурсов) уязвимости, основанного на алгоритме случайного леса.

Фиг. 3 иллюстрирует ROC-кривую (кривую ошибок) для классификатора Improper Restriction of Stored XXE Ref (неправильное ограничение ссылки на персистентный внешний объект XML) уязвимости, основанного на алгоритме случайного леса.

Фиг. 4 иллюстрирует ROC-кривую (кривую ошибок) для классификатора Reflected XSS All Clients (отраженный межсайтовый скриптинг для всех клиентов) уязвимости, основанного на алгоритме случайного леса.

Фиг. 5 иллюстрирует ROC-кривую (кривую ошибок) для классификатора Stored XXS (персистентный межсайтовый скриптинг) уязвимости, основанного на алгоритме случайного леса.

Фиг. 6 иллюстрирует ROC-кривую (кривую ошибок) для классификатора Improper Restriction of XXE Ref (неправильное ограничение ссылки на внешний объект XML) уязвимости, основанного на методе опорных векторов.

Фиг. 7 иллюстрирует ROC-кривую (кривую ошибок) для классификатора SSRF (подделка запроса на стороне сервера) уязвимости, основанного на методе опорных векторов.

Фиг. 8 иллюстрирует ROC-кривую (кривую ошибок) для классификатора SQL Injection (SQL-инъекция) уязвимости, основанного на методе дерева решений.

Фиг. 9 иллюстрирует матрицы ошибок (без нормализации) для классификаторов уязвимостей, основанных на методе случайного леса.

Фиг. 10 иллюстрирует матрицы ошибок (без нормализации) для классификаторов уязвимостей, основанных на методе опорных векторов.

Фиг. 11 иллюстрирует матрицы ошибок (без нормализации) для классификаторов уязвимостей, основанных на методе дерева решений.

Фиг. 12 иллюстрирует блок-схему заявленного способа.

Фиг. 13 иллюстрирует пример данных, полученных от инструмента SAST.

Фиг. 14 иллюстрирует пример общего вида вычислительной системы, которая обеспечивает реализацию заявленного изобретения.

Осуществление изобретения

Ниже будут описаны понятия и термины, необходимые для понимания данного изобретения.

Модель в машинном обучении (МО) - совокупность методов искусственного интеллекта, характерной чертой которых является не прямое решение задачи, а обучение в процессе применения решений множества сходных задач.

Уязвимость в программном обеспечении - недостаток в системе, используя который, можно намеренно нарушить её целостность и вызвать неправильную работу. Уязвимость может быть результатом ошибок программирования, недостатков, допущенных при проектировании системы, ненадежных паролей, вирусов и других вредоносных программ, скриптовых и SQL-инъекций. Уязвимости могут быть неэксплуатируемыми и эксплуатируемыми.

Эксплойт - компьютерная программа, фрагмент программного кода или последовательность команд, использующие уязвимости в программном обеспечении и применяемые для проведения атаки на вычислительную систему.

Эксплуатируемая уязвимость - уязвимость в программном обеспечении, для которой может быть создан и применен эксплойт.

AST - абстрактное синтаксическое дерево. Конечное помеченное ориентированное дерево, в котором внутренние вершины сопоставлены с операторами языка программирования, а листья с соответствующими операндами.

F-1 мера представляет собой совместную оценку точности и полноты.

ROC-кривая - графическая характеристика качества бинарного классификатора, отражающая зави-

симость доли истинно-положительных классификаций от доли ложно-положительных классификаций при варьировании порога решающего правила.

Матрица ошибок - это способ разбить классифицируемые объекты на четыре категории в зависимости от комбинации реального класса и ответа классификатора.

Коннекторы - программные компоненты, осуществляющие сбор данных от источников информации (система управления задачами /система для совместной работы над релизами /система управления версиями /система управления проектами /система управления сервисами предприятия /и др.) и приведение данных к необходимым структуре и формату.

Хранилище - система для хранения больших объемов собранных и обработанных коннекторами данных, а также генерируемой иными компонентами системы.

Данное изобретение может быть реализовано на компьютере, в виде автоматизированной информационной системы (АИС) или машиночитаемого носителя, содержащего инструкции для выполнения вышеупомянутого способа.

Изобретение может быть реализовано в виде распределенной компьютерной системы.

В данном изобретении под системой подразумевается компьютерная система, ЭВМ (электронно-вычислительная машина), ЧПУ (числовое программное управление), ПЛК (программируемый логический контроллер), компьютеризированные системы управления и любые другие устройства, способные выполнять заданную, чётко определённую последовательность вычислительных операций (действий, инструкций).

Под устройством обработки команд подразумевается электронный блок либо интегральная схема (микропроцессор), исполняющая машинные инструкции (программы)/

Устройство обработки команд считывает и выполняет машинные инструкции (программы) с одного или более устройства хранения данных, например таких устройств, как оперативно запоминающие устройства (ОЗУ) и/или постоянные запоминающие устройства (ПЗУ). В качестве ПЗУ могут выступать, но, не ограничиваясь, жесткие диски (HDD), флеш-память, твердотельные накопители (SSD), оптические носители данных (CD, DVD, BD, MD и т.п.) и др.

Программа - последовательность инструкций, предназначенных для исполнения устройством управления вычислительной машины или устройством обработки команд.

Подготовка данных для обучения.

Обучение модели проводилось на исторических данных об уязвимостях программного обеспечения, размеченных на 2 класса:

Confirmed или эксплуатируемая уязвимость (класс 1);

Not Exploitable или неэксплуатируемая уязвимость (класс 0).

Данные об уязвимостях получены из сканирований Checkmarx - SAST (Static Application Security Testing) инструмента. SAST- это инструмент для анализа кода или его части на наличие уязвимостей без запуска исследуемого приложения на исполнение.

Обучение модели МО для каждого типа уязвимости производится на заранее размеченных данных. Всего было доступно на момент создания модели от 2178 до 55949 уязвимостей в зависимости от типа уязвимости, обнаруженных в заданный временной диапазон, например, 1-3 месяца.

В обучающей выборке использовались исключительно уникальные уязвимости, количество которых варьируется от 153 до 2147 в зависимости от типа уязвимости. Для оценки качества модели набор данных был разбит на 2 части: тренировочную и контрольную выборки. Разбиение происходило случайным образом в отношении 70% на тренировочную выборку и 30% на контрольную выборку.

Взвешенная f-1 мера для всех классификаторов в среднем составляет около 0.89, точность - около 0.91.

На фиг. 1-8 приведены ROC-кривые (кривые ошибок) для классификаторов различных уязвимостей, основанных на различных алгоритмах. Кривые демонстрируют качество бинарной классификации и отображают соотношение между долей объектов от общего количества носителей признака, верно классифицированных как несущие признак, и долей объектов от общего количества объектов, не несущих признака, ошибочно классифицированных как несущие признак при варьировании порога решающего правила.

На фиг. 1 представлена ROC-кривая (кривая ошибок) для классификатора Absolute Path Traversal (Обход абсолютного пути) уязвимости, основанного на алгоритме случайного леса.

На фиг. 2 представлена ROC-кривая (кривая ошибок) для классификатора Improper Resource Shutdown or Release (неправильное отключение или освобождение ресурсов) уязвимости, основанного на алгоритме случайного леса.

На фиг. 3 представлена ROC-кривая (кривая ошибок) для классификатора Improper Restriction of Stored XXE Ref (неправильное ограничение ссылки на персистентный внешний объект XML) уязвимости, основанного на алгоритме случайного леса.

На фиг. 4 представлена ROC-кривая (кривая ошибок) для классификатора Reflected XSS All Clients (отраженный межсайтовый скриптинг для всех клиентов) уязвимости, основанного на алгоритме случайного леса.

На фиг. 5 представлена ROC-кривая (кривая ошибок) для классификатора Stored XXS (персистентный межсайтовый скриптинг) уязвимости, основанного на алгоритме случайного леса.

На фиг. 6 представлена ROC-кривая (кривая ошибок) для классификатора Improper Restriction of XXE Ref (неправильное ограничение ссылки на внешний объект XML) уязвимости, основанного на методе опорных векторов.

На фиг. 7 представлена ROC-кривая (кривая ошибок) для классификатора SSRF (подделка запроса на стороне сервера) уязвимости, основанного на методе опорных векторов.

На фиг. 8 представлена ROC-кривая (кривая ошибок) для классификатора SQL Injection (SQL-инъекция) уязвимости, основанного на методе дерева решений.

На фиг. 9-11 приведены матрицы ошибок (без нормализации) - таблица с 4 различными комбинациями прогнозируемых (строки) и фактических значений (колонки) -для классификаторов уязвимостей, основанных на различных алгоритмах.

На фиг. 9 представлены матрицы ошибок (без нормализации) для классификаторов уязвимостей, основанных на методе случайного леса.

На фиг. 10 представлены матрицы ошибок (без нормализации) для классификаторов уязвимостей, основанных на методе опорных векторов.

На фиг. 11 представлены матрицы ошибок (без нормализации) для классификаторов уязвимостей, основанных на методе дерева решений.

Как показано на фиг. 12 компьютерно-реализуемый способ выявления эксплуатируемых уязвимостей в программном коде (100) состоит из нескольких этапов, выполняемых по меньшей мере одним процессором.

На этапе (101) получают данные об уязвимостях в программном коде.

На данном этапе получают данные об уязвимостях (фиг. 13), полученных в результате сканирования программного кода с помощью инструмента SAST (Static Application Security Testing). Данные представлены в виде таблицы SQL-базы данных, в которой содержатся по крайней мере следующие поля:

ResultId - уникальный идентификатор уязвимости,

ScanId - идентификатор сканирования,

PathId - идентификатор уязвимости в рамках сканирования,

QueryId - идентификатор типа уязвимости,

Node_Id - порядковый номер (идентификатор) элемента в векторе атаки,

ProjectId - идентификатор проекта,

SourceId - идентификатор коммита в репозитории,

File_Name - файл с исходным кодом, содержащим уязвимость,

Short_Name - имя элемента вектора атаки,

Line - строка исходного кода, содержащая элемент вектора атаки,

Col - колонка исходного кода, содержащая элемент вектора атаки,

State - класс уязвимости по эксплуатируемости (0 - To Verify, 1 - Not Exploitable, 2 -Confirmed, 4 - Possibly Not Exploitable).

Далее на этапе (102) выполняют построение вектора атаки на уязвимость, где каждый элемент вектора атаки указывает на имя элемента в исходном коде и содержит его координату.

На данном этапе производится запрос об обнаруженных уязвимостях в базу данных инструмента SAST, получают список элементов исходного кода, указывающих на уязвимость, и строят вектор атаки на уязвимость по этим элементам с информацией об их положении (координате) в исходном коде (номер строки начала элемента, номер колонки начала элемента), а также наименованиях.

Далее на этапе (103) получают на основе предыдущего этапа (102) данные, содержащие по меньшей мере файл исходного кода и тип обнаруженной уязвимости.

На данном этапе производится запрос о каждом элементе вектора атаки на уязвимость в базу данных инструмента SAST, получают информацию о пути к файлу с исходным кодом, содержащим данный элемент вектора атаки, а также типе (классе) данной уязвимости.

Далее на этапе (104) преобразуют исходный код, содержащий уязвимость в дерево абстрактного синтаксиса (AST), в котором внутренние вершины сопоставлены с операторами языка программирования, а листья с соответствующими операндами.

На данном этапе производится синтаксический анализ (парсинг) исходного кода, содержащегося в файле, полученном на Этапе (103), и написанного на любом языке программирования (Java, C#, ASP, Visual Basic, C, C++, PHP, Apex, Ruby, JavaScript, VBScript, Perl, Swift, Python, Groovy, Scala и др.), результатом которого является дерево разбора или дерево абстрактного синтаксиса, отображающее зависимости между всеми элементами исходного кода, содержащими информацию об их положениях в исходном коде (номер строки начала элемента, номер колонки начала элемента, номер строки окончания элемента, номер колонки окончания элемента), их классах, таких как выражения (expressions), инструкции (statements), объявления (declarations) и др., типах их классов, их именах, их родительских и наследных элементах, комментариях к ним и пр.

Далее на этапе (105) осуществляют поиск элемента вектора атаки по координатам и имени среди

вершин AST, содержащий по меньшей мере информацию о типе вершины.

На данном этапе производится поиск элементов исходного кода, обнаруженных на Этапе (102) в дереве абстрактного синтаксиса, сформированного на Этапе (104) по сопоставлению их положения в исходном коде (номер строки начала элемента и номер колонки начала элемента), а также их наименований.

Далее на этапе (106) формируют путь в AST между найденными элементами вектора атаки.

На данном этапе производится поиск общего родительского элемента исходного кода для пар элементов, обнаруженных на этапе (105) на основе информации, полученной на этапе (104), и строится путь в AST между этими парами через общих родителей.

Далее на этапе (107) формируют упорядоченную последовательность, которая представляет собой типы каждого элемента пути.

На данном этапе из пути в AST, сформированного на этапе (106), в котором каждый элемент содержит информацию, полученную на этапе (104), формируется упорядоченная последовательность типов классов элементов (типов вершин) этого пути.

Далее на этапе (108) формируют векторное представление сформированной последовательности как количества всех возможных типов узлов AST внутри нее.

На данном этапе формируется словарь всех возможных типов классов (типов) вершин AST, полученных на Этапе (104). Значениями данного словаря для каждой последовательности, сформированной на этапе (107), будут являться количества соответствующих типов, имеющихся в данной последовательности и/или N-граммы таких типов. Из значений такого словаря формируется численный вектор с одинаковой размерностью для всех последовательностей по меньшей мере по одному из методов:

мешок слов (bag-of-words),

one-hot encoding (OHE),

кодирование словаря уникальными индексами,

word2Vec,

векторные представления типов - "Embedding"

и/или их совокупности.

Для получения Embedding типов могут быть использованы такие технологии как нейронные сети: полносвязанные, рекуррентные, сверточные, трансформеры. В одном из частных вариантов изобретения для получения векторных представлений типов используются предобученные нейронные сети, такие как DistilBERT: smaller, faster, cheaper, lighter; ALBERT (Lite BERT Google); TinyBERT; T-NLG (Turing Natural Language Generation); USE (Universal Sentence Encoder); ELMo (Embeddings from Language Models) или наследуемые от них сети.

Аугментация.

Так как обучение модели производилось на несбалансированных классах, то для увеличения количества примеров минорного класса использовалось оверсемплирование методом SMOTE (synthetic minority over-sampling technique) с параметром samplingstrategy = 0.7 Параметр был подобран опытным путем.

Векторизация.

Для извлечения признаков из сформированных словарей применялся метод векторизации TF-IDF (term-frequency times inverse document-frequency). Векторизатор обучался на обучающей выборке без использования стоп-слов (stop_words=None), без использования IDF, так как, исходя из размера обучающей выборки и наличия в каждом элементе обучающей выборки очень распространенных признаков, IDF-компонент не позволит вычислить особенные признаки для каждого словаря, использовались 1 и 2-граммы (параметр подбирался опытным путём).

Стандартизация.

Для некоторых моделей машинного обучения критически важно, чтобы используемые признаки были в одном масштабе, имели близкое к нормальному распределение. Особенно чувствительны к этому логистическая регрессия, метод опорных векторов (SVM). Для приведения признаков к одинаковому масштабу с математическим ожиданием 0 и дисперсией 1 применялся метод StandardScaler. При его применении каждое значение признака изменяется по формуле:

$$x_{st} = \frac{x-\mu}{\sigma},$$

где x_{st} - значение после стандартизации, x - значение до стандартизации, μ- среднее арифметическое по всем значениям признака, σ - стандартное отклонение по всем значениям признака.

Таким образом, значения всех признаков приводятся к математическому ожиданию 0 и дисперсии 1.

Обучение модели машинного обучения.

Для обучения исходный датасет предобрабатывался методом TF-IDF, разбивался на обучающую и тестовую выборки с соотношением 7:3, со стратификацией по классу, с перемешиванием.

Лучшая модель классификатора для каждого типа уязвимости выбиралась по наибольшему значению F-меры по классу 1 в результате обучения и тестирования следующих моделей:

DecisionTree (дерево решений) с Cost-Complexity Pruning 0,01 (параметр подбирался опытным пу-

тем),

LogRegression (логистическая регрессия),

Bayes (наивный байесовский классификатор),

SVM (метод опорных векторов),

RandomForest (случайный лес) с числом деревьев 1000 (параметр подбирался опытным путем).

С аугментацией или без нее и со стандартизацией или без нее. В случае близких друг к другу значений F-меры по классу 1 (с разницей менее 0,00999) для нескольких моделей, наилучшим классификатором выбирался тот, чья площадь под ROC-кривой (кривой ошибок) была больше.

Далее на этапе (109) осуществляют обработку полученных данных с помощью модели машинного обучения (МО), обученной на векторных представлениях данных об уязвимостях, в ходе которой осуществляется классификация уязвимостей по степени эксплуатации.

На данном этапе производится автоматический выбор обученной модели по типу уязвимости, которую необходимо классифицировать. После передачи вектора уязвимости в модель для классификации, модель возвращает значение вероятности попадания в класс 1 (эксплуатируемая уязвимость) от 0 до 1, где 0 - уязвимость однозначно неэксплуатируемая, 1 - уязвимость однозначно эксплуатируемая. В частном случае порог классификации уязвимости между неэксплуатируемой и эксплуатируемой составляет значение вероятности 0,5.

После классификации 2716 уникальных векторов уязвимостей для задач тестирования качества работы модели была выбрана произвольно 151 уязвимость для ручной валидации результатов классификации силами экспертов по кибербезопасности.

Доля корректной классификации составила 86%, при этом 95% ошибок приходится на диапазон вероятности попадания в класс 1 (эксплуатируемая уязвимость) от 40 до 60%, что соответствует нормальному распределению ошибки в классификаторе (Гауссово распределение).

Скорость классификации 1 уникальной уязвимости моделью составляет менее 0,1 с. Разработчику требуется в среднем 120 с на ручную валидацию 1 уникальной уязвимости. Точность модели обусловлена правильностью разметки обучающей выборки и не уступает точности ручной валидации уязвимости.

Метод	Точность	Время
Ручная классификация	0.9	120 сек
Классификация моделью МО	**0.91**	**0,1 сек**

Далее на этапе (110) выявляют эксплуатируемые уязвимости в программном коде на основе классификации на предыдущем этапе.

На данном этапе производится запись в базу данных инструмента SAST о статусе уязвимости (эксплуатируемая или неэксплуатируемая) и вероятность принадлежности к тому или иному классу или доверительный интервал (Confidence Level).

В результате, данный подход позволяет формировать и приоритизировать задачи по устранению недостатков в разрабатываемом программном обеспечении, тем самым повышая скорость и точность выявления эксплуатируемых и не эксплуатируемых уязвимостей, что обеспечивает повышение скорости обновления программного обеспечения или вывода программного обеспечения на рынок за счет исключения из анализа недостижимых (неэксплуатируемых) уязвимостей и увеличивая надежность программного обеспечения от действий злоумышленников, направленных на:

хищение чувствительной информации,

причинение репутационного или финансового ущерба организации или пользователю,

уничтожение важных данных или препятствование доступу к важным данным,

искажение информации,

хищение денежных средств за счет минимизации ошибки, при принятии решения об эксплуатируемости уязвимостей.

На фиг. 14 представлен пример общего вида вычислительной системы (300), которая обеспечивает реализацию заявленного способа или является частью компьютерной системы, например, сервером, персональным компьютером, частью вычислительного кластера, обрабатывающим необходимые данные для осуществления заявленного изобретения.

В общем случае, система (300) содержит объединенные общей шиной информационного обмена один или несколько процессоров (301), средства памяти, такие как ОЗУ (302) и ПЗУ (303), интерфейсы ввода/вывода (304), устройства ввода/вывода (1105), и устройство для сетевого взаимодействия (306).

Процессор (301) (или несколько процессоров, многоядерный процессор и т.п.) может выбираться из ассортимента устройств, широко применяемых в настоящее время, например, таких производителей, как: Intel™, AMD™, Apple™, Samsung Exynos™, MediaTEK™, Qualcomm Snapdragon™ и т.п. Под процессором или одним из используемых процессоров в системе (300) также необходимо учитывать графический процессор, например, GPU NVIDIA или Graphcore, тип которых также является пригодным для полного или частичного выполнения способа, а также может применяться для обучения и применения моделей

машинного обучения в различных информационных системах.

ОЗУ (302) представляет собой оперативную память и предназначено для хранения исполняемых процессором (301) машиночитаемых инструкций для выполнения необходимых операций по логической обработке данных. ОЗУ (302), как правило, содержит исполняемые инструкции операционной системы и соответствующих программных компонент (приложения, программные модули и т.п.). При этом, в качестве ОЗУ (302) может выступать доступный объем памяти графической карты или графического процессора.

ПЗУ (303) представляет собой одно или более устройств постоянного хранения данных, например, жесткий диск (HDD), твердотельный накопитель данных (SSD), флэш-память (EEPROM, NAND и т.п.), оптические носители информации (CD-R/RW, DVD-R/RW, BlueRay Disc, MD) и др.

Для организации работы компонентов системы (300) и организации работы внешних подключаемых устройств применяются различные виды интерфейсов В/В (304). Выбор соответствующих интерфейсов зависит от конкретного исполнения вычислительного устройства, которые могут представлять собой, не ограничиваясь: PCI, AGP, PS/2, IrDa, FireWire, LPT, COM, SATA, IDE, Lightning, USB (2.0, 3.0, 3.1, micro, mini, type C), TRS/Audio jack (2.5, 3.5, 6.35), HDMI, DVI, VGA, Display Port, RJ45, RS232 и т.п.

Для обеспечения взаимодействия пользователя с вычислительной системой (300) применяются различные средства (305) В/В информации, например, клавиатура, дисплей (монитор), сенсорный дисплей, тач-пад, джойстик, манипулятор мышь, световое перо, стилус, сенсорная панель, трекбол, динамики, микрофон, средства дополненной реальности, оптические сенсоры, планшет, световые индикаторы, проектор, камера, средства биометрической идентификации (сканер сетчатки глаза, сканер отпечатков пальцев, модуль распознавания голоса) и т.п.

Средство сетевого взаимодействия (306) обеспечивает передачу данных посредством внутренней или внешней вычислительной сети, например, Интранет, Интернет, ЛВС и т.п. В качестве одного или более средств (306) может использоваться, но не ограничиваться: Ethernet карта, GSM модем, GPRS модем, LTE модем, 5G модем, модуль спутниковой связи, NFC модуль, Bluetooth и/или BLE модуль, Wi-Fi модуль и др.

Представленные материалы изобретения раскрывают предпочтительные примеры реализации изобретения и не должны трактоваться как ограничивающие иные, частные примеры его воплощения, не выходящие за пределы испрашиваемой правовой охраны, которые являются очевидными для специалистов соответствующей области техники.

ФОРМУЛА ИЗОБРЕТЕНИЯ

1. Компьютерно-реализуемый способ выявления эксплуатируемых уязвимостей в программном коде, выполняемый с помощью по меньшей мере одного процессора и содержащий этапы, на которых:

получают данные об уязвимостях в программном коде;

выполняют построение вектора атаки на уязвимость, где каждый элемент вектора атаки указывает на имя элемента в исходном коде и содержит его координату;

получают на основе предыдущего этапа данные, содержащие по меньше мере файл исходного кода и тип обнаруженной уязвимости;

преобразуют исходный код, содержащий уязвимость в дерево абстрактного синтаксиса (AST), в котором внутренние вершины сопоставлены с операторами языка программирования, а листья - с соответствующими операндами;

осуществляют поиск элемента вектора атаки по координатам и имени среди вершин AST, содержащий по меньшей мере информацию о типе вершины;

формируют путь в AST между найденными элементами вектора атаки;

формируют упорядоченную последовательность, которая представляет собой типы каждого элемента пути;

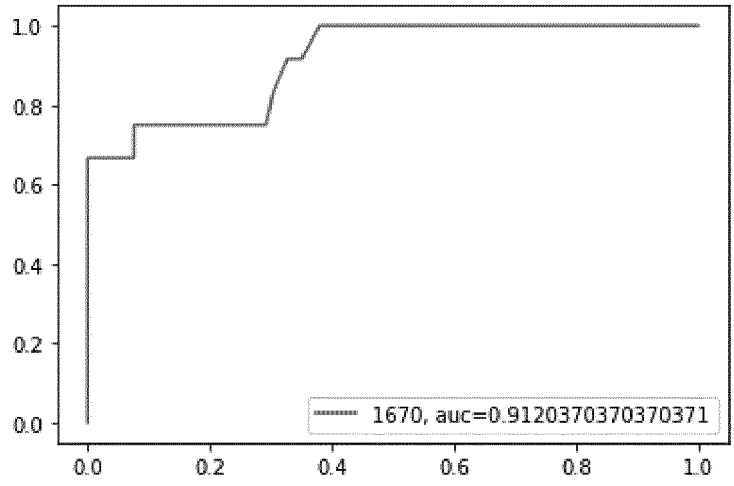
формируют векторное представление сформированной последовательности как количества всех возможных типов узлов AST внутри нее;

осуществляют обработку полученных данных с помощью модели машинного обучения (МО), обученной на векторных представлениях данных об уязвимостях, в ходе которой осуществляется классификация уязвимостей по степени эксплуатации, и

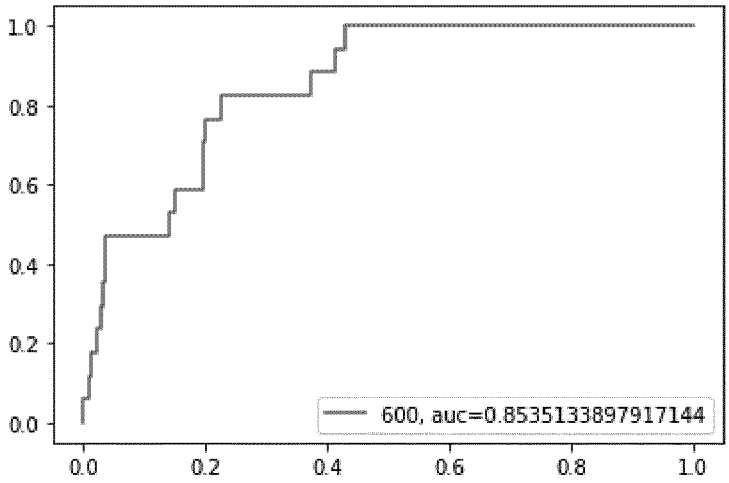
выявляют эксплуатируемые уязвимости в программном коде на основе классификации на предыдущем этапе.

2. Способ по п.1, характеризующийся тем, что при формировании векторного представления сформированной последовательности, кроме всех возможных типов узлов AST, используются количества N-грамм типов узлов AST.

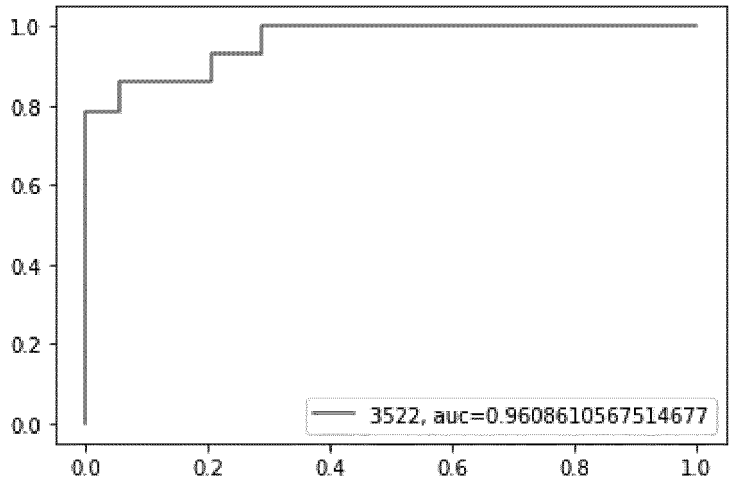
3. Система выявления эксплуатируемых уязвимостей в программном коде, содержащая по меньшей мере один процессор и память, хранящую машиночитаемые инструкции, которые при их выполнении процессором реализуют способ по пп.1, 2.



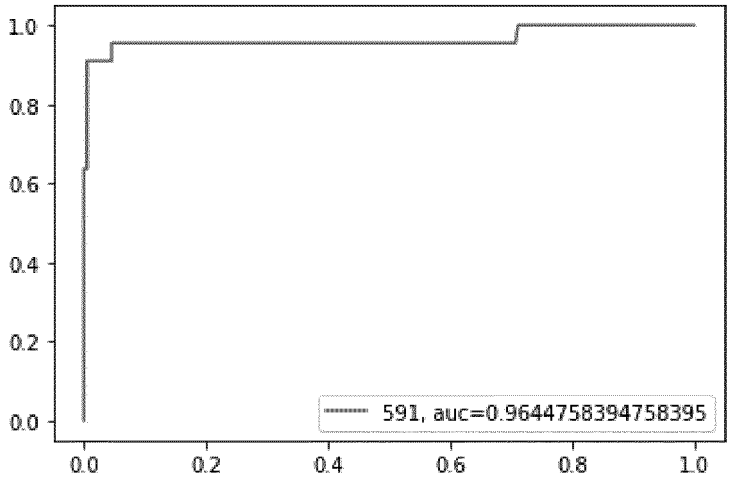
Фиг. 1



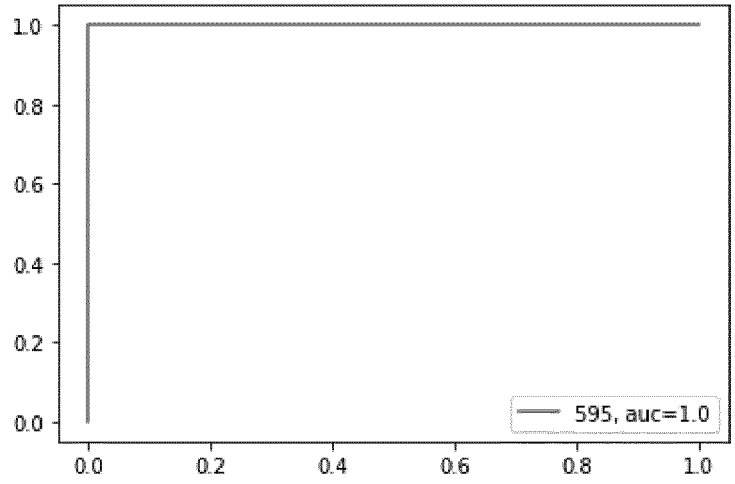
Фиг. 2



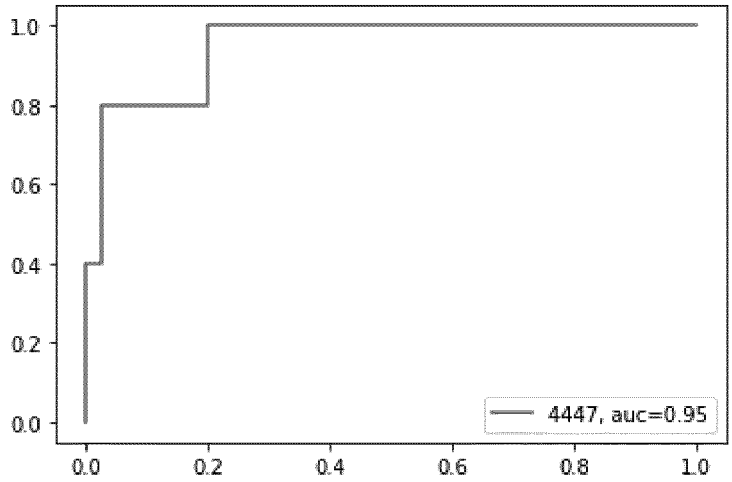
Фиг. 3



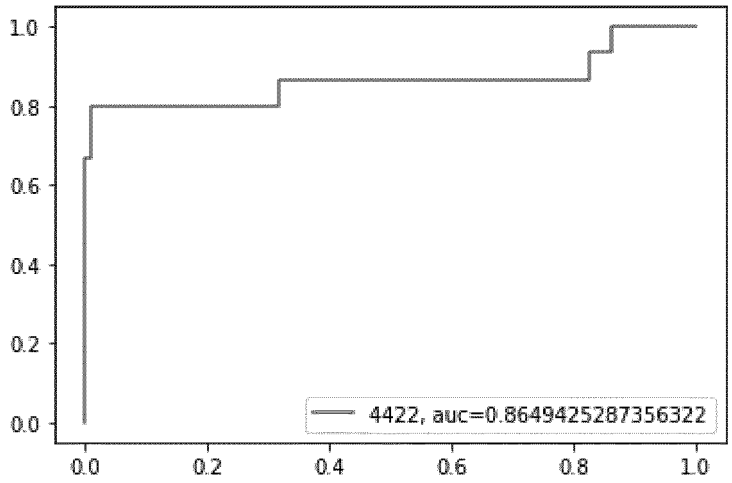
Фиг. 4



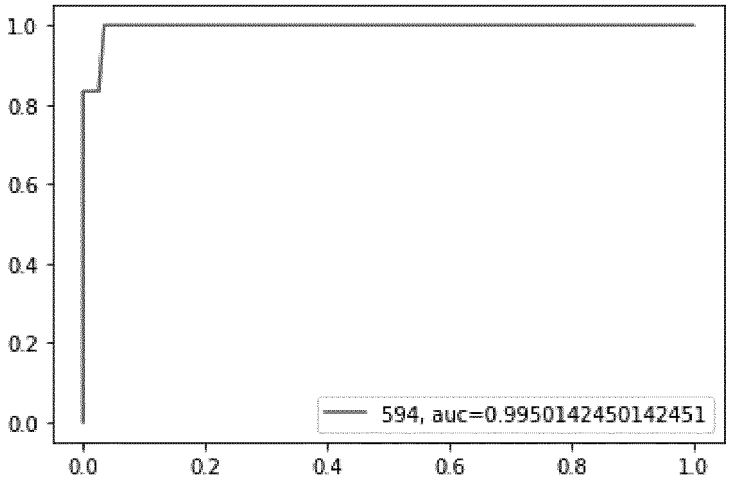
Фиг. 5



Фиг. 6


1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
4422, auc=0.8649425287356322

Фиг. 7


1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
594, auc=0.9950142450142451

Фиг. 8

Absolute_Path_Traversal

	Неэксплуатируемая	Эксплуатируемая
Неэксплуатируемая уязвимость	170	1
Эксплуатируемая уязвимость	4	8

Improper_Resource_Shutdown_or_Release

	Неэксплуатируемая	Эксплуатируемая
Неэксплуатируемая уязвимость	248	9
Эксплуатируемая уязвимость	9	8

Improper_Restriction_of_XXE_Ref

	Неэксплуатируемая	Эксплуатируемая
Неэксплуатируемая уязвимость	73	0
Эксплуатируемая уязвимость	3	11

Reflected_XSS_All_Clients

	Неэксплуатируемая	Эксплуатируемая
Неэксплуатируемая уязвимость	221	1
Эксплуатируемая уязвимость	2	20

Stored_XSS

	Неэксплуатируемая	Эксплуатируемая
Неэксплуатируемая уязвимость	573	0
Эксплуатируемая уязвимость	1	13

Фиг. 9

Improper_Restriction_of_Stored_XXE_Ref

	Неэксплуатируемая	Эксплуатируемая
Неэксплуатируемая уязвимость	49	1
Эксплуатируемая уязвимость	1	4

SSRF

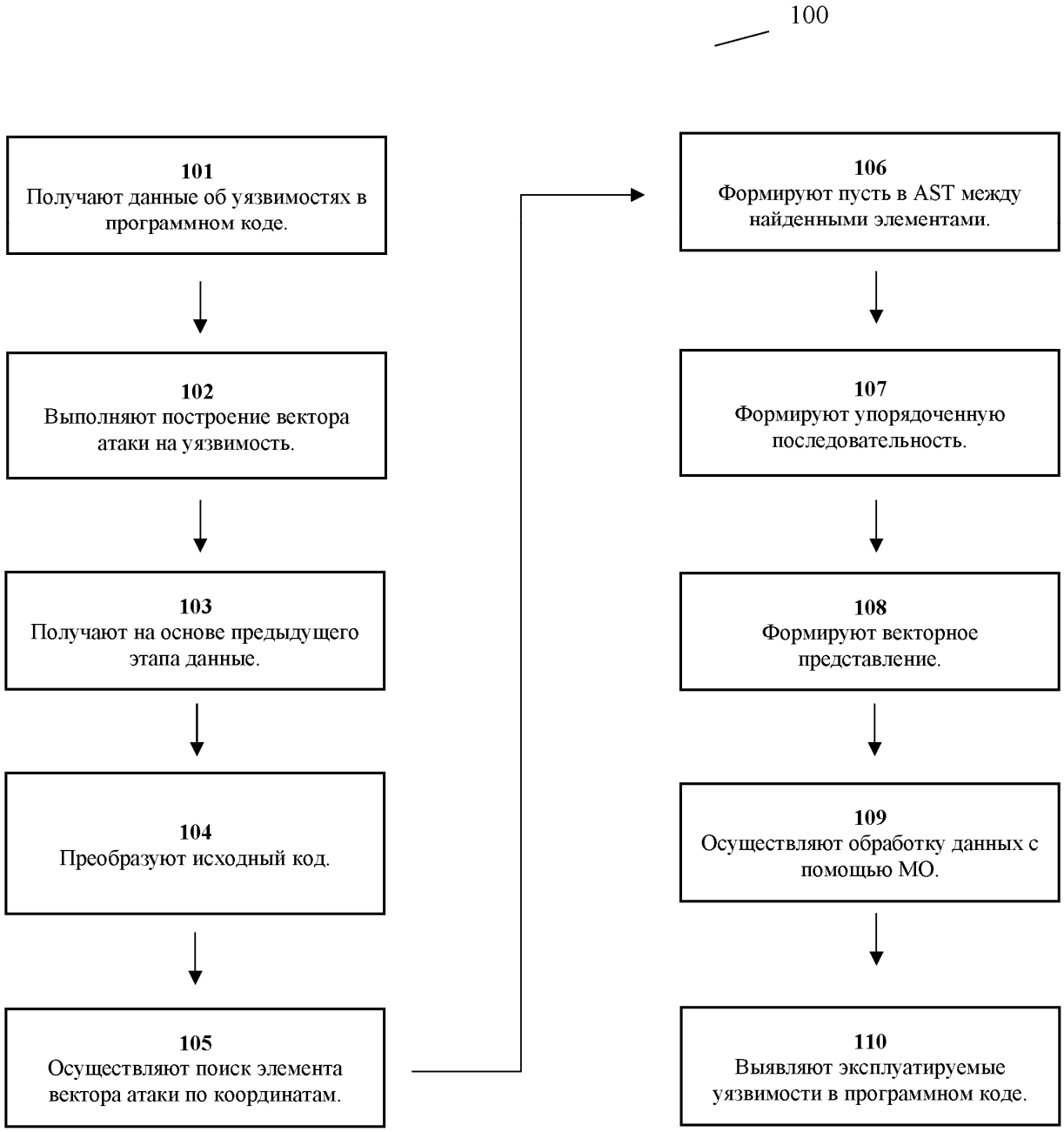
	Неэксплуатируемая	Эксплуатируемая
Неэксплуатируемая уязвимость	113	3
Эксплуатируемая уязвимость	3	12

Фиг. 10

SQL_Injection

	Неэксплуатируемая	Эксплуатируемая
Неэксплуатируемая уязвимость	116	1
Эксплуатируемая уязвимость	1	5

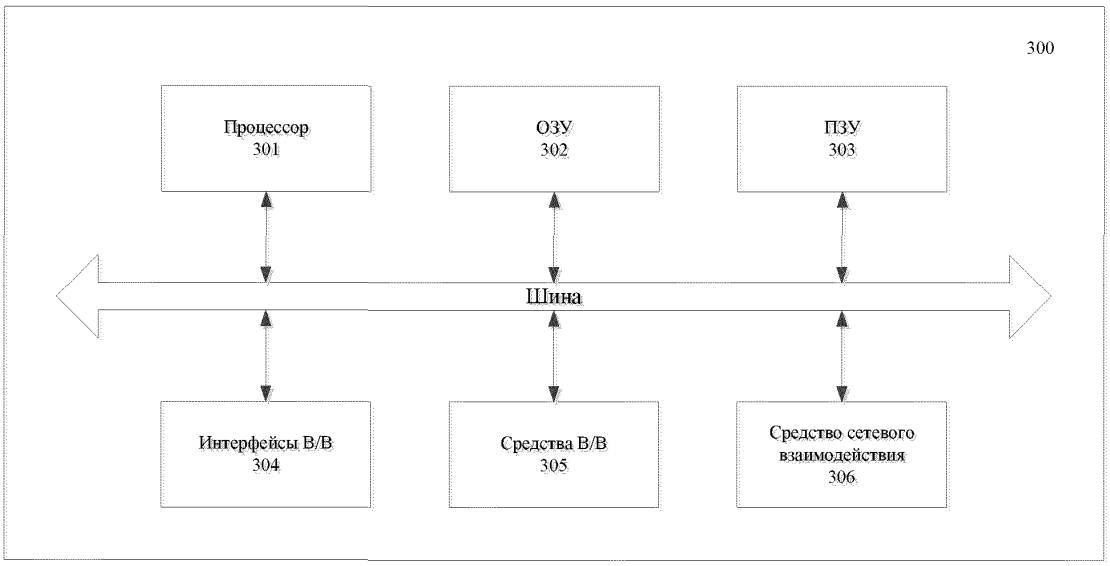
Фиг. 11



Фиг. 12

	ResultId	ScanId	PathId	QueryId	Node_Id	ProjectId	SourceId	File_Name	Short_Name	Line	Col.	State
1	1176362-714	1176362	714	595	18	23891	0000009147_0-2140556026_00047658146	\src\systems\is.oad-sbrus\solutions\docflow\services\singledocview\projects\lib\src\isida\oadsbrus\docflow\singledocview\vo\PhotoAlbumLinkVO.java	resultJSONString	442	32	4
2	1176362-714	1176362	714	595	19	23891	0000009147_0-2140556026_00047658146	\src\systems\is.oad-sbrus\solutions\docflow\services\singledocview\projects\lib\src\isida\oadsbrus\docflow\singledocview\vo\PhotoAlbumLinkVO.java	resultJSONString	442	13	4
3	1176362-714	1176362	714	595	20	23891	0000009147_0-2140556026_00047658146	\src\systems\is.oad-sbrus\solutions\docflow\services\singledocview\projects\lib\src\isida\oadsbrus\docflow\singledocview\vo\PhotoAlbumLinkVO.java	resultJSONString	448	16	4
4	1176362-714	1176362	714	595	21	23891	0000009147_0-2140556026_00047658146	\src\systems\is.oad-sbrus\solutions\docflow\services\singledocview\projects\web\src\isida\oadsbrus\docflow\singledocview\action\pa\AjaxRefreshTableAction.java	toJSON	52	45	4
5	1176362-714	1176362	714	595	22	23891	0000009147_0-2140556026_00047658146	\src\systems\is.oad-sbrus\solutions\docflow\services\singledocview\projects\web\src\isida\oadsbrus\docflow\singledocview\action\pa\AjaxRefreshTableAction.java	jsonResult	52	13	4
6	1176362-714	1176362	714	595	23	23891	0000009147_0-2140556026_00047658146	\src\systems\is.oad-sbrus\solutions\docflow\services\singledocview\projects\web\src\isida\oadsbrus\docflow\singledocview\action\pa\AjaxRefreshTableAction.java	jsonResult	58	22	4
7	1176362-714	1176362	714	595	24	23891	0000009147_0-2140556026_00047658146	\src\systems\is.oad-sbrus\solutions\docflow\services\singledocview\projects\web\src\isida\oadsbrus\docflow\singledocview\action\pa\AjaxRefreshTableAction.java	write	58	21	4
8	1176362-732	1176362	732	595	1	23891	0000009147_0-2140556026_00047658146	\src\systems\is.oad-sbrus\solutions\referencestmanagement\services\archiveplaces\projects\ejb\src\isida\oadsbrus\referencestmanagement\archiveplaces\ejb\ArchivePlacesBean.java	addOrder	179	38	4
9	1176362-732	1176362	732	595	2	23891	0000009147_0-2140556026_00047658146	\src\systems\is.oad-sbrus\solutions\referencestmanagement\services\archiveplaces\projects\ejb\src\isida\oadsbrus\referencestmanagement\archiveplaces\ejb\ArchivePlacesBean.java	criteraIter	181	52	4
10	1176362-732	1176362	732	595	3	23891	0000009147_0-2140556026_00047658146	\src\systems\is.oad-sbrus\solutions\referencestmanagement\services\archiveplaces\projects\ejb\src\isida\oadsbrus\referencestmanagement\archiveplaces\ejb\ArchivePlacesBean.java	list	181	69	4
11	1176362-732	1176362	732	595	4	23891	0000009147_0-2140556026_00047658146	\src\systems\is.oad-sbrus\solutions\referencestmanagement\services\archiveplaces\projects\ejb\src\isida\oadsbrus\referencestmanagement\archiveplaces\ejb\ArchivePlacesBean.java	boxCodesRandomOrder	181	30	4
12	1176362-732	1176362	732	595	5	23891	0000009147_0-2140556026_00047658146	\src\systems\is.oad-sbrus\solutions\referencestmanagement\services\archiveplaces\projects\ejb\src\isida\oadsbrus\referencestmanagement\archiveplaces\ejb\ArchivePlacesBean.java	boxCodesRandomOrder	185	39	4
13	1176362-732	1176362	732	595	6	23891	0000009147_0-2140556026_00047658146	\src\systems\is.oad-sbrus\solutions\referencestmanagement\services\archiveplaces\projects\ejb\src\isida\oadsbrus\referencestmanagement\archiveplaces\ejb\ArchivePlacesBean.java	boxCode	187	38	4
14	1176362-732	1176362	732	595	7	23891	0000009147_0-2140556026_00047658146	\src\systems\is.oad-sbrus\solutions\referencestmanagement\services\archiveplaces\projects\ejb\src\isida\oadsbrus\referencestmanagement\archiveplaces\ejb\ArchivePlacesBean.java	add	187	37	4
15	1176362-732	1176362	732	595	8	23891	0000009147_0-2140556026_00047658146	\src\systems\is.oad-sbrus\solutions\referencestmanagement\services\archiveplaces\projects\ejb\src\isida\oadsbrus\referencestmanagement\archiveplaces\ejb\ArchivePlacesBean.java	boxCodes	192	17	4
16	1176362-732	1176362	732	595	9	23891	0000009147_0-2140556026_00047658146	\src\systems\is.oad-sbrus\solutions\referencestmanagement\services\archiveplaces\projects\ejb\src\isida\oadsbrus\referencestmanagement\archiveplaces\ejb\ArchivePlacesBean.java	boxCodes	194	39	4
17	1176362-732	1176362	732	595	10	23891	0000009147_0-2140556026_00047658146	\src\systems\is.oad-sbrus\solutions\referencestmanagement\services\archiveplaces\projects\ejb\src\isida\oadsbrus\referencestmanagement\archiveplaces\ejb\ArchivePlacesBean.java	boxCode	197	72	4

Фиг. 13


300
Процессор
301
ОЗУ
302
ПЗУ
303
Шина
Интерфейсы В/В
304
Средства В/В
305
Средство сетевого
взаимодействия
306

Фиг. 14