УНИВЕРСИТЕТ ИТМО
НАУЧНО-ТЕХНИЧЕСКИЙ ВЕСТНИК ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ
март–апрель 2020 Том 20 № 2 ISSN 2226-1494 http://ntv.itmo.ru/
SCIENTIFIC AND TECHNICAL JOURNAL OF INFORMATION TECHNOLOGIES, MECHANICS AND OPTICS
March–April 2020 Vol. 20 No 2 ISSN 2226-1494 http://ntv.itmo.ru/en/
НАУЧНО-ТЕХНИЧЕСКИЙ ВЕСТНИК ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ

УДК 004.492.3 doi: 10.17586/2226-1494-2020-20-2-233–242

МЕТОДЫ ПОИСКА АНОМАЛЬНЫХ АКТИВНОСТЕЙ ВЕБ-ПРИЛОЖЕНИЙ

О.И. Михеева[a], Ю.А. Гатчин[a], С.В. Савков[b], Р.М. Хамматова[c], А.П. Нырков[d]

[a] Университет ИТМО, Санкт-Петербург, 197101, Российская Федерация
[b] ЗАО «Мобикон», Санкт-Петербург, 199178, Российская Федерация
[c] ООО «Удостоверяющий центр ГАЗИНФОРМСЕРВИС», Санкт-Петербург, 198096, Российская Федерация
[d] Государственный университет морского и речного флота им. адмирала С.О. Макарова, Санкт-Петербург, 198035, Российская Федерация
Автор для переписки: rapppersha@yandex.ru

Информация о статье
Поступила в редакцию 14.02.20, принята к печати 07.03.20
Язык статьи — русский

Ссылка для цитирования: Михеева О.И., Гатчин Ю.А., Савков С.В., Хамматова Р.М., Нырков А.П. Методы поиска аномальных активностей веб-приложений // Научно-технический вестник информационных технологий, механики и оптики. 2020. Т. 20. № 2. С. 233–242. doi: 10.17586/2226-1494-2020-20-2-233-242

Аннотация

Предмет исследования. Представлен обзор существующих методов выявления аномальных активностей веб-приложений. Приведены сравнительные характеристики. Показаны направления совершенствования средств защиты информации в веб-приложениях. **Метод.** Для оценки методов поиска аномальных активностей веб-приложений определены критерии выбора показателей. Особое внимание уделено таким показателям как скорость запуска веб-приложений после загрузки; скорость реакции веб-приложений на действия пользователя; количество найденных аномальных активностей в сравнении с количеством найденных ложных срабатываний. Выполнено сравнение трех методов поиска аномальных активностей: статистического сканирования кода; динамического сканирования кода; мониторинга сетевого трафика. Рассмотрены достоинства и недостатки каждого метода, примеры реализации. **Основные результаты.** Показано, что лучшими характеристиками обладает динамический метод поиска аномальных активностей. Метод позволяет выявлять аномалии, связанные с передачей трафика и аномалии, возникающие в процессе локальной работы веб-приложений. Метод реализуется в виде встроенного в движок браузера анализатора кода. Анализатор проверяет все обращения веб-приложения к движку и выявляет аномальную активность на основе таких обращений. В отличие от статического, динамическое сканирование позволяет определять аномалии в Web Workers, WebAssembly и в частях кода, которые загружаются по сети после запуска приложения. **Практическая значимость.** Работа может быть полезна специалистам по информационной безопасности, которые занимаются проблемами защиты веб-приложений, а также программистам и системным администраторам на этапе создания и внедрения приложений. Итоги работы могут найти применение при разработке веб-приложений, браузеров, программного обеспечения для защиты информации.

Ключевые слова
аномальная активность, браузер, браузерный движок, веб-приложения, движок JavaScript, анализ трафика между клиентом и сервером, статический анализ кода, динамический анализ кода, поиск аномальных активностей

doi: 10.17586/2226-1494-2020-20-2-233-242

SEARCH METHODS FOR ABNORMAL ACTIVITIES OF WEB APPLICATIONS

O.I. Mikheeva[a], Yu.A. Gatchin[a], S.V. Savkov[b], R.M. Khammatova[c], A.P. Nyrkov[d]

[a] ITMO University, Saint Petersburg, 197101, Russian Federation
[b] ZAO Mobicon, Saint Petersburg, 199178, Russian Federation
[c] LLC Certification Authority Gazinformservice, Saint Petersburg,198096, Russian Federation
[d] Admiral Makarov State University of Maritime and Inland Shipping, Saint Petersburg, 198035, Russian Federation
Corresponding author: rapppersha@yandex.ru

Article info
Received 14.02.20, accepted 07.03.20
Article in Russian

For citation: Mikheeva O.I., Gatchin Yu.A., Savkov S.V., Khammatova R.M., Nyrkov A.P. Search methods for abnormal activities of web applications. *Scientific and Technical Journal of Information Technologies, Mechanics and Optics,* 2020, vol. 20, no. 2, pp. 233–242 (in Russian). doi: 10.17586/2226-1494-2020-20-2-233-242

Abstract

Subject of Research. The paper presents a review of existing detection methods for abnormal activities of web applications. Comparative characteristics are given. Priorities for improving information security tools in web applications are shown. **Method.** For evaluation of search methods for abnormal activities of web applications, criteria for selecting indicators were defined. Particular attention was paid to such indicators as: the launching speed of web applications after loading, web application responsiveness to user actions and the number of abnormal activities found in comparison with the number of malfunctions found. Three methods of searching for abnormal activities were compared: statistical code scanning, dynamic code scanning and network traffic monitoring. We considered advantages and disadvantages of each method and implementation examples. **Main Results.** It is shown that the dynamic method of searching for abnormal activities has the best characteristics. The method provides the identification of anomalies associated with traffic transfer and anomalies that occur during the local operation of web applications. The method is implemented as a code analyzer built into the browser engine. The analyzer checks all calls of the web application to the engine and detects abnormal activity based on such calls. In contrast to static scanning, dynamic scanning identifies anomalies in Web Workers, WebAssembly and in the parts of code that are downloaded over the network after the application starts. **Practical Relevance.** The work can be useful to information security specialists who deal with the problems of protecting web applications, as well as programmers and system administrators at application creation and implementation stage. The results of the work can find practical use in the development of web applications, browsers, and information protection software.

Keywords

abnormal activity, browser, browser engine, web applications, JavaScript engine, traffic analysis between client and server, static code analysis, dynamic code analysis, search for abnormal activities

Введение

Веб-приложения начали использоваться в конце 1990-х — начале 2000-х годов и позволяют клиентам взаимодействовать с веб-сервером при помощи браузера. Со временем они получили широкое распространение в таких важных сферах как государственные услуги, банки, электростанции. Веб-приложения используются пользователями для доступа к большому массиву информации.

Технологии веб-приложений развиваются большими темпами. JavaScript, практически не использующийся в сайтах 20 лет назад, превратился в мощнейший язык программирования с широкой инфраструктурой разработки. Применение технологии WebAssembly привело к тому, что в сфере веб-приложений появились модули, написанные на различных языках программирования. Современные веб-приложения являются асинхронными и используют технологии Web Workers, с помощью которых код может выполняться в фоновом режиме.

В связи с активной разработкой технологии веб-приложений и растущим их количеством остро встает проблема защиты информации. Большое количество веб-приложений не проходит достаточный контроль со стороны разработчиков и выпускается в недоработанном виде на рынок. Пользователи используют его, зачастую не предполагая проблем, с которыми они столкнутся. По данным экспертов, 90 % веб-приложений подвержены угрозе атак на клиентов, в 9 случаях из 10 злоумышленники могут атаковать посетителей сайта, 16 % приложений содержат уязвимости, позволяющие получить полный контроль над системой, а в 8 % случаев — атаковать внутреннюю сеть компании[1,2].

Целью данной работы является проведение сравнительного анализа существующих методов поиска аномальной активности в веб-приложениях. Это необходимо для понимания происходящих в данной среде изменений и путей развития средств защиты информации в веб-приложениях.

Известны попытки предоставить методы анализа кода стандартных приложений на аномальные активности, однако эти методы не полностью подходят к веб-приложениям. Связано это с тем, что в случае классических приложений работа ведется с исходным кодом. В случае веб-приложений работа происходит только с кодом клиента, который сложно анализировать в силу специфики области применения.

Эта специфика заключается в автоматическом обновлении кода и широком использовании сетевых соединений. Код изменяется по мере использования приложения, в результате чего появляется необходимость в реальном времени предупреждать пользователя о возможном аномальном поведении браузера.

Сложность решения подобной задачи состоит в том, что сфера применения веб-приложений еще интенсивно развивается, в результате чего приходится постоянно использовать методы, применяемые для классических приложений. Они часто оказываются недостаточно эффективными в веб-приложениях с учетом постоянного сетевого обмена информацией и наличия динамического окружения [1–4][3,4].

[1] Positive Technologies: 82 % уязвимостей веб-приложений содержится в исходном коде [Электронный ресурс]. 2020 Режим доступа: http://www.itsec.ru/news/positive-technologies-82-uyazvimostey-veb-prilozheniy-soderzhitsia-v-ishodnom-kode (дата обращения: 02.03.2020).

[2] Введение в тему безопасности веб-приложений [Электронный ресурс]. 2011 Режим доступа: https://www.ptsecurity.com/ru-ru/research/webinar/1487/ (дата обращения: 02.03.2020).

[3] Types of XSS: Stored XSS, Reflected XSS and DOM-based XSS // Acunetix Blog [Электронный ресурс]. Режим доступа: https://www.acunetix.com/websitesecurity/xss/ (дата обращения: 22.01.2020).

[4] Berners-Lee T. Uniform Resource Locators // RFC 1738 – IETF [Электронный ресурс]. Режим доступа: https://www.ietf.org/rfc/rfc1738.txt (дата обращения: 23.01.2020).

Специфика веб-приложений

Веб-приложения доставляются посредством браузера, выполняются в браузере, имеют доступ к основным возможностям операционной системы через браузер, а также завершают свое выполнение в браузере и производят автоматическое обновление своих частей через браузер. Браузер при этом входит в состав операционной системы и представлен в виде широкой разновидности приложений. Главное, что объединяет все эти приложения в класс браузеров, — это выполнение определенных стандартов консорциума всемирной компьютерной сети [5][1,2].

Каждый браузер имеет в своем составе так называемый браузерный движок (layout engine). Он состоит из движков для выполнения программного кода, парсинга структуры CSS (Cascading Style Sheets — каскадные таблицы стилей) и HTML (HyperText Markup Language — язык гипертекстовой разметки), циклического движка для рендера и вспомогательных библиотек для доступа к функциям операционной системы (рис. 1). На этом движке выполняется основной код веб-приложения. Необходимо обратить внимание на построитель DOM-выражений (Document Object Model — объектная модель документа), так как здесь злоумышленники с помощью стороннего кода производят подмену элементов, обеспечивая незаметный для пользователя запуск аномальной активности. Точно также вызывает интерес возможность доступа через слой API (Application Programming Interface — интерфейс взаимодействия между сайтом и сторонними программами и серверами) к функциям операционной системы, т. е. возможность перехватить управление компьютером [6].

Основными частями современного браузера (рис. 1) являются такие компоненты, как Networking — компонент операционной системы для работы с сетью; Operating System — API-слой операционной системы; Resource Loader — компонент загрузки ресурсов, таких как сокеты, файлы, пайпы; Plugin Layer — компонент, обеспечивающий доступ к внешним плагинам для выполнения операций, например, управление потоком в WebSocket или работа с видеопотоком; CSS Parser — парсер CSS-выражений; HTML Parser — парсер HTML-выражений; JavaScript Engine — движок, который выполняет JavaScript-выражения; DOM Builder — билдер DOM-выражений для создания структуры страницы; Render Engine — движок, который осуществляет рендер и последующее обновление элементов страницы; User Interface — интерфейс веб-приложения поверх браузера.

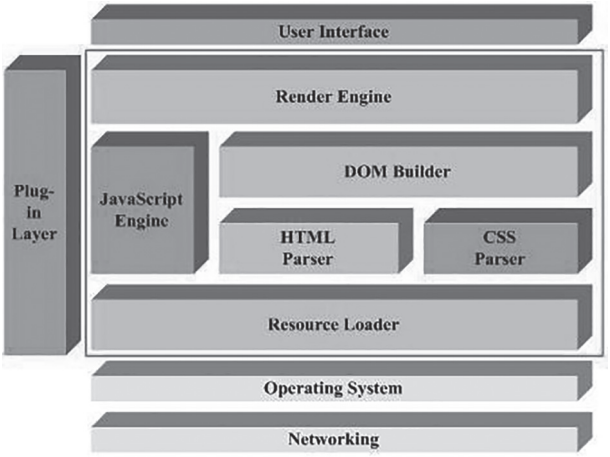

Рис. 1. Структура движка браузера[3]

Выделим такие движки для вывода веб-страниц:
- Blink (компания Google) — представлен в браузерах Google Chrome, Microsoft Edge, Opera, в многочисленных open-source проектах;
- Gecko (компания Mozilla) — представлен в браузерах Firefox;
- WebKit (компания Apple) — представлен в браузере Safari и в операционных системах Mac OS X, IOS.

Внутри браузерного движка находится движок JavaScript, который необходим для выполнения встроенных программ на языке программирования JavaScript. Для представляемого исследования он является наиболее важной частью, поскольку большая часть кода написана и исполняется именно на этом движке [7].

Основные движки JavaScript на данный момент:
- V8 (компания Google) — используется в браузерах Chrome;
- Chakra (компания Microsoft) — используется в браузерах Edge;
- Rhino (компания Mozilla) — используется в браузерах Firefox;
- SquirrelFish (компания Apple) — используется в браузерах Safari.

Перед непосредственным выполнением код проходит определенный поток преобразований. Это представляет интерес по той причине, что на каждом из этапов компиляции и оптимизации движком система защиты и злоумышленный код могут встроить свои конструкции, изначально не обнаруживаемые в исходном коде. Сначала движок JavaScript трансформирует код в абстрактное синтаксическое дерево (AST). Далее, основываясь на AST, начинается выполнение кода интерпретатором, на выходе которого получается не оптимизируемый байт-код. Его может анализировать система защиты.

В силу этого традиционные методы анализа кода постоянно сталкиваются с проблемами. В ходе работы на основе собранных профилирующих данных оптимизирующий компилятор генерирует более оптимизи-

[1] Cross-frame-scripting // OWASP the free and open software security community [Электронный ресурс]. Режим доступа: https://www.owasp.org/index.php/Cross_Frame_Scripting (дата обращения: 24.01.2020).

[2] Как хакеры атакуют веб-приложения: боты и простые уязвимости. Блог компании Positive Technologies [Электронный ресурс]. 25.04.2017. Режим доступа: https://habr.com/ru/company/pt/blog/327344/ (дата обращения: 24.01.2020).

[3] Язык JavaScript (введение) 2015 [Электронный ресурс]. Режим доступа: https://en.ppt-online.org/84862 (дата обращения: 01.02.2020).

рованный машинный код, из-за чего браузер способен выполнять код быстрее. Интерпретатор в распространенном движке V8 называется Ignition (зажигание), а оптимизирующий компилятор — Turbofan (вентилятор). Таким образом, представляет интерес анализ кода в двух точках выполнения, поэтому структура выполнения веб-приложений является достаточно сложной. При этом на все ее элементы может оказываться хакерское воздействие. Например, могут подменяться одни элементы другими, чтобы заставить пользователя выполнять непредумышленные действия.

Для всех движков и инфраструктуры выполнения кода общими являются, с точки зрения защиты информации, следующие отличительные черты:

— выполняемый код приходит в открытом виде и проходит несколько этапов компиляции специальной программой и оптимизации в процессе исполнения;
— движки браузеров постоянно добавляют новые технологии, значительно отличающиеся в поведении друг от друга, хотя и следуют общим рекомендациям консорциума всемирной паутины;
— для реализации многопоточного выполнения кода используется цикл событий Event Loop, посредством которого можно как зациклить выполнение кода, так и получить доступ к критически важной информации через прерывания [8];
— большое количество выполняющегося кода в фоне, особенно с приходом технологии Web Workers, создает проблемы для точного понимания текущей ситуации в веб-приложении [9, 10][1];
— достаточно частое общение с сервером посредством сетевого доступа осложняется широким выбором технологий для доступа к серверу, а также различными publisher-subscriber (шаблоны проектирования приложения, в которых отправители сообщений напрямую не привязаны к подписчикам, а производится обмен сообщениями через технологии подписки) технологиями;
— секретные данные пользователя (банковские карты, персональные данные, пароли, служебная информация) изначально передавались в открытом виде, сейчас же большая часть такой информации передается в шифрованном виде посредством HTTPS-соединения, но внутри клиента она находится в открытом виде [11];
— различные офлайн-режимы вынуждены накапливать секретную информацию пользователя и передавать ее при первом появлении связи на сервер.

Реализация угроз аномальных активностей, которые могут реализовываться во время выполнения кода веб-приложений в браузере, может привести как к небольшому дискомфорту с точки зрения использования приложений, так и к значительному ущербу в области защиты информации. Виды таких угроз весьма различны.

— **Нарушение конфиденциальности информации.** Конфиденциальность информации можно нарушить двумя способами:
  1) передача закрытой информации пользователя, полученной во время выполнения кода, на сервер злоумышленника;
  2) сбор статистики о совершаемых действиях пользователя, что позволяет злоумышленнику следить за пользователем.
— **Нарушение целостности информации в веб-приложениях.** В данном случае проблема также может возникнуть в виде реализации двух угроз:
  1) подмена страниц веб-приложения на страницы злоумышленника или непосредственная замена всей страницы, отдельных элементов страницы или выполняемой функциональности;
  2) распространение вредоносного программного обеспечения в виде вирусов, троянов, установка их на компьютер через веб-приложения [12, 13].
— **Выполнение вредоносных действий от имени пользователя.** Это действие может привести к таким действиям через веб-приложения, как удаление содержимого от имени пользователя или участие устройства пользователя в атаках сети, спам-атаках, а также кибератаках [14, 15].

Можно выделить основные методы поиска аномальных активностей, применимые к веб-приложениям. К ним относятся: мониторинг трафика между браузером и внешней сетью; статическое сканирование кода; динамическое сканирование исполняемого кода на этапе выполнения.

В каждом методе поиска могут использоваться в качестве дополнительных такие средства как сбор статистики по коду, анализ профилирующих данных, мониторинг вызовов API-функций операционной системы, а также снимков DOM-дерева браузера в разное время [16].

С точки зрения пользователя веб-приложений при сравнении методов поиска аномальной активности имеет смысл рассматривать определенные критерии их применимости. Важным моментом выбора того или иного метода является возможность применять средства защиты, не возвращаясь к использованию веб-приложений без каких-либо элементов защиты.

Для оценки методов поиска аномальных активностей веб-приложений выберем такие показатели как:

— падение скорости запуска веб-приложений после загрузки – разница времени между первым запуском «чистого» веб-приложения и веб-приложения с включенной системой защиты;
— падение скорости реакции веб-приложений на действия пользователя;
— сравнение количества найденных аномальных активностей с количеством найденных ложных срабатываний.

Мониторинг трафика между браузером и внешней сетью

Подобный способ поиска аномальных активностей заключается в использовании традиционного для

[1] Двухфакторная аутентификация: что это и зачем оно нужно? Блог Лаборатории Касперского [Электронный ресурс]. Режим доступа: https://www.kaspersky.ru/blog/what_is_two_factor_authenticatio/4272/ (дата обращения: 01.02.2020).

анализа сетевого трафика решения. Устанавливается прокси-сервер, который просматривает и анализирует трафик между веб-приложением и внешней сетью. Мониторинг выполняется посредством анализа содержимого в HTTP-пакете, анализа адресов источника и назначения, а также портов. Для анализа шифрованного трафика, передающегося через SSL-соединение (соединение Secure Sockets Layer — соединение, использующее уровень защищенных сокетов), применяется прокси, который либо встраивается в схему SSL-соединения, либо является его исходной точкой. Определение аномальной активности осуществляется через поиск в HTTP-пакетах определенных сигнатур, свидетельствующих об отклонении, либо использование черных списков для поиска конкретных хостов злоумышленников [17–20]. В качестве вспомогательного средства производится мониторинг API-вызовов операционной системы. В ряде случаев это может давать определенные результаты.

В виде примеров таких систем можно привести персональные антивирусные системы со встроенным в них модулем анализа HTTP-трафика: Flowmon Networks, OWASP ZAP, Kaspersky Internet Security, C-Teppa COB, Qualys Web Application Scanner.

Преимущества подобного способа поиска аномальных активностей заключаются в более быстром анализе трафика при поиске уязвимостей. При этом практически не снижается производительность веб-приложения. На данный метод не оказывает влияние обфускация кода, он позволяет находить большинство традиционных сетевых атак, таких как подключение веб-приложения в бот-сеть, обращение к другим хостам, не имеющим отношение к работе данного веб-приложения [21].

К сожалению, данный метод не позволяет выявлять аномальную активность, не связанную с передачей трафика. Например, известная атака, когда одни элементы подменяются другими для того, чтобы пользователь выполнил авторизационное действие. Само действие не является аномальным, поэтому выделить его как аномальное практически невозможно [22]. Также он плохо определяет большинство аномальных активностей, специфичных для веб-приложений, например, воровство cookies и XSS-атаки.

Статическое сканирование кода

Статическое сканирование кода выполняется перед тем, как очередная порция программного кода веб-приложения попадает в движок браузера. Во время статического кода происходит анализ приложения, выявление возможных аномальных активностей [23].

Такой поиск осуществляется по известным базам сигнатур. Также происходит поиск в коде вызовов к операционной системе для обращения к различным авторизованным действиям, что является непосредственным признаком веб-приложения. Код анализируется по базам данных уязвимостей, доступным еще в традиционных языках программирования [24].

Однако в современных реалиях при экспоненциальном росте компьютерных технологий затруднено создание полноценной базы сигнатур аномальных активностей веб-приложений, что не позволяет с достаточной эффективностью выявлять их аномальное поведение [25]. Фактически приходится постоянно находить компромисс между анализом кода, его расширяемостью и быстротой исполнения, а это влияет на эффективность поиска аномальных активностей [26, 27].

При статическом сканировании на входе принимается исходный код, а на выходе выдается информация о найденных аномальных активностях. Метод ограничен теоретической проблемой, сформулированной в теореме Райса.

Согласно теории алгоритмов, «для любого свойства, для которого существуют вычислимые функции с этим свойством и вычислимые функции без этого свойства, определить, является ли произвольный алгоритм функцией с таким свойством, является неразрешимой задачей».

Иначе говоря, невозможно сказать, исходя из исходного произвольного кода, все ли проблемы безопасности, которые появятся в процессе его выполнения, будут найдены.

Статический анализатор плохо обнаруживает аномальные активности, использующие нулевой указатель исключения, или активности, связанные с передачей управления коду, полученному из стека. Все, что может статический анализатор – выделять частные случаи, но при этом проверка на аномальную активность все равно должна проводиться другим методом.

Статический анализ может производиться как на основе заранее заготовленных шаблонов аномалий, так и на основании потока выполнения кода, где в зависимости от определенных прыжков и условий можно распознать аномальную активность. Также существует статистический подход, основанный на анализе метрик — аномальная активность выявляется по статистическим показателям.

Прекрасным примером применения данных подходов является инструмент Mashup Developer Tool (рис. 2), в котором, с одной стороны, по определенным шаблонам анализируется код, с другой стороны — производится анализ потока выполнения, а с третьей стороны — происходит сбор метрик по внешнему обращению к сети. Данные анализируются, выявляются различные аномальные активности и о них сообщается пользователю.

На рис. 2 показан процесс статического анализа кода (System Workflow) с помощью инструмента Mashup Developer Tool. В начале процесса пользователь (Mashup Developer) использует три источника для анализа приложения: External JavaScript Library URLs (внешние JavaScript URL, которые разрешено использовать в коде); Mashup Application (приложение для анализа); Trust Policy (авторизационные правила, которые применяются для анализа приложения). С помощью этих источников происходит оценка приложения (Policy Adherence Assessment). В ходе оценки код подвергается статическому анализу (Static Code Analysis). Выявленные попытки неразрешенных обращений к другим сайтам (Dynamic Code Injection Sites) и нарушения созданных пользователем политик доступа приложения к API (Potential Trust Violation Sites) со-

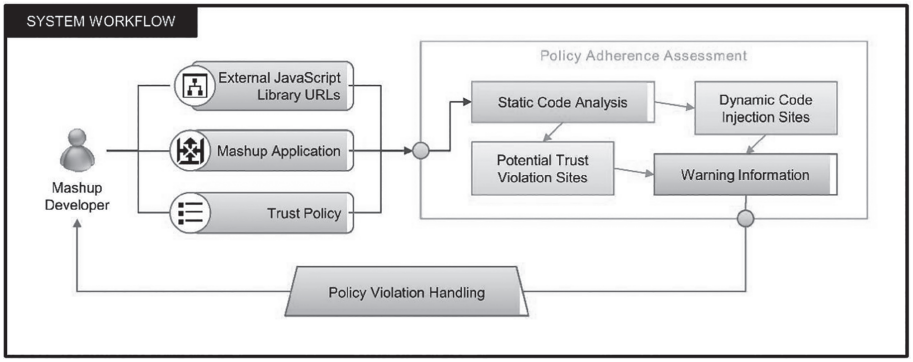

Рис. 2. Статический анализ кода на практике Mashup Developer Tool [28]

ставляют аномальные активности. Информация о них в виде предупреждений (Warning Information) передается через обработку связанных правил (Policy Violation Handling) обратно к пользователю.

Наиболее применимый способ на данный момент — проверка по базе шаблонов. Другой способ — более сложный, заключается в попытке предсказать, каким образом будет выполняться программа, зачастую имеет значительный показатель ложных срабатываний. Анализ таких метрик как количество строк кода, а также вызовов сторонних функций мало эффективен и используется лишь в виде дополнительного средства анализа кода [29].

С 2014 года компания Mozilla пытается построить статическое средство анализа кода, в том числе с точки зрения безопасности пользователя, ScanJS. Однако результаты внедрения этого способа показали его неэффективность, и Mozilla отказалась от дальнейшей разработки[1].

С точки зрения конкретного пользователя статический анализ кода неэффективен. Время ожидания проверки и возможная модификация исполнения кода после сетевого взаимодействия осложняют практику применения данного метода.

Примерами практической реализации данного метода поиска уязвимостей являются платформы: NodeJsScan, EsLint, Node Security Platform, Retire.js, IBM Security AppScan Source, Checkmarx CxSAST, WhiteHat Security.

Линтеры представляют собой важнейший инструмент для разработчиков программного обеспечения. Они позволяют анализировать код на различные аномальности в процессе разработки и особенно при выпуске программного обеспечения. Одним из наиболее признанных инструментов статического анализа кода для выявления проблемных шаблонов, обнаруженных в коде JavaScript, является EsLint. Однако, с точки зрения пользователя веб-приложения, такой метод недостаточно эффективен ввиду неспособности находить динамически изменяющийся код и его влияние при первом запуске на производительность веб-приложения.

[1] The importance of web application scanning // Важность сканирования веб-приложений. 2019 [Электронный ресурс]. Режим доступа: https://www.acunetix.com/websitesecurity/the-importance-of-web-application-scanning/ (дата обращения: 22.01.2020).

Преимущество данного метода заключается в том, что он выполняется только один раз при загрузке кода. В последующем ему достаточно проверять только сохраненные контрольные суммы и сравнивать их с текущими. Такой же метод позволяет доверять серверным базам контрольные суммы уже проверенных файлов. В сравнении с сетевым мониторингом метод позволяет выявлять больше аномальных активностей кода.

Проблема применения подобного метода заключается в большом разнообразии веб-приложений и их архитектур, в результате чего построение базы сигнатур и аномальных активностей затруднено. Кроме того, быстрое развитие технологий и постоянное внедрение новых архитектурных стандартов делает невозможным достижение достаточной скорости нахождения сетевых аномалий в коде. Плохо находятся аномальные активности в обфусцированном и самозагружающемся коде. Речь идет о таком довольно распространенном способе построения приложения, когда код приложения достаточно быстро перестраивается и становится отличным от того, каким был на момент загрузки. Дополнительную сложность создают технологии WebAssembly, поскольку проанализировать подключаемый код на этапе предварительной компиляции становится невозможным [30–32].

Динамическое сканирование исполняемого кода на этапе выполнения

Динамическое сканирование исполняемого кода подразумевает постоянное помодульное сканирование в момент выполнения инструкции браузерным движком и движком JavaScript. Анализ может производиться в моменты выполнения модулей Ignition и TurboFan. Первый служит для процесса компиляции кода, второй обеспечивает процесс оптимизации кода (рис. 3).

JavaScript-код в процессе выполнения сначала проходит этап интерпретации кода (Interpreted). Как исходный файл (JavaScript Source) он проходит через движок компиляции (Ignition). В процессе формируется низкоуровневый байт-код (Bytecode). После этого начинает работать компилятор первого уровня (Full-codegen), который производит код для непосредственного выполнения в браузере, но не оптимизированный (Unoptimized code). Это второй этап компиляции кода, после которого начинает работать веб-приложение (Baseline). На треть-

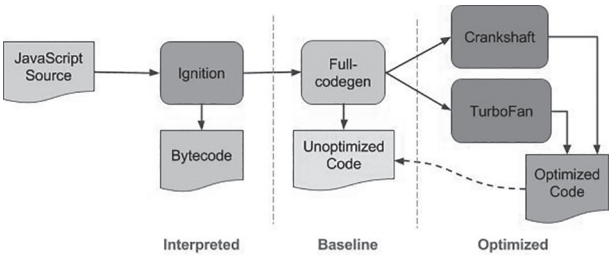

Рис. 3. Схема потока выполнения JavaScript-кода[1]

ем этапе код анализируется браузером с целью ускорения выполнения. Этот этап называется оптимизацией кода (Optimized). Код на данном этапе попадает сразу в два разных оптимизирующих компилятора: TurboFan и Crankshaft. Разница между ними заключается в том, что TurboFan перегенерирует оптимизированный машинный код, а Crankshaft пользуется для оптимизации исходным кодом. В результате получается новый оптимизированный скомпилированный машинный код JavaScript (Optimized Code), который со временем теряет свою актуальность. Тогда оптимизация проходит заново.

В обоих модулях имеется возможность анализа как бинарного кода, что довольно затруднительно, так и кода, получаемого на языке C, а также возможность применять анализ JavaScript-кода на входе в JavaScript-движок. С точки зрения разработчика программного обеспечения есть возможность запускать код в браузере, подключать систему динамического сканирования и получать результаты после того, как код отработает. Тем самым гарантируется, что полученные результаты сканирования подобны полученным пользователем. После проверки программному обеспечению присваивается определенная категория надежности. В итоге можно обеспечить информирование пользователя об уровне безопасности через контрольную сумму и категорию безопасности. Однако для пользователя данный способ проверки затратен из-за большого времени выполнения [33–35].

Такой метод на практике предоставляет информацию о времени выполнения дерева вызова функции, актуальных типах данных объектов и переменных, неявных преобразованиях типов, а также деструктуризации переменных [36]. Метод работает при встраивании в код таким образом, чтобы возможность получения записи всех происходящих событий не меняла логику программы и схему ее выполнения. Он добавляет свои функции наблюдателя во все события для последующего отслеживания и реакции на действия. Поведение кода можно отследить в режиме выполнения, при встраивании в определенные моменты с помощью перехватов глобальных событий и идентификаторов. Вместе с тем это требует настройку браузера для выполнения или наличия движка JavaScript.

Поскольку динамический анализ основан на анализе входного набора данных, его эффективность зависит от качества и количества входных данных, а также зависит от способа анализа. Входными данными могут быть такие метрики, как:

— время исполнения программы и ее отдельных модулей, количество внешних запросов, количество используемой памяти и др.;
— цикломатическая сложность кода программы, цикломатическая динамическая сложность (т. е. количество ветвей в программе по мере выполнения);
— возникающие программные ошибки и доступ за пределами выделенной памяти и стека;
— объем передаваемых данных и объем загруженного кода для исполнения.

Динамическое сканирование применяется как последнее средство перед выполнением кода в браузере или используется для анализа кода после зафиксированной аномальной активности. И в том, и в другом случае динамическое сканирование затрудняет работу пользователя с веб-приложением. С другой стороны, этот вид сканирования кода наиболее приближен к пользователю и представляет полную информацию о выполняемом коде. В ряде случаев в ходе динамического сканирования используют снимки состояния браузера при выполнении кода, полученные или от разработчика, или от предыдущих запусков веб-приложения. Эти снимки позволяют более эффективно разделять нормальную и аномальную активность, а также обходить такие сложные случаи, как высоко оптимизированный код в WebAssembly, работающий с OpenGl, и напрямую с видеокартой.

У динамического анализа кода есть неоспоримое преимущество — точное знание, какая функция и какими аргументами вызвана. Также можно проверить корректность вызова, но это затруднено тем, на каком этапе в потоке выполнения JavaScript-движка будет осуществляться этот анализ. Если динамический анализ будет выполнен на этапе не оптимизируемого интерпретатора, то можно будет упустить из рассмотрения функции и аргументы, появившиеся после запуска оптимизирующего компилятора.

С другой стороны, если код будет проанализирован после оптимизации, существует риск не увидеть всю картину анализа кода и не разобраться, какую непосредственную роль в обработке события играет данная функция. При статическом анализе это утверждать с уверенностью невозможно, но картина анализа кода получается более высокоуровневой.

Инструментами для поиска аномальных активностей с применением метода динамического сканирования служат: Iroh.js, Clinic.js, PVS-Studio.

Достоинства данного метода состоят в том, что он разрешает выполнять анализ и манипуляцию кода в процессе выполнения, при этом нет необходимости модифицировать исходный код или снабжать его аннотациями как в случае со средствами статической типизации. Способствует выявить аномальную активность до ее непосредственного выполнения и тем самым предотвратить злоумышленное действие. Разные примеры подобного подхода, такие как онлайн IDE (Integrated Development Environment — интегрированная среда разработки), позволяют находить аномальные актив-

[1] JavaScript V8 Engine Explained // Объяснение работы движка Javascript V8. 2019. [Электронный ресурс]. Режим доступа: https://hackernoon.com/javascript-v8-engine-explained-3f940148d4ef (дата обращения: 02.03.2020).

ности в ходе редактирования кода. В большинстве реализаций появление ложных срабатываний исключено, так как обнаружение ошибки происходит в момент ее возникновения в программе. Таким образом, обнаруженная ошибка является не предсказанием, сделанным на основе анализа модели программы, а констатацией факта ее возникновения [37].

Недостатки данного метода поиска аномальной активности заключаются в том, что он является экспериментальным, и его применение сопряжено с проблемами обеспечения производительности. Его эффективность по сравнению с другими методами довольно низкая, база данных сигнатур растет быстро, и поиск новых сигнатур затруднен постоянным развитием сферы веб-приложений. Проблема состоит в том, что, в отличие от компилируемых языков, объектные сигнатуры интерпретируемого языка JavaScript и DOM-движка еще не разработаны достаточно для продуктивного анализа.

Заключение

Наиболее эффективным методом анализа веб-приложений является динамический способ анализа кода, реализуемый в виде встроенного в движок браузера анализатора кода, который проверяет все обращения веб-приложения к движку и выявляет аномальную активность на основе таких обращений. Из-за неизбежного снижения производительности браузера и необходимости его постоянного адаптирования под конкретные веб-приложения подобный метод так и не был доработан в полноценное коммерческое решение. Вместе с тем потенциально этот метод позволяет выявлять с большой точностью активности, не являющиеся обычными для веб-приложений.

На сегодняшний день защиту веб-приложений выполняет сам браузер, предоставляя возможность создания и выполнения различных движков. При этом затруднен доступ к нативным функциям операционной системы. Использование Secure Sockets Layer соединения обеспечивает надежную защиту от подделки кода и атак «человек посередине». Тем самым получается код, переданный разработчиком, и этот код имеет доступ не ко всем функциям операционной системы.

Таким образом, динамический метод анализа кода является самым перспективным способом анализа аномальных активностей, и необходим дальнейший более глубокий анализ этого метода с точки зрения производительности и выявления необходимых аномалий [38–40].

Литература

1. Низамутдинов М.Ф. Тактика защиты и нападения на Web-приложения. СПб.: БХВ-Петербург, 2005. 432 с.
2. Зайцев А.С., Малюк А.А. Разработка классификации внутренних угроз информационной безопасности посредством кластеризации инцидентов // Безопасность информационных технологий. 2016. Т. 23. № 3. С. 20–33.
3. Garin E.V., Meshcheryakov R.V. Method for determination of the social graph orientation by the analysis of the vertices valence in the connectivity component // Вестник Южно-Уральского государственного университета. Серия: Математика. Механика. Физика. 2017. Т. 9. № 4. С. 5–12. doi: 10.14529/mmph170401
4. Stuttard D., Pinto M. The Web Application Hacker's Handbook: Discovering and Exploiting Security Flaws. John Wiley & Sons, 2011. 912 p.
5. Мейксин С.М. Безопасность банков // Вестник науки и образования. 2019. № 4-2(58). С. 53–55 [Электронный ресурс]. URL: https://cyberleninka.ru/article/n/bezopasnost-bankov (дата обращения: 23.02.2020).
6. Сукиасян В.М., Придиус Е.С. Современные принципы и подходы к frontend архитектуре веб-приложений // Наука, техника и образование. 2019. № 10(63). С. 54–57.
7. Томилов И.О., Трифанов А.В. Фаззинг. Поиск уязвимостей в программном обеспечении без наличия исходного кода // Интерэкспо ГЕО-Сибирь. 2017. Т. 9. № 2. С. 75–80.
8. Мельников В.Г., Гребень А.Е., Макарова Д.Г. Исследование межсетевых экранов для веб-приложений с открытым исходным кодом // Интерэкспо ГЕО-Сибирь. 2018. № 7. С. 233–236.
9. Мельников В.Г., Трифанов А.В. Методы обхода межсетевых экранов для приложений // Интерэкспо ГЕО-Сибирь. 2017. Т. 9. № 2. С. 113–117.
10. Семенова З.В., Данилова О.Т., Ковшарь И.Р. Анализ безопасности стека технологий для разработки web-ресурсов // Динамика систем, механизмов и машин. 2019. Т. 7. № 4. С. 98–105 [Электронный ресурс]. URL: https://cyberleninka.ru/article/n/analiz-bezopasnosti-steka-tehnologiy-dlya-razrabotki-web-resursov (дата обращения: 23.02.2020).
11. Fogie S., Grossman J., Hansen R., Rager A., Petkov P. XSS Attacks: Cross Site Scripting Exploits and Defense. Oxford: Elsevier, 2007. 448 p.

References

1. Nizamutdinov M.F. *Tactics of Defense and Attack on Web Applications*. St.Petersburg, BHV Publ., 2005, 432 p. (in Russian)
2. Zaytsev A.S., Malyuk A.A. Development of information security insider threat classification using incident clustering. *IT Security*, 2016, vol. 23, no. 3, pp. 20–33. (in Russian)
3. Garin E.V., Meshcheryakov R.V. Method for determination of the social graph orientation by the analysis of the vertices valence in the connectivity component. *Bulletin of the South Ural State University. Series: Mathematics. Mechanics. Physics*, 2017, vol. 9, no. 4, pp. 5–12. doi: 10.14529/mmph170401
4. Stuttard D., Pinto M. *The Web Application Hacker's Handbook: Discovering and Exploiting Security Flaws*. Wiley Publ., 2011, 912 p.
5. Meyksin S.M. Bank security. *Bulletin of Science and Education*, 2019, no. 4-2(58), pp. 53–55. Available at: https://cyberleninka.ru/article/n/bezopasnost-bankov (accessed: 23.02.2020). (in Russian)
6. Sukiasyan V.M., Pridius E.S. Contemporary principles and approaches to frontend web architecture. *Science, Technology and Education*, 2019, no 10(63). pp. 54–57. (in Russian)
7. Tomilov I.O., Trifanov A.V. Fuzzing. Gray-box method. *Interexpo GEO-Siberia*, 2017, vol. 9, no. 2, pp. 75–80. (in Russian)
8. Melnikov V.G., Greben A.E., Makarova D.G. Investigation of open source firewalls for web applications. *Interexpo GEO-Siberia*, 2018, no. 7, pp. 233–236. (in Russian)
9. Melnikov V.G., Trifanov A.V. WAF bypass. *Interexpo GEO-Siberia*, 2017, vol. 9, no. 2, pp. 113–117. (in Russian)
10. Semenova Z.V., Danilova O.T., Kovshar I.R. The analysis of security of a stack of technologies for development of web-resources. *Dynamics of Systems, Mechanisms and Machines*, 2019, vol. 7, no. 4, pp. 98–105. Available at: https://cyberleninka.ru/article/n/analiz-bezopasnosti-steka-tehnologiy-dlya-razrabotki-web-resursov (accessed: 23.02.2020). (in Russian)
11. Fogie S., Grossman J., Hansen R., Rager A., Petkov P. *XSS Attacks: Cross Site Scripting Exploits and Defense*, Oxford, Elsevier, 2007, 448 p.
12. Alcorn W., Frichot Ch., Orrù M. *The Browser Hacker's Handbook*. John Wiley & Sons, 2014, 648 p.
13. Cross M. *Developer's guide to web application security*. Elsevier, 2007, 500 p. doi: 10.1016/B978-1-59749-061-0.X5000-1

12. Alcorn W., Frichot Ch., Orrù M. The Browser Hacker's Handbook. John Wiley & Sons, 2014. 648 p.
13. Cross M. Developer's guide to web application security. Elsevier, 2007. 500 p. doi: 10.1016/B978-1-59749-061-0.X5000-1
14. Беляев А., Петренко С. Системы обнаружения аномалий: новые идеи в защите информации [Электронный ресурс]. URL: http://citforum.ru/security/articles/anomalis/ (дата обращения: 14.02.2020).
15. Цыганенко Н.П. Статический анализ кода мобильных приложений как средство выявления его уязвимостей // Труды БГТУ. Физико-математические науки и информатика. 2015. № 6(179). С. 200–203.
16. Марков А.С., Матвеев В.А., Фадин А.А., Цирлов В.Л. Эвристический анализ безопасности программного кода // Вестник МГТУ им. Н.Э. Баумана. Серия: Приборостроение. 2016. № 1. С. 98–111.
17. Иконников М.А., Карманов И.Н. Меры и требования к защищенным веб-приложениям // Интерэкспо Гео-Сибирь. 2019. Т. 6. № 2. С. 13–19. doi: 10.33764/2618-981X-2019-6-2-13-19
18. Барабанов А.В., Лавров А.И., Марков А.С., Полотнянщиков И.А. Исследование атак типа «межсайтовая подделка запросов» // Вопросы кибербезопасности. 2016. № 5. С 43–49. doi: 10.21581/2311-3456-2016-5-43-50.
19. Барабанов А.В., Марков А.С., Фадин А.А., Цирлов В.Л. Статистика выявления уязвимостей программного обеспечения при проведении сертификационных испытаний // Вопросы кибербезопасности. 2017. № 2. С 2–8. doi: 10.21581/2311-3456-2017-2-2-8
20. OWASP TOP 10 — 2017. The Ten Most Critical Web Application Security Risks. OWASP Foundation, 2017. 23 p.
21. Geetha K., Sreenath N. SYN flooding attack – Identification and analysis // Proc. International Conference on Information Communication and Embedded Systems (ICICES 2014). 2014. P. 1–7. doi: 10.1109/ICICES.2014.7033828
22. Марков А.С., Цирлов В.Л. Опыт выявления уязвимостей в зарубежных программных продуктах // Вопросы кибербезопасности. 2013. № 1. С. 42–48.
23. Яковлев Г.О., Батетников И.А. Обеспечение безопасности сторонних компонентов веб приложений // Вестник науки и образования. 2019. № 9-2(63). С. 6–9.
24. Чукляев Е.И. Современные технологии статического и динамического анализа программного обеспечения // Наукоемкие технологии в космических исследованиях Земли. 2016. Т. 8. № S2. С. 56–60.
25. Шишкин Ю.Е. Оптимизация выявления аномалий облачных сервисов // Наука, техника и образование. 2017. № 4(34). С. 62–65. doi: 10.20861/2312-8267-2017-34-002
26. Скабцов Н. Аудит безопасности информационных систем. СПб.: Питер, 2018. 272 с. (Библиотека программиста).
27. Артамонов А.С., Иванов А.Ю. Перспективные методы анализа информационных потоков в сфере безопасности автоматизированных систем МЧС России (информационно-аналитически обзор часть 2) // Научно-аналитический журнал «Вестник Санкт-Петербургского университета Государственной противопожарной службы МЧС России». 2017. № 1. С. 74–83. doi: 10.24411/2218-130X-2017-00035
28. Chang J., Venkatasubramanian K., West A.G., Kannan S., Sokolsky O., Kim M.J., Lee I. ToMaTo: A trustworthy code mashup development tool. // Proc. 5th International Workshop on Web APIs and Service, Mashups'11. 2011. P. 18 [Электронный ресурс]. URL: https://dl.acm.org/doi/10.1145/2076006.2076012 (дата обращения: 31.03.2020). doi: 10.1145/2076006.2076012
29. Намиот Д.Е., Романов В.Ю. Анализ данных для программных репозиториев // International Journal of Open Information Technologies. 2018. Т. 6. № 4. С. 1823 [Электронный ресурс]. URL: https://cyberleninka.ru/article/n/analiz-dannyh-dlya-programmnyh-repozitoriev (дата обращения: 23.02.2020).
30. Wang H., Zhang D., Shin K.G. Detecting syn flooding attacks // Proc. 21st Annual Joint Conference of the IEEE Computer and Communications Societies. INFOCOM-2002. 2002. V. 3. P. 1530–1539. doi: 10.1109/INFCOM.2002.1019404
31. Ertaul L., Martirosyan Y. Implementation of a Web Application for Evaluation of Web Application Security Scanners // Proc. International Conference on Security and Management. 2012. P. 82–89.
32. Rafique S., Humayun M., Gul Z., Abbas A., Javed H. Systematic Review of Web Application Security Vulnerabilities Detection Methods // Journal of Computer and Communications. 2015. V. 3. N 9. P. 28–40. doi: 10.4236/jcc.2015.39004
33. Одинокая М.А. Об эффективном использовании современных технологий создания интерактивных веб-документов // Интер-
14. Beliaev A., Petrenko S. *Anomaly detection systems: novel Ideas for data protection*. Available at: http://citforum.ru/security/articles/anomalis/ (accessed: 14.02.2020). (in Russian)
15. Tsyganenko N.P. The static analysis of mobile applications code as vulnerabilities detection method. *Proceedings of BSTU. Physics and Mathematics. Informatics*, 2015, no. 6(179), pp. 200–203. (in Russian)
16. Markov A.S., Matveev V.A., Fadin A.A., Tsirlov V.L. Heuristic analysis of source code security. *Herald of the Bauman Moscow State Technical University. Series Instrument Engineering*, 2016, no. 1, pp. 98–111. (in Russian)
17. Ikonnikov M.A., Karmanov I.N. Measures and requirements to protected web applications. *Interexpo GEO-Siberia*, 2019, vol. 6, no. 2, pp. 13–19. (in Russian). doi: 10.33764/2618-981X-2019-6-2-13-19
18. Barabanov A., Lavrov A., Markov A., Polotnyanschikov I. A Study of Cross-Site Request Forgeries During Software Security Evaluation. *Voprosy kiberbezopasnosti*, 2016, no. 5, pp. 43–49. (in Russian). doi: 10.21581/2311-3456-2016-5-43-50.
19. Barabanov A., Markov A., Fadin A., Tsirlov V. Statistics of Software Vulnerabilities Detection During Certified Testing. *Voprosy kiberbezopasnosti*, 2017, no. 2, pp. 2–8. (in Russian). doi: 10.21581/2311-3456-2017-2-2-8
20. *OWASP TOP 10 – 2017. The Ten Most Critical Web Application Security Risks.* OWASP Foundation, 2017, 23 p.
21. Geetha K., Sreenath N. SYN flooding attack — Identification and analysis. *Proc. International Conference on Information Communication and Embedded Systems (ICICES 2014)*, 2014, pp. 1–7. doi: 10.1109/ICICES.2014.7033828
22. Markov A., Tsirlov V. Experience in identifying vulnerabilities in software. *Voprosy kiberbezopasnosti*, 2013, no. 1, pp. 42–48. (in Russian)
23. Yakovlev G.O., Batetnikov I.A. Securing third-party web application components. *Bulletin of Science and Education*, 2019, no. 9-2(63), pp. 6–9. (in Russian)
24. Chuklyaev E.I. The modern technologies of static and dynamic analysis of software. *H&ES Research*, 2016, vol. 8, no. S2, pp. 56–60. (in Russian)
25. Shishkin Yu.E. Optimization of cloud services anomalies detection. *Science, Technology and Education*, 2017, no. 4(34), pp. 62–65. (in Russian). doi: 10.20861/2312-8267-2017-34-002
26. Skabtcov N. *Audit of Information Systems Security*. St.Petersburg, Piter Publ., 2018, 272 p. (in Russian)
27. Artamonov A.S., Ivanov A.Yu. Advanced methods of analysis of information flows in the sphere of security of the automated systems of emercom of Russia (information-analytical review - part 2). *Vestnik sankt-peterburgskogo universiteta GPS MCHS Rossii*, 2017, no. 1, pp. 74–83. (in Russian). doi: 10.24411/2218-130X-2017-00035
28. Chang J., Venkatasubramanian K., West A.G., Kannan S., Sokolsky O., Kim M.J., Lee I. ToMaTo: A trustworthy code mashup development tool. *Proc. 5th International Workshop on Web APIs and Service, Mashups'11, 2011*, pp. 18. Available at: https://dl.acm.org/doi/10.1145/2076006.2076012 (accessed: 31.03.2020). doi: 10.1145/2076006.2076012
29. Namiot D., Romanov V. On data mining for software repositories. *International Journal of Open Information Technologies*. 2018, vol. 6. no 4. pp. 1823. Available at: https://cyberleninka.ru/article/n/analiz-dannyh-dlya-programmnyh-repozitoriev (accessed: 23.02.2020). (in Russian)
30. Wang H., Zhang D., Shin K.G. Detecting syn flooding attacks. *Proc. 21st Annual Joint Conference of the IEEE Computer and Communications Societies*. INFOCOM-2002, 2002, vol. 3, pp. 1530–1539. doi: 10.1109/INFCOM.2002.1019404
31. Ertaul L., Martirosyan Y. Implementation of a Web Application for Evaluation of Web Application Security Scanners. *Proc. International Conference on Security and Management*, 2012, pp. 82–89.
32. Rafique S., Humayun M., Gul Z., Abbas A., Javed H. Systematic Review of Web Application Security Vulnerabilities Detection Methods. *Journal of Computer and Communications*, 2015, vol. 3, no. 9, pp. 28–40. doi: 10.4236/jcc.2015.39004
33. Odinokaya M.A. About the effective usage of modern tech nologies of the creation of interactive web-documents. *Interactive science*, 2017, no. 3(13), pp. 55–56. Available at: https://cyberleninka.ru/article/n/ob-effektivnom-ispolzovanii-sovremennyh-tehnologiy-sozdaniya-interaktivnyh-veb-dokumentov-1 (accessed: 22.02.2020). (in Russian). doi: 10.21661/r-118243
34. Novozhylov A.V., Akulov G.S. Browsers support of HTML5 and CSS3. *Russian Universities Reports. Mathematics*. 2014, vol. 19,

активная наука. 2017. № 3(13). С. 55–56 [Электронный ресурс]. URL: https://cyberleninka.ru/article/n/ob-effektivnom-ispolzovanii-sovremennyh-tehnologiy-sozdaniya-interaktivnyh-veb-dokumentov-1 (дата обращения: 22.02.2020). doi: 10.21661/r-118243

34. Новожилов А.В., Акулов Г.С. Поддержка браузерами технологий HTML5 и CSS3 // Вестник российских университетов. Математика. 2014. Т. 19. № 2. С. 663–665 [Электронный ресурс]. URL: https://cyberleninka.ru/article/n/podderzhka-brauzerami-tehnologiy-html5-i-css3 (дата обращения: 22.02.2020).
35. Бутин А.А. Методические аспекты разработки систем защиты программного обеспечения // Вестник науки и образования. 2018. № 16-1(52). С. 30–36 [Электронный ресурс]. URL: https://cyberleninka.ru/article/n/metodicheskie-aspekty-razrabotki-sistem-zaschity-programmnogo-obespecheniya (дата обращения: 22.02.2020).
36. Григорьев С.В., Ковалев Д.А. Алгоритм синтаксического анализа контекстно-свободной аппроксимации динамически формируемого кода // Известия вузов. Северо-Кавказский регион. Технические науки. 2017. № 3(195). С. 43–48 [Электронный ресурс]. URL: https://cyberleninka.ru/article/n/algoritm-sintaksicheskogo-analiza-kontekstno-svobodnoy-approksimatsii-dinamicheski-formiruemogo-koda (дата обращения: 22.02.2020). doi: 10.17213/0321-2653-2017-3-43-48
37. Кулясов Н.В., Исаев С.В. Исследование сетевых аномалий корпоративной сети Красноярского научного центра // Сибирский журнал науки и технологий. 2018. Т. 19. № 3. С. 412–422 [Электронный ресурс]. URL: https://cyberleninka.ru/article/n/issledovanie-setevyh-anomaliy-korporativnoy-seti-krasnoyarskogo-nauchnogo-tsentra (дата обращения: 22.02.2020). doi: 10.31772/2587-6066-2018-19-3-412-422
38. Зуев В.Н., Ефимов А.Ю. Нейросетевой поведенческий анализ действий пользователя в целях обнаружения вторжений уровня узла // Программные продукты и системы. 2019. № 2. С. 268–272 [Электронный ресурс]. URL: https://cyberleninka.ru/article/n/neyrosetevoy-povedencheskiy-analiz-deystviy-polzovatelya-v-tselyah-obnaruzheniya-vtorzheniy-urovnya-uzla (дата обращения: 22.02.2020). doi: 10.15827/0236-235X.126.268-272
39. Вишневский А.С. Обманная система для выявления хакерских атак, основанная на анализе поведения посетителей веб-сайтов // Вопросы кибербезопасности. 2018. № 3(27). С. 54–62 [Электронный ресурс]. URL: https://cyberleninka.ru/article/n/obmannaya-sistema-dlya-vyyavleniya-hakerskih-atak-osnovannaya-na-analize-povedeniya-posetiteley-veb-saytov (дата обращения: 22.02.2020). doi: 10.21681/2311-3456-2018-3-54-62
40. Бурлаков М.Е., Ивкин А.Н. Система обнаружения вторжения на основе искусственной иммунной системы // Вестник ПНИПУ. Электротехника, информационные технологии, системы управления. 2019. № 29. С. 209–224 [Электронный ресурс]. URL: https://cyberleninka.ru/article/n/sistema-obnaruzheniya-vtorzheniya-na-osnove-iskusstvennoy-immunnoy-sistemy (дата обращения: 22.02.2020).

no. 2, pp. 663–665. Available at: https://cyberleninka.ru/article/n/podderzhka-brauzerami-tehnologiy-html5-i-css3 (accessed: 22.02.2020). (in Russian)

35. Butin A.A. Methodical aspects of development of protection systems software. *Bulletin of Science and Education*, 2018, no. 16-1(52), pp. 30–36. Available at: https://cyberleninka.ru/article/n/metodicheskie-aspekty-razrabotki-sistem-zaschity-programmnogo-obespecheniya (accessed: 22.02.2020). (in Russian)
36. Grigorev S.V., Kovalev D.A. Syntax analysis of context-free approximation of dynamically generated code. *University news. North-Caucasian region. Technical sciences series*, 2017, no. 3(195), pp. 43–48. Available at: https://cyberleninka.ru/article/n/algoritm-sintaksicheskogo-analiza-kontekstno-svobodnoy-approksimatsii-dinamicheski-formiruemogo-koda (accesed: 22.02.2020). (in Russian). doi: 10.17213/0321-2653-2017-3-43-48
37. Kulyasov N.V., Isaev S.V. Investigation of the network anomalies of the corporate network of Krasnoyarsk scientific center. *Siberian Journal of Science and Technology*, vol. 19, no. 3, pp. 412–422. Available at: https://cyberleninka.ru/article/n/issledovanie-setevyh-anomaliy-korporativnoy-seti-krasnoyarskogo-nauchnogo-tsentra (accessed: 22.02.2020). (in Russian). doi: 10.31772/2587-6066-2018-19-3-412-422
38. Zuev V.N., Efimov A.Yu. Neural network user behavior analysis for detecting host-level intrusion. *Software & Systems*, 2019, no. 2, pp. 268–272. Available at: https://cyberleninka.ru/article/n/neyrosetevoy-povedencheskiy-analiz-deystviy-polzovatelya-v-tselyah-obnaruzheniya-vtorzheniy-urovnya-uzla (accessed: 22.02.2020). (in Russian). doi: 10.15827/0236-235X.126.268-272
39. Vishnevsky A. Content Based Attack Detection in Web-Oriented Honeypots. *Voprosy kiberbezopasnosti*, 2018, no. 3(27), pp. 54–62. Available at: https://cyberleninka.ru/article/n/obmannaya-sistema-dlya-vyyavleniya-hakerskih-atak-osnovannaya-na-analize-povedeniya-posetiteley-veb-saytov (accessed: 22.02.2020). (in Russian). doi: 10.21681/2311-3456-2018-3-54-62
40. Burlakov M.E., Ivkin A.N. Intrusion detection system based on the artificial immune system. *PNRPU Bulletin. Electrotechnics, Informational Technologies, Control Systems*, 2019, no. 29, pp. 209–224. Available at: https://cyberleninka.ru/article/n/sistema-obnaruzheniya-vtorzheniya-na-osnove-iskusstvennoy-immunnoy-sistemy (accessed: 22.02.2020). (in Russian)

Авторы

Михеева Олеся Игоревна — студент, Университет ИТМО, Санкт-Петербург, 197101, Российская Федерация, ORCID ID: 0000-0001-8370-529X, rapppersha@yandex.ru

Гатчин Юрий Арменакович — доктор технических наук, профессор, профессор, Университет ИТМО, Санкт-Петербург, 197101, Российская Федерация, Scopus ID: 56127302800, ORCID ID: 0000-0002-1067-619X, gatchin@mail.ifmo.ru

Савков Сергей Витальевич — директор по информационным технологиям, ЗАО «Мобикон», Санкт-Петербург, 199178, Российская Федерация, Scopus ID: 57212837033, ORCID ID: 0000-0002-9438-4564, sergsavkov@gmail.com

Хамматова Регина Марсельевна — инженер, ООО «Удостоверяющий центр ГАЗИНФОРМСЕРВИС», Санкт-Петербург, 198096, Российская Федерация, ORCID ID: 0000-0001-7236-9024, r.hamatova@mail.ru

Ныркov Анатолий Павлович — доктор технических наук, профессор, профессор, Государственный университет морского и речного флота им. адмирала С.О. Макарова, Санкт-Петербург, 198035, Российская Федерация, Scopus ID: 56607396800, ORCID ID: 0000-0002-9803-6284, NyrkowAP@gumrf.ru

Authors

Olesia I. Mikheeva — Student, ITMO University, Saint Petersburg, 197101, Russian Federation, ORCID ID: 0000-0001-8370-529X, rapppersha@yandex.ru

Yury A. Gatchin — D.Sc., Full Professor, ITMO University, Saint Petersburg, 197101, Russian Federation, Scopus ID: 56127302800, ORCID ID: 0000-0002-1067-619X, gatchin@mail.ifmo.ru

Sergey V. Savkov — Chief Information Officer, ZAO Mobicon, Saint Petersburg, 199178, Russian Federation, Scopus ID: 57212837033, ORCID ID: 0000-0002-9438-4564, sergsavkov@gmail.com

Regina M. Khammatova — Engineer, LLC Certification Authority Gazinformservice, Saint Petersburg,198096, Russian Federation, ORCID ID: 0000-0001-7236-9024, r.hamatova@mail.ru

Anatoliy P. Nyrkov — D.Sc., Full Professor, Admiral Makarov State University of Maritime and Inland Shipping, Saint Petersburg, 198035, Russian Federation, Scopus ID: 56607396800, ORCID ID: 0000-0002-9803-6284, NyrkowAP@gumrf.ru