ПАТТЕРН ДЛЯ ОБЕСПЕЧЕНИЯ БЕЗОПАСНОСТИ ПРИЛОЖЕНИЯ ПРИ УГРОЗЕ МОДИФИКАЦИИ МОДЕЛИ МАШИННОГО ОБУЧЕНИЯ

Корнеев Н. В.[1], Котрини Е. С.[2]

DOI: 10.21681/2311-3456-2025-1-117-127

Цель статьи: *разработка шаблонного механизма защиты для обеспечения безопасности приложения при угрозе модификации модели машинного обучения.*

Метод исследования: *анализ принципов проведения атак с искажением модели ML и возможностей нарушителя на этапе обучения модели. Синтез сценария атаки с помощью двух стратегий атаки: стратегии ввода данных и модификации данных модели ML. Основу модели ML сформировали на базе прогнозной модели Bank Customer Churn Prediction, а для угрозы был выбран внешний нарушитель, который способен изменить выборку данных для модели машинного обучения через сеть путем реализации сценария атаки отравления обучающих данных. С использованием методов криптографии предложен новый механизм защиты, обеспечивающий целостность дата-сета обучающих данных благодаря хешированию и хранению подписанных электронной цифровой подписью хэш-сумм. Исследование выполнено путем натурного моделирования приложения на основе Docker в средах с поддержкой контейнеризации, его развёртывания и тестирования при угрозе модификации модели машинного обучения.*

Результат: *проведен анализ угрозы модификации модели машинного обучения и показана актуальность проблемы разработки универсальных шаблонных механизмов безопасности, называемых паттернами. В частности, рассмотрены три применимые стратегии атаки для модификации модели машинного обучения, основанные на возможностях нарушителя – adversarial example, evasion attack, и модификации данных модели машинного обучения – adversarial example, evasion attack. Рассмотрен сценарий атаки отравления обучающих данных. Построена микросервисная архитектура для обеспечения безопасности приложения при угрозе модификации модели машинного обучения для широкого круга приложений в облачной инфраструктуре. Разработан паттерн безопасности для защиты приложения от атаки отравления обучающих данных на основе микросервисов, интегрированных в контейнеры и стека технологий: Java 17; Spring 5; Docker, docker-compose; PostgreSQL; RabbitMQ; Git; Log4j2; Logstash; Elasticsearch; Kibana; Swagger. В рамках проведённого исследования были разработаны 5 микросервисов: eureka – service, users – api, api – gateway, wrapper – api, config – server. С целью защиты данных, поступающих в модель машинного обучения, разработан микросервис wrapper – api. Механизм защиты заключается в том, что все вызовы к микросервису машинного обучения проходят через него и проверяются на факт порчи и/или подмены, данные, поступающие извне также проходят валидацию на стороне микросервиса. Перед добавлением в БД обучающих данных запись усиливается электронной подписью, происходит хэширование исходных данных, соединённых с секретным ключом. Разработан программный код микросервисов, включая коды специальных методов и алгоритмы их реализации, обеспечивающие механизм защиты приложения от атаки отравления обучающих данных. Развёрнута система мониторинга атаки отравления обучающих данных на базе открытого программного обеспечения Elasticsearch, Logstash, Kibana через объекты журнала событий (appender): ошибку (warn) и информирование (info), которая может быть использована в системах SIEM.*

Практическая ценность: *практическая значимость предлагаемого решения включает шаблонный механизм защиты в виде паттерна, который можно применить для широкого круга приложений, в том числе перенести разрабатываемое решение на любую отрасль: топливно-энергетическую, экономическую и не только, ввиду кроссплатформенности самого решения.*

Ключевые слова: *облачные вычисления, набор данных, шаблон, атака с искажением модели машинного обучения, состязательный пример, атака уклонения, каузативная атака, контейнер, машинное обучение, журнал событий, система мониторинга.*

Введение

Использование новых технологий, связанных с искусственным интеллектом и машинным обучением, активно применяется во многих областях деятельности человека. Новые алгоритмы помогают распознавать знаки дорожного движения, водить беспилотники, фильтровать спам, распознавать лица и т.д. Однако с развитием таких технологий встает вопрос об их безопасности и надежности. По мере роста потребления продуктов и услуг, созданных на основе машинного обучения, стало необходимо предпринимать специальные меры, чтобы защитить не только пользователей и их данные, но и сам искусственный интеллект и алгоритмы от нарушения их работоспособности.

1 Корнеев Николай Владимирович, доктор технических наук, доцент, РГУ нефти и газа (НИУ) имени И. М. Губкина, Москва, Россия. E-mail: niccyper@mail.ru
2 Котрини Елена Сергеевна, студент РГУ нефти и газа (НИУ) имени И. М. Губкина, Москва, Россия. E-mail: kotrini.elena@mail.ru

За последние годы многими исследователями было предложено и реализовано значительное количество работ, касающихся безопасности технологий, связанных с искусственным интеллектом и машинным обучением, например [1–6]. Однако пробелом остается разработка универсальных шаблонных механизмов безопасности, называемыми паттернами. В частности в данной статье мы ставим целью разработку шаблонного механизма защиты для обеспечения безопасности приложения при угрозе модификации модели машинного обучения.

Паттерн безопасности описывает конкретную повторяющуюся проблему безопасности, которая возникает в определенных известных контекстах, а также предлагает хорошо зарекомендовавшую себя общую схему решения такой проблемы безопасности.

Машинное обучение (ML) – это техника обучения информационной системы на основе предоставленных наборов данных (dataset) без использования предопределенных правил, является частным случаем искусственного интеллекта. Общей задачей машинного обучения является построение алгоритма (программы) на основании предоставленных входных данных и заданных верных результатов: таким образом, процесс работы ML-системы разделен на первоначальное обучение на предоставляемых наборах данных и на последующее принятие решений уже обученной системой.

Системы машинного обучения напрямую зависят от данных. Изменение данных, в свою очередь, изменяет работу модели. Однако модели машинного обучения всегда обучаются на некотором подмножестве данных – тренировочном наборе (dataset). А уже затем на этапе эксплуатации модель встречается с реальными данными из генеральной совокупности. И вот эти данные могут по своим характеристикам отличаться от тех, на которых модель обучалась и тренировалась. Изменение данных приводит к модификации работы модели и появления угрозы безопасности информации.

Согласно банку угроз безопасности информации ФСТЭК России, такая угроза безопасности информации называется УБИ 221. Она может быть осуществлена внешним нарушителем (с высоким потенциалом) или внутренним нарушителем (со средним или высоким потенциалом) путем искажения («отравления») обучающих данных. Угроза обусловлена недостатками алгоритмов машинного обучения и осуществления процесса машинного обучения. Реализация угрозы возможна при наличии у нарушителя возможности воздействовать на процесс машинного обучения.

Анализ и методы исследования

Атаки с искажением модели ML (Poisoning Attack) [7] представляют собой целенаправленное злоумышленное изменение данных во время машинного обучения для компрометации всего процесса машинного обучения. Анализ возможностей нарушителя на этапе обучения состоят в следующем. Нарушитель пытается модифицировать модель ML, изменяя набор данных, используемый для обучения. Сценарий атаки включает доступ к частичным или полным данным обучения. На сегодня выделяются две применимые стратегии атаки для модификации модели ML, основанные на возможностях нарушителя – это стратегии ввода данных и модификации данных модели ML.

Стратегия ввода данных реализуется через состязательный пример (adversarial example) [8] или через атаку уклонения (evasion attack) [9]. Она используется, когда нарушитель не имеет никакого доступа к обучающим данным, а также к алгоритму обучения, но имеет возможность добавить новые данные в обучающий набор. Нарушитель может исказить целевую модель ML, вставив ложные выборки в обучающий набор данных. Например, к изображению он может добавить специально подобранный незначительный шум, после чего результат предсказания модели полностью изменится. Это влечет за собой некорректность работы приложения с использованием модели машинного обучения.

Стратегия модификации данных используется, когда нарушитель не имеет доступа к алгоритму обучения, но имеет полный доступ к обучающим данным. Атаки с изменением обучающих данных носят название атак отравления (data poisoning) [10], или каузативных атак (causative attack) [11]. В этом случае нарушитель напрямую искажает обучающие данные, например, путем прямого изменения меток обучающих данных, изменяя их до того, как они будут использованы целевой моделью ML. С помощью подобной атаки он воздействует на рабочую модель в процессе обучения с целью влияния на ее дальнейшее поведение. Благодаря такому подходу нарушитель может опубликовать неверные данные для искажения рабочих процессов в приложении. Стоит также отметить, что данная угроза особенно серьёзна в связи с тем, что многие разработчики ML-решений, включая крупных производителей систем безопасности, используют публичные обучающие выборки, которые могут быть легко «отравлены» третьими сторонами. Соответственно, изменение этих данных повлечет за собой некорректность работы приложений, созданных на основе обученной модели ML.

Для ограничения зоны применимости разрабатываемого шаблонного механизма защиты определим, что нами будет рассматривается приложение

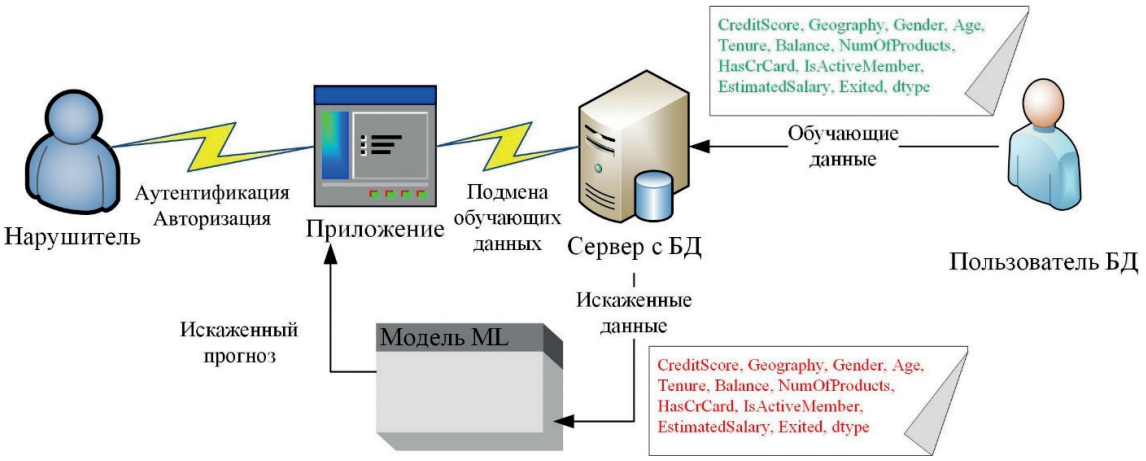

Рис. 1. Сценарий атаки отравления обучающих данных

для компании, оказывающей финансовые услуги, например, банк, который использует модель ML для прогнозирования оттока клиентов. Основу модели ML сформируем на базе прогнозной модели Bank Customer Churn Prediction [12], разработанной сообществом специалистов Kaggle. Выбор модели обусловлен ее широким применением на практике. Данная модель будет анализировать поведение клиентов, исходя из их транзакций, активности, истории обращений и других факторов, чтобы предсказать вероятность того, что клиент может уйти из банка в ближайшем будущем. В качестве входных данных будут использованы следующие переменные с типами данных: CreditScore – int64, Geography – object, Gender – object, Age – int64, Tenure – int64, Balance – float64, NumOfProducts – int64, HasCrCard – int64, IsActiveMember – int64, EstimatedSalary – float64, Exited – int64, dtype – object. На основании этих данных модель машинного обучения должна предсказать, что 20% клиентов уйдут из банка [12].

Идентификация клиентов, которые склонны к оттоку, позволяет банку принимать меры по их удержанию. Раннее обнаружение потенциальных уходящих клиентов позволяет банку предпринимать действия для совершенствования опыта обслуживания, предлагать персонализированные услуги или бонусов для уменьшения оттока и удержания клиентов.

В статье в качестве угрозы был выбран внешний нарушитель, который способен изменить выборку данных для модели машинного обучения через сеть. В качестве внешнего нарушителя будем рассматривать сотрудника банка имеющего доступ к БД обучающих данных (рис. 1). Рассмотрим сценарий атаки отравления обучающих данных, когда нарушитель подменит обучающие данные, например, внедрив неверные или искаженные данные о клиентах, то это приводит к искажению прогнозов модели.

Нарушитель изменяет данные о поведении клиентов, чтобы модель считала, что надежные клиенты склонны к оттоку, или наоборот. В результате банк

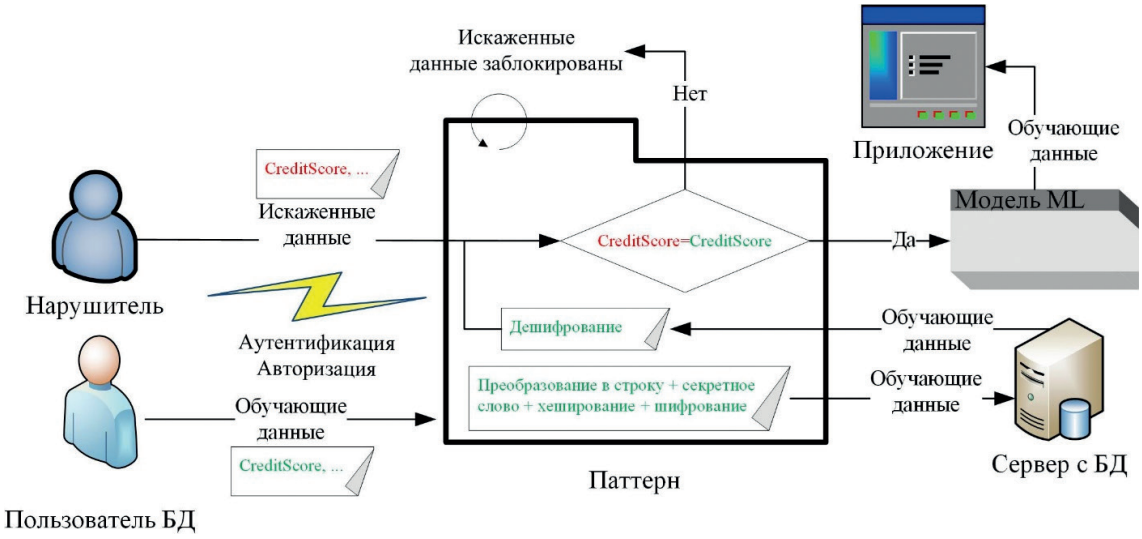

Рис. 2. Сценарий защиты приложения от атаки отравления обучающих данных

принимает неправильные решения на основе искаженных прогнозов, например, направляет усилия на удержание клиентов, которые на самом деле не собираются уходить, или наоборот, упускает возможности предотвратить отток уязвимых клиентов. В свою очередь это приведет к финансовым потерям для банка, ухудшению качества обслуживания клиентов, потере доверия со стороны клиентов и ущербу репутации банка. Поэтому важно обеспечить безопасность данных, использованных для обучения модели машинного обучения, чтобы избежать подобных проблем и сохранить качество работы модели.

Для решения данной проблемы необходимо реализовать паттерн безопасности, включающий в себя механизм защиты приложения (рис. 2), Механизм защиты строится на применении методов криптографии. Так, целостность дата-сета обучающих данных будет обеспечена благодаря хешированию и хранению подписанных электронной цифровой подписью хэш-сумм.

На рис. 2 видно, что обучающие данные проходят операцию хеширования, т.е. преобразования массива входных данных произвольной длины в выходную битовую строку установленной длины. В таком процессе генерации применяется набор методов хеширования с использованием математических формул (хеш-функций). Суть процесса заключается в том, что определенные данные проходят проверку на их соответствие оригиналу, причем сам подлинник в этом действии не участвует. При сравнении информации оценивается идентичность хеш-значений. Важно упомянуть, что при любом изменении данных (например, изменении слова), хеш-значение также изменится, что позволит обнаружить действия нарушителя и сообщить о компрометации целостности. В этом случае искаженные данные будут заблокированы.

Разрабатываемый нами паттерн может быть использован не только для защиты описанного приложения, но и в более широком контексте – для обеспечения безопасности приложений в целом, например, в системе SIEM (Security Information and Event Management).

Новизна предлагаемого решения определяется возможностью обеспечения безопасности приложения при угрозе модификации модели машинного обучения в облачной информационной инфраструктуре России при переходе на импортозамещение.

Практическая значимость предлагаемого решение включает шаблонный механизм защиты в виде паттерна, который можно применить для широкого круга приложений, в том числе перенести разрабатываемое решение на любую отрасль: топливно-энергетическую, экономическую и не только, ввиду кроссплатформенности самого решения.

Паттерн безопасности для приложения

Для реализации паттерна использован следующий стек технологий: Java 17; Spring 5; Docker, docker-compose; PostgreSQL; RabbitMQ; Git; Log4j2; Logstash; Elasticsearch; Kibana; Swagger. Паттерн безопасности (рис. 3) построен на основе микросервисной архитектуры. За основу взят язык Java, основной фреймворк – Spring, в частности, Spring Cloud, необходимый для создания систем, на основе микросервисной архитектуры.

Все входящие HTTP запросы проходят и обрабатываются в шлюзе, который создан на основе Spring Cloud Gateway, который также выступает в роли балансировщика нагрузки (Load balancer) для распределения трафика между несколькими экземплярами одного и того же микросервиса (рис. 3). На уровне шлюза отрабатывает фильтр авторизации (gateway), который пропускает неавторизованные запросы

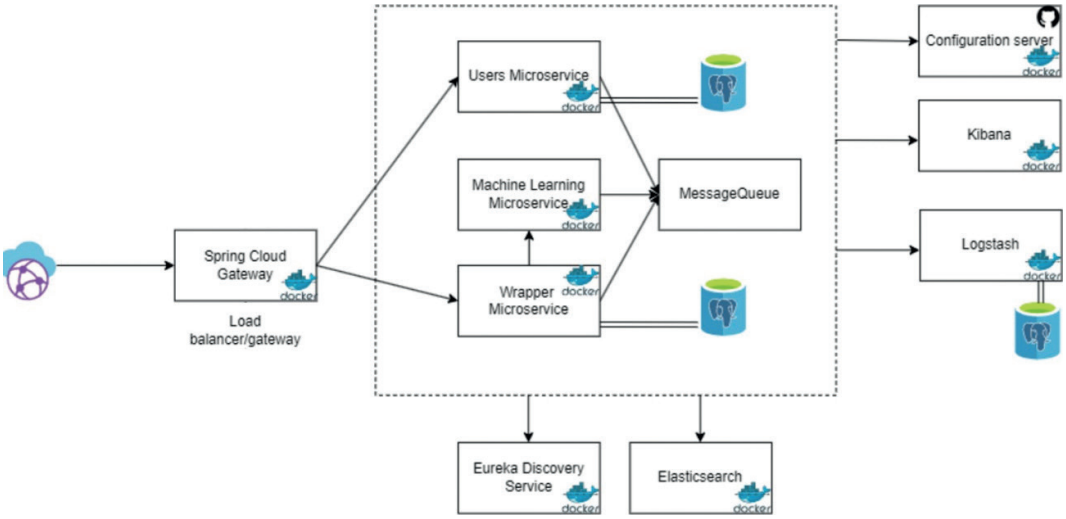

Рис. 3. Микросервисная архитектура паттерна безопасности для приложения

только на те URI, которые необходимы для прохождения аутентификации и авторизации, т.е. работает на JSON Web Token (JWT) аутентификации [13].

За аутентификацию и авторизацию пользователей отвечает пользовательский микросервис (Users Mircoservice). Работа последнего основана на части Spring framework – Spring Security, работающего в режиме Stateless аутентификации, т.е. без хранения сессии пользователя и без назначения cookies. Выбор сделан в пользу JWT, так как для работы в Statefull режиме сессия должна храниться на каждом сервисе, и все они должны обновляться синхронно, что затратно в плане памяти и быстродействия. Данный сервис хранит информацию о пользователях в базе данных PostgreSQL, конфиденциальные данные пользователя шифруются при помощи класса шифрования, интегрированного в Spring Security – BCryptPasswordEncouder.

Микросервис машинного обучения (Machine Learning Mircoservice) отвечает за генерацию предсказаний о заинтересованности того или иного клиента в дальнейшем использовании услуг банка. Модель ML предварительно обучается на заготовленном наборе данных.

С целью защиты данных, поступающих в модель ML, разработан отдельный микросервис на основе паттерна проектирования Wrapper. Механизм защиты заключается в том, что все вызовы к микросервису машинного обучения проходят через него. За счёт этого появляется возможность реализации предварительных проверок поступающих данных на факт порчи и/или подмены. Кроме того, данные, поступающие извне, также проходят валидацию на стороне микросервиса. Вместе с этим перед добавлением данных в БД обучающих данных запись усиливается электронной подписью, то есть, происходит хэширование исходных данных, соединённых с секретным ключом. За счёт добавления данного микросервиса появляется прослойка между данными и моделью ML, способствующая защите данных, а также контролю над ними.

За отправку асинхронных сообщений отвечает микросервис MessageQueue, в паттерне такая функциональность необходима только во время обновления конфигурации, то есть при изменении файлов конфигурации на сервере конфигурации в очередь отправляется сообщение о необходимости повторного чтения файлов свойств; сервисы, подписанные на данную очередь, получают данное сообщение и выполняют обновление. Это позволяет выполнять обновление свойств приложения в горячем режиме, то есть без необходимости останавливать работу кластера, на котором развернуты микросервисы. В качестве брокера сообщений выбран RabbitMQ [14], так как в условиях рассматриваемого сценария атаки нет необходимости создавать шину данных, а достаточно простой пересылки асинхронных сообщений.

Вместе с сервисом отправки асинхронных сообщений стоит упомянуть сервис, отвечающий за конфигурацию всей системы (паттерн и приложение). С целью облегчения конфигурирования принято решение вынести все файлы свойств каждого микросервиса, а также общие настройки в приватный Git репозиторий. Для работы с ним используется микросервис ConfigurationServer, который основан на Spring Cloud Config Server. За счёт последнего осуществляется считывание данных из репозитория и их чтения микросервисами. Также именно эта структура отвечает за обновление конфигурации в горячем режиме.

Для агрегации логирования в едином месте и наблюдения за состоянием системы будет использован ряд микросервисов, основанных на стеке ELK – Elasticsearch, Logstash, Kibana [15]. Каждый микросервис в системе генерирует свой файл логов. Для удобства работы необходимо агрегировать все файлы журнала в одном месте, для этого используется Logstash. Он собирает файлы журнала, сортирует их, фильтрует и отправляет выше по цепочке в Elasticsearch, который, в свою очередь, индексирует файлы журнала и сохраняет их. Kibana позволяет визуализировать полученные данные. Чтобы собирать данные с каждого микросервиса, используется библиотека log4j2, предоставляющая функционал для журналирования. Все указанное в совокупности дает возможность реализовать мониторинг атаки отравления обучающих данных.

Ещё один микросервис, необходимый для функционирования всей системы – Eureka Discovery Service. Данный микросервис был разработан нами для регистрации всех остальных микросервисов и обеспечения связи между ними, а также он даёт возможность контролировать состояние каждого микросервиса.

Таким образом, согласно сценарию защиты (рис. 2) система развёрнута в облаке, на распределяющем сервере установлена защита от вредоносного трафика (Firewall), таких как DDoS, DoS, ReDoS, SSI. Группа пользователей, включая нарушителя, взаимодействуют с системой при помощи сетевых устройств связи, те, в свою очередь, отправляют HTTP запрос, который приходит на локальный маршрутизатор, где, в свою очередь, происходит преобразование доменного имени по протоколу DNS в IP-адрес распределяющего сервера. После прохождения запроса через защиту последний попадает на сетевой балансировщик нагрузки (в рассматриваемом примере – это

один и тот же микросервис), который используется для маршрутизации TCP-трафика (трафика четвертого уровня OSI). Они работают на транспортном уровне и способны принимать решения о маршрутизации на основе IP-адреса и номера порта запроса. Последние зачастую используются в облачных средах.

После прохождения через балансировщик нагрузки запрос попадает на наименее загруженную ноду кластера. Физически все ноды представлены кластером серверов, а количество вычислительных машин на базе одного кластера в каждом конкретном случае определяется нагрузкой [16]. Все ноды повторяют друг друга и внутри представлены сущностью микросервиса, каждый из которых работает под управлением docker-compose. Он, в свою очередь, организует взаимодействие между микросервисами, отвечает за их дублирование и непрерывную работу. Все микросервисы представлены Docker контейнерами, которые обеспечивают изолированную работу сервисов друг от друга, также на базе каждой ноды работает Logstash, Kibana, Elasticsearch, необходимые для отслеживания работы приложения.

К каждой машине можно получить доступ при помощи ssh тунеля, который позволяет получить доступ к удалённой машине с любого устройства, на котором есть необходимые ssh ключи. Все ноды работают с базами данных, которые представлены отдельным кластером, в данном случае шардирование базы данных не используется по причине отсутствия значительной нагрузки на узел данных.

Разработка сервисов паттерна безопасности

В рамках проведённого исследования были разработаны 5 микросервисов: eureka – service, users – api, api – gateway, wrapper – api, config – server. В дальнейшем мы более подробно рассмотрим только один из них – wrapper – api, так как именно он обеспечивает основной механизм защиты данных, поступающих в модель ML от атаки отравления обучающих данных.

Микросервис eureka – service является основным. Он отвечает за регистрацию всех остальных сервисов и обеспечивает связь между ними, а также даёт возможность контролировать состояние каждого микросервиса. Остальные сервисы – вспомогательные. Без них работа первого невозможна.

Микросервис config – service предоставляет конфигурацию для всех остальных микросервисов. Все конфигурации лежат на удаленном git – репозитории.

Микросервис api – gateway – это шлюз, через который проходят все запросы в систему. Он откидывает все неавторизованные запросы, то есть те, у которых токен либо неверный, либо его нет, и пропускает запросы только на сервис, который авторизует пользователей. Как только пользователь получает этот токен, он может проходить дальше. Код этого сервиса содержит только описание фильтра JwtAuthorizationFilter. Этот фильтр отвечает за авторизацию пользователей, используя JWT. Он обеспечивает безопасность системы, отклоняя неавторизованные запросы и проверяя валидность JWT токенов перед передачей запросов дальше по цепочке обработки.

Микросервис user – api выполняет функцию управления пользователями в системе, обеспечивая регистрацию пользователей, аутентификацию и авторизацию, а также предоставляя доступ к защищенным ресурсам с помощью JWT токенов.

Микросервис wrapper – api включает в себя:

1. класс DataRouterController – это контроллер, который отвечает за обработку HTTP запросов;
2. класс PersonDTO – это DTO (Data Transfer Object), который представляет данные пользователя. Он содержит поля, соответствующие определенным нами ранее входным данным: кредитный рейтинг (CreditScore), страна проживания (Geography), пол (Gender), возраст (Age) и т.д.;
3. класс PersonDTOBuilder – это строитель для объекта PersonDTO. Он предоставляет методы для установки различных полей объекта PersonDTO поэтапно;
4. класс EncryptedPerson представляет модель данных, которая описывает сущность «зашифрованный пользователь» в контексте базы данных. Модель содержит поля, которые соответствуют атрибутам зашифрованного пользователя, таким как CreditScore, Geography, Gender, Age и т.д. Модель используется для работы с данными в базе данных, выполнения операций CRUD (создание, чтение, обновление, удаление) и отображения данных в приложении;
5. класс EncryptedPersonBuilder – это строитель для объекта EncryptedPerson;
6. классы ошибок (Exeptions), которые могут быть в процессе выполнения программы;
7. класс EncryptedPeopleRepository – это репозиторий, который используется для взаимодействия с базой данных и выполнения операций CRUD сущностей типа EncryptedPerson;
8. классы ответов Responses: IntegrityViolationOfDataResponse – класс, который представляет ответ на ситуацию, когда происходит нарушение целостности данных; NoSuchPersonResponse – класс представляет ответ на ситуацию, когда запрашиваемый пользователь не найден; PersonNotValidResponse – класс представляет ответ на ситуацию, когда данные пользователя не прошли валидацию; PersonSaveProcessingResponse –класс представляет ответ на ситуацию, когда возникает ошибка при обработке сохранения данных пользователя;

9. PeopleService – это класс контроллера принятия запроса на добавление пользователя, и его обработку;

Класс PeopleService включает в себя два основных метода: addPerson(PersonDTO person) и getPerson(long id).

Код метода addPerson(PersonDTO person):

```
public void addPerson(PersonDTO person) {
   EncryptedPerson personToAdd;
   try {
      personToAdd = messageCoder.
      code(person);
      encryptedPeopleRepository.
      save(personToAdd);
   } catch (RuntimeException e) {
      logger.error(e.getCause().
      getMessage());
      throw e;
   }
}
```

Метод addPerson(PersonDTO person) принимает объект PersonDTO, что представляет данные о пользователе, которого необходимо добавить. Сначала данные пользователя кодируются с помощью объекта messageCoder, чтобы зашифровать информацию о пользователе. Затем зашифрованные данные сохраняются в базе данных с помощью метода save() репозитория encryptedPeopleRepository. В случае возникновения исключения при сохранении данных ошибка логируется и исключение пробрасывается дальше.

Код метода getPerson(long id):

```
public PersonDTO getPerson(long id) {
   EncryptedPerson person =
   encryptedPeopleRepository.findById(id).
   orElseThrow(
      () -> new NoSuchPersonException
      (String.format("Person with id %d not
      found", id))
   );
   Optional<PersonDTO> personToReturn;
   try {
      personToReturn = messageDecoder.
      decode(person);
   } catch (RuntimeException e) {
      throw new IntegrityViolationOfData
      Exception(String.format("Person with
      id %d has data violation", id));
   }
        return personToReturn.get();
}
```

Метод getPerson(long id) принимает уникальный идентификатор пользователя (id) и возвращает информацию о пользователе. Сначала происходит поиск зашифрованных данных пользователя в базе данных по id с помощью метода findById() репозитория encryptedPeopleRepository. Если пользователь не найден, выбрасывается исключение NoSuchPersonException. Затем зашифрованные данные декодируются с помощью объекта messageDecoder, чтобы получить исходные данные пользователя. В случае возникновения ошибки при декодировании данных выбрасывается исключение IntegrityViolationOfDataException и информация о пользователе возвращается в виде объекта PersonDTO.

10. класс MessageCoder отвечает за шифрование данных объекта PersonDTO в объект EncryptedPerson, чтобы сохранить его в базе данных;

Основной метод класса MessageCoder – метод code(PersonDTO dto), который принимает объект PersonDTO и возвращает зашифрованный объект EncryptedPerson. Код метода code(PersonDTO dto):

```
public EncryptedPerson code(PersonDTO dto) {
   KeyStore keyStore; //Private key
   loading PrivateKey privateKey;
   try {
      keyStore= KeyStore.getInstance
      ("PKCS12");
      char[] passwordToKeyFile =
      environment.getProperty("encryption.
      passwordToPrivateKeyFile").
      toCharArray();
   keyStore.load(new FileInputStream("../
   wrapper-api/src/main/resources/sender_
   keystore.p12"), passwordToKeyFile);
      privateKey =
         (PrivateKey) keyStore.getKey
         ("senderKeyPair", passwordToKey
         File);
   } catch (IOException | NoSuchAlgorithm
      Exception | CertificateException |
      KeyStoreException |
         UnrecoverableKeyException e) {
      throw new RuntimeException(e);}
```

В методе реализуется следующий алгоритм:

1. загружается закрытый ключ (privateKey) из хранилища ключей (KeyStore);
2. объект PersonDTO преобразуется в строку (string ToEncrypt) с помощью метода toString();
3. Генерируется хеш сообщения (messageHash) из строки stringToEncrypt с использованием алгоритма SHA-256;
4. хеш сообщения шифруется с помощью закрытого ключа (privateKey) и алгоритма RSA с использованием объекта Cipher;
5. полученная цифровая подпись (digitalSignature) преобразуется в строку и сохраняется в поле signature объекта EncryptedPerson;
6. создается объект EncryptedPerson с зашифрованными данными и возвращается в качестве

результата метода, а впоследствии этот объект будет добавлен в БД.

11. класс MessageDecoder отвечает за декодирование зашифрованных данных объекта EncryptedPerson, которые хранятся в базе данных, а также за проверку подлинности цифровой подписи.

Основной метод класса MessageDecoder – метод decode(EncryptedPerson encryptedPerson), который получает зашифрованный объект EncryptedPerson из базы данных и возвращает объект PersonDTO после декодирования.

В методе реализуется следующий алгоритм:

1. загружается открытый ключ (publicKey) из хранилища ключей (KeyStore);
2. зашифрованная цифровая подпись (encryptedMessageHash) декодируется из объекта EncryptedPerson. Таким образом, она декодируется из формата Base64 в виде строки;
3. расшифровывается подпись с использованием открытого ключа (publicKey) и алгоритма RSA с помощью объекта Cipher. Получается дешифрованный хеш (decryptedMessageHash);
4. вычисляется хеш (messageHash) строки объекта EncryptedPerson;
5. сравнивается дешифрованный хеш с вычисленным хешем для проверки подлинности данных;
6. если подпись корректна, создается объект PersonDTO на основе данных объекта EncryptedPerson с помощью PersonDTOBuilder;
7. возвращается объект PersonDTO, если подпись верна, иначе логируется ошибка валидации подписи.

Мониторинг атаки отравления обучающих данных и обсуждение результатов

Была разработана система мониторинга атаки отравления обучающих данных, основанная на стеке ELK [17]. Рассмотрим логику работы этой системы: все микросервисы пишут свои логи либо через стандартный логгер от spring – logback, либо через log4j2. Данные логгеры конфигурируются следующим образом: добавляется один объект журнала событий (appender), который отправляет логи на host logstash, на порт 5000, один appender – стандартный, в консоль, порог логов для первого appender выставляется на ошибку (warn), для второго – информирование (info).

Для отображения этой информации необходимо настроить logstash, в нём необходимо указать, что логи будут приходить на порт 5000, а отправляются на хост elasticsearch, порт 9200, в формате json. Также нужно явно указать, что logstash будет слушать подключения от всех хостов, не только от локального.

На этом этапе важно убедиться, что в docker – compose данные микросервисы находятся в одной сети, иначе логи не будут доходить до logstash, а затем до elasticsearch. В настройках kibana нужно указать, что необходимо слушать подключения на порт 5601, а также явно указать, где находится elasticsearch, хост и порт, соответственно. Таким образом, мы получаем рабочую систему агрегации логов. Для просмотра графиков необходимо перейти на хост, на котором запущен кластер ELK, на порт 5601. Таким образом будет получен доступ к графическому интерфейсу kibana. Далее необходимо настроить data view, представления данных, перейдя на вкладку index management мы увидим, что основные микросервисы начали писать свои логи (рис. 4). На рис. 4 показан журнал и графики логов для микросервиса users – api.

Анализ журнала и графиков логов в режиме реального времени позволяет видеть попытки атак отравления обучающих данных через информирование (info) и ошибки (warn). Метрики этих ошибок могут быть настроены более гибко через соответствующие события, например, для метода addPerson(PersonDTO person) в случае возникновения исключения при сохранении данных; для метода getPerson(long id) в случае возникновения ошибки при декодировании данных на основе исключения IntegrityViolationOfDataException; для метода decode(EncryptedPerson encryptedPerson) в случае когда логируется ошибка валидации подписи.

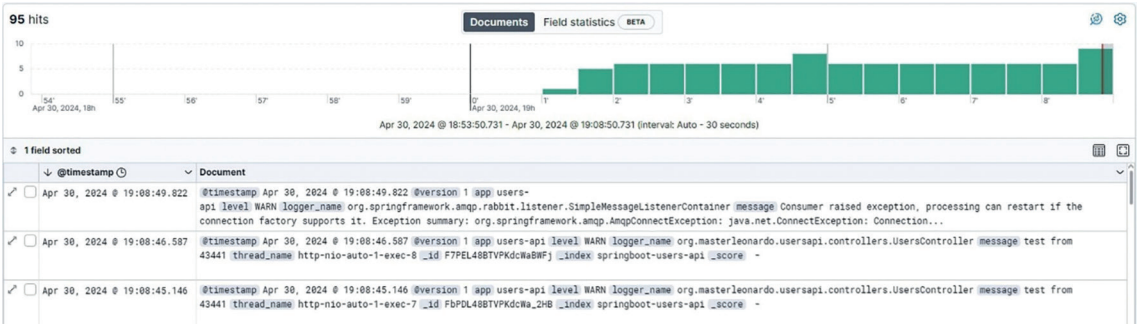

Рис. 4. Витрина с результатами мониторинга микросервиса users – api

Выводы

Построена микросервисная архитектура для обеспечения безопасности приложения при угрозе модификации модели машинного обучения для широкого круга приложений в облачной инфраструктуре. Рассмотрен сценарий атаки отравления обучающих данных. Разработан паттерн безопасности для защиты приложения от атаки отравления обучающих данных на основе микросервисов, интегрированных в контейнеры. В рамках проведённого исследования были разработаны 5 микросервисов: eureka – service, users – api, api – gateway, wrapper – api, config – server. С целью защиты данных, поступающих в модель ML, разработан микросервис wrapper – api. Механизм защиты заключается в том, что все вызовы к микросервису машинного обучения проходят через него и проверяются на факт порчи и/или подмены, данные, поступающие извне, также проходят валидацию на стороне микросервиса. Перед добавлением данных в БД обучающих данных запись усиливается электронной подписью, происходит хэширование исходных данных, соединённых с секретным ключом. Разработан программный код микросервисов, включая коды специальных методов и алгоритмы их реализации, обеспечивающих механизм защиты приложения от атаки отравления обучающих данных. Развёрнута система мониторинга атаки отравления обучающих данных на базе открытого программного обеспечения через объекты журнала событий (appender): ошибку (warn) и информирование (info), которые могут быть использованы в системах SIEM. Метрики этих ошибок могут быть гибко настроены через соответствующие события специальных методов, что дает возможность гибко настраивать сам паттерн и применять его для широкого круга приложений.

Литература

1. *Shameer Mohammed, S. Nanthini, N. Bala Krishna, Inumarthi V. Srinivas, Manikandan Rajagopal, M. Ashok Kumar, A new lightweight data security system for data security in the cloud computing, Measurement: Sensors, Volume 29, 2023, 100856. DOI: 10.1016/j.measen.2023.100856.*
2. *S. Achar, Cloud computing security for multi-cloud service providers: controls and techniques in our modern threat landscape, International Journal of Computer and Systems Engineering, 16(9), 2022, 379–384. DOI: 10.5281/zenodo.7084251.*
3. *Oludare Isaac Abiodun, Moatsum Alawida, Abiodun Esther Omolara, Abdulatif Alabdulatif, Data provenance for cloud forensic investigations, security, challenges, solutions and future perspectives: A survey, Journal of King Saud University – Computer and Information Sciences, Volume 34, Issue 10, Part B, 2022, 10217–10245. DOI: 10.1016/j.jksuci.2022.10.018.*
4. *Chakraborti, A., Curtmola, R., Katz, J., Nieh, J., Sadeghi, A. R., Sion, R., Zhang, Y., Cloud Computing Security: Foundations and Research Directions. Foundations and Trends in Privacy and Security, 3(2), 2022, 103–213. DOI: 10.1561/3300000028.*
5. *Ukeje, N., Gutierrez, J., Petrova, K., Information security and privacy challenges of cloud computing for government adoption: a systematic review, International Journal of Information Security, Volume 23, 2024, 1459–1475. DOI: 10.21203/rs.3.rs-3351319/v1.*
6. *Fatemeh Khoda Parast, Chandni Sindhav, Seema Nikam, Hadiseh Izadi Yekta, Kenneth B. Kent, Saqib Hakak, Cloud computing security: A survey of service-based models, Computers & Security,Volume 114, 2022, 102580. DOI: 10.1016/j.cose.2021.102580.*
7. *Ting Zhou, Hanshu Yan, Bo Han, Lei Liu, Jingfeng Zhang, Learning a robust foundation model against clean-label data poisoning attacks at downstream tasks, Neural Networks, Volume 169, 2024, 756–763. DOI: 10.1016/j.neunet.2023.10.034.*
8. *Ade Kurniawan, Yuichi Ohsita, Masayuki Murata, Detection of sensors used for adversarial examples against machine learning models, Results in Engineering, Volume 24, 2024, 103021. DOI: 10.1016/j.rineng.2024.103021.*
9. *Hamid Bostani, Veelasha Moonsamy, EvadeDroid: A practical evasion attack on machine learning for black-box Android malware detection, Computers & Security, Volume 139, 2024, 103676. DOI: 10.1016/j.cose.2023.103676.*
10. *Mahdee Jodayree, Wenbo He, Dr. Ryszard Janicki, Preventing Image Data Poisoning Attacks in Federated Machine Learning by an Encrypted Verification Key, Procedia Computer Science, Volume 225, 2023, 2723–2732. DOI: 10.1016/j.procs.2023.10.264.*
11. *Michael Gallagher, Nikolaos Pitropakis, Christos Chrysoulas, Pavlos Papadopoulos, Alexios Mylonas, Sokratis Katsikas, Investigating machine learning attacks on financial time series models, Computers & Security, Volume 123, 2022, 102933. DOI: 10.1016/j.cose.2022.102933.*
12. *Pahul Preet Singh, Fahim Islam Anik, Rahul Senapati, Arnav Sinha, Nazmus Sakib, Eklas Hossain, Investigating customer churn in banking: a machine learning approach and visualization app for data science and management, Data Science and Management, Volume 7, Issue 1, 2024, 7–16. DOI: 10.1016/j.dsm.2023.09.002.*
13. *Badr Eddine Sabir, Mohamed Youssfi, Omar Bouattane, Hakim Allali, Authentication and load balancing scheme based on JSON Token For Multi-Agent Systems, Procedia Computer Science, Volume 148, 2019, 562–570. DOI: 10.1016/j.procs.2019.01.029.*
14. *Esquembre F., Chacón J., Saenz J., Vega J., Dormido-Canto S., A programmable web platform for distributed access, analysis, and visualization of data, Fusion Engineering and Design, Volume 197, 2023, 114049. DOI: 10.1016/j.fusengdes.2023.114049.*
15. *Dongyeop Lee, Daesik Lim, Jongseok Park, Soojeong Woo, Youngho Moon, Aesol Jung, Management Architecture With Multimodal Ensemble AI Models for Worker Safety, Safety and Health at Work, Volume 15, Issue 3, 2024, 373–378. DOI: 10.1016/j.shaw.2024.04.008.*
16. *Miguel Correia, Wellington Oliveira, José Cecílio, Monintainer: An orchestration-independent extensible container-based monitoring solution for large clusters, Journal of Systems Architecture, Volume 145, 2023, 103035. DOI: 10.1016/j.sysarc.2023.103035.*
17. *Adabi Raihan Muhammad, Parman Sukarno, Aulia Arif Wardana, Integrated Security Information and Event Management (SIEM) with Intrusion Detection System (IDS) for Live Analysis based on Machine Learning, Procedia Computer Science, Volume 217, 2023, 1406–1415. DOI: 10.1016/j.procs.2022.12.339.*

PATTERN FOR SECURING APPLICATIONS UNDER THREAT OF MODIFICATION MACHINE LEARNING MODEL

Korneev N. V.[3], *Kotrini E. S.*[4]

Keywords: *cloud computing, dataset, template, poisoning attack, adversarial example, evasion attack, causative attack, container, machine learning, event log, monitoring system.*

The purpose of this article: *development of a template protection mechanism to ensure application security in the event of a threat of modification of a machine learning model.*

Research method: *analysis of the principles of attacks with ML model distortion and the capabilities of the intruder at the model training stage. Synthesis of an attack scenario using two attack strategies: data input strategy and ML model data modification strategy. The ML model was based on the Bank Customer Churn Prediction model, and an external intruder was selected for the threat, which is capable of changing the data sample for the machine learning model via the network by implementing a training data poisoning attack scenario. Using cryptographic methods, a new protection mechanism is proposed that ensures the integrity of the training data set due to hashing and storing hash sums signed with an electronic digital signature. The study was carried out by natural modeling of a Docker-based application in environments with containerization support, its deployment and testing in the event of a threat of modification of the machine learning model.*

Result: *the analysis threat of modification machine learning model and shows the relevance of the problem developing universal template security mechanisms, called patterns. In particular, three applicable attack strategies for modifying a machine learning model based on the capabilities of the intruder are considered – adversarial example, evasion attack and modification of machine learning model data – adversarial example, evasion attack. Scenario of poisoning attack on training data is considered. The article analyzes the threat of modification of machine learning model and shows the relevance of the problem of developing universal template security mechanisms, called patterns. In particular, three applicable attack strategies for modifying a machine learning model based on the capabilities of the intruder are considered – adversarial example, evasion attack and modification of machine learning model data – adversarial example, evasion attack. A scenario of an attack on training data poisoning is considered. A microservice architecture is built to ensure application security under the threat of modification of a machine learning model for a wide range of applications in the cloud infrastructure. A security pattern is developed to protect the application from an attack on poisoning training data based on microservices integrated into containers and a stack of technologies: Java 17; Spring 5; Docker, docker-compose; PostgreSQL; RabbitMQ; Git; Log4j2; Logstash; Elasticsearch; Kibana; Swagger. As part of the study, 5 microservices were developed: eureka – service, users – api, api – gateway, wrapper – api, config – server. In order to protect data coming into the machine learning model, the wrapper – api microservice was developed. The protection mechanism is that all calls to the machine learning microservice go through it and are checked for damage and/or substitution, data coming from the outside is also validated on the microservice side. Before adding data to the training data DB, the record is reinforced with an electronic signature, the source data is hashed, connected with a secret key. The program code of the microservices has been developed, including codes of special methods and algorithms for their implementation, providing a mechanism for protecting the application from a training data poisoning attack. A monitoring system for a training data poisoning attack was deployed based on the open source software Elasticsearch, Logstash, Kibana through event log objects (appender): error (warn) and information (info), which can be used in SIEM systems.*

Practical value: *the practical value of the proposed solution includes a template protection mechanism in the form of a pattern that can be applied to a wide range of applications, including transferring the developed solution to any industry: fuel and energy, economics and more, due to the cross-platform nature of the solution itself.*

References

1. *Shameer Mohammed, S. Nanthini, N. Bala Krishna, Inumarthi V. Srinivas, Manikandan Rajagopal, M. Ashok Kumar, A new lightweight data security system for data security in the cloud computing, Measurement: Sensors, Volume 29, 2023, 100856. DOI: 10.1016/j.measen.2023.100856.*
2. *S. Achar, Cloud computing security for multi-cloud service providers: controls and techniques in our modern threat landscape, International Journal of Computer and Systems Engineering, 16(9), 2022, 379–384. DOI: 10.5281/zenodo.7084251.*
3. *Oludare Isaac Abiodun, Moatsum Alawida, Abiodun Esther Omolara, Abdulatif Alabdulatif, Data provenance for cloud forensic investigations, security, challenges, solutions and future perspectives: A survey, Journal of King Saud University – Computer and Information Sciences, Volume 34, Issue 10, Part B, 2022, 10217–10245. DOI: 10.1016/j.jksuci.2022.10.018.*
4. *Chakraborti, A., Curtmola, R., Katz, J., Nieh, J., Sadeghi, A. R., Sion, R., Zhang, Y., Cloud Computing Security: Foundations and Research Directions. Foundations and Trends in Privacy and Security, 3(2), 2022, 103–213. DOI: 10.1561/3300000028.*
5. *Ukeje, N., Gutierrez, J., Petrova, K., Information security and privacy challenges of cloud computing for government adoption: a systematic review, International Journal of Information Security, Volume 23, 2024, 1459–1475. DOI: 10.21203/rs.3.rs-3351319/v1.*

3 Nikolai V. Korneev, Dr.Sc., Professor, Gubkin Russian State University of Oil and Gas (National Research University), Moscow, Russia. E-mail: niccyper@mail.ru
4 Elena S. Kotrini, student, Gubkin Russian State University of Oil and Gas (National Research University), Moscow, Russia. E-mail: kotrini.elena@mail.ru

6. *Fatemeh Khoda Parast, Chandni Sindhav, Seema Nikam, Hadiseh Izadi Yekta, Kenneth B. Kent, Saqib Hakak, Cloud computing security: A survey of service-based models, Computers & Security,Volume 114, 2022, 102580. DOI: 10.1016/j.cose.2021.102580.*
7. *Ting Zhou, Hanshu Yan, Bo Han, Lei Liu, Jingfeng Zhang, Learning a robust foundation model against clean-label data poisoning attacks at downstream tasks, Neural Networks, Volume 169, 2024, 756-763. DOI: 10.1016/j.neunet.2023.10.034.*
8. *Ade Kurniawan, Yuichi Ohsita, Masayuki Murata, Detection of sensors used for adversarial examples against machine learning models, Results in Engineering, Volume 24, 2024, 103021. DOI: 10.1016/j.rineng.2024.103021.*
9. *Hamid Bostani, Veelasha Moonsamy, EvadeDroid: A practical evasion attack on machine learning for black-box Android malware detection, Computers & Security, Volume 139, 2024, 103676. DOI: 10.1016/j.cose.2023.103676.*
10. *Mahdee Jodayree, Wenbo He, Dr. Ryszard Janicki, Preventing Image Data Poisoning Attacks in Federated Machine Learning by an Encrypted Verification Key, Procedia Computer Science, Volume 225, 2023, 2723–2732. DOI: 10.1016/j.procs.2023.10.264.*
11. *Michael Gallagher, Nikolaos Pitropakis, Christos Chrysoulas, Pavlos Papadopoulos, Alexios Mylonas, Sokratis Katsikas, Investigating machine learning attacks on financial time series models, Computers & Security, Volume 123, 2022, 102933. DOI: 10.1016/j.cose.2022.102933.*
12. *Pahul Preet Singh, Fahim Islam Anik, Rahul Senapati, Arnav Sinha, Nazmus Sakib, Eklas Hossain, Investigating customer churn in banking: a machine learning approach and visualization app for data science and management, Data Science and Management, Volume 7, Issue 1, 2024, 7–16. DOI: 10.1016/j.dsm.2023.09.002*
13. *Badr Eddine Sabir, Mohamed Youssfi, Omar Bouattane, Hakim Allali, Authentication and load balancing scheme based on JSON Token For Multi-Agent Systems, Procedia Computer Science, Volume 148, 2019, 562–570. DOI: 10.1016/j.procs.2019.01.029.*
14. *Esquembre F., Chacón J., Saenz J., Vega J., Dormido-Canto S., A programmable web platform for distributed access, analysis, and visualization of data, Fusion Engineering and Design, Volume 197, 2023, 114049. DOI: 10.1016/j.fusengdes.2023.114049.*
15. *Dongyeop Lee, Daesik Lim, Jongseok Park, Soojeong Woo, Youngho Moon, Aesol Jung, Management Architecture With Multi-modal Ensemble AI Models for Worker Safety, Safety and Health at Work, Volume 15, Issue 3, 2024, 373–378. DOI: 10.1016/j.shaw.2024.04.008.*
16. *Miguel Correia, Wellington Oliveira, José Cecílio, Monintainer: An orchestration-independent extensible container-based monitoring solution for large clusters, Journal of Systems Architecture, Volume 145, 2023, 103035. DOI: 10.1016/j.sysarc.2023.103035.*
17. *Adabi Raihan Muhammad, Parman Sukarno, Aulia Arif Wardana, Integrated Security Information and Event Management (SIEM) with Intrusion Detection System (IDS) for Live Analysis based on Machine Learning, Procedia Computer Science, Volume 217, 2023, 1406–1415. DOI: 10.1016/j.procs.2022.12.339.*