(19) Евразийское патентное ведомство

(11) 047814 (13) B1

(12) ОПИСАНИЕ ИЗОБРЕТЕНИЯ К ЕВРАЗИЙСКОМУ ПАТЕНТУ

(45) Дата публикации и выдачи патента
2024.09.13

(21) Номер заявки
202393225

(22) Дата подачи заявки
2023.12.13

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

(54) СПОСОБ И СИСТЕМА УСТРАНЕНИЯ УЯЗВИМОСТЕЙ В ПРОГРАММНОМ КОДЕ

(31) **2023112128**
(32) **2023.05.11**
(33) **RU**
(43) **2024.09.11**

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

(72) Изобретатель:
Вышегородцев Кирилл Евгеньевич, Кузьмин Александр Михайлович (RU)

(56) RU-C1-2790005
RU-C2-2755675
US-A1-20190258803
RU-C2-2364930
US-B2-8566805

(57) Изобретение в общем относится к области вычислительной техники, а в частности к автоматизированному способу и системе устранения уязвимостей в программном коде с помощью алгоритмов машинного обучения. Техническим результатом, достигающимся при решении данной проблемы, является повышение безопасности программного обеспечения за счет устранения уязвимостей в программном коде. Указанный технический результат достигается благодаря осуществлению компьютерно-реализуемого способа устранения уязвимостей в программном коде, выполняемого с помощью по меньшей мере одного процессора и содержащего этапы, на которых: получают данные об уязвимостях в программном коде; получают на основе предыдущего этапа данные, содержащие по меньшей мере блок 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; формируют путь обхода вершин в AST; формируют упорядоченную последовательность, которая представляет собой типы каждого элемента пути; осуществляют обработку упорядоченной последовательности с помощью кодирующей модели машинного обучения, обученной на упорядоченных последовательностях данных об уязвимостях, в ходе которой получают представление упорядоченной последовательности в виде матрицы скрытых состояний; осуществляют обработку матрицы скрытых последовательностей с помощью генеративной модели машинного обучения, обученной на матрицах скрытых состояний последовательностей для уязвимостей, последовательностей без уязвимостей и их исходных кодах, в ходе которой получают новую упорядоченную последовательность, соответствующую исходному программному коду по выполняемым функциям, но с устранённой в нем уязвимостью; переводят полученную на предыдущем этапе упорядоченную последовательность в блок исходного кода, эквивалентный по функционалу изначальному блоку исходного кода.

047814 B1

047814 B1

Область техники

Изобретение в общем относится к области вычислительной техники, а в частности к автоматизированному способу и системе устранения 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 человеческий фактор.

Из уровня техники известен патент US 8631384B2 "Creating a test progression plan", патентообладатель: IBM, опубликовано: 01.12.2011. В данном решении описывается автоматизированный процесс составления планов тестирования программных продуктов. Известное решение 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 устранения уязвимостей в программном коде.

Раскрытие изобретения

В заявленном техническом решении предлагается новый подход к устранению уязвимостей в программном коде. В данном решении используется алгоритм машинного обучения, который позволяет автоматизировать процесс проверки программного кода и значительно ускорить процесс устранения уязвимостей в программном коде.

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

Техническим результатом, достигающимся при решении данной проблемы, является повышение безопасности программного обеспечения за счет устранения уязвимостей в программном коде.

Указанный технический результат достигается благодаря осуществлению компьютерно-реализуемого способа устранения уязвимостей в программном коде, выполняемый с помощью, по меньшей мере одного процессора и содержащий этапы, на которых:

получают данные об уязвимостях в программном коде;

получают на основе предыдущего этапа данные, содержащие по меньшей мере блок 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;

формируют путь обхода вершин в AST;

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

осуществляют обработку упорядоченной последовательности с помощью кодирующей модели машинного обучения, обученной на упорядоченных последовательностях данных об уязвимостях, в ходе которой получают

представление упорядоченной последовательности в виде матрицы скрытых состояний;

осуществляют обработку матрицы скрытых последовательностей с помощью генеративной модели машинного обучения, обученной на матрицах скрытых состояний последовательностей для уязвимостей, последовательностей без уязвимостей и их исходных кодах, в ходе которой получают новую упорядоченную последовательность, соответствующую исходному программному коду по выполняемым функциям, но с устранённой в нем уязвимостью;

переводят полученную на предыдущем этапе упорядоченную последовательность в блок исходного кода, эквивалентный по функционалу изначальному блоку исходного кода.

В одном из частных вариантов реализации способа упорядоченная последовательность, которая представляет собой типы каждого элемента пути, является многомерной и представляет собой граф.

В другом частном варианте реализации способа преобразуют исходный код, содержащий уязвимость в граф потока управления (CFG - control flow graph), в котором внутренние вершины сопоставлены с операторами языка программирования, а листья - с соответствующими операндами.

В другом частном варианте реализации способа преобразуют исходный код, содержащий уязвимость в граф зависимостей управления (CDG - Control Dependence Graphs), в котором внутренние вершины сопоставлены с операторами языка программирования, а листья - с соответствующими операндами.

В другом частном варианте реализации способа преобразуют исходный код, содержащий уязвимость в граф зависимости данных (DDG - Data Dependence Graph), в котором внутренние вершины сопоставлены с операторами языка программирования, а листья - с соответствующими операндами.

В другом частном варианте реализации способа преобразуют исходный код, содержащий уязвимость в граф зависимости программы (PDG - Program Dependence graphs), в котором внутренние вершины сопоставлены с операторами языка программирования, а листья - с соответствующими операндами.

В другом частном варианте реализации способа преобразуют исходный код, содержащий уязвимость в граф свойств кода (CPG - Code Property Graphs), в котором внутренние вершины сопоставлены с операторами языка программирования, а листья - с соответствующими операндами.

Кроме того, заявленный технический результат достигается за счет системы устранения 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

Фиг. 1 иллюстрирует блок-схему заявленного способа.

Фиг. 2 иллюстрирует пример общего вида вычислительной системы, которая обеспечивает реализацию заявленного решения.

Осуществление изобретения

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

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

Техническое решение может быть реализовано в виде распределенной компьютерной системы.

В данном решении 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:

код с уязвимостью (класс 1),

код без уязвимости (класс 0).

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.

Как показано на фиг. 1 компьютерно-реализуемый способ устранения 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. В одном из частных вариантов изобретения эти данные получены в результате сканирования программного кода с помощью инструмента SAST (Static Application Security Testing).

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

Далее на этапе (103) 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.

В одном из частных вариантов реализации заявленного технического решения, преобразуют исходный код, содержащий уязвимость в граф потока управления (CFG -control flow graph), в котором внутренние вершины сопоставлены с операторами языка программирования, а листья - с соответствующими операндами.

В другом частном варианте реализации заявленного технического решения преобразуют исходный код, содержащий уязвимость в граф зависимостей управления (CDG - Control Dependence Graphs), в котором внутренние вершины сопоставлены с операторами языка программирования, а листья - с соответствующими операндами.

В другом частном варианте реализации заявленного технического решения преобразуют исходный код, содержащий уязвимость в граф зависимости данных (DDG -Data Dependence Graph), в котором внутренние вершины сопоставлены с операторами языка программирования, а листья - с соответствующими операндами.

В другом частном варианте реализации заявленного технического решения преобразуют исходный код, содержащий уязвимость в граф зависимости программы (PDG - Program Dependence graphs), в кото-

ром внутренние вершины сопоставлены с операторами языка программирования, а листья - с соответствующими операндами.

В другом частном варианте реализации заявленного технического решения преобразуют исходный код, содержащий уязвимость в граф свойств кода (CPG - Code Property Graphs), в котором внутренние вершины сопоставлены с операторами языка программирования, а листья - с соответствующими операндами.

Далее на этапе (104) формируют путь обхода вершин AST.

Далее на этапе (105) формируют упорядоченную последовательность, которая представляет собой типы каждого элемента.

На данном этапе из пути обхода вершин в AST, сформированного на этапе (105), в котором каждый элемент содержит информацию, полученную на этапе (104), формируется упорядоченная последовательность типов классов элементов (типов вершин) этого пути.

В одном из частных вариантов изобретения упорядоченная последовательность, представляющая собой типы каждого элемента пути, является многомерной и представляет собой граф (представление дерева абстрактного синтаксиса (AST) в виде графа).

Далее на этапе (106) осуществляют обработку упорядоченной последовательности с помощью кодирующей модели машинного обучения (МО), обученной на упорядоченных последовательностях данных об уязвимостях, в ходе которой получают представление упорядоченной последовательности в виде матрицы скрытых состояний.

На данном этапе формируется словарь всех возможных типов классов (типов) вершин AST, полученных на Этапе (103). Значениями данного словаря для каждой последовательности, сформированной на этапе (105), 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.

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).

Обучение модели машинного обучения

Для получения матрицы скрытых последовательностей используется блок энкодера (кодировки) генеративной модели машинного обучения. Данный блок может представлять собой слой Embedding, случайно обученный или инициализированный. Далее могут идти нейронные сети: Полносвязанные, Рекуррентные, Сверточные, Трансформеры. В одном из частных вариантов изобретения блок энкодера представляет собой по меньшей мере один слой нейронные сети из типов Рекуррентные нейронные сети (Recurrent neural networks, RNN), Долгая краткосрочная память (Long short term memory, LSTM), Управляемые рекуррентные нейроны (Gated recurrent units, GRU), Нейронные машины Тьюринга (Neural Turing machines, NMT), Двунаправленные RNN, LSTM и GRU (BiRNN, BiLSTM и BiGRU), Глубокие остаточные сети (Deep residual networks, DRN), Нейронные эхо-сети (Echo state networks, ESN), Машины неустойчивых состояний (Liquid state machines, LSM), самоорганизующаяся карта Кохонена (Kohonen networks, KN, или organising (feature) map, SOM, SOFM).

Далее на этапе (107) осуществляют обработку матрицы скрытых последовательностей с помощью генеративной модели машинного обучения (МО), обученной на матрицах скрытых состояний последовательностей для уязвимостей, последовательностей без уязвимостей и их исходных кодах, в ходе которой получают новую упорядоченную последовательность, соответствующую исходному программному коду по выполняемым функциям, но с устранённой в нем уязвимостью.

Данная обработка осуществляется блоком генератора (декодера) генеративной модели машинного обучения. Этот блок может представлять собой декодер типа Вариационный автоэнкодер (VAE), Методы

глубокого обучения, ограниченную машину Больцмана (RBM), Глубокую сеть доверия (Deep Belief Network - DBN), нейронные сети: Полносвязанные, Рекуррентные, Сверточные, Трансформеры.

В одном из частных вариантов изобретения блок декодера представляет собой по меньшей мере один слой нейронные сети из типов Сверточных нейронных сете (CNN), Рекуррентные нейронные сети (Recurrent neural networks, RNN), Долгая краткосрочная память (Long short term memory, LSTM), Управляемые рекуррентные нейроны (Gated recurrent units, GRU), Нейронные машины Тьюринга (Neural Turing machines, NMT), Двунаправленные RNN, LSTM и GRU (BiRNN, BiLSTM и BiGRU), Глубокие остаточные сети (Deep residual networks, DRN), Нейронные эхо-сети (Echo state networks, ESN), Машины неустойчивых состояний (Liquid state machines, LSM), самоорганизующаяся карта Кохонена (Kohonen networks, KN, или organising (feature) map, SOM, SOFM).

В одном из частных вариантов изобретения в качестве модели машинного обучения используются генеративно-состязательные сети (англ. Generative adversarial networks, сокр. GAN). Данная генеративно-состязательная сеть может содержать блок энкодера со слоями, указанными в разделе выше. Может содержать блок декодера со слоями, указанными в разделе выше. Блок дискриминатора (дискриминантной модели) в одном из частных вариантов изобретения содержит нейронные сети: Полносвязанные, Рекуррентные, Сверточные, Трансформеры; Классификаторы: DecisionTree (дерево решений), LogRegression (логистическая регрессия), Bayes (Наивный байесовский классификатор), SVM (метод опорных векторов), K-means (к-соседей), RandomForest (случайный лес), градиентные бустинги: XGBoost, LightGBM.

Обучение происходит следующим образом: операции производятся над двумя группами блоков программного кода, где первые блоки программного кода - это код с уязвимостью, а вторые блоки программного кода - это код без уязвимости. Данные блоки переводятся в упорядоченную последовательность элементов AST дерева в соответствии с этапами (101)-(106).

Далее получают последовательность кода с уязвимостью - А, соответствующую ей последовательность кода без уязвимости - Б. Полученную последовательность А (кода с уязвимостью) подают в блок энкодера (кодировки) генеративной модели машинного обучения. Получают матрицу скрытых состояний (107). Данную матрицу скрытых состояний подают в блок генератора (декодера) генеративной модели машинного обучения (108). Получают новую (восстановленную) упорядоченную последовательность - В.

Данную последовательность (В) сравнивают с исходной последовательностью Б (кода без уязвимости). Полученные несоответствия пересчитывают в количественную характеристику по выбранной функции потерь. Данную количественную характеристику используют для обучения нейронной сети по выбранному алгоритму обучения.

В одном из частных вариантов изобретения используется по меньшей мере одна из функций потерь: KLD (Вычисляет потерю дивергенции Кулбека-Лейблера между истинным значением и предсказанным значением), MAE (Вычисляет среднюю абсолютную ошибку между метками и прогнозами), MAPE (Вычисляет среднюю абсолютную процентную ошибку между истинным значением и предсказанным значением), MSE (Вычисляет среднеквадратичную ошибку между метками и прогнозами), MSLE (Вычисляет среднеквадратичную логарифмическую ошибку между истинным значением и предсказанным значением), binary_crossentropy (Вычисляет двоичную потерю кроссэнтропии), binary_focal_crossentropy (Вычисляет двоичную потерю фокальной кроссэнтропии), categorical_crossentropy (Вычисляет категориальную потерю кроссэнтропии), categorical_hinge (Вычисляет категориальную потерю "лассо" между истинным значением и предсказанным значением), cosine_similarity (Вычисляет косинусное сходство между метками и предсказаниями), hinge (Вычисляет потери в шарнире между истинным значением и предсказанным значением), huber (Вычисляет величину потерь по Хуберу), kl_divergence (Вычисляет потерю дивергенции Кулбека-Лейблера между истинным значением и предсказанным значением), kullback_leibler_divergence (Вычисляет потерю дивергенции Кулбека-Лейблера между истинным значением и предсказанным значением), log_cosh (Логарифм гиперболического косинуса ошибки прогнозирования), logcosh (Логарифм гиперболического косинуса ошибки прогнозирования), mean_absolute_error (Вычисляет среднюю абсолютную ошибку между метками и прогнозами), mean_absolute_percentage_error (Вычисляет среднюю абсолютную процентную ошибку между истинным значением и предсказанным значением), mean_squared_error (Вычисляет среднеквадратичную ошибку между метками и прогнозами), mean_squared_logarithmic_error (Вычисляет среднеквадратичную логарифмическую ошибку между истинным значением и предсказанным значением), poisson (Вычисляет потери Пуассона между истинным значением и предсказанным значением), sparse_categorical_crossentropy (Вычисляет разреженную категориальную потерю кроссэнтропии), squared_hinge (Вычисляет квадрат потерь на "лассо" между истинным значением и предсказанным значением). Данные функции потерь часто имеют следующие обозначения: binary_cross_entropy, binary_cross_entropy_with_logits, poisson_nll_loss, cosine_embedding_loss, cross_entropy, ctc_loss, gaussian_nll_loss, hinge_embedding_loss, kl_div, ll_loss, mse_loss, margin_ranking_loss, multilabel_margin_loss, multilabel_soft_margin_loss, multi_margin_loss, nll_loss, huber_loss, smooth_ll_loss, soft_margin_loss, triplet_margin_loss, triplet_margin_with_distance_loss.

В одном из частных вариантов изобретения используется по меньшей мере один из алгоритмов обучения: Adadelta, Adagrad, Adam, AdamW, SparseAdam, Adamax, ASGD, LBFGS, NAdam, RAdam,

RMSprop, Rprop, FTRL, SGD, FastSGD, SGD-Nesterov, SAGA, SAGA+.

Далее на этапе (108) переводят полученную на предыдущем этапе упорядоченную последовательность в блок исходного кода, эквивалентный по функционалу изначальному блоку исходного кода.

В результате, данный подход позволяет формировать и приоритизировать задачи по устранению недостатков в разрабатываемом программном обеспечении, тем самым повышая скорость разработки программного кода, уменьшает количество ошибок в программном коде, а также напрямую влияет на безопасность программного кода, что обеспечивает повышение скорости обновления программного обеспечения или вывода программного обеспечения на рынок за счет исключения из 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.

На фиг. 2 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 технического решения.

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 заявки раскрывают предпочтительные примеры реализации технического решения 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 устранения уязвимостей в программном коде, выполняемый с помощью по меньшей мере одного процессора и содержащий этапы, на которых:

получают данные об уязвимостях в программном коде;

получают на основе предыдущего этапа данные, содержащие по меньшей мере блок 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;

формируют путь обхода вершин в AST;

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

осуществляют обработку упорядоченной последовательности с помощью кодирующей модели машинного обучения, обученной на упорядоченных последовательностях данных об уязвимостях, в ходе которой получают представление упорядоченной последовательности в виде матрицы скрытых состояний;

осуществляют обработку матрицы скрытых последовательностей с помощью генеративной модели машинного обучения, обученной на матрицах скрытых состояний последовательностей для уязвимостей, последовательностей без уязвимостей и их исходных кодах, в ходе которой получают новую упорядоченную последовательность, соответствующую исходному программному коду по выполняемым функциям, но с устранённой в нем уязвимостью;

переводят полученную на предыдущем этапе упорядоченную последовательность в блок исходного кода, эквивалентный по функционалу изначальному блоку исходного кода.

2. Способ по п.1, характеризующийся тем, что упорядоченная последовательность, которая представляет собой типы каждого элемента пути, является многомерной и представляет собой граф.

3. Способ по п.1, характеризующийся тем, что преобразуют исходный код, содержащий уязвимость в граф потока управления (CFG - control flow graph), в котором внутренние вершины сопоставлены с операторами языка программирования, а листья - с соответствующими операндами.

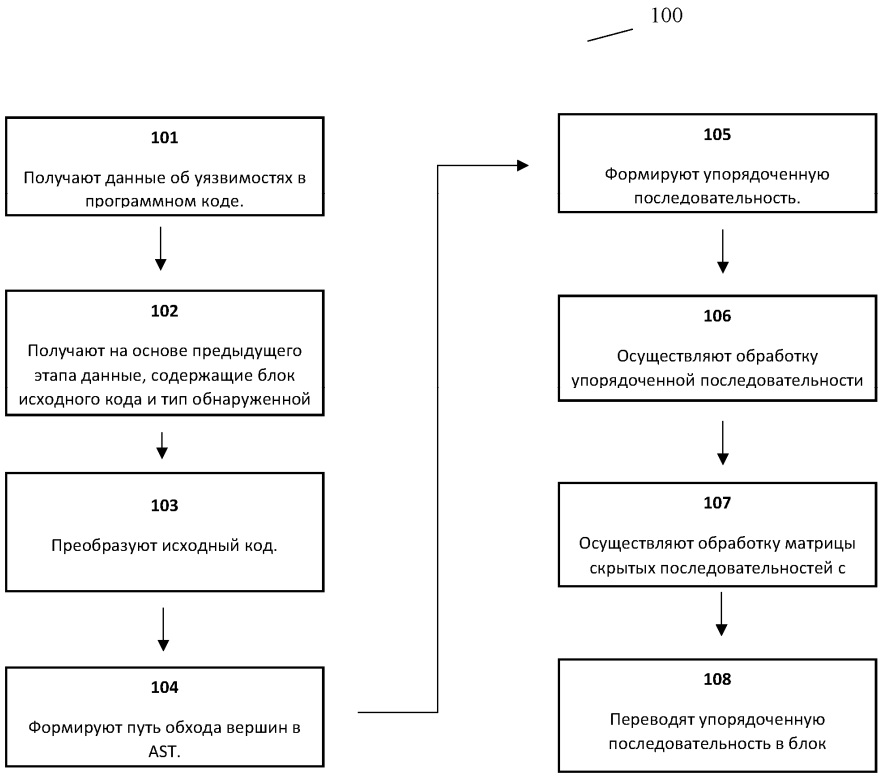
4. Способ по п.1, характеризующийся тем, что преобразуют исходный код, содержащий уязвимость в граф зависимостей управления (CDG - Control Dependence Graphs), в котором внутренние вершины сопоставлены с операторами языка программирования, а листья - с соответствующими операндами.

5. Способ по п.1, характеризующийся тем, что преобразуют исходный код, содержащий уязвимость в граф зависимости данных (DDG - Data Dependence Graph), в котором внутренние вершины сопоставлены с операторами языка программирования, а листья - с соответствующими операндами.

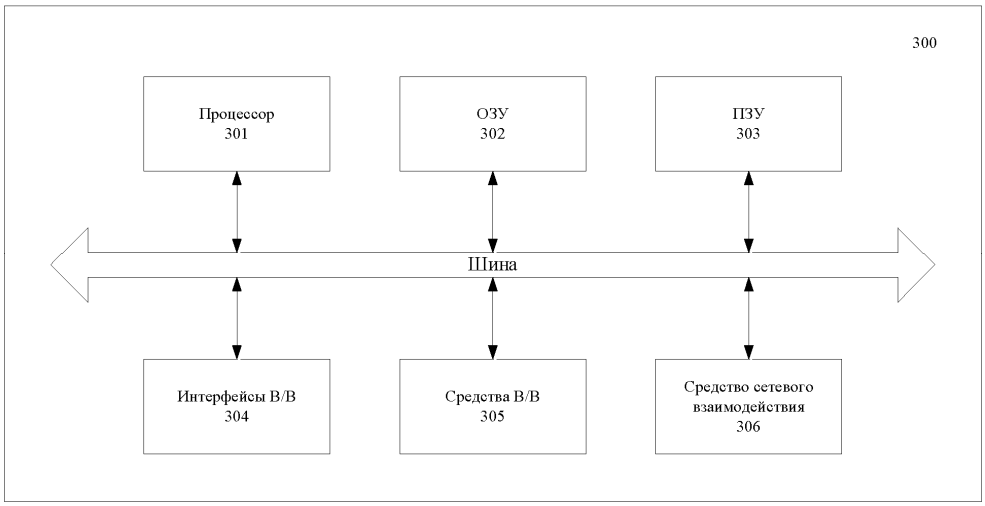
6. Способ по п.1, характеризующийся тем, что преобразуют исходный код, содержащий уязвимость в граф зависимости программы (PDG - Program Dependence graphs), в котором внутренние вершины сопоставлены с операторами языка программирования, а листья - с соответствующими операндами.

7. Способ по п.1, характеризующийся тем, что преобразуют исходный код, содержащий уязвимость в граф свойств кода (CPG - Code Property Graphs), в котором внутренние вершины сопоставлены с операторами языка программирования, а листья - с соответствующими операндами.

8. Система устранения уязвимостей в программном коде, содержащая по меньшей мере один процессор и память, хранящую машиночитаемые инструкции, которые при их выполнении процессором реализуют способ по любому из пп.1-7.


100
101
Получают данные об уязвимостях в программном коде.
102
Получают на основе предыдущего этапа данные, содержащие блок исходного кода и тип обнаруженной
103
Преобразуют исходный код.
104
Формируют путь обхода вершин в AST.
105
Формируют упорядоченную последовательность.
106
Осуществляют обработку упорядоченной последовательности
107
Осуществляют обработку матрицы скрытых последовательностей с
108
Переводят упорядоченную последовательность в блок

Фиг. 1


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

Фиг. 2