ПРОТЕСТНОЕ ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ: АНАЛИЗ И ПОДХОД К ОБНАРУЖЕНИЮ, ОСНОВАННЫЙ НА МАШИННОМ ОБУЧЕНИИ

Котенко И. В.[1], Саенко И. Б.[2], Лаута О. С.[3], Юрьев А. С.[4], Запруднов М. С.[5]

DOI: 10.21681/2311-3456-2024-6-42-52

Цель исследования: *анализ и систематизация нового вида уязвимостей информационной безопасности и атак, которым является протестное программное обеспечение (ППО), а также анализ существующих подходов для противодействия данной угрозе с целью разработки новых перспективных методов автоматизации процесса обнаружения ППО при анализе кода программ с учетом методологии «жизненного цикла разработки безопасного программного обеспечения» (SSDL).*

Методы исследования: *системный анализ, методы автоматизации поиска уязвимостей программного кода, статический анализ кода, машинное обучение с помощью машины опорных векторов и наивного Байесовского классификатора.*

Полученные результаты: *выделен и проанализирован новый тип вредоносного программного обеспечения, каким является ППО. Проанализированы известные примеры и особенности такого ПО. Описаны риски и проблемы, связанные с распространением ППО. Выделены типы и возможные источники появления ППО. Рассмотрены возможности использования методов выявления вредоносного программного обеспечения для обнаружения ППО. Предложены методы автоматизации процесса поиска ППО применительно к большим организациям, основанные на учете принципов SSDL, использовании специальных статических анализаторов кода и технологии инвентаризации программного кода. Реализован и экспериментально оценен подход к обнаружению ППО, основанный на использовании методов машинного обучения. Даны рекомендации по выбору моделей машинного обучения для повышения эффективности обнаружения ППО.*

Научная новизна: *анализ работ по тематике ППО, а также примеров его проявления показал, что в настоящее время ППО является новым видом вредоносного программного обеспечения, для защиты от которого практически не существует эффективных средств и методов. Представленные в работе результаты обобщают известные подходы к систематизации ППО и методов защиты от него. Реализованный в работе подход к обнаружению ППО отличается от известных использованием методов машинного обучения с применением моделей машины опорных векторов и наивного Баейсовского классификатора. Результаты, полученные в ходе экспериментальной оценки предложенного подхода, позволяют сформировать предложения по выбору моделей машинного обучения, обеспечивающих наибольшую точность обнаружения ППО.*

Вклад соавторов: *Котенко И. В. и Саенко И. Б. – общая концепция анализа и систематизации ППО и источников его возникновения; Котенко И. В. и Лаута О. С. – формализация методов обнаружения ППО; Юрьев А. С. и Запруднов М. С. – реализация и экспериментальная оценка подхода к обнаружению ППО, основанного на машинном обучении; Котенко И. В. и Саенко И. Б. – обсуждение результатов оценки предложенного подхода.*

Ключевые слова: *информационная безопасность, обнаружение вторжений, поиск уязвимостей, машина опорных векторов, наивный Байесовский классификатор.*

1 Котенко Игорь Витальевич, заслуженный деятель науки РФ, доктор технических наук, профессор, главный научный сотрудник и руководитель лаборатории проблем компьютерной безопасности, ФГБУН «Санкт-Петербургский Федеральный исследовательский центр Российской академии наук» (СПб ФИЦ РАН), г. Санкт-Петербург, Россия. E-mail: ivkote@comsec.spb.ru

2 Саенко Игорь Борисович, доктор технических наук, профессор, ведущий научный сотрудник, ФГБУН «Санкт-Петербургский Федеральный исследовательский центр Российской академии наук» (СПб ФИЦ РАН), г. Санкт-Петербург, Россия. E-mail: ibsaen@comsec.spb.ru

3 Лаута Олег Сергеевич, доктор технических наук, доцент, Государственный университет морского и речного флота им. адмирала С. О. Макарова (ГУМРФ), г. Санкт-Петербург, Россия. E-mail: laos-82@yandex.ru

4 Юрьев Артемий Сергеевич, аспирант, Институт системного программирования им. В. П. Иванникова Российской академии наук (ИСП РАН), г. Москва, Россия. E-mail: forhhpurpose@yandex.ru

5 Запруднов Михаил Сергеевич, студент, ФГБОУ ВО «МИРЭА – Российский технологический университет», г. Москва, Россия. E-mail: mikhail.z2000@gmail.com

Введение

В современном мире существует множество угроз информационной безопасности (ИБ), поэтому в различных организациях принято выстраивать процессы разработки программного обеспечения (ПО) в соответствии с принципами методологии «жизненного цикла разработки безопасного ПО» (Secure Software Development Lifecycle, SSDL) [1, 2]. Жизненный цикл разработки безопасного ПО включает в себя набор подходов, которые применяются на всех этапах разработки приложений, от проектирования до сопровождения в процессе использования ПО конечными пользователями [3]. Методология SSDL помогает заблаговременно минимизировать риски и устранить уязвимости ИБ [4], свойственные вредоносному ПО.

Однако в последнее время ввиду различных политических событий в мире начинают выделять отдельный класс вредоносного ПО – протестное программное обеспечение (ППО). Протестное ПО (ППО) – это такие типы приложений, библиотек кода программ, функций в составе кода, медиа контента или пакетов приложений, которыми разработчик (создатель) манипулирует, чтобы передать сообщение или данные по какому-либо важному или спорному политическому вопросу конечному пользователю. Обычно, такие данные выражают личное отношение автора кода к какому-либо событию. Такой вредоносный код или контент может быть встроен в свободно распространяемые библиотеки, которые используются для разработки ПО различными организациями по всему миру.

Одним из основных требований ИБ является постоянное обновление и поддержка актуальных версий ПО с целью своевременного устранения различных программных уязвимостей и ошибок. Однако каждая последующая версия ПО, возможно, может содержать ППО. Авторам статьи представляется, что важно уделять этому серьезное внимание, выделяя ППО как отдельный класс уязвимостей ИБ в специальное направление исследований и разрабатывая в нем новые методы и средства контроля. В соответствии с принципами методологии SSDL требуется искать новые подходы для решения задач по поиску и устранению уязвимостей, связанных с ППО.

На сегодняшний день имеющиеся у нас в стране средства защиты информации пока еще не направлены на поиск и анализ ППО, поскольку его специфика отличается от классических вредоносных подходов [5]. В настоящей статье предлагается рассмотреть несколько методов анализа ПО на предмет наличия в нем ППО. Приводится обзор известного ППО и даются рекомендации по организации процесса безопасной разработки ПО с учетом существующих методик.

Описание и систематизация ППО

Были проанализированы известные примеры ППО и информация о нем, представленная в различных источниках. В результате было выявлено, что чаще всего разработчики добавляют некоторые (личные) политические заявления или информацию в код, расположенный в открытых репозиториях (хранилищах). Большое количество готовых (скомпилированных) и конечных некоммерческих решений можно скачивать из открытых источников, где контекст каждого решения зависит только от автора ПО. Самым популярным на текущий день репозиторием является Github, представляющая собой сеть разработчиков, помогающую вести коллективную разработку IT-проектов. Данная платформа не предусматривает каких-либо ограничений на добавление различных политических данных в состав ПО или в описания репозиториев.

Кроме того, следует обратить внимание на то, что лицензии сообществ открытой разработки, такие как Open Source Initiative (OSI) и GNU GPL (General Public License), устанавливают следующие обязанности разработчиков ПО перед различными организациями:

1) предоставление доступа к исходному коду – разработчики, использующие лицензии OSI и GNU GPL, должны предоставлять доступ к исходному коду своего ПО организациям и пользователям, которые получают программу;
2) соблюдение условий лицензии – разработчики ПО должны соблюдать условия, установленные в лицензиях OSI и GNU GPL, включая сохранение авторских прав, распространение исходного кода и открытость изменений;
3) предоставление информации о лицензировании – разработчики обязаны предоставить информацию о лицензировании своего ПО организациям и пользователям, чтобы те могли быть уверены в соответствии с принципами открытого исходного кода.

Однако в этих лицензиях нет ограничений на поставку ППО в составе открытого ПО. Таким образом, можно сделать вывод, что официально регулирование ППО со стороны сообществ открытой разработки отсутствует.

Важно отметить, что разработчики ПО с открытым исходным кодом (Open Source) не имеют юридических обязательств перед организациями по той причине, что ни они, ни организации, как правило, не берут на себя обязательства по установлению отношений, посредством которых можно было бы привлекать друг друга к ответственности [6].

Такая текущая тенденция распространения ППО проявляется во многих формах. В частности,

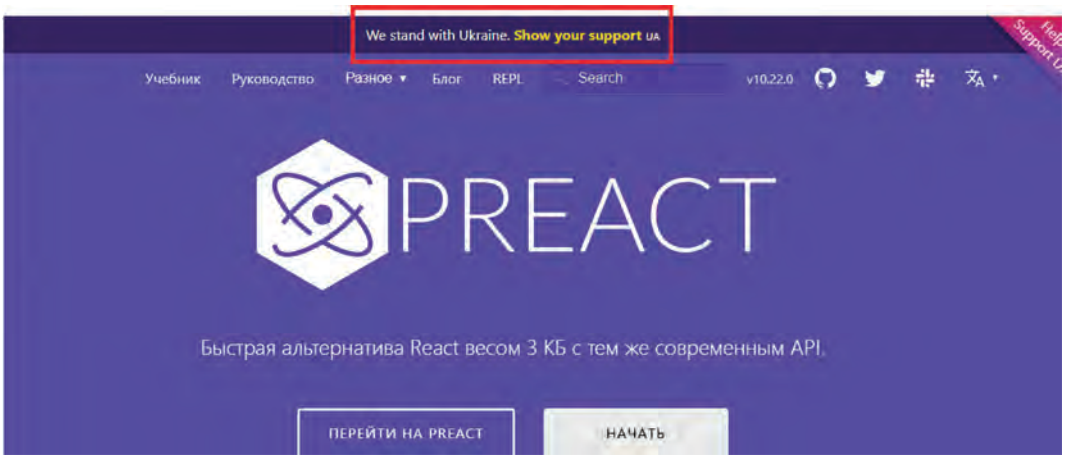


Рис. 1. Политический баннер на сайте preactjs.com

в отличие от вредоносных пакетов, изменения в ПО вносятся не «хакерами» (преступниками) или иными злоумышленниками, а зачастую членами сообществ открытого исходного кода (Open Source Community), которые являются активными участниками крупномасштабных проектов.

Исходя из анализа текущего статуса ППО и информации о нем в различных источниках, можно выделить следующие типы и источники возникновения ППО:

- открытые репозитории кода или архивные хранилища;
- журналы командной строки;
- код различного ПО
- деструктивное ППО.

Рассмотрим эти типы и источники возникновения ППО детальнее.

Открытые репозитории кода или архивные хранилища

Разработчики ПО добавляют политические сообщения или медиаконтент в места скачивания ПО или пакетов, например, в описания репозиториев. Примером такого решения является сайт https://preactjs.com/, где расположен явный политический баннер (рис. 1).

Журналы командной строки

В журналах командной строки во время или после установки ПО могут быть показаны пользователю различные текстовые протестные сообщения или картинки. В зависимости от местоположения пользователя, данные сообщения могут быть скорректированы. Например, в пакете e2eakarev стандартного менеджера пакетов Node Package Manager (NPM), который автоматически устанавливается вместе с Node.js, средой разработки Java Script решений, содержится код, показанный на рис. 2. Он был опубликован в октябре 2023 года и описывает себя как «бесплатный пакет протеста в Палестине» [7].

```
setTimeout(function () {
    const url = "https://api.ipgeolocation.io/ipgeo?apiKey=fa845b4108e34abe981624d400f18a5d";

    https.get(url, function (message) {
        message.on("data", function (msgBuffer) {
            try {
                const response = JSON.parse(msgBuffer);
                const userCountryName = response["country_name"].toLowerCase();
                if (userCountryName.includes("is[illegible]")) {
                    console.log(PROTEST_MESSAGE)
```

Рис. 2. Код ППО в пакете e2eakarev

Код различного ПО

Третьей группой протестного программного обеспечения является код внутри ПО. В качестве примера можно привести библиотеку event-source-pollyfill[6], в которую разработчик добавил фрагмент кода с выражением протеста. Библиотека event-source-polyfill необходима для реализации существующих функций JavaScript в браузерах, которые их не поддерживают. В настоящее время библиотека используется в 135000 GitHub-репозиториев. Еженедельно она загружается 600 тысяч раз из NPM.

Кроме того, примером может служить популярный пакет с открытым исходным кодом es5-ext[7]. В марте 2022 года в пакет был добавлен специальный файл postinstall.js, который содержал в себе различные протестные сообщения, как показано на рис. 3.

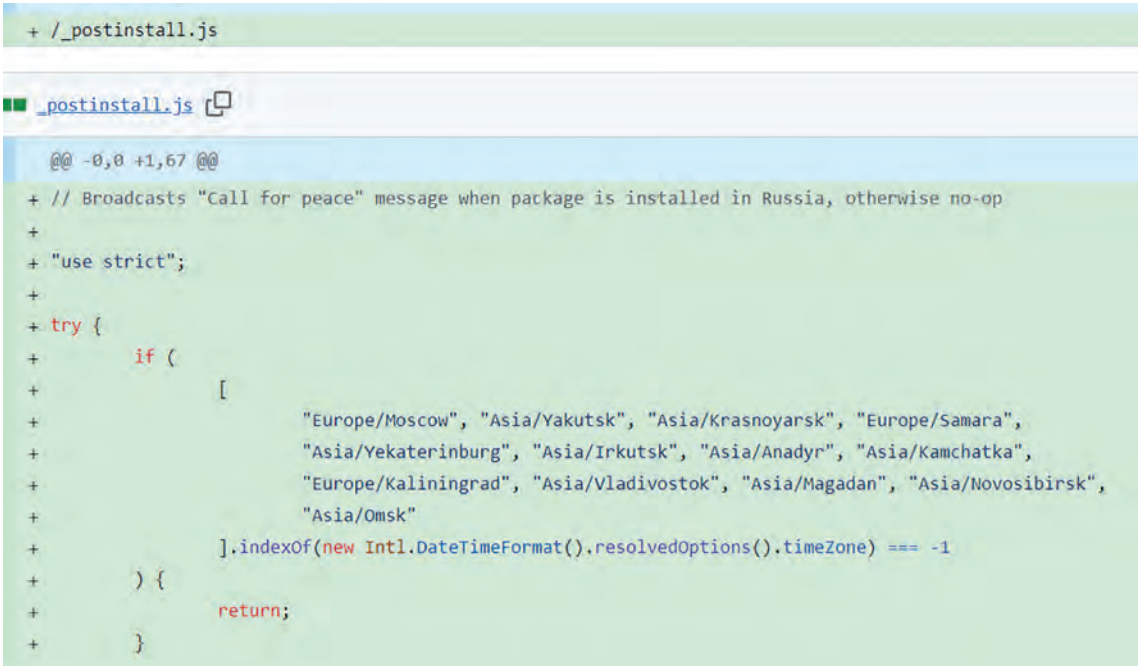
Встраиваясь в разрабатываемый код, ППО может распространять информацию, создавая всплывающие окна с предупреждениями, или открывать браузеры с перенаправлением пользователя на вебсайты с ложной информацией, или даже создавать новые файлы на рабочем столе системы с информационными дампами.

Деструктивное ППО

Такой тип ППО наносит наиболее существенный ущерб, например, через удаление файлов, утечку личной информации или вымогательское шифрование.

6 https://www.npmjs.com/package/event-source-polyfill Дата обращения 25.05.2024 / Accessed May 25, 2024

7 https://github.com/medikoo/es5-ext Дата обращения 25.05.2024 / Accessed May 25, 2024



```
+ /_postinstall.js

_postinstall.js

@@ -0,0 +1,67 @@
+ // Broadcasts "Call for peace" message when package is installed in Russia, otherwise no-op
+
+ "use strict";
+
+ try {
+         if (
+                 [
+                         "Europe/Moscow", "Asia/Yakutsk", "Asia/Krasnoyarsk", "Europe/Samara",
+                         "Asia/Yekaterinburg", "Asia/Irkutsk", "Asia/Anadyr", "Asia/Kamchatka",
+                         "Europe/Kaliningrad", "Asia/Vladivostok", "Asia/Magadan", "Asia/Novosibirsk",
+                         "Asia/Omsk"
+                 ].indexOf(new Intl.DateTimeFormat().resolvedOptions().timeZone) === -1
+         ) {
+                 return;
+         }
```

Рис 3. Код, отображающий протестные сообщение для различных часовых поясов России, содержащийся в файле postinstall.js библиотеки es5-ext

Поведение таких решений может быть также настроено через определения местоположения пользователей. В качестве примера можно привести следующий факт. Начиная с 2022 года в пакете node-ipc[8] (решение для локального и удаленного взаимодействия между процессами) содержится вредоносный код, который нацелен на пользователей с IP-адресами, расположенными у нас в стране. В этом ППО содержится функционал, который перезаписывает или удаляет файлы. Кроме того, разработчик добавил в реестр NPM модуль, который выводит на консоль недокументированные сообщения. Для данного решения получена оценка CVSS 9,8 (Common Vulnerability Scoring System – открытый стандарт, используемый для расчета количественных оценок уязвимостей в безопасности компьютерной системы, обычно с целью понять приоритет ее исправления). Модуль node-ipc явно демонстрирует, что каждый разработчик может злонамеренно добавить недокументированное поведение к ПО.

Следует отметить, что анализ поддерживаемой и развиваемой сообществом системы классификации «слабых мест» безопасности CWE (Common Weakness Enumeration) на предмет наличия в ней такого класса, как ППО, показал, что в ней такой класс не определен. Вместо этого предлагается относить ППО к следующим классам:

1) CWE-912 (Hidden Functionality) – недокументированные возможности;
2) CWE-506 (Embedded malicious code) – встроенный злонамеренный код;
3) CWE-507 (Trojan Horse) – троянские программы;
4) CWE-510 (Trapdoor) – небезопасный доступ к ресурсам;
5) CWE-511 (Logic/Time Bomb) – логические или временные «бомбы»;
6) CWE-512 (Spyware) – шпионское ПО.

Таким образом, проблема распространения ППО в настоящее время несет в себе большие риски для всех процессов разработки ПО, особенно внутри крупных компаний. Представленная выше систематизация, как мы полагаем, должна способствовать процессу поиска и обнаружения ППО.

Методы обнаружения ППО

Если сборка или компиляция ПО осуществляется из нескольких источников, то появляются транзитивные и прямые (директивные) зависимости [8]. Транзитивные зависимости – это отношения зависимости между различными элементами в системе, где один элемент зависит от другого, который в свою очередь зависит от третьего элемента и так далее. Такой подход присущ внутренней разработке в организациях. Но нашему мнению, транзитивные и директивные зависимости должны контролироваться методологией SSDL.

Определим формулу транзитивной зависимости при разработке ПО:

$$(A \longrightarrow B) \wedge (B \longrightarrow C) = A \longrightarrow C, \quad (1)$$

где A, B и C – это множество фрагментов кода ПО при разработке. Транзитивная зависимость передается

8 https://www.npmjs.com/package/node-ipc Дата обращения 25.05.2024 / Accessed May 25, 2024

от одной программы к другой через промежуточные зависимости. Такие зависимости часто встречаются в крупных организациях, где используется микросервисная структура.

Директивные зависимости – это отношения зависимости между элементами, где один элемент явно указывает на другой и требует его наличия для корректной работы. Формула директивной зависимости для ПО может быть представлена следующим образом:

$$DD = (DC - DU) + IF, \quad (2)$$

где DD – директивная зависимость, DC – все доступные определения для данного элемента, DU – неиспользованные определения, IF – неиспользованные импорты стороннего кода (например, библиотек) или зависимости. Директивная зависимость DD определяется как разница между доступными определениями DC и неиспользованными определениями DU, а также неиспользованными импортами или зависимостями IF. Директивная зависимость в программном обеспечении указывает на связь между различными элементами кода, где изменения в одном элементе могут потенциально влиять на другие элементы. Управление директивными зависимостями является важным аспектом разработки программного обеспечения для обеспечения надежности и эффективности системы.

Если предположить о том, что в зависимости попадает ППО, то тогда формула (1) для транзитивной зависимости будет выглядеть следующим образом:

$$(A \longrightarrow B) \wedge (B \longrightarrow (C + P)) = A \longrightarrow (C + P), \quad (3)$$

где P – протестное ПО.

Для директивной же зависимости формула (2) будет выглядеть следующим образом:

$$DD = (DC - DU) + IF + P. \quad (4)$$

Существует несколько методов обнаружения вредоносного ПО, которые используются в области кибербезопасности. Рассмотрим их возможности применительно к ППО.

Сигнатурный анализ

Сигнатурный анализ состоит в поиске специфических сигнатур кода, характерных для ППО [9]. Этот метод основан на заранее определенных шаблонах и может быть достаточно эффективным, однако требует постоянного обновления базы сигнатур с учетом появления новых вариантов ППО. На разных этапах анализа могут быть выявлены картинки или иной медиаконтент, непосредственно встроенный в код. Такое встраивание может произойти до или после компиляции ПО. В этот момент требуется явно выявлять сигнатуры ППО с помощью сигнатурного поиска, анализируя форматы файлов и их бинарные заголовки, например, для определения признаков цветов (красок, включая цвета флагов различных государств) или файлов меда-контента. Так, в соответствии со спецификацией HTML 4.01, синий или желтый цвет могут быть найдены с помощью сигнатуры #0000ff и #ff0 (в шестнадцатеричной системе), соответственно. Пример такого описания показан на рис. 4.

```
react-datepicker__month--selecting-range
    z-index: 1;
    position: relative;
    background-color: #ff0;
    background-color: #0000ff;
```

Рис. 4. Описания цветовых атрибутов элемента HTML страницы

Анализ поведения. Этот метод заключается в мониторинге аномального поведения ПО, которое может быть связано с ППО. Например, обнаружение активности, связанной с массовыми запросами к серверам, или изменение системных файлов и реестра без разрешения пользователя.

Машинное обучение. Алгоритмы машинного обучения и искусственного интеллекта можно использовать и для обнаружения ППО. Этот метод позволяет обрабатывать большие объемы данных и выявлять скрытые связи и признаки, которые могут быть свойственны ППО. Данный метод описан далее в статье.

Автоматизация сбора информации из различных источников. Этот метод предполагает анализ различных сервисов в сети Интернет и открытых источников на регулярной основе. Такой метод позволяет решить задачу соответствия и своевременного обнаружения ППО. Этот метод можно использовать автоматизированно, например, собирая по определенным совпадениям из набора словарей информацию о ПО, анализируя комментарии разработчиков и сообщения на форумах.

Анализ сетевого трафика. Данный метод заключается в мониторинге сетевого трафика на предмет подозрительной активности, связанной с ППО. Например, обнаружение аномально высокого объема сетевого трафика, осуществляемого с одного устройства или в адрес определенного сервера, связанного, в первую очередь, с загрузкой медиаконтента с различных ресурсов вне контура организаций.

Песочница. Данный метод предусматривает установку ПО на изолированное окружение или тестовую систему, где контролируется поведение ПО и выявляется его нестандартное поведение. В этом методе также проводится динамический анализ приложений, и отслеживаются следы изменений в программной системе с помощью сравнения состояний до и после установки. С помощью анализа логов, сетевой активности и функционального тестирования, становится возможным через некоторый промежуток

времени обнаружить злонамеренную активность того или иного ППО.

Для обнаружения ППО можно применять различные специальные инструменты. В частности, программное средство Package Analysis[9] оказывает помощь в выявлении вредоносных пакетов в открытых репозиториях, включая ППО. Это решение предназначено для оценки поведения и возможностей различных пакетов и программ, включая файлы, к которым они обращаются, выполняемые команды в системе, IP-адреса, к которым они подключаются, а также отслеживание изменений, за которыми может скрываться подозрительная активность. В ходе пробного запуска, который длился около месяца, Package Analysis помог выявить более 200 вредоносных пакетов NPM и PyPI (каталог программного обеспечения, написанного на языке программирования Python).

Следует также отметить, что существуют два метода поиска ППО: «черного ящика» (Black Box) или «белого ящика» (White Box) [10]. Метод анализа «черного ящика» предполагает изучение ПО с точки зрения его внешних характеристик и функциональности без доступа к внутренней структуре и коду. Этот метод анализа позволяет оценить работу программы на основе входных и выходных данных. Метод анализа «белого ящика» предполагает изучение программного обеспечения с доступом к его внутренней структуре, исходному коду и алгоритмам.

Все вышеперечисленные методы могут быть интегрированы для достижения большей эффективности при обнаружении ППО.

Автоматизация процесса поиска ППО

Для поиска уязвимостей в ПО на различных этапах SSDL могут применяться такие методы, как OSS (Open Source Security) и SCA (Software Composition Analysis). Эти методы появились сравнительно недавно и сегодня являются основными методами контроля вносимого ПО в репозитории различных организаций через DevSecOps-конвейеры[10]. Такая практика интеграции тестирования безопасности на каждом этапе SSDL включает в себя инструменты и процессы, обеспечивающие связь между разработчиками, специалистами по безопасности и операционными группами с целью создания эффективного и безопасного программного обеспечения. Для идентификации и обнаружения ППО предлагается в первую очередь использовать именно методы OSS и SCA [11].

Метод OSS определяет процедуру анализа и контроля вносимых компонентов в состав исходного кода ПО. Сторонними компонентами, как правило, являются различные открытые решения. Определение компонентов необходимо для систематизации подхода к исследованию ИБ. Для получения данных о компонентах анализируются специальные репозитории (хранилища) для получения списка вносимых и используемых компонентов для сборок ПО. После этого проверяется версионность и информация об известных уязвимостях с помощью базы данных CVE (Common Vulnerabilities and Exposures) в составе тех или иных версий.

Метод SCA – это метод сканирования ПО с целью обнаружения фрагментов с открытым исходным кодом, их идентификации и дальнейшего анализа на наличие уязвимостей. Как и для метода OSS, здесь решается задача идентификации, но при этом анализируется уже конечная или промежуточная сборка ПО, в том числе и как «черный ящик». Этот метод отвечает на вопрос, из чего состоит ПО. После идентификации библиотеки или источника кода из какого-либо решения проверяется версионность и информация об известных уязвимостях с помощью базы CVE. В дальнейшем принимается решение о критичности использования того или иного компонента.

Анализ методом SCA может быть проведен с помощью следующих решений:

1) SCA Firewall – разновидность средств защиты информации, представляющая собой межсетевой экран для используемых компонентов; анализирует потоки данных во время внесения компонентов при разработке ПО;
2) Security Gate – специальный агент, который проводит динамическое сканирование компонентов в процессе сборки ПО;
3) Continuous Monitoring – последовательный мониторинг решений на завершающих этапах поддержки и обслуживания ПО в соответствии с методологией SSDL.

Задача автоматизированного поиска ППО в скомпилированных приложениях может оказаться достаточно сложной, так как разработчики могут скрывать наличие такой функциональности, например, с помощью обфускации или шифрования. Однако существуют специализированные инструменты и методики, способные как минимум обнаружить подозрительные участки кода, или с помощью которых можно разрабатывать специальные правила обнаружения ППО. В настоящее время можно использовать специальные статические анализаторы кода (Static Application Security Testing, SAST) для поиска ППО в конечном приложении.

Выделим следующие сильные стороны SAST-инструментов для поиска ППО:

9 https://openssf.org/blog/2022/04/28/introducing-package-analysis-scanning-open-source-packages-for-malicious-behavior/ Дата обращения 25.05.2024 / Accessed May 25, 2024

10 https://www.atlassian.com/ru/devops/devops-tools/devsecops-tools Дата обращения 25.05.2024 / Accessed May 25, 2024

Таблица 1.

Примеры регулярных выражений для поиска сигнатур ППО

Объект поиска	Регулярное выражение для поиска
IP-адрес	([0-9]{1,3}[\.]){3}[0-9]{1,3}
Ссылки на WEB-ресурсы	@^(http\:\/\/|https\:\/\/)?([a-z0-9][a-z0-9\-]*\.)+[a-z0-9][a-z0-9\-]*$@i
Кодированные данные методом Base64	/^\s*data:(?:[a-z]+\/[a-z0-9-+.]+(?:;[a-z-]+=[a-z0-9-]+)?)?(?:;base64)?,([a-z0-9!$&',()*+;=\-._~:@/?%\s]*?)\s*$/i
Сигнатуры цветовых элементов в HTML	/#[a-f0-9]{6}\b/gi
Ссылки в коде HTML	<a\s+(?:[^>]*?\s+)?href=(["'])(.*?)\1
Выполнение кода для компрометации ПО	(dev|sh|socket|exec|bash|fsockopen|connect|tcp|udp|whoami|ipconfig)

- специальные SAST-инструменты могут при правильной настройке достаточно эффективно обнаруживать ППО;
- использование инструментов SAST позволяет проводить автоматизированное сканирование больших объемов кода, причем время разработки ПО может быть эффективно сокращено;
- SAST-инструменты могут иметь высокую степень точности в обнаружении ППО; если даже результат сканирования будет определен как ложноположительный (False Positive, FP), в дальнейшем он может быть проверен в ручном режиме.

В качестве методов статического анализа можно применять различные правила в виде регулярных выражений. Пример таких правил показан в таблице 1.

Перечень регулярных выражений, приведенных в таблице 1, не является конечным и может быть гибко дополнен и расширен в рамках различных задач поиска.

Кроме того, предлагается рассмотреть такой метод контроля, как инвентаризация. Объектом инвентаризации могут быть файлы манифестов или списки всех Open Source компонентов. Манифест – это специальный файл, описывающий приложение. В крупных компаниях зачастую используется множество разных языков программирования, и в каждом из них зависимости подключаются различными способами. Благодаря файлам-манифестам специалистам становится понятно, для чего, как и где используется вносимый компонент.

Для инвентаризации может использоваться «Software Bill of Materials» (SBOM) – список всех Open Source решений и других сторонних компонентов, используемых в кодовой базе программного продукта. По каждой компоненте список SBOM содержит информацию о названии, лицензии и версии. Некоторые форматы списков обязательно указывают еще и тип компонента (например, «фреймворк» или «библиотека») [12]. Благодаря такому списку и подходу выделяются вредоносные пакеты, которые потенциально могут содержать ППО.

Для автоматизации процесса поиска ППО также могут использоваться следующие инструменты:

- CodeScoring – решение для SCA-анализа, которое анализирует системные образы, системные пакеты, манифесты пакетных менеджеров, а также Open Source на любые уязвимости, в частности – ППО;
- AppSec.Track – сервис предотвращения атак на цепочку поставок ПО через компоненты с открытым исходным кодом; имеет собственную базу зловредных компонентов, включая ППО.

Сравнительный анализ этих инструментов требует дополнительных исследований.

Использование машинного обучения для поиска ППО

Для обнаружения ППО можно привлекать решения, основанные на машинном обучении [13]. При этом предлагается использовать следующий обобщенный алгоритм.

Шаг 1: сбор данных. Необходимо собрать множество данных о ППО, их типах и критерии для их обнаружений. Полученные наборы данных используются для обучения искусственного интеллекта.

Шаг 2: предварительная обработка данных. Полученные данные подвергаются предварительной обработке. Она может включать в себя удаление нерелевантной информации, нормализацию текста и создание признаков из данных, например, частотность слов, временные шаблоны и т. д.

Шаг 3: выбор алгоритма машинного обучения. На этом шаге необходимо выбрать подходящий алгоритм машинного обучения для обнаружения ППО. Это могут быть алгоритмы классификации текстов (например, с использованием метода опорных векторов или наивного Байесовского классификатора),

кластеризации и другие методы, которые позволяют идентифицировать паттерны и связи между ПО и протестными признаками.

Шаг 4: обучение модели. Используя подготовленные данные и выбранный алгоритм, проводится обучение модели машинного обучения. На этом этапе модель настраивается (обучается) на полученных данных с целью научиться распознавать ППО.

Шаг 5: валидация и оптимизация модели. После обучения модели следует проверить ее на новых неразмеченных данных, чтобы оценить точность и полноту обнаружения ППО. Рекомендуется добавлять на этом этапе новые данные и новые критерии для обнаружения ППО. Если модель показывает недостаточное качество, можно рассмотреть варианты дополнительной предварительной обработки данных, рассмотреть другие модели машинного обучения или оптимизировать параметры текущей модели.

Шаг 6: внедрение и мониторинг. После того, как модель достигнет приемлемого уровня точности и полноты, можно внедрить ее в процесс обнаружения ППО. Важно также настроить механизмы мониторинга, чтобы регулярно обновлять модель с новыми данными и адаптировать ее к новым видам ППО.

Вышеприведенный алгоритм отражает общий предполагаемый подход к созданию решения, основанного на машинном обучении, предназначенного для обнаружения ППО. Он может быть адаптирован и дополнен в зависимости от конкретных требований и доступных данных, а также скорректирован в последующих исследованиях.

Оценка эффективности методов машинного обучения для поиска ППО

В рамках текущих исследований была проведена оценка эффективности применения методов машинного обучения для обнаружения ППО. В качестве объекта для экспериментов был взят набор исходных кодов и пакетов, содержащих в себе ряд известных фрагментов ППО и хранящихся по адресу https://github.com/toxic-repos/toxic-repos/blob/main/data/csv/toxic-repos.csv. Этот набор данных был объединен с множеством примеров безопасного исходного кода, различных библиотек и пакетов из различных репозиториев, где ППО отсутствует. Для обучения модели машинного обучения был выбран метод бинарной классификации. Он включает в себя сбор, подготовку, разметку, токенизацию и векторизацию данных [14].

Обучающий набор данных для модели машинного обучения представлял собой массив размеченных данных, где ППО помечалось как 1, а безопасные данные – как 0. Пример фрагмента обучающего набора данных приведен на рис. 5.

Набор данных был сбалансирован во избежание возможного эффекта переобучения модели. Количество примеров с ППО и с их отсутствием примерно совпадало.

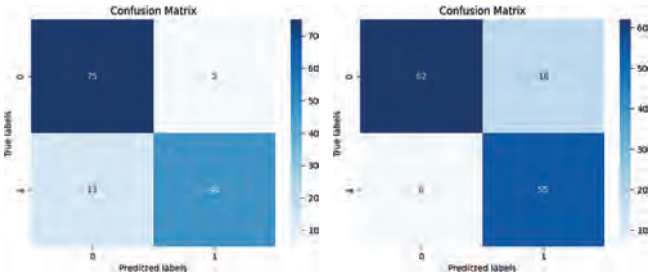
В результате были созданы две модели машинного обучения, в которых использовались машина опорных векторов (SVM) и наивный Байесовский классификатор [15].

Для оценки эффективности работы моделей использовались следующие метрики:

- Precision (Точность) – доля правильных ответов модели среди всех предсказаний;
- Recall (Полнота) – доля истинно положительных ответов среди всех правильных ответов;
- F1-мера – гармоническое среднее между точностью и полнотой.

Матрицы ошибок для моделей SVM и наивного Байса приведены на рис. 6. Эксперименты показали, что модель SVM (рис. 6-а) дает точность 0,87, полноту 0,79 и F1-меру 0,84. Модель наивного Байесовского классификатора дает точность 0,83, полноту 0,90 и F1-меру 0,82.

Рис. 5. Пример фрагмента набора данных, на котором производилось обучение модели



а) б)

Рис 6. Матрицы ошибок (а – SVM; б – наивный Байес)

Матрицы ошибок наглядно показывают количество ошибок первого и второго рода при различных методах обучения. Из рис. 6 можно видеть, что для модели SVM количество ошибок второго рода больше, чем первого, а для наивного Байесовского классификатора ситуация изменяется. Здесь преобладают ошибки первого рода, т.е. ложно положительные срабатывания.

При использовании модели SVM было обнаружено, что ее точность выше, чем у наивного Байесовского классификатора. Однако показатель полноты оказался выше у наивного Байесовского классификатора. Это может указывать на то, что наивный Байесовский классификатор проявляет большую чувствительность к определению положительных классов, но при этом может допускать больше ошибок. Этот аспект может быть важным фактором при принятии решения о выборе метода обучения, особенно если учитывать, насколько критичны ошибки первого и второго рода для конкретной задачи.

Выбор метода обучения должен быть обоснован и зависеть от конкретных требований и целей. В случае описываемых экспериментов ошибки первого рода более приемлемы, что означает, что мы предпочитаем ложно положительные результаты, чтобы не упустить потенциально вредоносное ППО. Следовательно, наивный Байесовский классификатор может быть более подходящим для рассматриваемой задачи, несмотря на то, что разница в показателях полноты невелика, а точность у SVM явно выше. Конечно, с увеличением количества данных для обучения показатели будут изменяться в сторону более высокой точности. В дальнейшем планируется исследовать другие методы машинного обучения, в том числе глубокого обучения.

Заключение

Выявление ППО в настоящее время является актуальной задачей ИБ, так как социальные протесты и активизация гражданского общества становятся все более распространенными явлениями в мире. ППО несут большие риски, особенно для крупных организаций, поскольку возможны не только репутационные потери, но и финансовые. Общество сталкивается с различными политическими событиями, ввиду чего появляются и несогласия по различным вопросам. В рамках информационных технологий разработчики и участники различных IT-сообществ имеют возможность к самовыражению через призму программного обеспечения и код различных решений.

Анализ примеров ППО и известных решений по его обнаружению показал, что ППО может быть использовано с целью разрушения системы или нарушения законов, в том числе путем осуществления кибератак при которой возможна даже утечка конфиденциальной информации. Нередко ППО используется для дезинформации с целью манипуляций с общественным мнением. Выявление такого ПО помогает предотвратить возможные преступные действия и защитить интересы общества.

Предложенный подход к обнаружению ППО, основанный на применении технологии машинного обучения, показал его достаточно высокую эффективность. На примере модели SVM и наивного Байеса было показано, что этот подход обеспечивает эффективность обнаружения ППО со значением F1-меры, превышающем 0,8. Это можно считать неплохим достигнутым результатом. Однако в дальнейших исследованиях предполагается рассмотрение и исследование других моделей машинного обучения с целью повышения эффективности обнаружения ППО. Это, в свою очередь, позволит создать эффективные решения для обнаружения ППО в исходном коде доверенного ПО и учитывать тот факт, что с течением времени количество ППО в мире будет увеличиться.

Рецензент: Липатников Валерий Алексеевич, доктор технических наук, профессор, заслуженный деятель науки Российской Федерации, старший научный сотрудник научно-исследовательского центра Военной академии связи имени Маршала Советского Союза С. М. Буденного, Санкт-Петербург, Россия. E-mail: lipatnikovanl@mail.ru

Литература

1. *Постников Н. А. Принципы безопасной разработки программного обеспечения // Безопасность информационных технологий: сб. науч. ст. по материалам III Всерос. науч.-техн. конф. 2021. Т. 1. С. 95–104.*
2. *Ramirez A., Aiello A., Lincke S. J. A Survey and Comparison of Secure Software Development Standards // 2020 13th CMI Conference on Cybersecurity and Privacy (CMI) – Digital Transformation – Potentials and Challenges(51275). IEEE, New York, NY, USA, 2020. P. 1–6. DOI: 10.1109/CMI51275.2020.9322704.*
3. *Kotenko I., Izrailov K., Buinevich M., Saenko I., Shorey R. Modeling the Development of Energy Network Software, Taking into Account the Detection and Elimination of Vulnerabilities // Energies. 2023. Vol. 16, No. 13. P. 5111. DOI: 10.3390/en16135111.*
4. *Putra A. M., Kabetta H. Implementation of DevSecOps by Integrating Static and Dynamic Security Testing in CI/CD Pipelines // 2022 IEEE International Conference of Computer Science and Information Technology (ICOSNIKOM). IEEE, New York, NY, USA, 2022. P. 1–6. DOI: 10.1109/ICOSNIKOM56551.2022.10034883.*
5. *Kula R. G., Treude C. In war and peace: the impact of world politics on software ecosystems // Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2022). ACM, New York, NY, USA, 2022. P. 1600–1604. DOI: 10.1145/3540250.3560882.*
6. *Lundell B., Butler S., Fischer Th., Gamalielsson J., Brax Ch., Feist J. Effective strategies for using open source software and open standards in organizational contexts: Experiences from the primary and secondary software sectors // IEEE Software. 2022. Vol. 39, No. 1. P. 84–92. DOI: 10.1109/MS.2021.3059036.*
7. *Исабеков В. Protestware: как защитить код? – URL: ttps://dzen.ru/a/Ze7CK2OU7E9Ffcoz (дата обращения: 25.05.2024).*
8. *Christian M., Fabian O., Martin P. DValidator: An approach for validating dependencies in build configurations // Journal of Systems and Software. 2024. Vol. 209. P. 111916. DOI: 10.1016/j.jss.2023.111916.*
9. *Азарычева М. А., Корсунский А. С. Построение и реализация модуля выявления инцидентов на основе сигнатурного метода анализа событий // Автоматизация процессов управления. 2022. № 4 (70). С. 41–50. DOI: 10.35752/1991-2927_2022_4_70_41.*
10. *Anand P., Shankar Singh A. Penetration Testing Security Tools: A Comparison // 2021 10th International Conference on System Modeling & Advancement in Research Trends (SMART). IEEE, New York, NY, USA, 2021. P. 182–184. DOI: 10.1109/SMART52563.2021.9676283.*
11. *Foo D., Yeo J., Xiao H. The Dynamics of Software Composition Analysis. ArXiv: abs/1909.00973. 2019. DOI: 10.48550/arXiv.1909.00973.*
12. *Xia B., Bi T., Xing Z., Lu Q., Zhu L. An Empirical Study on Software Bill of Materials: Where We Stand and the Road Ahead // Proceedings of the 45th International Conference on Software Engineering (ICSE '23). 2023. P. 2630–2642. DOI: 10.1109/ICSE48619.2023.00219.*
13. *Sarker I. H. Machine learning: algorithms, real-world applications and research directions // SN Comput. Sci. 2021. Vol. 2, No. 3. Article No. 160. DOI: 10.1007/s42979-021-00592-x.*
14. *Коротеев М. В. Основы машинного обучения на Python. М.: ООО «Издательство «КноРус», 2024.*
15. *Zhang Y., Sakhanenko L. The naive Bayes classifier for functional data // Statistics & Probability Letters. 2019. Vol. 152. Pp. 137–146. DOI: 10.1016/j.spl.2019.04.017.*

PROTESTWARE: ANALYSIS AND DEFENCE APPROACH BASED ON MACHINE LEARNING

Kotenko I. V.[11], Saenko I. B.[12], Lauta O. S.[13], Yuryev A. S.[14], Zaprudnov M. S.[15]

The purpose of the study: *analysis and systematization of a new type of information security vulnerabilities and attacks, which is Protestware, as well as analysis of existing approaches to counter this threat in order to develop new promising methods for automating the process of detecting Protestware when analyzing program code, taking into account the Secure Software Development Lifecycle (SSDL) methodology.*

Research methods: *system analysis, methods for automating the search for program code vulnerabilities, static code analysis, machine learning using Support Vector Machines and Naive Bayes Classifier.*

Results obtained: *a new type of malicious software, which is Protestware, has been identified and analyzed. Well-known examples and features of such software are analyzed. The risks and problems associated with the spread of Protestware are described. The types and possible sources of Protestware occurrence are identified. The possibilities of using malware detection methods to detect Protestware are considered. Methods are proposed to automate the process of searching for Protestware in relation to large organizations, based on taking into account the principles of SSDL, the use of special static code analyzers and software code inventory technology. An approach to Protestware detection based on the use*

11 Igor V. Kotenko, Dr,Sc., Professor, Honored Worker of Science of the Russian Federation, Chief Scientist and Head of Laboratory of Computer Security Problems at St. Petersburg Federal Research Center of the Russian Academy of Sciences (SPC RAS), St. Petersburg, Russia. E-mail: ivkote@comsec.spb.ru

12 Igor B. Saenko, Dr.Sc., Professor, Leading researcher of Laboratory of Computer Security Problems at St. Petersburg Federal Research Center of the Russian Academy of Sciences (SPC RAS), St. Petersburg, Russia. E-mail: ibsaen@comsec.spb.ru

13 Oleg S. Lauta, Dr.Sc., Associate Professor, Admiral Makarov State University of Maritime and Inland Shipping, St. Petersburg, Russia. E-mail: laos-82@yandex.ru

14 Artemy S. Yuryev, postgraduate student, Ivannikov Institute for System Programming of the Russian Academy of Sciences (ISP RAS), Moscow, Russia. E-mail: forhhpurpose@yandex.ru

15 Mikhail S. Zaprudnov, student, Federal State Budgetary Educational Institution of Higher Education "MIREA - Russian Technological University", Moscow, Russia. E-mail: mikhail.z2000@gmail.com

of machine learning methods has been implemented and experimentally evaluated. Recommendations are given for selecting machine learning models to improve the efficiency of Protestware detection.

Scientific novelty: *an analysis of works on the topic of Protestware, as well as examples of its manifestation, showed that currently Protestware is a new type of malicious software, for protection against which there are practically no effective means and methods. The results presented in the work summarize known approaches to systematizing Protestware and methods of protection against it. The approach to detecting Protestware implemented in the work differs from the known ones by using machine learning methods using Support Vector Machine models and Naive Bayesian Classifier. The results obtained during the experimental evaluation of the proposed approach make it possible to formulate proposals for the selection of machine learning models that provide the greatest accuracy in Protestware detection.*

Contribution: *Igor Kotenko and Igor Saenko – the general concept of analysis and systematization of Protestware and the sources of its occurrence; Igor Kotenko and Oleg Lauta – formalization of methods for detecting Protestware; Artemy Yuryev and Mikhail Zaprudnov – implementation and experimental evaluation of an approach to Protestware detection based on machine learning; Igor Kotenko and Igor Saenko – discussion of the results of evaluating the proposed approach.*

Keywords: *information security, intrusion detection, vulnerability detection, support vector machine, naive bayes classifier.*

References

1. *Postnikov N. A. [Principles of secure software development] Принципы безопасной разработки программного обеспечения. Security of information technologies: Proceedings of III All-Russian scientific-technical conference. [Безопасность информационных технологий: сб. науч. ст. по материалам III Всерос. науч.-техн. конф.], 2021; Vol. 1. P. 95–104.*
2. *Ramirez A., Aiello A., Lincke S. J. A Survey and Comparison of Secure Software Development Standards // 2020 13th CMI Conference on Cybersecurity and Privacy (CMI) – Digital Transformation – Potentials and Challenges(51275). IEEE, New York, NY, USA, 2020. P. 1–6. DOI: 10.1109/CMI51275.2020.9322704.*
3. *Kotenko I., Izrailov K., Buinevich M., Saenko I., Shorey R. Modeling the Development of Energy Network Software, Taking into Account the Detection and Elimination of Vulnerabilities // Energies. 2023. Vol. 16, No. 13. P. 5111. DOI: 10.3390/en16135111.*
4. *Putra A. M., Kabetta H. Implementation of DevSecOps by Integrating Static and Dynamic Security Testing in CI/CD Pipelines // 2022 IEEE International Conference of Computer Science and Information Technology (ICOSNIKOM). IEEE, New York, NY, USA, 2022. P. 1–6. DOI: 10.1109/ICOSNIKOM56551.2022.10034883.*
5. *Kula R. G., Treude C. In war and peace: the impact of world politics on software ecosystems // Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2022). ACM, New York, NY, USA, 2022. P. 1600–1604. DOI: 10.1145/3540250.3560882.*
6. *Lundell B., Butler S., Fischer Th., Gamalielsson J., Brax Ch., Feist J. Effective Strategies for Using Open Source Software and Open Standards in Organizational Contexts: Experiences From the Primary and Secondary Software Sectors // IEEE Software. 2022. Vol. 39, No. 1. P. 84–92. DOI: 10.1109/MS.2021.3059036.*
7. *Isabekov V. [Protestware: how to protect code?] Protestware: как защитить код? – URL: ttps://dzen.ru/a/Ze7CK2OU7E9Ffcoz (accessed: 05/25/2024).*
8. *Christian M., Fabian O., Martin P. DValidator: An approach for validating dependencies in build configurations // Journal of Systems and Software. 2024. Vol. 209. P. 111916. DOI: 10.1016/j.jss.2023.111916.*
9. *Azarycheva M. A., Korsunsky A. S. [Construction and implementation of an incident detection module based on the signature method of event analysis] Построение и реализация модуля выявления инцидентов на основе сигнатурного метода анализа событий. Automation of Control Processes [Автоматизация процессов управления]. 2022. No. 4 (70). P. 41–50. DOI: 10.35752/1991-2927_2022_4_70_41.*
10. *Anand P., Shankar Singh A. Penetration Testing Security Tools: A Comparison // 2021 10th International Conference on System Modeling & Advancement in Research Trends (SMART). IEEE, New York, NY, USA, 2021. P. 182–184. DOI: 10.1109/SMART52563.2021.9676283.*
11. *Foo D., Yeo J., Xiao H. The Dynamics of Software Composition Analysis. ArXiv: abs/1909.00973. 2019. DOI: 10.48550/arXiv.1909.00973.*
12. *Xia B., Bi T., Xing Z., Lu Q., Zhu L. An Empirical Study on Software Bill of Materials: Where We Stand and the Road Ahead // Proceedings of the 45th International Conference on Software Engineering (ICSE '23). 2023. P. 2630–2642. DOI: 10.1109/ICSE48619.2023.00219.*
13. *Sarker I. H. Machine learning: algorithms, real-world applications and research directions // SN Comput. Sci. 2021. Vol. 2, No. 3. Article No. 160. DOI: 10.1007/s42979-021-00592-x.*
14. *Koroteev M. V. [Machine learning basics in Python] Основы машинного обучения на Python. М.: LLC «Publishing House «KnoRus», 2024.*
15. *Zhang Y., Sakhanenko L. The naive Bayes classifier for functional data // Statistics & Probability Letters. 2019. Vol. 152. P. 137–146. DOI: 10.1016/j.spl.2019.04.017.*