MALWARE DETECTION & CLASSIfiCATION USING MACHINE LEARNING

Sobirjonov Umidjon Javlon o'g'li

Abstract

In today's internet world, malware is still the most harmful threat to the internet users. The new malware developed are distinct from conventional one, more dynamic in design and usually inherits the properties from two or more malware types, these type of malware are called polymorphic. Polymorphic malware is a form of malware which constantly modifies its recognisable features to fool detection using traditional signature-based models. Behavior-based identification of ransomware tests not just the file's identity, but also the operation it intends to take after some time span or at specific time. Now everyone wanted to get the behavioural pattern that can be derived from static analysis or dynamic analysis, with these pattern various machine learning models can be used to predict whether it is a malware or not, or identify its family of malware. In this work, behavior-based detection methods are address and how these various machine learning techniques are used to develop behavior-based malware detection and classification methods.

Keywords: Static Analysis; Machine Learning; Detection &Classification;

I. Введение

В наши дни использование Интернета, компьютеров и интеллектуальных устройств очень распространено, и многие люди используют эти устройства каждый день. Куда бы мы ни пошли, везде есть люди с хорошими и плохими намерениями, это тот же случай в мире Интернета, что и люди с плохими мотивами, которые хотят воспользоваться настоящими пользователями для своей личной выгоды [8].В последние годы атаки вредоносных программ становятся все более сложными с точки зрения сроков. Обнаружение и классификация вредоносных программ важны, поскольку это должно быть сделано правильно, к какому семейству вредоносных программ они принадлежат, и на этой основе могут быть разработаны решения для

предотвращения вредоносных программ или защиты от вредоносных программ с уникальной сигнатурой для их идентификации. Вредоносное ПО имеет различные типы в зависимости от того, каковы их мотивы разработки, например, для шпионажа используются шпионские программы, для финансовых целей используются программы-вымогатели и т.д. Чтобы провести анализ вредоносных программ с использованием машинного обучения, мы должны иметь базовое представление о типах вредоносных программ и методах, которые они использовали. Как они разделились в классе, основываясь на их поведении.

Как указано ниже:

Вирус - такой же, как и другие программы. Основное отличие заключается в том, что программа запускается в системе без предварительного разрешения пользователя и также реплицирует себя для заражения других программ на компьютере.

Червь - это не что иное, как улучшенная версия вируса. Основное отличие заключается в том, что все машины, подключенные по сети, подвергаются риску, и существует вероятность того, что они будут заражены червем.

Троянец – Основная цель дизайна трояна состоит в том, чтобы он выглядел как законное программное обеспечение, чтобы пользователи были обмануты, рассматривая вредоносное как законное.

Бэкдор – это тип вредоносного ПО, которое используется для создания задней двери для входа в целевую машину. Это не сильно влияет на систему.

А. Необходимость машинного обучения

Доступные антивирусные решения в основном обнаруживают вредоносное ПО на основе сигнатур. Эти сигнатуры для обнаружения вредоносного ПО являются производными от ранее собранных образцов вредоносного ПО. Эти решения, основанные на сигнатурах, работают очень хорошо, если вредоносное ПО известно ранее, но если вредоносное ПО ранее не было известно, то оно не может обнаружить новые образцы. В результате решений

на основе сигнатур не всегда достаточно. Частота ложноположительных и негативов также высока [3]. Для противодействия угрозе улучшенных версий вредоносных программ необходимы новые механизмы обнаружения. Возможными решениями этой проблемы являются [9] анализ на основе сигнатур, который может быть интегрирован с методами машинного обучения, которые могут обеспечить более высокую точность при обнаружении, чем только однократное использование подхода, основанного на сигнатурах когда используется подход без сигнатур, необходимо учитывать некоторые предопределенные ориентиры, чтобы классифицировать программу как вредоносную.

II. МЕТОДЫ АНАЛИЗА ВРЕДОНОСНЫХ ПРОГРАММ

В этом разделе перечисленные здесь методы машинного обучения используются при анализе вредоносных программ. Согласно автору [7] анализ вредоносного ПО состоит из трех основных этапов, а именно: Цели, функции и алгоритм машинного обучения. При обнаружении вредоносных программ основное внимание уделяется обнаружению, является ли данный образец вредоносным? При анализе сходства вредоносных программ здесь мы можем знать, что файл является вредоносным, и искать функции, которые могут четко классифицировать образец и помочь в обнаружении вредоносных программ. После завершения достижения цели второй этап анализа предназначен для извлечения функций на основе потребности. После этого последнего этапа необходимо применить алгоритм машинного обучения для достижения поставленной цели [7].

A. Цели анализа вредоносных программ

При анализе вредоносных программ основное внимание уделяется обнаружению. Вредоносные программы обнаруживаются на основе уникальной сигнатуры, разработанной на основе предыдущих собранных образцов и правильно классифицированной как вредоносная или

доброкачественная [7]. Первая главная цель всегда состоит в том, чтобы определить, является ли данный образец вредоносным или нет. Основная цель большей части обзорной работы заключается в обнаружении вредоносного ПО [1], [2], [5].

B. Извлечение функций

Для извлечения функции из двоичного файла вредоносного ПО существует два подхода, в основном статический или динамический анализ, используемый отдельно или в комбинации. При статическом анализе вредоносный файл проверяется без выполнения, все функции извлекаются в основном из заголовка PE или разбираются на исполняемый файл и анализируются на языке ассемблера. При динамическом анализе исполняемый файл запускается в среде управления, и его поведение отслеживается, например, какие системные вызовы динамически вызываются, которые не являются кодом, пытается подключиться к какой-либо внешней сети, пытается выполнить какие-либо изменения в файле реестра. Все эти следы и действия используются для анализа вредоносных программ [6].

Ниже приведены некоторые функции, которые в основном используются в рецензируемых статьях для достижения конечной цели анализа вредоносных программ.

III. МЕТОДЫ МАШИННОГО ОБУЧЕНИЯ

A. Light GBM(классификатор повышения градиента)

Light GBM - это быстрый, легковесный алгоритм, который обучается на основе древовидной структуры. Повышение градиента в основном используется для классификации и регрессии. Это комбинация различных алгоритмов, на основе которых создается новая итеративная модель. Использование градиентного бустинга увеличивает эффективность машинного обучения, поскольку оно обеспечивает более высокую точность и эффективность [4]. Основное различие между light GBM и другим алгоритмом заключается в том, что он выращивает дерево вертикально, т.е. оно растет по

листьям, как показано на рисунке 1, в то время как другие алгоритмы повышения градиента растут горизонтально, т.е. они растут по уровню, как показано на рисунке 2. Это ясно объясняется на следующих рисунках:

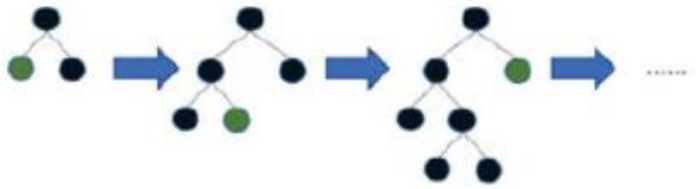

Рисунок 1 - Рост ЛГБМ по листьям

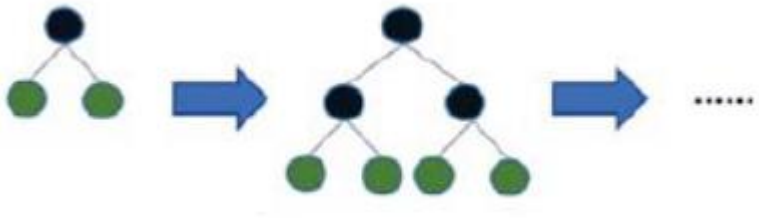

Рисунок 2 - Рост по другим уровням

Математический анализ для метода GOSS (вычисление прироста дисперсии при разделении признака j).Давайте рассмотрим обучающий набор, имеющий n экземпляров x_1, …, x_n, где каждый x_i представляет собой вектор с размерами в пространстве X_S.На каждой итерации повышения градиента градиенты, которые дают отрицательный результат в функции потерь по отношению к выходным данным модели, идентифицируются как $g_1,\cdots,g_n$. В этом подходе GOSS во время обучения экземпляры ранжируются в соответствии с результатом их соответствующих градиентов в порядке убывания. Тогда

$$(\text{top} - a^x) \text{ x } 100\%$$

Подмножеством экземпляров является форма, которая включает градиенты, имеющие большие значения. Затем дополнение множества A, т.е. A^c, состоящее

$$(1 - a^x) \text{ x } 100\%$$

экземпляры, имеющие градиенты с меньшими значениями. После этого подмножество B является случайной выборкой, имеющей размер (b x A^c)

Наконец, экземпляры разделяются в соответствии с оцененным коэффициентом усиления дисперсии в векторе V_j (d) по подмножеству A ? B. Приведенное ниже уравнение (1) предназначено для усиления дисперсии с использованием метода GOSS.

$$\bar{V}_j = \frac{1}{n}\left(\frac{(\sum_{x_{i\in A_l}} g_i + \frac{(1-a)}{b}\sum_{x_{i\in b_l}} g_i)^2}{{n_r}^j(d)} + \frac{(\sum_{x_{i\in A_r}} g_i + \frac{(1-a)}{b}\sum_{x_{i\in B_r}} g_i)^2}{{n_r}^j(d)}\right)$$

Где A_l={x_i ? A : x_{ij} ? d }, A_r={ x_i ? A : x_{ij} > d }, B_l={ x_i ? B : x_{ij} ? d }, B_r = { x_i ? B : x_{ij} > d }, и коэффициент $\frac{(1-a)}{b}$ используется для нормализации суммы градиентов по B обратно к размеру A^c

B. Дерево решений

Дерево решений в основном используется для построения классификационных и регрессионных моделей. В проект для классификации вредоносного ПО используется классификатор дерева решений. Модель классификации, которую он строит, похожа на древовидную структуру. Дерево решений - это контролируемое обучение. На рисунке 3 показано дерево решений, которое предсказывает погоду для одного дня.

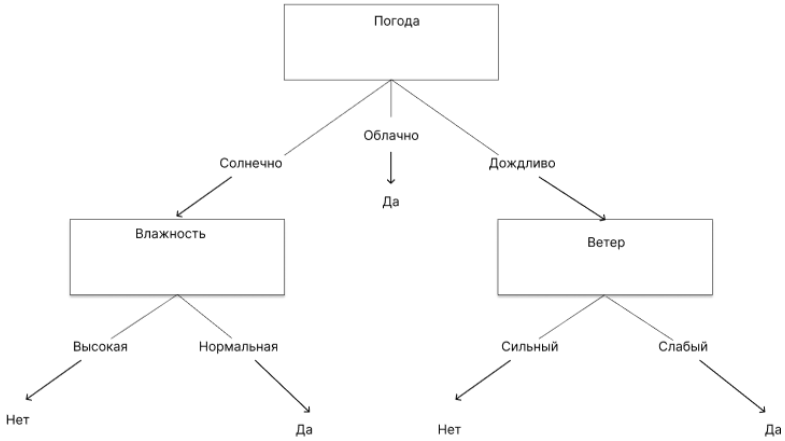

Рисунок 3 – Дерево решений

C. Метод случайного леса(Random Forest)

Случайный лес - это контролируемый алгоритм классификации. Случайный лес напрямую связан с алгоритмом дерева решений, поскольку он больше похож на коллективные деревья решений, которые могут образовывать лес. Точность модели случайного леса сильно зависит от количества деревьев в лесу, небольшое количество деревьев дает низкую точность, но это быстро с точки зрения производительности, в то время как большое количество деревьев дает более высокую точность, но это медленнее с точки зрения исполнения. На рисунке 4 объясняется структура для случайного формирования леса.

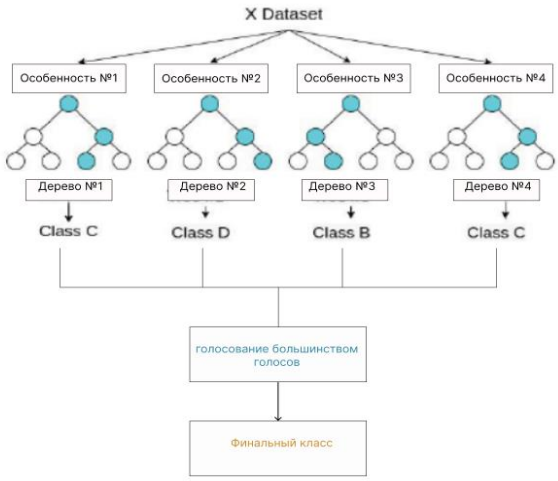

Рисунок 4 – Классификатор случайного леса

IV. ПРЕДЛАГАЕМАЯ АРХИТЕКТУРА

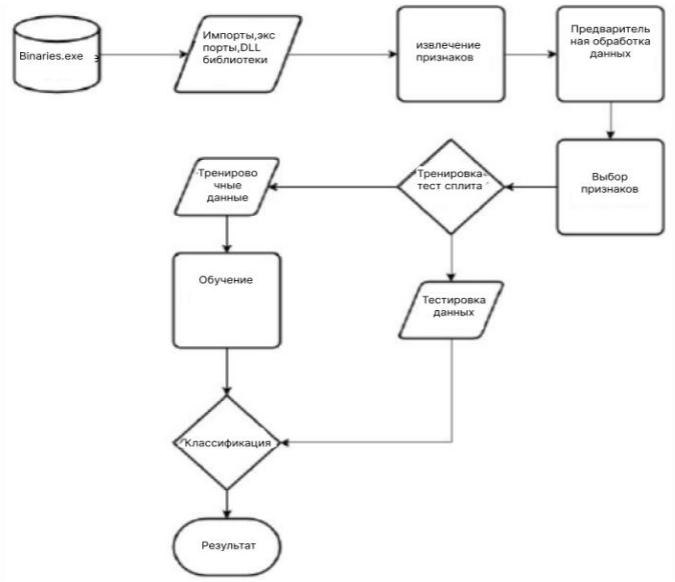

Рисунок 5 – Архитектура

Предлагаемая архитектура для обнаружения и классификации вредоносных программ показана на рисунке 5. Исходя из архитектуры, общий процесс делится на следующие этапы:

1) Создание набора данных

Извлечение статической информации о программе или программном обеспечении с помощью PE-файла библиотеки python и генерирование excel для

хранения информации для каждой программы или программного обеспечения

2) Предварительная обработка данных

Предварительная обработка данных является одним из важных этапов перед выбором функции для модели. При предварительной обработке данных нужно удалить нулевые значения из набора данных, если они присутствуют. Нужно удалить столбцы, содержащие категориальные данные.

3) Выбор функции

Выбор признаков, которые тесно связаны с выходными данными или важны для прогнозирования или классификации заданных входных данных. После выбора подходящих функций разделение набора данных на два раздела для целей обучения и тестирования в зависимости от необходимости.

4) Обучение

После успешного завершения вышеуказанного шага нужно обучить набор обучающих данных желаемому алгоритму, выбранному для классификации.

5) Классификация

Это последний шаг, на котором фактическая классификация прогнозируется на основе обучения, проведенного на предыдущем шаге.

V. ПРАКТИЧЕСКАЯ РАБОТА И РЕЗУЛЬТАТЫ

A. Практическая работа

1) Предварительная обработка данных: В этой части данные подготавливаются для процесса обучения. Удаляются первые столбцы, содержащие строковые значения, а также последний столбец, содержащий значение, которое сообщает о том, является ли двоичный файл вредоносным или законным. Из 54 признаков 13 признаков выбираются с помощью дополнительного древовидного классификатора.

2) Обучение: Для обучения модели рассматриваются 3 различных алгоритма, а именно. Дерево решений, Случайный лес и Light GBM.

B. Результаты

1) Дерево решений. Классификатор дерева решений задается с входными значениями X_train для обучения. После завершения процесса обучения

точность алгоритма дерева решений проверяется с помощью X_test значений. С помощью классификатора дерева решений достигается точность 99,14%. На рисунке 6 показана матрица неточностей для классификатора дерева решений.

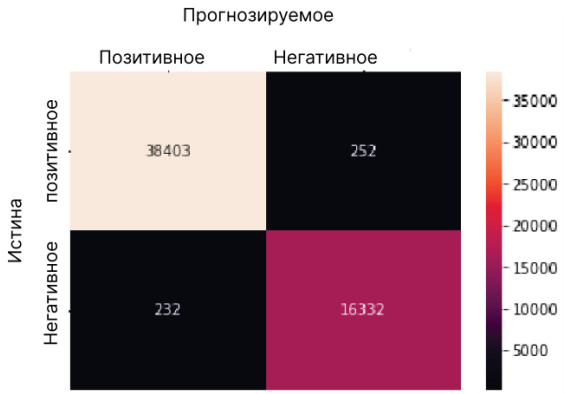

Рисунок 6 – матрица неточностей для классификатора дерева решений

2) Случайный лес. Классификатор случайного леса задается с входными значениями X_train для обучения. Классификатор случайного леса требует больше времени для обучения по сравнению с алгоритмом дерева решений и Light GBM, но достигаемая точность выше, чем у дерева решений, но ниже, чем у Light GBM. После завершения процесса обучения точность алгоритма случайного леса проверяется с помощью X_test значений. С помощью классификатора случайных леса достигается точность 99,47%. На рисунке 7 показана матрица неточностей для классификатора случайных лесов.

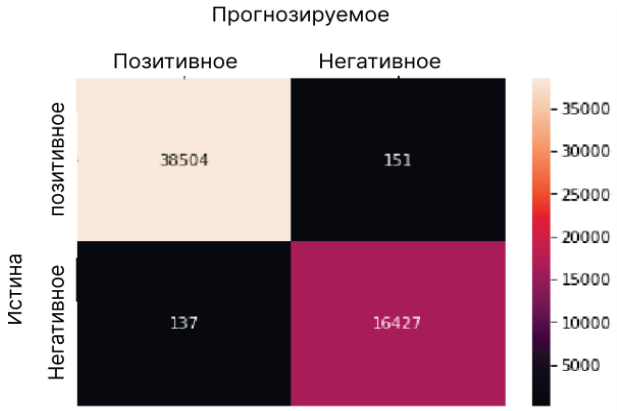

Рисунок 7 – матрица неточностей для классификатора случайного леса

3) Light GBM. Light GBM задается с входными значениями X_train для обучения. Light GBM при обучении работает быстрее, чем алгоритмы дерева

решений и случайного леса. Кроме того, достигаемая точность является самой высокой среди двух других. После завершения процесса обучения точность алгоритма Light GBM проверяется с помощью значений X_test. С помощью классификатора Light GBM достигается точность 99,50%. На рисунке 8 показана матрица неточностей для классификатора Light GBM.

Рисунок 8 – матрица неточностей для классификатора Light GBM

C. Сравнение результатов

В таблице I приведены результаты по всем трем использованным классификаторам. Из таблицы I видно, что в случае точности LightGBM имеет самый высокий показатель, но ЛПР и ИНР немного ниже, чем у классификатора случайного леса.

Classifier	Accuracy	TPR	FNR	FPR	TNR
Decision Tree	99.14%	98.61%	1.39%	0.65%	99.35%
Random Forest	99.47%	99.17%	0.83%	0.39%	99.61%
Light GBM	99.50%	99.29%	0.71%	0.41%	99.59%

ТАБЛИЦА I

ТАБЛИЦА:СРАВНЕНИЕ РЕЗУЛЬТАТОВ

Из таблицы I видно, что алгоритм light GBM обладает наивысшей точностью по сравнению с моделью случайного леса и деревом решений. Кроме того, если мы сравним время, затрачиваемое алгоритмом случайного леса на обучение, оно будет самым высоким по сравнению с двумя другими. Это может увеличить накладные расходы на производительность при обучении очень больших наборов данных. В таких сценариях Light GBM лучше всего

подходит для обучения больших наборов данных с меньшими затратами на производительность.

VI. Заключение

Цель, поставленная перед исследованием, была достигнута. Рассмотренный алгоритм машинного обучения был успешно оценен.

Алгоритм машинного обучения был применен к статическим функциям, извлеченным из законных и вредоносных исполняемых файлов. При таком подходе, является данный файл вредоносным или нет, будет предсказано очень быстро. Дерево решений имеет более низкую точность около 99,14%, в то время как случайный лес имеет лучшую точность около 99,47%. Машинный алгоритм Light GBM достиг наивысшей точности среди всех - примерно на 99,50%, что немного больше, чем алгоритм случайного леса. Машинный алгоритм Light GBM эффективен с точки зрения точности, а также времени, затрачиваемого на обучение модели. Случайный лес работает намного медленнее по сравнению с Light GBM. Ложноотрицательный (предсказывающий вредоносный код как законный) показатель Light GBM является самым низким среди всех, что является одним плюсом, который необходимо учитывать при выборе основного алгоритма классификации. Хотя желательно, чтобы в таких исследованиях ложноотрицательные результаты были равны нулю или почти близки к нулю. Если частота ложных срабатываний выше, то модель не используется в производственной среде. Я хотел бы завершить эту работу, заявив, что есть некоторые области, которые требуют дополнительных исследований в данном направлении. Поскольку использование Интернета растет и каждый день разрабатываются новые вредоносные программы, для борьбы с ними необходимы новые методы.

Список использованных литератур

[1] Blake Anderson, Daniel Quist, Joshua Neil, Curtis Storlie, and Terran Lane. Graph-based malware detection using dynamic analysis. Journal

in computer Virology, 7(4):247–258, 2011.

[2] Jinrong Bai, Junfeng Wang, and Guozhong Zou. A malware detection scheme based on mining format information. The Scientific World Journal, 2014.

[3] Balaji Baskaran and Anca Ralescu. A study of android malware detection techniques and machine learning. In Proceedings of the 27th Modern Artificial Intelligence and Cognitive Science Conference, volume 1584 of CEUR Workshop Proceedings, pages 15–23. CEUR-WS.org, 2016.

[4] Mouhammd Al-kasassbeh Mohammad A. Abbadi, Ahmed M. Al-Bustanji. Robust intelligent malware detection using lightgbm algo-rithm. International Journal of Innovative Technology and Exploring Engineering (IJITEE), 9, 04 2020.

[5] Igor Santos, Jaime Devesa, Felix Brezo, Javier Nieves, and Pablo Garcia Bringas. Opem: A static-dynamic approach for machine-learning-based malware detection. In International Joint Conference CISIS'12-ICEUTE 12-SOCO 12 Special Sessions, pages 271–280. Springer, 2013.
(MALWARE), pages 11–20. IEEE, 2015.

[6] Michael Sikorski and Andrew Honig. Practical Malware Analysis: The Hands-On Guide to Dissecting Malicious Software. No Starch Press, USA, 1st edition, 2012.

[7] Daniele Ucci, Leonardo Aniello, and Roberto Baldoni. Survey of machine learning techniques for malware analysis. Computers & Security, 81:123–147, 2019.

[8] Yanfang Ye, Tao Li, Donald Adjeroh, and S Sitharama Iyengar. A survey on malware detection using data mining techniques. ACM Computing Surveys (CSUR), 50(3):1–40, 2017.