МАТЕМАТИЧЕСКИЕ ОСНОВЫ КОМПЬЮТЕРНОЙ БЕЗОПАСНОСТИ

УДК 004.85 DOI 10.17223/20710410/56/5

ОБНАРУЖЕНИЕ АНОМАЛИЙ В СОСТАВНЫХ ОБЪЕКТАХ В ФОРМАТЕ JSON[1]

Е. А. Шляхтина*,**, Д. Ю. Гамаюнов*

*Московский государственный университет имени М. В. Ломоносова, г. Москва, Россия
**ООО «Солидсофт», г. Москва, Россия

E-mail: elena.shlyakhtina@solidwall.io, gamajun@seclab.cs.msu.su

Работа посвящена проблеме защиты от компьютерных атак современных веб-приложений и мобильных приложений с «облачной» серверной частью. Рассматривается задача обнаружения вредоносного содержимого в данных в формате JSON, который стал одним из самых распространённых способов сериализации и передачи объектов между клиентской и серверной частями приложений. Предложен метод построения эталонной модели для некоторой заданной коллекции JSON-объектов, на основе которой можно обнаруживать аномалии различных типов. Эталонная модель строится на основе моделей простых типов, входящих в состав объектов из коллекции, а также схемы, которая обобщает их структуру. Экспериментально исследован метод построения эталонной модели с использованием модификаций, затрагивающих структуру JSON-объекта, а также внедрения SQL-инъекций, инъекций команд ОС и JavaScript/HTML-инъекций. Проведён анализ статистической значимости предсказаний модели, измерено качество работы модели, определяемое коэффициентом корреляции Мэтьюса, на тестовых выборках, состоящих из данных, взятых из трафика реальных веб-приложений.

Ключевые слова: *безопасность веб-трафика, обнаружение аномалий, машинное обучение.*

ANOMALY DETECTION IN JSON STRUCTURED DATA

E. A. Shliakhtina*,**, D. Y. Gamayunov*

*Lomonosov Moscow State University, Moscow, Russia
**SolidSoft, LLC, Moscow, Russia

In this paper, we address the problem of intrusion detection for modern web applications and mobile applications with the cloud-based server side, using malicious content detection in JSON data, which is currently one of the most popular data serialization and exchange formats between client and server parts of an application. We propose a method for building a JSON model for the given set of JSON objects capable of detection of structure and type anomalies. The model is based on the models for basic data types inside JSON collection objects and schema model that generalizes objects' structure in the collection. We performed experiments using modifications of objects'

structures and insertions of code injection attack vectors such as SQL injections, OS command injections, and JavaScript/HTML injections. The analysis showed statistical significance between the model's predictions and the presence of anomalies in the data gathered from the real web applications' traffic. The quality of the model's predictions was measured using the Matthews correlation coefficient (MCC). The MCC values computed on the data were close to one which indicates the model's high efficiency in solving the problem of anomaly detection in JSON objects.

Keywords: *web traffic security, anomaly detection, machine learning*

Введение

В настоящее время одним из самых распространённых способов сериализации и обмена данными между клиентской и серверной частями приложений является формат данных JavaScript Object Notation (JSON) — он используется как в динамических веб-приложениях, так и для обмена между мобильным клиентом и «облачной» серверной частью приложения, а также для реализации внешних API-приложений. При этом различного рода вредоносная активность и компьютерные атаки, которые эксплуатируют некоторую уязвимость в приложении, также используют формат JSON для взаимодействия с серверной частью приложения — например, автоматизированная переборная активность для некоторого API будет заключаться в отправке запросов с перебираемым значением некоторого параметра внутри JSON-объекта, а поиск уязвимостей внедрения кода или выхода из контекста данных в контекст команд злоумышленник будет в том числе реализовывать подстановкой атакующих векторов в поля значений в JSON-объектах. Таким образом, можно пытаться обнаружить такие атаки за счёт обнаружения аномалий в структуре или типах данных в JSON-объектах внутри HTTP-запросов к приложению. Так возникает задача построения эталонной модели для коллекции JSON-объектов с целью обнаружения недопустимого вредоносного содержимого в данных такого типа. Впоследствии такая модель может быть использована в системах защиты приложений и автоматизированного API для контроля потоков данных в формате JSON.

Типичным средством защиты веб-приложений от компьютерных атак в настоящее время является межсетевой экран WAF (Web Application Firewall), работающий на прикладном уровне и анализирующий HTTP-трафик. WAF располагается перед защищаемым веб-сервером и работает в режиме обратного прокси-сервера. Просматривая пакеты информации в трафике, WAF строит эталонную модель данных, что в режиме анализа позволяет ему принимать решение о допустимости анализируемого трафика, который может содержать различные атаки на уровне приложений. Существует и другой подход для обнаружения вредоносного содержимого в трафике — это распознавание атак на основе определённых паттернов или сигнатур, однако существенным недостатком таких систем является то, что ранее неизвестные атаки могут быть найдены с меньшей вероятностью, в отличие от систем обнаружения аномалий.

Данная работа посвящена построению модели, определяющей наличие аномального содержимого в данных в формате JSON. В её основе лежат модели простых типов, входящих в состав JSON-объектов из коллекции, а также схема JSON-объектов, которая описывает структуру всех объектов коллекции.

1. Постановка задачи

Цель данной работы — разработка модели для коллекции составных объектов в формате JSON, определяющей класс («аномальный», «нормальный»), к которому принадлежит объект, лучше, чем случайное гадание.

Объект считается аномальным, если содержит одну из следующих атак, которые подробно описаны в п. 2:

1) изменение имени JSON-атрибута;
2) добавление JSON-атрибута;
3) удаление JSON-атрибута;
4) изменение значения JSON-атрибута: внедрение SQL-инъекции;
5) изменение значения JSON-атрибута: внедрение инъекции команд ОС;
6) изменение значения JSON-атрибута: внедрение JavaScript/HTML-инъекции.

Условия, в рамках которых проводится исследование:

1) рассматриваются синтаксически корректные JSON-объекты, как аномальные, так и нормальные;
2) аномальный JSON-объект содержит только одну атаку из списка выше;
3) все данные до их модификации считаются нормальными.

Сформулируем нулевую и альтернативную гипотезы:

— $\mathbf{H_0}$: предсказание модели об аномальности объекта из JSON-коллекции и его истинный класс являются независимыми при любых выбранных атаках из списка выше, их носителях из коллекции и самой коллекции;
— $\mathbf{H_1}$: предсказание модели об аномальности объекта из JSON-коллекции и его истинный класс не являются независимыми хотя бы при некоторых выбранных атаках из списка выше, их носителях из коллекции и самой коллекции.

Выберем уровень значимости $\alpha = 0{,}01$ и в соответствии с ним в результате исследования с использованием критерия хи-квадрат примем решение о принятии или отклонении нулевой гипотезы.

2. Основные понятия

2.1. JSON-формат

JSON — это текстовый формат данных, основанный на синтаксисе объекта JavaScript. Он часто используется для передачи данных в веб-приложениях, поскольку многие среды программирования имеют функционал для работы с ним.

Данные в формате JSON представляют собой один из следующих объектов:

— record (запись) — набор пар (атрибутов) *ключ* : *значение*, разделённых запятыми, и заключенный в фигурные скобки ({ });
— array (массив) — упорядоченный набор значений, разделённых запятыми, и заключенный в прямые скобки ([ ]).

Составными типами будем называть record и array.

Простыми типами будем называть:

— string (строка) — заключённое в двойные кавычки упорядоченное множество символов юникода;
— number (число) — целое или вещественное число;
— boolean (булевский тип) — принимает значения `true`/`false`;
— null — ключевое слово в JavaScript, которое обозначает отсутствующий объект.

Типом ключа может быть только string, а типом значения — любой (составной или простой). В листинге 1 приведён пример JSON-объекта.

```
{
    "string": "Hello World!",
    "number": 12.05,
    "boolean": true,
    "record": {
        "number": 2,
        "array": [true, 7, null]
    }
}
```

Листинг 1. Пример данных в формате JSON

2.2. JSON-схема

JSON-схема — это объект в формате JSON, описывающий типы данных и структуру некоторого JSON-объекта. Схема может быть использована для валидации структуры данных, обрабатываемых на веб-сервере. Существуют различные стандарты описания структуры JSON-объекта, например JSON Schema [1]. В листинге 3 приведён пример схемы этого стандарта для JSON-объекта листинга 2.

```
{
    "name": "Tom",
    "age": 33,
    "kids": ["Sam", "Tony"]
}
```

Листинг 2. Пример данных в формате JSON

```
{
    "$schema": "http://json-schema.org/draft-07/schema",
    "$id": "/example/",
    "type": "object",
    "required": [
        "name",
        "age",
        "kids"
    ],
    "properties": {
        "name": {
            "$id": "#name",
            "type": "string"
        },
        "age": {
            "$id": "#age",
            "type": "number"
        },
        "kids": {
            "$id": "#kids",
            "type": "array",
            "items": {
```

```
23                  "$id": "#items",
24                  "anyOf": [
25                      {
26                          "$id": "#0",
27                          "type": "string"
28                      }
29                  ]
30              }
31          }
32      },
33      "additionalProperties": true
34  }
```

Листинг 3. Пример схемы JSON-объекта листинга 2

Схема фиксирует структуру объекта и ограничивает типы данных для значений, соответствующих конкретным ключам, могут быть указаны также дополнительные сведения, например максимальное и минимальное допустимые значения типа number или обязательность/опциональность ключей в записи объекта, всё это зависит от выбора алгоритма построения схемы.

2.3. Атаки на веб-приложения

Рассмотрим ряд атак на веб-приложения, векторы которых могут быть встроены в JSON-объект, попадающий на веб-сервер. В рейтинге самых распространённых уязвимостей веб-приложений «OWASP Top 10 2021» сообщества Open Web Application Security Project (OWASP) инъекционные атаки входят в первую тройку. Главной причиной уязвимости, связанной с использованием инъекций, является отсутствие проверки данных, полученных и используемых веб-приложением, что ведёт к необходимости валидации данных.

SQL-инъекция

Structured Query Language (SQL) — это специальный язык, используемый в большинстве реляционных баз данных для запросов к ним, обработки данных и контроля доступа. SQL-инъекция — это атака, при которой в процессе конструировния SQL-выражения на веб-сервере, благодаря особенному пользовательскому вводу, полностью меняется логика выполнения SQL-запроса. При недостаточной проверке данных от пользователя злоумышленник может внедрить в форму веб-интерфейса приложения специальный код, содержащий кусок SQL-запроса. Такая манипуляция позволяет злоумышленнику исполнить произвольный запрос к базе данных и получить незаконный доступ к данным, хранящимся в этой базе.

Инъекция команд ОС

Инъекция команд ОС (Command Injection) — это атака, целью которой является выполнение произвольных команд операционной системы с помощью уязвимого веб-приложения. Например, если при разработке веб-приложения существует функционал, позволяющий пользователю взаимодействовать с ОС веб-сервера, то злоумышленнику достаточно сформировать ввод, содержащий команды ОС, которые в итоге будут выполняться с привилегиями уязвимого приложения.

JavaScript/HTML-инъекция

JavaScript/HTML-инъекция, также известная как Cross Site Scripting (XSS), — это атака, при которой вредоносный код встраивается в выдаваемую веб-системой стра-

ницу и исполняется в браузере пользователя. Этот код обычно является JavaScript, HTML или любым другим кодом, который может быть исполнен в браузере. XSS позволяет злоумышленникам перехватывать пользовательские сессии, подменять веб-страницы или перенаправлять пользователей на вредоносные сайты.

3. Обзор методов

Приведём обзор существующих методов по автоматизации построения схемы JSON-объектов и обнаружению вредоносного содержимого в них.

В работах [2 – 4] предлагаются алгоритмы построения схем JSON-объектов. В [2, 3] используется схожий подход: для каждого объекта в коллекции генерируется схема, на основе полученных схем строится древовидная структура, которая хранит информацию о схемах объектов из коллекции. Информация — это атрибуты объектов и их типы, путь от корня объекта к атрибуту, число встреч атрибута, позвляющее делать выводы о его опциональности. В результате получается схема, содержащая в себе все возможные атрибуты, находящиеся в коллекции, с пометкой об их опциональности.

В [4] разработан язык для описания структуры JSON-объектов. Предлагается параметрический и допускающий параллельную работу способ построения схемы. Алгоритм на основе механизма map-reduce применяется к большой коллекции. Он основан на параллельном извлечении схемы для каждого элемента коллекции и объединении этих схем, причём на reduce-этапе происходит объединение только эквивалентных схем. Отношение эквивалентности (ER) является параметром алгоритма, регулирующим баланс между детальностью и компактностью полученной схемы. В работе подробно рассматриваются два вида ER: Kind-equivalence и Label-equivalence:

- Kind-equivalence:
  - простые типы: в случае идентичности сливаются в один, а разные добавляются в объединение;
  - составные типы: если ключи не совпадают, то они добавляются к новой записи и помечаются как опциональные, если же ключи совпадают, то они сливаются в один, а их типы рекурсивно обрабатываются.
- Label-equivalence: отличие этого подхода в том, что слияние двух схем происходит, только если множества ключей обеих схем совпадают, иначе схемы добавляются в объединение.

Вводится формальное описание типов данных и возможных операций над ними. Такой строгий подход позволяет авторам приводить формальные доказательства свойств полученного метода. Таким образом, на этапе map происходит генерация схемы для каждого JSON-объекта из коллекции, а на reduce — слияние полученных типов. Нужно отметить, что приведенные в [4] отношения эквивалентности, по мнению авторов, приемлемы для большинства задач. Ясно, что в случае, когда в коллекции содержатся максимально разнородные объекты, подход Kind-equivalence создаст компактную, человеко-читаемую схему, в отличие от Label-equivalence. Иначе, подход Label-equivalence может дать более точное описание объектов коллекции, так как учитывается совместная встречаемость ключей объектов. Можно заметить, что итоговая схема, получаемая в [2, 3], представляет собой подход Kind-equivalence работы [4].

В [5] описана модель, в основе которой лежит ансамбль случайных деревьев (Random Forest ensemble algorithm). Он агрегирует результаты работы классификаторов, выявляющих аномалии при помощи таких методов, как измерение энтропии Шеннона, анализ n-грамм строкового представления объекта и измерение его длины, измерение схожести структуры JSON-объектов. Утверждается, что для обнаружения

вредоносного контента разработанный метод использует подход выявления аномалий, а не поиск паттернов конкретных атак. Однако для построения пространства признаков для тренировочной и тестовой выборки используются n-граммы векторов атак, а именно для определения таких признаков, как общее число n-грамм векторов атак в объекте и их число в каждом простом типе этого объекта. Таким образом, для анализа некоторого JSON-объекта нужно строить пространство признаков и вычислять данные параметры, основываясь на некотором словаре n-грамм векторов рассматриваемых атак. Кроме того, анализ важности признаков, приведённый в [5], показал, что число n-грамм векторов атак находится на втором месте в рейтинге признаков, оказывающих наибольшее влияние на решение модели.

В данной работе мы ставим задачу разработать модель, которая определяла бы наличие аномального содержимого в объекте без какой-либо дополнительной информации о векторах атак.

4. Моделирование нормы для коллекции JSON-объектов

Приведём подробное описание предлагаемой модели. Она представляет собой совокупность классификаторов, а результат её работы — агрегация результатов работы классификаторов, среди которых:

1) модель схемы;
2) модель строкового представления объектов;
3) модель длины строкового представления объектов;
4) модели простых типов внутри объектов:
   - булевского типа;
   - численного типа;
   - строкового типа.

Каждая из этих моделей возвращает значение `true`, если аномалий нет, `false` — если есть. Результатом работы всей модели является конъюнкция результатов работы перечисленных моделей. Далее подробно опишем каждую из них.

4.1. Модель схемы

В качестве метода построения схемы JSON коллекции выбран Label-equivalence-подход [4]. Постановка задачи предполагает большой набор однородных данных для анализа, поэтому размер схемы даже при этом подходе не будет большим, в то же время учёт совместной встречаемости ключей даст возможность точнее представлять структуру ожидаемых на веб-сервере данных. Далее определим вид схемы JSON-объекта, приведём алгоритм её построения, алгоритм слияния двух схем и введём меру схожести двух схем.

Выработка схемы

Определим множества типов, которые будут использоваться при построении схемы SCHEMA_TYPES = {"string", "number", "boolean", "null", "record", "array", "union"}, и разобьём их на следующие группы:

- простые типы: BASIC_TYPES = {"string", "number", "boolean", "null"};
- составные типы:
  - RECORD_TYPE = "record";
  - ARRAY_TYPE = "array";
- тип, обозначающий объединение схем, которые не удалось слить в одну: UNION_TYPE = "union".

Будем строить схему объекта рекурсивно в процессе обхода этого объекта в глубину. Сначала определяется тип текущего объекта из множества SCHEMA_TYPES, далее в зависимости от типа строится конкретная структура (т. е. схема), описывающая текущий объект. Соответствие типа и его схемы представлено в табл. 1. Схема объекта листинга 2 находится в листинге 4.

Таблица 1

Вид схем различных типов

Тип	Схема	Описание

BASIC_TYPES	`{`
	`  "type": "number"`
	`}`
		В поле "type" записывается текущий тип

RECORD_TYPE	`{`
	`  "type": "record",`
	`  "properties": {`
	`  "prop_1": {...},`
	`  ...`
	`  "prop_k": {...}`
	`  }`
	`}`
		В поле "type" записывается текущий тип, а поле "properties" содержит список ключей в рассматриваемом объекте с соответствующими схемами их значений

ARRAY_TYPE	`{`
	`  "type": "array",`
	`  "items": {...}`
	`}`
		В поле "type" записывается текущий тип, а поле "items" содержит схему значений, содержащихся в массиве

UNION_TYPE	`{`
	`  "type": "union",`
	`  "subtypes": [...]`
	`}`
		В поле "type" записывается текущий тип, а поле "subtypes" содержит список схем, которые не удалось слить в одну

```
{
    "type": "record",
    "properties": {
        "name": {
            "type": "string"
        },
        "age": {
            "type": "number"
        },
        "kids": {
            "type": "array",
            "items": {
                "type": "string"
            }
        }
    }
}
```

Листинг 4. Пример схемы JSON-объекта листинга 2

Задача слияния двух схем в одну возникает, например, когда нужно построить схему значений массива, содержащего элементы разных типов. Согласно методу [4], возможно слияние только эквивалентных схем. При Label-equivalence-подходе эквивалентность схем означает:

1) совпадение значений в поле "type";
2) совпадение множества ключей в поле "properties" при наличии этого поля.

Приведём описание алгоритма слияния двух схем. Если две схемы, поданные на вход алгоритму, являются эквивалентными, то происходит их слияние, а именно:

— простые типы из BASIC_TYPES сливаются в один;
— две схемы типа RECORD_TYPE образуют новую схему типа RECORD_TYPE с одинаковым множеством ключей, значения которых далее рекурсивно обрабатываются алгоритмом слияния;
— две схемы типа ARRAY_TYPE образуют новую схему типа ARRAY_TYPE, где схема значений вырабатывается слиянием схем значений в этих массивах при помощи алгоритма слияния;
— две схемы типа UNION_TYPE дают в результате данного алгоритма схему типа UNION_TYPE, где схемы в составе поля "subtypes" максимально по возможности слиты.

Если же схемы не эквивалентны, то они объединяются в схему типа UNION_TYPE.

Построение схемы JSON-объекта происходит при обходе этого объекта в глубину. Схема для коллекции JSON-объектов может быть построена итеративно, где на каждом шаге строится схема текущего объекта и затем сливается со схемой, выработанной на основе предыдущих объектов коллекции.

Мера сходства схем

Схема, построенная для коллекции JSON-объектов, представляет собой ожидаемую структуру объектов. Необходимо задать меру, определяющую уровень сходства структуры анализируемого объекта с выработанной схемой. В [5] для измерения сходства структур двух JSON-объектов используется коэффициент Жаккара, принимающий значения из отрезка $[0, 1]$.

Определение 1. Коэффициент сходства Жаккара (Jaccard similarity coefficient) — бинарная мера сходства двух множеств A и B, определяемая формулой

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|}.$$

Если $A = \varnothing$ и $B = \varnothing$, то $J(A, B) = 1$.

Чтобы была возможность использовать этот коэффициент, нужно представить схему в виде некоторого множества. В [5] предлагается представлять объект как множество всех путей в нём, где путь — это упорядоченный набор ключей, ведущих к простому типу внутри объекта. Будем использовать тот же подход для представления схемы (которая по определению тоже JSON-объект) в виде множества путей в ней.

Множество путей для некоторой схемы есть набор строк фиксированной структуры: первая часть строки состоит из списка ключей, находящихся по данному пути, в квадратных скобках; затем подстрока `=>`; далее указывается значение, находящееся по данному пути.

Приведём пример измерения сходства двух схем. В листингах 6 и 8 представлены наборы путей, выделенных для схем из листингов 5 и 7 соответственно.

```
{
    "type": "record",
    "properties": {
        "name": {
            "type": "string"
        },
        "age": {
            "type": "number"
        }
    }
}
```

Листинг 5. Схема A

```
[
    "["type"] => "record"",
    "["properties"]["name"]["type"] => "string"",
    "["properties"]["age"]["type"] => "number""
]
```

Листинг 6. Множество путей в схеме A

```
{
    "type": "record",
    "properties": {
        "name": {
            "type": "string"
        },
        "age": {
            "type": "string"
        }
    }
}
```

Листинг 7. Схема B

```
[
    "["type"] => "record"",
    "["properties"]["name"]["type"] => "string"",
    "["properties"]["age"]["type"] => "string""
]
```

Листинг 8. Множество путей в схеме B

Тогда коэффициент сходства схем A и B запишется так:

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{2}{4} = 0{,}5.$$

Однако использование такого подхода в чистом виде было бы некорректным, так как при построении схемы может быть использован тип UNION_TYPE, тогда в соответствующем поле "subtypes" находятся все возможные структуры, которые могут присутствовать в этом конкретном месте схемы одного объекта. Поэтому, чтобы

иметь возможность сравнивать пути в схеме анализируемого объекта и пути в схеме, построенной для коллекции, нужен алгоритм, который из любой схемы, содержащей тип UNION_TYPE, формирует список всех возможных вариантов схем без типа UNION_TYPE внутри и содержащихся в исходной схеме. То есть для формирования списка схем используются всевозможные сочетания подсхем, выбранных из элементов типа UNION_TYPE исходной схемы. Эти подсхемы и будут находиться на том месте, где в схеме коллекции присутствовал тип UNION_TYPE. В листинге 10 представлен результат применения этого алгоритма к схеме из листинга 9.

```
{
    "type": "record",
    "properties": {
        "a": { "type": "string" },
        "b": {
            "type": "union",
            "subtypes": [
                { "type": "string" },
                { "type": "number" }
            ]
        }
    }
}
```

Листинг 9. Схема с типом UNION_TYPE

```
{
    "type": "record",
    "properties": {
        "a": { "type": "string" },
        "b": { "type": "string" }
    }
}

{
    "type": "record",
    "properties": {
        "a": { "type": "string" },
        "b": { "type": "number" }
    }
}
```

Листинг 10. Простые схемы в составе схемы из листинга 9

Измерение сходства схемы объекта со схемой коллекции проводится следующим образом. Для каждой из схем формируется список всех возможных вариантов схем без типа UNION_TYPE внутри. Затем все полученные объекты в этих списках преобразуются в наборы путей по принципу, описанному выше. Теперь можно вычислять схожесть между наборами с использованием коэффициента Жаккара. Если схема объекта не содержит тип UNION_TYPE, то для неё подбирается максимально похожая, т. е. с максимальны коэффициентом Жаккара, схема из списка вариантов для схемы коллекции. Если схема объекта не содержит тип UNION_TYPE, то для каждого варианта схемы объекта подбирается максимально похожий вариант схемы коллекции,

а затем из полученного списка выбирается вариант, дающий наименьший коэффициент Жаккара. Такой подход обусловлен тем, что поставленная задача — это выявить отклонения в схеме объекта от нормы, выраженной схемой коллекции.

Модель схемы определяет соответствие или несоответствие данного JSON-объекта заданной схеме на основе введённой меры и её порогового значения, являющегося гиперпараметром модели. В случае соответствия схеме модель возвращает `true`, иначе — `false`.

4.2. Модель строкового представления объектов

В качестве модели строкового представления объекта используется статистическая модель, которая подстраивается под обучающую выборку из строк и вычисляет вероятность принадлежности новой строки к модели.

Пусть t — конкретная n-грамма, $n \in \{1, 2\}$. Вероятность того, что случайно взятая n-грамма строки имеет значение t, равна

$$\mathsf{P}[y = t] = \theta_t.$$

Воспользуемся байесовским подходом к оценке неизвестного параметра модели θ_t. В качестве априорного распределения θ_t возьмём стандартное равномерное, т. е. $U[0, 1]$, которое является частным случаем бета-распределения:

$$\mathbf{U}[0, 1] \equiv \mathbf{B}(1, 1).$$

С помощью формулы Байеса можно получить апостериорное распределение:

$$\mathsf{P}[\theta_t|Y] = \frac{\mathsf{P}[Y|\theta_t] \cdot \mathsf{P}[\theta_t]}{\mathsf{P}[Y]},$$

где Y — строка, представляющая собой обучающую выборку.

Пусть Y содержит N n-грамм и N_t из них есть t, тогда правдоподобие имеет распределение по Бернулли:

$$\mathsf{P}[Y|\theta_t] = \binom{N}{N_t} \theta_t^{N_t} (1 - \theta_t)^{N - N_t}.$$

Бета-распределение является сопряжённым априорным распределением для распределения Бернулли. Его использование в качестве априорного гарантирует, что апостериорное распределение тоже будет бета-распределением. Это существенно упрощает вычисление апостериорной вероятности простым добавлением количества появления и непоявления конкретной n-граммы t к существующим параметрам бета-распределения, а именно:

$$\mathsf{P}[\theta_t|Y] = \frac{\binom{N}{N_t} \theta_t^{N_t} (1 - \theta_t)^{N - N_t} \dfrac{\theta_t^{1-1} (1 - \theta_t)^{1-1}}{B(1, 1)}}{\displaystyle\int_0^1 \binom{N}{N_t} x^{N_t} (1 - x)^{N - N_t} \frac{x^{1-1} (1 - x)^{1-1}}{B(1, 1)}\, dx} = \frac{\theta_t^{N_t} (1 - \theta_t)^{N - N_t}}{\displaystyle\int_0^1 x^{N_t} (1 - x)^{N - N_t}\, dx} =$$

$$= \frac{\theta_t^{N_t} (1 - \theta_t)^{N - N_t}}{B(N_t + 1, N - N_t + 1)} \sim \mathbf{B}(N_t + 1, N - N_t + 1).$$

Взяв математическое ожидание по апостериорному распредению $p(\theta_t|Y)$, получим байесовскую оценку на параметр θ_t:

$$\int_0^1 \mathsf{P}[\theta_t|Y] \cdot \theta_t \, d\theta_t = \frac{N_t + 1}{N + 2}.$$

Описанный процесс можно итерационно повторять при появлении новых данных для дообучения модели. В таком случае текущее распределение параметра считается априорным и вычисляется новое апостериорное распределение.

Вероятность принадлежности новой строки y к модели можно рассчитать по формуле

$$\mathsf{P}[y|\Theta] = \prod_{t \in n\text{-grams}} \theta_t^{N_t}.$$

Рассчитанная по этой формуле вероятность для коротких строк будет иметь большее значение, чем для длинных, поэтому будем использовать следующую оценку на $\mathsf{P}[y|\Theta]$, которая получается из $\mathsf{P}[y|\Theta]$ взятием корня степени N и последующим логарифмированием:

$$\text{score} = \frac{\sum\limits_{t \in n\text{-grams}} N_t \, \log \theta_t}{N}.$$

Решение о принадлежности строки к модели выносится сравнением вычисленного для неё score с некоторым порогом. Предполагая, что оценки score распределены нормально, воспользуемся правилом трёх сигм, где mean — операция взятия выборочного среднего:

$$\mathsf{P}[\text{mean(scores)} - 3\sigma < \text{score} < \text{mean(scores)} + 3\sigma] \approx 0{,}997.$$

Для стандартного отклонения σ используется следующая оценка:

$$\sigma = \big(\text{mean(scores)} - \min(\text{scores})\big)/2.$$

Положим порог равным $\text{mean(scores)} - 3\sigma$. Для всех строк, для которых обе величины score (для одограмм и биграмм) выше данного порога, модель возвращает `true`, иначе — `false`.

4.3. Модель длины строкового представления объектов

Моделью длины строкового представления объекта является модель численного типа, которая в качестве данных использует последовательность длин строкового представления JSON-объектов из коллекции. Модель численного типа описана в п. 4.4.

4.4. Модели простых типов

Опишем модели типов "boolean", "number" и "string". Они позволяют определять соответствие наблюдаемого значения, находящегося по конкретному пути в объекте, ожидаемым значениям по такому пути. Примеры путей в объекте представлены в листинге 6. Для удобства работы с моделью будем хранить пары («путь», «модель простого типа»), что позволит быстро по пути отыскать соответствующую модель простого типа и осуществить проверку значения.

Модель булевского типа

В процессе обучения осуществляется подсчёт частот встречаемости значений `true` и `false`. Если частота значения, поданного на анализ, больше некторой границы, являющейся гиперпараметром модели, то значение считается нормальным и модель возвращает `true`, в противном случае — аномальным и модель возвращает `false`.

Модель численного типа

В процессе обучения определяется интервал, содержащий все значения из тренировочной выборки. Значение считается аномальным, если не попадает в этот интервал. В случае аномальности модель возвращает `false`, иначе `true`.

Модель строкового типа

В качестве модели строкового типа выступает одна из следующих моделей:

1) модель ограниченного множества значений;
2) модель целых чисел и чисел с плавающей точкой;
3) модель даты и времени;
4) модель номера банковской карты;
5) модель адреса электронной почты;
6) модель URL-адреса;
7) модель имени файла;
8) модель номера телефона;
9) модель идентификаторов (GUID, globally unique identifier);
10) модель строк из шестнадцатеричных цифр;
11) модель коротких строк.

Проверка строкового значения каждой из моделей означает проверку этого значения на соответствие формату строки, который отражён в названии модели. Идея подхода в том, чтобы выбрать более узкую модель из списка, если это возможно. В противном случае в качестве модели строкового типа используется статистическая модель, описанная в п. 4.2.

Имеющийся набор строк делится на две группы: тренировочный (20 %) и валидационный (80 %). В порядке, указанном в списке, значения из тренировочной выборки проверяются на соответствие выбранной модели. У каждой из них определён гиперпараметр — максимально допустимая доля значений, которые не соответствуют модели. Если при проверке доля значений выборки, не подходящих модели, меньше максимально допустимой доли, то модель считается подходящей для выборки. Таким образом, с использованием тренировочной выборки создается список из подходящих моделей. Далее каждая из них проверяется на валидационной выборке. В качестве модели строкового типа выбирается первая из подходящих моделей, прошедшая валидационную проверку.

5. Методика экспериментального исследования

Коллекции JSON-объектов, используемых для тестирования, были собраны из трафика реальных веб-приложений. Информация об используемых для тестирования наборах данных представлена в табл. 2.

Для обучения модели используется 60 % данных, для тестирования — 40 %. Опишем формирование тестовых наборов. Для каждой атаки, рассматриваемой в работе, формируются отдельные тестовые наборы. Процент модифицированных данных составляет от 5 до 95 % тестовой выборки с шагом 5 %. В зависимости от атаки модификация JSON-объекта означает:

1) изменение имени JSON-атрибута: у случайно выбранного атрибута изменяется имя добавлением к нему строки «_CHANGED»;
2) добавление JSON-атрибута: по случайно выбранному пути размещается дополнительный атрибут {"new_key": "new_value"};

Таблица 2

Коллекции JSON-объектов

Имя коллекции	Информация о веб-приложении	Количество объектов
data_0	Онлайн-магазин бытовой техники и электроники	1436
data_1	Онлайн-сервис для покупки авиабилетов	589
data_2	Онлайн-магазин товаров для дома и садоводства	2689
data_3	Онлайн-сервис для покупки туров	1000
data_4	Онлайн-сервис для покупки туров	1000
data_5	Онлайн-сервис для продажи билетов на поезда и самолеты	1431
data_6	Веб-приложение для покупки авиабилетов	7508

3) удаление JSON-атрибута: удаляется случайно выбранный атрибут из списка обязательных; если в коллекции таких нет, то удаляется любой случайно выбранный атрибут;
4) изменение значения JSON-атрибута (внедрение SQL-инъекции, команд ОС, JavaScript/HTML-инъекции): случайно выбирается атрибут и в его значение внедряется вектор атаки, причём если значение не строкового типа, то происходит полное замещение, в случае строки вектор атаки либо полностью замещает собой значение, либо внедряется в случайную позицию этой строки (этот выбор также происходит случайно).

Для случайного выбора используется функция choice из модуля random языка Python3. Векторы атак, используемые для тестирования, взяты из источников [6 – 9]. Их количество представлено в табл. 3.

Таблица 3

Количество векторов атак

Тип вектора атаки	Количество векторов
SQL-инъекция	12842
JavaScript/HTML-инъекции	12968
Инъекция команд ОС	558

Для оценки качества работы модели предлагается использовать коэффициент корреляции Мэтьюса (MCC) [10], который используется для измерения качества бинарных классификаций, особенно если размеры классов сильно различаются. Он определяется следующей формулой:

$$\text{MCC} = \frac{TP \cdot TN - FP \cdot FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}}.$$

В табл. 4 приведена матрица ошибок, где определены классы TN, FP, FN, TP.

Таблица 4

Матрица ошибок

Реальность	Предсказания модели	
	Есть аномалии (`false`)	Нет аномалий (`true`)
Есть аномалии (`false`)	TN	FP
Нет аномалий (`true`)	FN	TP

MCC принимает значение на интервале $[-1, 1]$. Значение 1 соответствует идеальному предсказанию, 0 — ситуации случайного предсказания, -1 — полностью противоположному предсказанию.

В работах [11, 12] показано, что данная мера в случае бинарной классификации является более информативной, чем F1-мера, Accuracy, Balanced accuracy, Bookmaker informedness, Markedness. Кроме того, существует связь между статистикой хи-квадрат и коэффициентом корреляции Мэтьюса для матрицы ошибок размера 2×2 [10]:

$$\chi^2 = N \cdot \mathrm{MCC}^2,$$

где N — объём тестовой выборки. Поэтому для проверки гипотезы H_0 достаточно сравнить MCC, рассчитанный для тестовой выборки, с критическим значением MCC, который соответствует заданному уровню значимости α. Для $\alpha = 0{,}01$ и одной степени свободы, как в нашем случае, $\chi^2 = 6{,}6349$. Значит, критическое значение MCC равно

$$\mathrm{MCC}_{\mathrm{crit}} = \pm\sqrt{\chi^2/N} = \pm\sqrt{66349/N}.$$

Если рассчитанный MCC удовлетворяет неравенству $|\mathrm{MCC}| \leqslant |\mathrm{MCC}_{\mathrm{crit}}|$, то гипотеза H_0 принимается, иначе — отклоняется.

6. Результаты экспериментального исследования

Приведём результаты тестирования предлагаемой модели со следующими гиперпараметрами:

— модель схемы: пороговое значение для меры сходства со схемой 0,99;
— модель булевского типа: пороговое значение для частоты встречаемости булевского значения 0,05;
— модель строкового типа: пороговое значение для доли несоответствующих модели значений 0,03.

При тестировании на имеющихся наборах данных получены значения MCC (рис. 1–7), где по оси абсцисс номерами отмечены следующие модификации:

1) изменение имени JSON-атрибута;
2) добавление JSON-атрибута;
3) удаление JSON-атрибута;
4) изменение значения JSON-атрибута: внедрение SQL-инъекции;
5) изменение значения JSON-атрибута: внедрение инъекции команд ОС;
6) изменение значения JSON-атрибута: внедрение JavaScript/HTML-инъекции.

Из диаграмм видно, что на всех проведённых тестах значение MCC значительно превысило критический MCC, а значит, гипотеза H_0 отклоняется, т. е. решения предлагаемой модели об аномальности объектов являются статистически значимыми.

На рис. 8 приведена диаграмма размаха MCC под названием «Experimental Group Posttest», полученная в [5]; тестирование проводилось с использованием тех же модификаций, что и в данной работе, с дополнительной модификацией — внедрением строк для переполнения буфера в значение JSON-атрибута. Процент модифицированных данных также менялся от 5 до 95 % с шагом 5 %.

Верхний квартиль MCC, полученный при тестировании модели из [5], при сравнении со всеми тестами предлагаемой модели находится ниже нижнего квартиля MCC, полученного нами на тестовых наборах данных, что говорит о лучшем качестве работы предлагаемой модели.

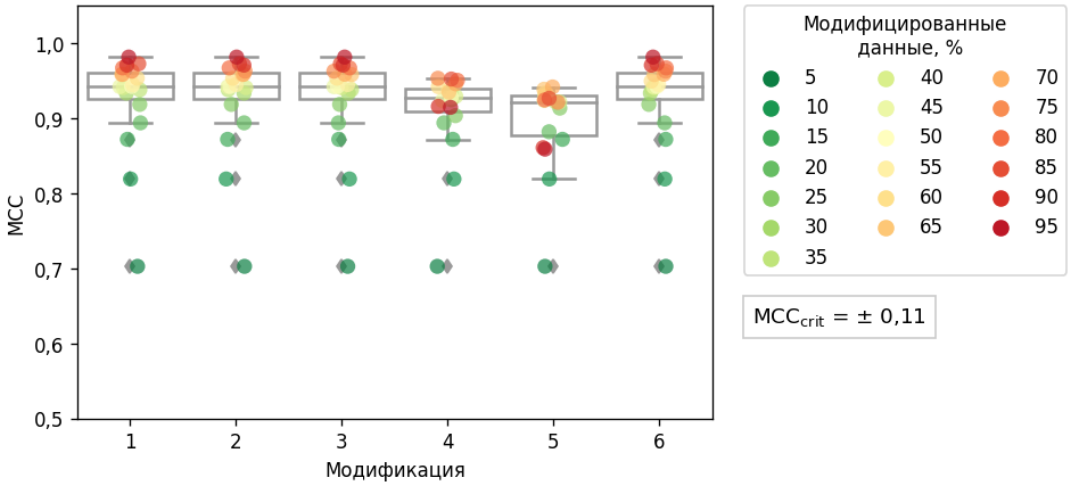

Рис. 1. Диаграмма размаха MCC для data_0

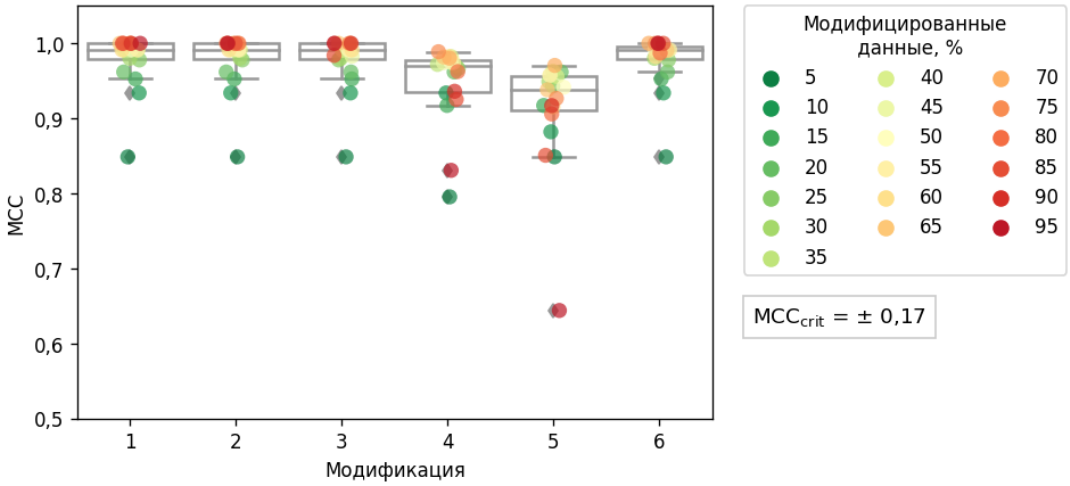

Рис. 2. Диаграмма размаха MCC для data_1

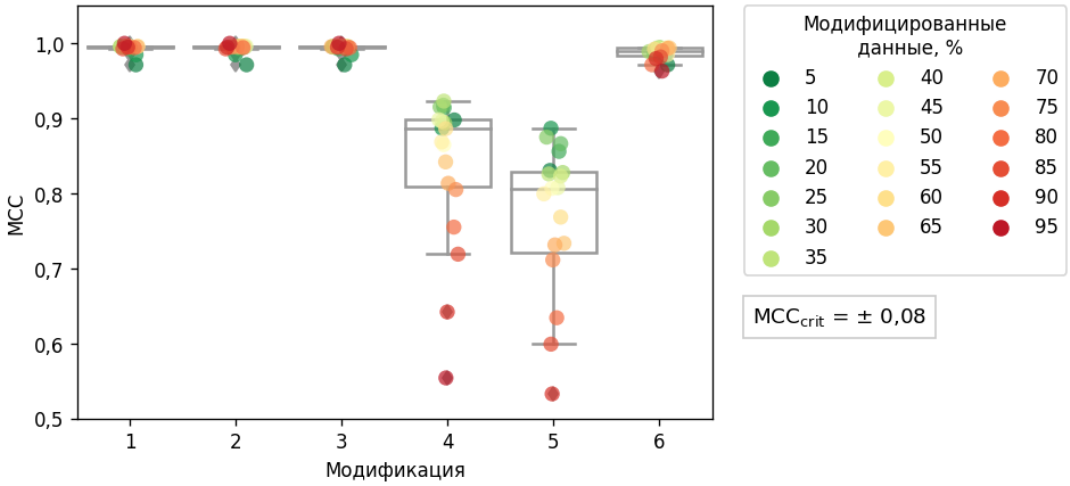

Рис. 3. Диаграмма размаха MCC для data_2

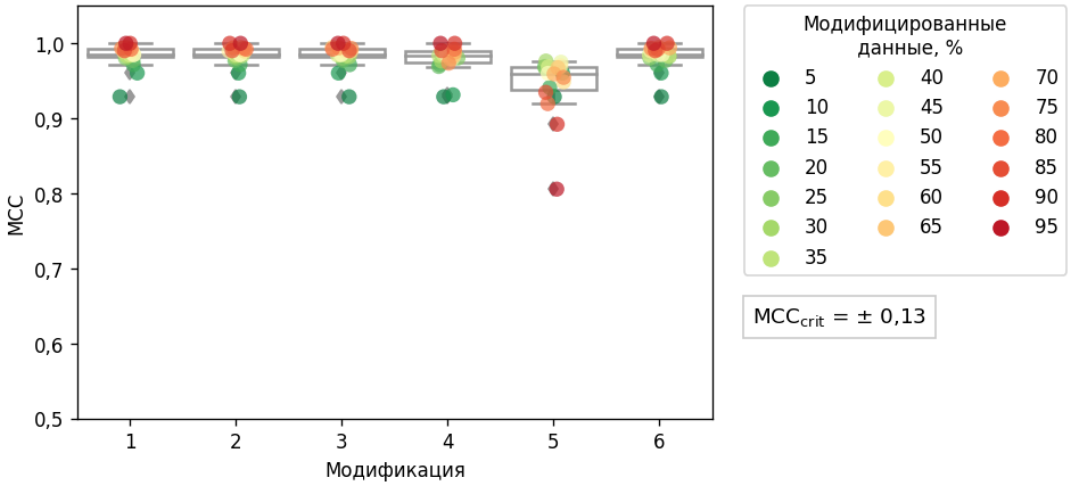

Рис. 4. Диаграмма размаха MCC для data_3

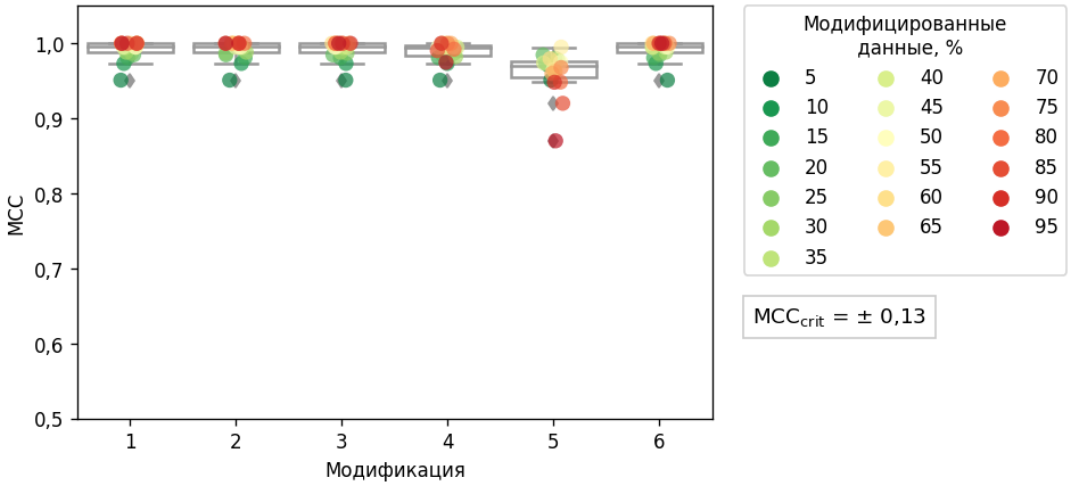

Рис. 5. Диаграмма размаха MCC для data_4

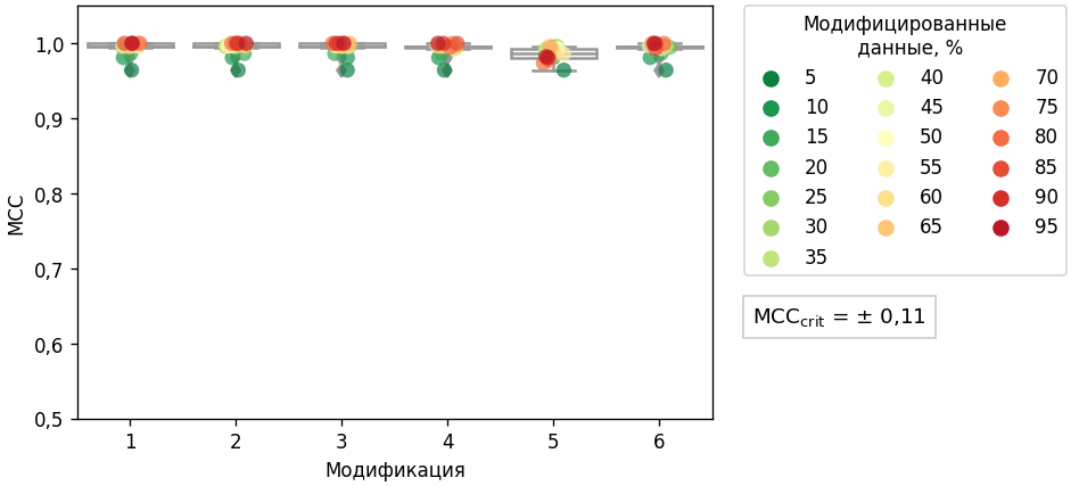

Рис. 6. Диаграмма размаха MCC для data_5

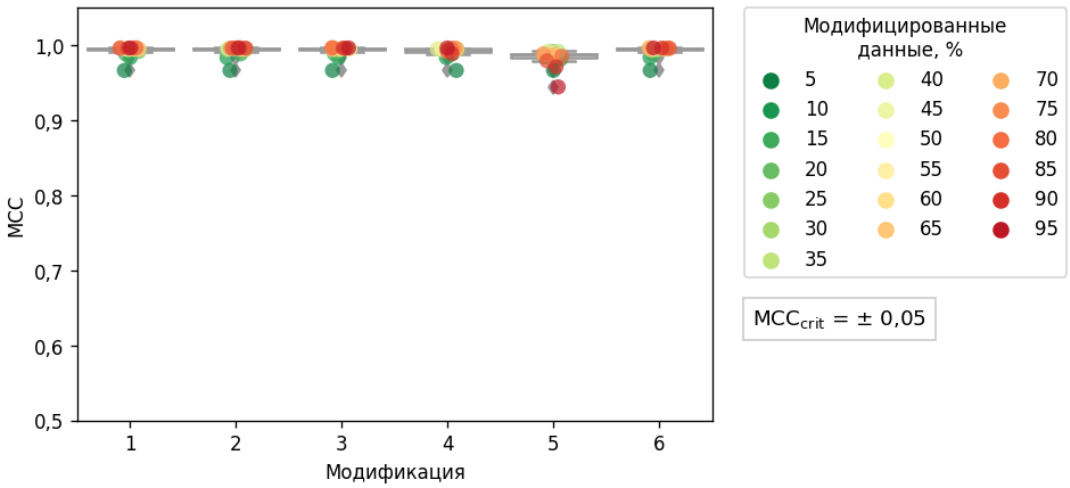

Рис. 7. Диаграмма размаха MCC для data_6

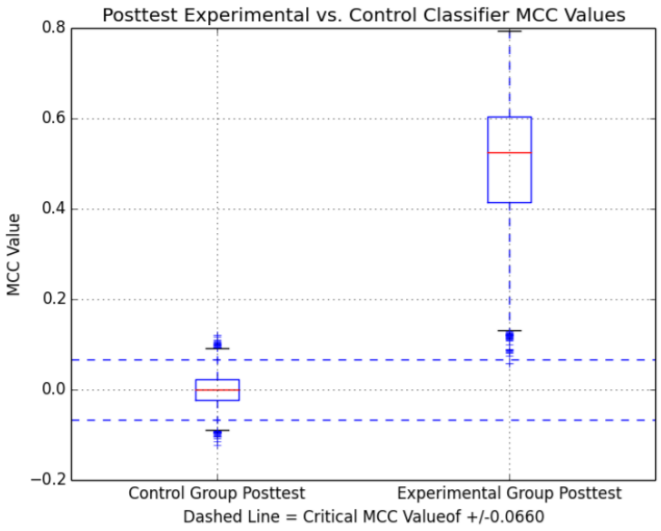

Рис. 8. Диаграмма размаха MCC, полученная в работе [5]

Сравнивая значения MCC при различных модификациях, можно сделать вывод, что для изменения имени JSON-атрибута, добавления JSON-атрибута, удаления JSON-атрибута модель показывает стабильно высокие значения, близкие к единице. Кроме того, с ростом процента модифицированных данных растёт MCC. Результаты модели при внедрении SQL-инъекции и JavaScript/HTML-инъекции также высокие, выбросы наблюдаются при минимальных долях модифицированных данных. Внедрение инъекции команд ОС показало средние значения MCC в сравнении с остальными модификациями, особенностью является то, что выбросы в этом случае относятся к данным с большим процентом модификации. Тем не менее медианы, полученные при всех экспериментах, выше 0,8, что говорит о стабильно хорошем качестве обнаружения аномалий моделью.

В табл. 5 представлено время проверки JSON-объекта из тестовых наборов данных, усреднённое по всем проведённым тестам для данной коллекции.

Таблица 5

Имя коллекции	Время, с
data_0	0,005461
data_1	0,000967
data_2	0,003866
data_3	0,002013
data_4	0,001439
data_5	0,000591
data_6	0,002125

Заключение

В работе рассмотрена проблема выявления вредоносного содержимого в составных объектах в формате JSON. Предложен метод построения модели, подстраивающейся под коллекцию JSON-объектов и позволяющей обнаружить аномальные, нетипичные для этой коллекции данные. Тестирование показало, что решения предлагаемой модели об аномальности объектов являются статистически значимыми. Значения MCC практически на всех тестовых данных близки к единице, что говорит о высокой результативности модели для решения задачи обнаружения структурных аномалий в JSON-объектах или аномалий нарушения типов данных для элементов JSON-объекта. Представленный подход может быть применён не только к формату JSON, но к любому типу данных, который может быть отображен в данный формат, например XML, который также часто используется для обмена информацией в веб-приложениях. Дальнейшие исследования могут быть связаны с разработкой дополнительных строковых моделей и расширением спектра возможных атак.

ЛИТЕРАТУРА

1. `www.json-schema.org`. JSON Schema. 2021.
2. *Frozza A. A., dos Santos Mello R., and da Costa F. S.* An approach for schema extraction of JSON and extended JSON document collections // IEEE Intern. Conf. IRI. 6–9 July 2018. P. 356–363.
3. *Klettke M., Störl U., and Scherzinger S.* Schema extraction and structural outlier detection for JSON-based NoSQL data stores // Conf. BTW, Hamburg, Germany, 4–6 March 2015. P. 425–444.
4. *Baazizi M. A., Colazzo D., Ghelli G., et al.* Parametric schema inference for massive JSON datasets // VLDB J. 2019. V. 28. No. 4. P. 497–521.
5. *Miller B. N.* Detection of Malicious Content in JSON Structured Data using Multiple Concurrent Anomaly Detection Methods. Dissertation. Eastern Michigan University, 2016. 125 p.
6. `www.github.com/payloadbox`. Payload Box. 2021.
7. `www.github.com/fuzzdb-project/fuzzdb`. FuzzDB Project. 2021.
8. `www.kaggle.com/syedsaqlainhussain/sql-injection-dataset`. SQL injection dataset. 2021.
9. `www.kaggle.com/syedsaqlainhussain/cross-site-scripting-xss-dataset-for-deep-learning`. Cross site scripting XSS dataset for Deep learning. 2021.
10. *Baldi P., Brunak S., Chauvin Y., et al.* Assessing the accuracy of prediction algorithms for classification: an overview // Bioinformatics. 2000. V. 16. No. 5. P. 412–424.

11. *Chicco D. and Jurman G.* The advantages of the Matthews correlation coefficient (MCC) over F1 score and accuracy in binary classification evaluation // BMC Genomics. 2020. V. 21. No. 1. P. 1–13.
12. *Chicco D., Totsch N., and Jurman G.* The Matthews correlation coefficient (MCC) is more reliable than balanced accuracy, bookmaker informedness, and markedness in two-class confusion matrix evaluation // BioData Mining. 2021. V. 14. No. 1. P. 1–22.

REFERENCES

1. `www.json-schema.org`. JSON Schema, 2021.
2. *Frozza A. A., dos Santos Mello R., and da Costa F. S.* An approach for schema extraction of JSON and extended JSON document collections. IEEE Intern. Conf. IRI, 6–9 July 2018, pp. 356–363.
3. *Klettke M., Störl U., and Scherzinger S.* Schema extraction and structural outlier detection for JSON-based NoSQL data stores. Conf. BTW, Hamburg, Germany, 4–6 March 2015, pp. 425–444.
4. *Baazizi M. A., Colazzo D., Ghelli G., et al.* Parametric schema inference for massive JSON datasets. VLDB J., 2019, vol. 28, no. 4, pp. 497–521.
5. *Miller B. N.* Detection of Malicious Content in JSON Structured Data using Multiple Concurrent Anomaly Detection Methods. Dissertation. Eastern Michigan University, 2016. 125 p.
6. `www.github.com/payloadbox`. Payload Box. 2021.
7. `www.github.com/fuzzdb-project/fuzzdb`. FuzzDB Project. 2021.
8. `www.kaggle.com/syedsaqlainhussain/sql-injection-dataset`. SQL injection dataset. 2021.
9. `www.kaggle.com/syedsaqlainhussain/cross-site-scripting-xss-dataset-for-deep-learning`. Cross site scripting XSS dataset for Deep learning. 2021.
10. *Baldi P., Brunak S., Chauvin Y., et al.* Assessing the accuracy of prediction algorithms for classification: an overview. Bioinformatics, 2000, vol. 16, no. 5, pp. 412–424.
11. *Chicco D. and Jurman G.* The advantages of the Matthews correlation coefficient (MCC) over F1 score and accuracy in binary classification evaluation. BMC Genomics, 2020, vol. 21, no. 1, pp. 1–13.
12. *Chicco D., Totsch N., and Jurman G.* The Matthews correlation coefficient (MCC) is more reliable than balanced accuracy, bookmaker informedness, and markedness in two-class confusion matrix evaluation. BioData Mining, 2021, vol. 14, no. 1, pp. 1–22.