Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное
учреждение высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»
Институт естественных и точных наук
Факультет математики, механики и компьютерных технологий
Кафедра прикладной математики и программирования
Направление подготовки: 09.04.04 Программная инженерия

РАБОТА ПРОВЕРЕНА
Рецензент, к.ф.-м.н., доцент кафедры «Системное программирование»
_____В.А. Голодов
«____»_____2019 г.

ДОПУСТИТЬ К ЗАЩИТЕ
Заведующий кафедрой, д.ф.-м.н., доцент
_____А.А. Замышляева
«____»_____2019 г.

# Разработка плагина для браузера Chrome, прогнозирующего поведение пользователя на веб-странице, с безопасным хранением статистики

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ
ЮУрГУ–09.04.04.2019.112.ПЗ ВКР

Руководитель работы, к.ф.-м.н.,
доцент кафедры ПМиП
_____/С.М. Елсаков
«____»_____2019 г.
Автор работы
Студент группы ЕТ-225
_____/ М.Ю. Усенко
«____»_____2019 г.
Нормоконтролер, ассистент
_____/Н.С. Мидоночева
«____»_____2019 г.

Челябинск
2019

# АННОТАЦИЯ

Усенко М.Ю. Разработка плагина для браузера Chrome, прогнозирующего поведение пользователя на веб-странице с безопасным хранением статистики. – Челябинск: ЮУрГУ, ЕТ-225, 50 с., 16 ил., библиогр. список – 18 наим., 2 прил.

Целью данной работы является разработка плагина для браузера Google Chrome, прогнозирующего поведение пользователя на веб-странице с безопасным хранением статистики. Отличительной особенностью данного расширения является возможность автоматизации действий, выполняемых пользователем, на стороне клиента.

В первой главе, которая является обзорной, описаны существующие решения для автоматизации действий пользователя, описаны способы хранения данных на стороне клиента, существующие угрозы безопасности согласно OWASP Top-10, рассмотрены различные алгоритмы симметричного шифрования.

Во второй главе представлена архитектура приложения, спроектирована база данных, разработано три модуля: модуль сбора данных, модуль выявления закономерностей, модуль автоматизации действий. А также разработана защита расширения.

В третьей главе проведено тестирование разработанного расширения: тестирование корректности работы и тестирование защищённости.

ОГЛАВЛЕНИЕ

# ВВЕДЕНИЕ

В настоящее время всё больше компаний предоставляют свои услуги в интернете и всё больше повседневных действий пользователи совершают онлайн: от заказа еды до поиска жилья и совершения банковских операций. При этом большинство компаний и разработчиков стараются упростить эти процессы: уменьшить количество необходимых действий и затрачиваемое время.

С одной стороны, эта задача решается привлечением user experience специалистов, которые прописывают user journey и оптимизируют интерфейс сервиса и тем самым упрощают взаимодействие с ним. Другой способ – оптимизировать бизнес-процессы: так, например, в крупных интернет-магазинах появилась возможность «заказать в один клик» на основе предоставленных ранее пользователем данных. Третий подход, который позволяет упрощать взаимодействие с веб-сервисами, это программы и расширения, которые используют технологии веб-майнинга и веб-скраппинга.

Актуальность данной работы обусловлена тем, что большинство существующих решений для прогнозирования действий пользователя и их автоматизации либо внедряются на стороне сервера, либо обеспечивают только получение данных с серверов и веб-страниц, не реализуя возможность отправки. При этом пользовательские данные, как правило, хранятся на серверах сторонних компаний. То есть пользователь, единожды передав им данные, будет вынужден полагаться на ответственное отношение компании к информационной безопасности.

Целью данной выпускной квалификационной работы является разработка плагина для браузера Google Chrome, прогнозирующего поведение пользователя на web-странице, с безопасным хранением статистики.

Для достижения поставленной цели нам необходимо выполнить следующие задачи:

– рассмотреть существующие системы для автоматизации поведения пользователя;

– рассмотреть существующие способы хранения данных в браузере;

– выбрать способ хранения данных;

– разработать алгоритм сбора информации о действиях пользователя на web-странице;

– разработать базу данных для хранения собранной информации;

– разработать алгоритмы прогнозирования поведения пользователя и автоматизации выполнения действий;

– изучить возможные угрозы для информационной безопасности расширения и разработать защиту расширения.

# 1 ТЕХНОЛОГИИ И УЯЗВИМОСТИ ВЕБ-ПРИЛОЖЕНИЙ

Главная задача разрабатываемого расширения – автоматизировать действия пользователя на веб-странице. Под действиями мы понимаем заполнение форм на сайтах и их отправку. Целевые web-страницы и данные для отправки формируются на основе статистики предыдущего поведения пользователя. При этом нам необходимо обезопасить собранную информацию о поведении пользователя от злоумышленников.

В этой главе будут рассмотрены существующие решения для автоматизации действий пользователя на web-странице, методы хранения данных, потенциальные уязвимости и алгоритмы шифрования.

## 1.1 Решения для и автоматизации действий пользователя

В современных браузерах, таких как Google Chrome и Opera, есть механизм автоматического заполнения форм. Есть два способа реализации автозаполнения.

Первый способ – атрибут autocomplete тэга input, который появился в спецификации HTML5. Он имеет следующий вид: <input type="text" name="foo" id="bar" autocomplete="organization"> [1]. Значение атрибута сообщает браузеру смысл заполняемого поля. Всего в спецификации HTML5 есть 53 значения [1] поля autocomplete, такие как address, name и другие. Как только пользователь отправляет форму с атрибутами autocomplete, браузер предлагает запомнить данные [2]. Однако он не автоматизирует отправку данных на сервер и, в случае, если пользователю необходимо ввести данные на нескольких формах, каждую из них придётся заполнять отдельно. Ранее, пользовательские данные (в том числе сохранённые пароли) хранились на стороне пользователя. На момент написания работы Google хранит пользовательские данные на стороне сервера, так как доступ к аккаунту может совершаться с разных устройств [3].

Другой способ автоматизации действий пользователя используется в крупных интернет-магазинах, например, возможность «заказать в один клик», данные для которой берутся из предыдущих действий пользователя на сайте. Но такой функционал внедряется на стороне сервера сугубо по желанию его владельцев.

Существуют различные системы, работающие на методах Web-Mining и Web-Scrapping, такие как DataCol или Dexi.io, но они специализируются только на получении данных с web-страниц.

В связи с тем, что ни в одном из рассмотренных решений нет возможности автоматического заполнения и отправки форм, было принято решение о разработке плагина для браузера, прогнозирующего поведение пользователя с безопасным хранением статистики. Среди всех браузеров мы выбрали Google Chrome, так как по данным статистики использования браузеров [4] им пользуется более 62% всех пользователей сети интернет, что делает его самым используемым браузером в мире.

## 1.2 Методы хранения данных на стороне клиента

Разрабатываемое расширение предполагает хранение пользовательских данных. Мы можем хранить их либо на удалённом сервере, либо на стороне пользователя. Данная работа не предполагает разработку серверной части и превращение расширения в полноценный веб-сервис. К тому же, отправка данных на сервер создаёт дополнительные риски для информационной безопасности.

В разрабатываемом расширении мы храним данные на стороне пользователя. Хранение данных в локальной реляционной базе данных нам не подходит. Во-первых, при установке расширения пользователю также придётся устанавливать СУБД и создавать базу данных на своём компьютере. Во-вторых, приложения не могут подключаться к базе данных напрямую, для этого им

требуется прослойка в виде веб-приложения [1], которое уже будет обращаться к базе данных.

Второй вариант: хранить данные на стороне пользователя в браузере. Для этого существует несколько технологий, которые будут рассмотрены ниже.

### 1.2.1 Cookies

Куки – это классический способ хранения простых строчных данных внутри документа. Обычно куки отсылаются с сервера на клиент, который может сохранять их, а затем отправлять обратно на сервер в ответ на последующие запросы. Это может быть использовано для таких вещей, как управление сессиями аккаунта или отслеживание пользовательской информации.

Дополнительно куки можно использовать и для простого хранения данных на клиентской стороне. Поэтому они также часто используются для хранения общих данных типа пользовательских настроек.

Преимущества Cookies:

– их можно использовать для коммуникации с сервером [6];

– мы можем определить для куки срок их автоматического окончания вместо того, чтобы удалять вручную [6].

Недостатки Cookies:

– они добавляются к загрузке страницы документа;

– они могут хранить небольшое количество данных;

– они могут содержать только строки;

– потенциальные проблемы с безопасностью;

– этот метод не рекомендуется для хранения данных на клиенте с момента появления WebStorageAPI (локальное и сессионное хранилище).

### 1.2.2 WebSQL

WebSQL – это API для реляционной базы на клиенте, сходное с SQLite, то есть это полноценная реляционная база данных на стороне клиента, запросы к которой мы можем отправлять с помощью SQL. С 2010 года данный API не поддерживается [7], поэтому современные браузеры не гарантируют корректную работу с WebSQL.

### 1.2.3 LocalStorage и SessionStorage

Локальное хранилище – это одна из разновидностей WebStorageAPI [8], специального API для хранения данных в браузере в формате ключ-значение. Этот API был создан как решение для проблем с куки и является более интуитивным и безопасным способом хранения простых данных внутри браузера.

Хотя технически мы можем хранить в локальном хранилище только строки, это обходится за счет преобразования в JSON. Таким образом мы можем сохранять в локальном хранилище более сложные данные по сравнению с куки.

Преимущества локального хранилища:

– предлагает более простой и интуитивный интерфейс хранения данных;

– более безопасно для хранения данных на клиенте;

– позволяет хранить больше данных.

Недостатки локального хранилища:

– позволяет хранить только строки.

Сессионное хранилище это вторая разновидность WebStorage API [8]. Оно точно такое же, как и локальное хранилище за исключением того, что данные хранятся только для сессии вкладки браузера. Как только пользователь уходит со страницы и закрывает браузер, данные очищаются.

### 1.2.4 IndexedDB

IndexedDB это намного более сложное и проработанное решение для хранения данных в браузере, так как это низкоуровневый API для хранения на клиенте значительного количества структурированных данных [9]. По своей сути это объектно-ориентированная база данных, основанная на JavaScript, которая позволяет нам легко сохранять и извлекать данные, проиндексированные по ключу. В качестве аналогов таблицы выступают объекты типа ObjectStorage [9].

Выполнение операций использующих IndexedDB происходит асинхронно, то есть не блокирует приложение.

Преимущества IndexedDB:

− могут обрабатывать более сложные структурированные данные;

− больше объем хранения;

− больше контроля по взаимодействию с хранилищем.

Недостатки IndexedDB:

− сложнее в освоении по сравнению с WebStorage API.

Сравнение рассматриваемых методов хранение данных на стороне пользователей рассматривается в таблице (таблица 1).

Таблица 1 – Сравнение методов хранения данных на стороне пользователя

| | Cookies | Local Storage | Session Storage | IndexedDB |
|---|---|---|---|---|
| Лимит хранимых данных | ~4кБ | ~5мБ | ~5мБ | До половины жёсткого диска |
| Постоянное хранение | Да | Да | Нет | Да |
| Тип хранимых данных | Строка | Строка | Строка | Любые данные |
| Индексируемость | Нет | Нет | Нет | Да |

Среди рассмотренных способов хранения данных на стороне пользователя нам подходит IndexedDB, так как она даёт возможность не просто хранить данные в виде ключ/данные, а является индексируемой, что даёт больше возможностей для выполнения CRUD-операций и структурирования данных.

## 1.3 Алгоритмы симметричного шифрования

База данных IndexedDB хранит данные ровно в том виде, в каком мы их туда помещаем. Это создаёт угрозы безопасности. Чтобы их устранить, необходимо шифровать данные. Исследования по этой теме [10, 11] предлагают использовать ассиметричный алгоритм RSA, однако в их системах присутствует сервер, который занимается выдачей ключей, что нам не подходит. В нашей системе предусмотрена авторизация пользователя по логину и паролю, поэтому мы можем использовать пользовательский пароль как ключ в симметричном шифровании. Ниже будут рассмотрены алгоритмы симметричного шифрования.

Алгоритм AES, Advanced Encryption System – это стандарт шифрования, принятый в США в 2001 году и пришедший на смену алгоритму DES. Опубликован под названием FIPS 197. AES является алгоритмом блочного шифрования. Длина блока – 128 бит, а длина ключей может быть равной 128, 192 или 256 бит. Количество раундов шифрования при этом зависит от длины ключа: для 128 бит – 10 раундов, для 192 – 12, для 256 – 14 раундов [12].

Алгоритм AES построен на основе подстановочно-перестановочной сети (SP-сеть), которая за один раунд шифрует весь блок сразу. В AES шифруемый блок и его промежуточные состояния в ходе шифрования представляются в виде матрицы байтов 4×4 (рисунок 1.1). На каждом раунде шифрования происходят следующие действия: сложение с раундовым ключом, байтовая подстановка, побайтовый циклический сдвиг в строках матрицы и умножение матриц.

Генерация раундовых ключей происходят следующим образом. Ключ и ключевая последовательность представляются в виде матриц 4-х байтовых слов, и начальный участок последовательности заполняется словами из ключа. Последующие слова ключевой последовательности вырабатываются по рекуррентному соотношению группами, кратными размеру ключа. Полученные из потока 4-байтовые слова группируются в ключи необходимого размера, который равен размеру шифруемого блока, и используются как раундовые ключи.

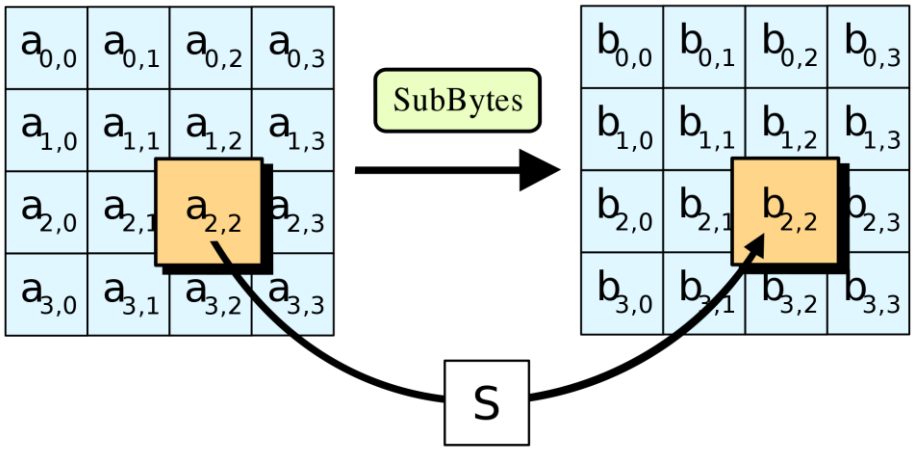


Рисунок 1.1 – Схема раунда в алгоритме AES

Алгоритм «Магма», ранее известный как ГОСТ 28147–89 был обнародован в 1994 году, а сейчас входит в состав ГОСТ Р 34.12–2015. Алгоритм «Магма» также является алгоритмом блочного шифрования. Длина блока составляет 64 бита, длина ключа 256 бит, число раундов – фиксированное, 32 раунда [13].

В основе алгоритма «Магма» лежит архитектура сети Фейстеля, состоящая из однотипных раундов. На каждом раунде блок разбивается на две половины одинаковой длины,  и  . К левой половине применяется функция fc ключом, результат складывается с правой половиной по модулю 2, результат меняется местами: на место  становится  и наоборот (рисунок 1.2). Из-за разницы в архитектурах алгоритмам, основанным на сети Фейстеля, в том

числе «Магме», требуется в два раза больше раундов для шифрования по сравнению с алгоритмами, основанными на подстановочно-перестановочной сети.

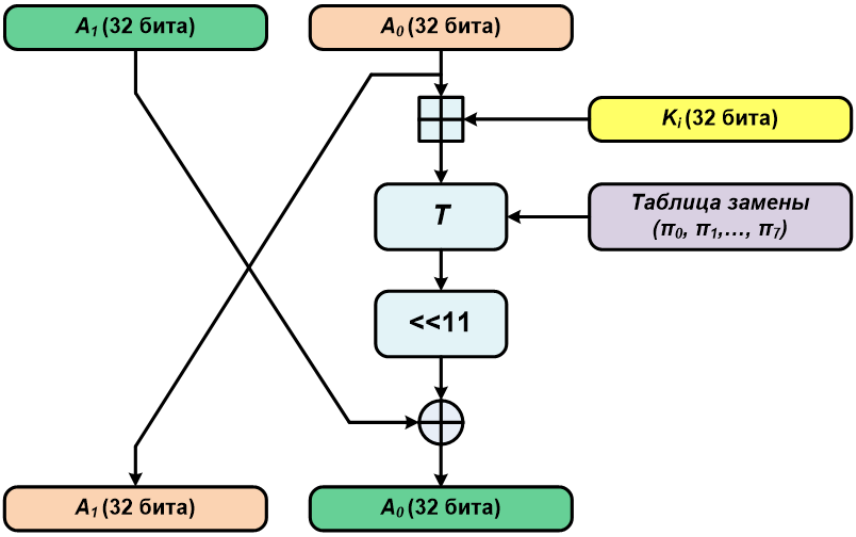


Рисунок 1.2 – Схема раунда в алгоритме «Магма»

В алгоритме «Магма» применён очень простой подход к получению раундовых ключей. Входной ключ представляется как массив из 8 элементов по 32 бита. Для раундов с 1 по 8, с 9 по 16 и с 17 по 24 используются ключи из матрицы с 1 по 8 соответственно. Для раундов с 25 по 32 эти ключи используются в обратном порядке [13].

Алгоритм «Кузнечик», как и «Магма», входит ГОСТ Р 34.12–2015и является блочным шифром. Длина блока – 128 бит, длина ключа – 256 бит, количество раундов – 10.

Как и алгоритм AES, «Кузнечик» основан на SP-сети. Раунд содержит три преобразования: сложение с раундовым ключом, преобразование блоком подстановок и линейное преобразование.

Генерация раундового ключа в алгоритме «Кузнечик» происходит следующим образом. Первые два раундовых ключа получаются разбиением входного ключа на две равные части (рисунок 1.3). Затем, для выработки новых пар раундовых ключей используется 8 итераций сети Фейстеля, где в качестве

раундовых ключей используется последовательность, прошедшая через линейное преобразование алгоритма.

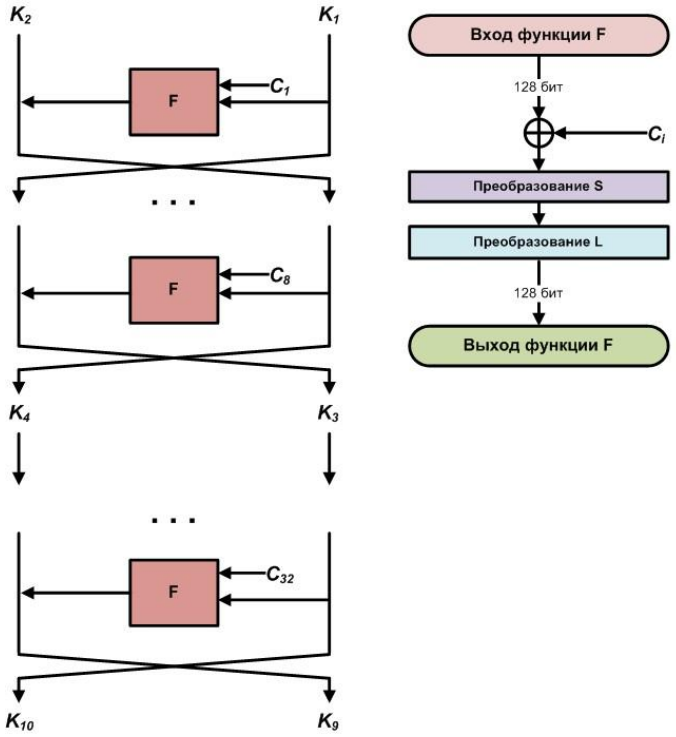


Рисунок 1.3 – Схема раунда в алгоритме «Кузнечик»

Сравнение алгоритмов шифрования по основным характеристикам представлено в таблице (таблица 2).

Таблица 2 – Сравнение основных алгоритмов симметричного шифрования

| | AES | Магма | Кузнечик |
|---|---|---|---|
| Архитектура | SP-сеть | Сеть Фейстеля | SP-сеть |
| Длина блока, бит | 128 | 64 | 128 |
| Длина ключа, бит | 128, 192, 256 | 256 | 256 |
| Количество раундов | В зависимости от длины ключа: 10, 12 и 14 раундов соответственно. | 32 | 10 |

## 1.4 OWASP топ-10

Исследованием уязвимостей и атак на веб-приложения, а также сбором статистики занимается OWASP – открытый проект по обеспечению безопасности веб-приложений. Все исследования проводятся на основе данных, собранных более чем от 40 компаний, специализирующихся на информационной безопасности, на отраслевых исследованиях и работах независимых исследователей. OWASP выступает за подход к безопасности приложений с точки зрения проблемы людей, процессов и технологий, поскольку для наиболее эффективного обеспечения безопасности приложений требуются улучшения во всех этих областях.

OWASP регулярно проводит исследования на тему информационной безопасности, самым их известным проектом является OWASP Top-10 с самыми распространёнными и опасными уязвимостями. Основной целью Top-10 OWASP является ознакомление разработчиков, проектировщиков, архитекторов, менеджеров и организаций в целом с рисками, связанными с наиболее распространенными и существенными недостатками в безопасности веб-приложений. Top-10 также предлагает базовые способы защиты от подобных рисков и руководства по дальнейшим действиям.

Методика оценки степени опасности уязвимостей для списка Top-10 основана на методике оценки рисков OWASP. Для каждой категории угроз оценивались характерные для стандартного веб-приложения недостатки, исходя из факторов их вероятности и риска. Затем угрозы группировались по степени опасности для веб-приложений. Список уязвимостей обновляется с каждым новым выпуском Топ-10, по мере изменения среды и условий эксплуатации.

Методика оценки рисков OWASP описывает множество факторов, помогающих оценить опасность обнаруженной уязвимости. Топ-10 предоставляет лишь обобщенные данные, а не информацию о конкретных

уязвимостях в реальных приложениях и API. Список составляется на основе глобальной статистики и обновляется по мере необходимости. Последняя версия OWASP Top-10, представленная в 2017 году, будет рассмотрена ниже.

### 1.4.1 Инъекции (внедрение)

Уязвимости, связанные, например, с внедрением SQL, NoSQL, OS и LDAP, возникают, когда непроверенные данные отправляются интерпретатору в составе команды или запроса. Вредоносные данные могут заставить интерпретатор выполнить непредусмотренные команды или обратиться к данным без прохождения соответствующей авторизации.

Почти любой источник данных может оказаться вектором для внедрения: переменные окружения, параметры, внешние и внутренние веб-службы, а также все типы пользователей. Внедрения становятся возможными, если злоумышленник может отправлять интерпретатору вредоносные данные.

Внедрения особенно распространены в старом коде. Уязвимости часто встречаются в SQL- LDAP-, XPath или NoSQL-запросах, системных командах, обработчиках XML, и запросах ORM [14]. Внедрения легко обнаружить при анализе кода. Сканеры и фаззеры могут помочь злоумышленникам найти подобные уязвимости.

Внедрения могут привести к потере данных, их повреждению или разглашению третьим лицам, а также к отказу в обслуживании. В некоторых случаях контроль над узлом может быть полностью перехвачен. Последствия для бизнеса зависят от критичности приложения и данных.

Приложение уязвимо, если:

− вводимые пользователем данные не проверяются, не фильтруются или не очищаются;

− динамические запросы или непараметризованные вызовы без контекстного экранирования напрямую используются в интерпретаторе;

– вредоносные данные используются в поисковых параметрах объектно-реляционного отображения для извлечения дополнительной, критичной информации;

– вредоносные данные используются или добавляются таким образом, что SQL код или команды содержат структурные и вредоносные данные в динамических запросах, командах или хранимых процедурах.

Анализ исходного кода является лучшим способом обнаружения внедрений, за которым следует полное автоматизированное тестирование всех вводимых параметров, заголовков, URL, cookie, JSON-, SOAP- и XML-данных [14]. Организации также могут включать в процесс непрерывной интеграции и развертывания ПО (CI/CD) статическое (SAST) и динамическое (DAST) тестирование кода и приложений для обнаружения новых уязвимостей перед внедрением приложений в производство.

### 1.4.2 Недостатки аутентификации

Функции приложений, связанные с аутентификацией и управлением сессиями, часто реализуются некорректно, позволяя злоумышленникам скомпрометировать данные учётных записей, ключи или сессионные токены авторизации, а также эксплуатировать другие недостатки аутентификации для временного или постоянного перехвата учетных записей пользователей.

Недостатки аутентификации очень распространены из-за исполнения и реализации большинства средств идентификации и контроля доступа. Управление сессиями является основой аутентификации и контроля доступа и присутствует во всех приложениях с контролем состояния.

Атакующие могут обнаружить недостатки аутентификации вручную и эксплуатировать их, используя автоматизированные инструменты, списки паролей и атаки по словарю.

Для компрометации системы злоумышленнику достаточно получить доступ к нескольким обычным или одной администраторской учетной записи.

В зависимости от области использования приложения результатом может стать отмывание денег, мошенничество в сфере социального обеспечения или кража персональных данных, а также разглашение охраняемой законом, конфиденциальной информации.

Подтверждение личности пользователя, аутентификация и управление сессиями играют важную роль в защите от атак, связанных с аутентификацией. Приложение имеет недостатки в аутентификации, если:

– допускается проведение автоматизированных атак, например, на учетные записи, когда у атакующего есть список действующих имен и паролей пользователей [14];

– допускается проведение атак методом подбора или других автоматизированных атак;

– допускается использование стандартных, ненадежных или хорошо известных паролей, например, «Password1» или «admin/admin»;

– используются ненадежные или неэффективные методы восстановления учетных данных и паролей, например, «ответы на основе знаний», которые являются небезопасными;

– используются незашифрованные, зашифрованные или ненадежно хешированные пароли;

– отсутствует или является неэффективной многофакторная аутентификация [14];

– отображаются идентификаторы сессии в URL (перезапись URL);

– не меняются идентификаторы сессий после успешного входа в систему;

– некорректно аннулируются идентификаторы сессий. Пользовательские сессии или токены аутентификации (в частности, токены единого входа (SSO)) неправильно аннулируются при выходе из системы или бездействии.

### 1.4.3 Разглашение конфиденциальных данных

Многие веб-приложения и API имеют плохую защиту критичных финансовых или персональных данных. Конфиденциальные данные требуют дополнительных мер защиты, например, их шифрования при хранении или передаче, а также специальных мер предосторожности при работе с браузером.

Вместо взлома механизмов шифрования злоумышленники крадут ключи, проводят атаки по принципу «человек посередине» или получают данные в незашифрованном виде с сервера, в процессе их передачи или из клиента пользователя, например, браузера. Подобные атаки обычно проводятся вручную. Ранее полученные базы данных паролей могут быть взломаны методом подбора с использованием графических процессоров.

На протяжении последних лет данная атака является самой распространенной и опасной. Чаще всего встречается отсутствие шифрования конфиденциальных данных, а при наличии часто используются ненадежные алгоритмы, протоколы, шифры, методы хранения хешированных паролей или методы создания и управления ключами. Также легко обнаружить уязвимости на стороне сервера для передаваемых данных, но не хранимых.

Из-за уязвимости часто страдают все персональные данные (медицинские записи, учетные данные, данные кредитных карт), которые должны быть защищены по закону, например, в соответствии с Общим регламентом ЕС по защите персональных данных (GDPR) или локальными законами о неприкосновенности данных.

### 1.4.4 Внешние сущности XML (XXE)

Старые или плохо настроенные XML-процессоры обрабатывают ссылки на внешние сущности внутри документов. Эти сущности могут быть использованы для доступа к внутренним файлам через обработчики URL-

файлов, общие папки, сканирование портов, удаленное выполнения кода и отказ в обслуживании.

Злоумышленники могут эксплуатировать уязвимые обработчики XML через загрузку XML или внедрение вредоносного контента в XML-документы, используя уязвимый код, зависимости или компоненты.

По умолчанию, большинство старых обработчиков XML позволяют задавать внешние сущности, URL, которые разыменовываются и вычисляются при обработке XML.

Инструменты SAST позволяют обнаружить уязвимость путем проверки зависимостей и конфигурации. Инструменты DAST требуют дополнительных операций, выполняемых вручную, для обнаружения и эксплуатации уязвимости. Тестировщиков, выполняющих проверки вручную, необходимо обучать XXE-тестированию, поскольку подобные проверки, по информации 2017 года, обычно не проводятся.

Приложения, в особенности веб-службы или компоненты на основе XML, являются уязвимыми в следующих случаях:

− приложение принимает XML напрямую или через выгрузку, особенно от недоверенных источников, или включает непроверенные данные в XML-документы, которые затем обрабатываются XML-обработчиком;

− хотя бы один из XML-обработчиков приложения или веб-службы на основе SOAP использует определение типа документов (DTD). Поскольку механизм отключения DTD зависит от обработчика, рекомендуется воспользоваться справочной информацией, например, «Памяткой OWASP по предотвращению XXE»;

− приложение использует SAML для идентификации в рамках федеративной безопасности или технологии единого входа (SSO). SAML использует XML для подтверждения идентификаторов, поэтому может быть уязвим;

– приложение использует SOAP версии ниже 1.2. Оно может быть уязвимо для XXE-атак, если XML-сущности передаются фреймворку SOAP;

– если приложение уязвимо для XXE-атак, то злоумышленник может также вызвать отказ в обслуживании или осуществить атаку с использованием XML-сущностей.

### 1.4.5 Недостатки контроля доступа

Действия, разрешенные аутентифицированным пользователям, зачастую некорректно контролируются. Злоумышленники могут воспользоваться этими недостатками и получить несанкционированный доступ к учетным записям других пользователей или конфиденциальной информации, а также изменить пользовательские данные или права доступа.

Эксплуатация контроля доступа является основным навыком злоумышленников. Инструменты SAST и DAST могут обнаружить отсутствие контроля доступа, но не могут проверить его работоспособность при его наличии. Наличие контроля доступа можно обнаружить вручную, а его отсутствие можно обнаружить автоматически в некоторых фреймворках.

Уязвимости, связанные с контролем доступа, довольно распространены из-за отсутствия автоматического обнаружения и эффективного функционального тестирования разработчиками. Тестирование вручную считается наилучшим способом обнаружения отсутствия или неэффективности контроля доступа, включая методы HTTP (GET, PUT и т. п.), контроллеры, прямые ссылки на объекты и т. д.

Технические последствия: выполнение злоумышленником действий с правами пользователя или администратора; использование пользователем привилегированных функций; создание, просмотр, обновление или удаление любых записей. Последствия для бизнеса зависят от критичности защиты приложения и данных.

Контроль доступа предполагает наличие политики, определяющей права пользователей. Обход ограничений доступа обычно приводит к несанкционированному разглашению, изменению или уничтожению данных, а также выполнению непредусмотренных полномочиями бизнес-функций. Наиболее распространенные уязвимости контроля доступа включают:

– обход ограничений доступа путем изменения URL, внутреннего состояния приложения или HTML-страницы, а также с помощью специально разработанных API;

– возможность изменения первичного ключа для доступа к записям других пользователей, включая просмотр или редактирование чужой учетной записи;

– повышение привилегий. Выполнение операций с правами пользователя, не входя в систему, или с правами администратора, войдя в систему с правами пользователя [14];

– манипуляции с метаданными, например, повторное воспроизведение или подмена токенов контроля доступа JWT или cookie-файлов, а также изменение скрытых полей для повышения привилегий или некорректное аннулирование JWT-токенов;

– несанкционированный доступ к API из-за некорректной настройки междоменного использования ресурсов (CORS) [15];

– доступ неаутентифицированных пользователей к страницам, требующим аутентификации, или доступ непривилегированных пользователей к привилегированным страницам. Доступ к API с отсутствующим контролем привилегий для POST-, PUT- и DELETE-методов/запросов [14].

### 1.4.6 Некорректная настройка параметров безопасности

Некорректная настройка безопасности является распространенной ошибкой. Это происходит из-за использования стандартных параметров безопасности, неполной или специфичной настройки, открытого облачного

хранения, некорректных HTTP-заголовков и подробных сообщений об ошибках, содержащих критичные данные. Все ОС, фреймворки, библиотеки и приложения должны быть не только настроены должным образом, но и своевременно корректироваться и обновляться.

Злоумышленники часто пытаются эксплуатировать неисправленные уязвимости, настроенные по умолчанию учетные записи, неиспользуемые страницы, незащищенные файлы и каталоги для получения несанкционированного доступа или информации о системе.

Подобные уязвимости позволяют злоумышленникам получить несанкционированный доступ к системным данным или функциям, а также могут привести к полной компрометации системы. Последствия для бизнеса зависят от критичности защиты приложения и данных.

Приложение уязвимо, если:

– любой из компонентов приложения недостаточно защищен или разрешения облачных сервисов некорректно настроены;

– включены или присутствуют лишние функции (например, неиспользуемые порты, службы, страницы, учетные записи или привилегии);

– учетные записи и пароли, создаваемые по умолчанию, используются без изменений;

– обработка ошибок позволяет осуществить трассировку стека или получить слишком подробные сообщения об ошибках;

– отключены или некорректно настроены последние обновления безопасности;

– не выбраны безопасные значения параметров защиты серверов приложений, фреймворков (например, Struts, Spring, ASP.NET), библиотек и прочих файлов, в которых могут содержаться настройки [14];

– сервер не использует безопасные заголовки или директивы, а также если они некорректно настроены;

– ПО устарело или имеет уязвимости.

Без организованной и регулярно выполняемой проверки безопасности приложений системы подвержены большему риску.

### 1.4.7 Межсайтовое выполнение сценариев (XSS)

XSS-атака происходит в тот момент, когда приложение добавляет непроверенные данные на новую веб-страницу без их соответствующей проверки или преобразования, или когда обновляет открытую страницу через API браузера, используя предоставленные пользователем данные, содержащие HTML- или JavaScript-код. С помощью внедрения на страницу вредоносного кода злоумышленники могут выполнять сценарии в браузере жертвы, которые позволяют им перехватывать данные пользовательских сессий, подменять страницы сайта или перенаправлять пользователей на вредоносные сайты.

Автоматизированные инструменты могут обнаруживать и эксплуатировать все три вида межсайтового выполнения сценариев, более того, фреймворки для их эксплуатации можно найти в открытом доступе.

Межсайтовое выполнение сценариев (XSS) является второй по распространенности уязвимостью из Top-10 OWASP и обнаруживается в 2/3 всех приложений.

Автоматизированные инструменты могут обнаруживать XSS автоматически, особенно в случае проработанных технологий, таких как PHP, J2EE / JSP и ASP.NET [14].

Межсайтовое выполнение сценариев будет иметь последствия средней степени тяжести в случае отраженного XSS или XSS на основе объектной модели документа и серьезные последствия в случае межсайтового выполнения хранимых сценариев с удаленным выполнением кода в браузере пользователя, например, кража учетных данных, перехват сессий или установка вредоносного программного обеспечения.

Существует три типа XSS, обычно эксплуатируемых в браузерах:

– отраженное межсайтовое выполнение сценариев (Reflected XSS). Приложение или API включает непроверенные и непреобразованные данные в состав HTML. Успешная атака может привести к выполнению произвольного HTML и JavaScript-кода в браузере жертвы. Обычно, злоумышленнику необходимо убедить пользователя перейти по ссылке, ведущей на вредоносную страницу, например, через рекламу;

– межсайтовое выполнение хранимых сценариев (Stored XSS). Приложение или API сохраняет необработанные входные данные, с которыми затем взаимодействуют пользователи или администраторы. Межсайтовое выполнение хранимых сценариев обычно считается очень опасной уязвимостью;

– межсайтовое выполнение сценариев на основе объектной модели документа (DOM XSS): JavaScript-фреймворки, одностраничные приложения и API, динамически добавляющие вредоносные данные на страницы, подвержены XSS на основе DOM. В идеале, приложение недолжно отправлять вредоносные данные небезопасным JavaScript API.

Обычно XSS используется для перехвата сессий, кражи учетных записей, обхода МФА, замены или подмены DOM-узлов (например, троянские панели входа в систему), а также атак на браузеры, например, для загрузки вредоносного ПО, регистрации нажатий и других атак на стороне клиента.

### 1.4.8 Небезопасная десериализация

Небезопасная десериализация часто приводит к удаленному выполнению кода. Ошибки десериализации, не приводящие к удаленному выполнению кода, могут быть использованы для атак с повторным воспроизведением, внедрением и повышением привилегий. Эксплуатировать десериализацию сложно, поскольку готовые эксплойты редко можно использовать без их изменения или доработки.

Некоторые инструменты могут обнаруживать ошибки десериализации, но для их подтверждения обычно требуется участие специалиста. Ожидается, что по мере разработки новых инструментов обнаружения и устранения ошибок десериализации данных об их распространенности станет больше.

Приложения и API уязвимы, если осуществляют десериализацию вредоносных или модифицированных объектов, предоставляемых злоумышленником.

Это позволяет осуществить два основных типа атак:

– атаки, связанные со структурой объектов и данных, когда злоумышленник изменяет логику приложения или удаленно выполняет произвольный код при наличии доступных приложению классов, поведение которых может меняться во время или после десериализации;

– атаки с подменой данных, например, связанные с управлением доступом, когда используются существующие структуры данных, но изменяется содержимое.

Сериализация может использоваться в приложениях для:

– удаленного и межпроцессного взаимодействия (RPC/IPC);

– проводных протоколов, веб-служб, брокеров сообщений;

– кэширования или сохранения данных;

– баз данных, серверов кэширования, файловых систем;

– cookie-файлов HTTP, параметров форм, токенов аутентификации API.

### 1.4.9 Использование компонентов с известными уязвимостями

Фреймворки, библиотеки и программные модули, запускаются с привилегиями приложения. Использование уязвимых компонентов может привести к потере данных или перехвату контроля над сервером. Использование приложениями и API компонентов с известными уязвимостями может нарушить защиту приложения и привести к серьезным последствиям.

Несмотря на простоту поиска уже готовых эксплойтов для большинства известных уязвимостей, некоторые из них требуют создания специальных средств для их эксплуатации.

Данная уязвимость является очень распространенной. Шаблоны для разработчиков, содержащие большое количество компонентов, могут привести к непониманию того, какие компоненты реально используются в приложении или API.

Несмотря на то, что не все уязвимости приводят к серьезным последствиям, причиной некоторых масштабных взломов стали именно компоненты, содержащие известные уязвимости. В зависимости от защищаемых активов подобная угроза может оказаться на вершине вашего списка.

Приложение уязвимо, если:

– вы не знаете версии всех используемых (на стороне клиента и на стороне сервера) компонентов. Сюда относятся сами компоненты и встроенные зависимости;

– ПО содержит уязвимости, не поддерживается или устарело. Сюда относятся ОС, веб-серверы, серверы приложений, СУБД, приложения, API, а также все компоненты, среды исполнения и библиотеки;

– поиск уязвимостей выполняется нерегулярно, а также отсутствует подписка на бюллетени по безопасности используемых компонентов;

– своевременно не устанавливаются исправления или обновления для используемых платформ, фреймворков и зависимостей. Обычно такое происходит, когда наличие обновлений

– проверяется раз в месяц или квартал, в результате чего организации неделями или месяцами не устраняют исправленные уязвимости;

– разработчики ПО не тестируют совместимость обновленных или исправленных библиотек;

– не обеспечивается безопасность компонентов.

### 1.4.10 Недостатки журналирования и мониторинга

Недостатки журналирования и мониторинга, а также отсутствие или неэффективное использование системы реагирования на инциденты, позволяет злоумышленникам развить атаку, скрыть свое присутствие и проникнуть в другие системы, а также изменить, извлечь или уничтожить данные. Проникновение в систему обычно обнаруживают только через 200 дней и, как правило, сторонние исследователи, а не в рамках внутренних проверок или мониторинга.

Эксплуатация недостатков журналирования и мониторинга лежит в основе почти всех крупных взломов. При проведении атак злоумышленники полагаются на отсутствие контроля и своевременного реагирования на инциденты.

Одним из способов определить качество мониторинга является анализ журналов после проведения теста на проникновение. Для определения возможного ущерба все действия тестировщиков должны регистрироваться соответствующим образом.

Большинство атак начинаются с анализа уязвимостей. Возможность проведения подобного анализа повышает вероятность удачной эксплуатации уязвимости практически до 100%. В 2016 году обнаружение факта проникновения занимало в среднем 191 день – нанесенный за это время ущерб мог быть огромным.

## 1.5 Выводы по разделу

В начале главы мы рассмотрели существующие механизмы и способы автоматизации действий в браузере, на отдельных сайтах и с помощью специальных программ и веб-сервисов.

Разрабатываемое расширение предполагает хранение пользовательских данных на стороне клиента в браузере. Среди всех рассмотренных технологий хранения мы выбрали IndexedDB по следующим причинам:

– в отличие от LocalStorage, которое также хранит данные в формате ключ/значение, IndexedDB позволяет хранить данные любого типа, а не только строковые;

– данные в IndexedDB можно проиндексировать, что упрощает их структуризацию и процесс записи/чтения;

– indexedDB является актуальной технологией, совместимой с большинством современных браузеров.

Сравнение алгоритмов шифрования по параметрам показало, что все рассматриваемые алгоритмы: AES, «Магма» и «Кузнечик» схожи по своим характеристикам. В качестве алгоритма шифрования в разрабатываемой программе был выбран «Кузнечик». По сравнению с алгоритмом «Магма» ему требуется меньше раундов для шифрования, так как в его основе лежит архитектура сети Фейстеля. Алгоритмы AESи «Кузнечик» основаны на подстановочно-перестановочной сети, однако отечественный алгоритм проще в реализации и освоении.

Далее в работе были рассмотрены самые опасные и распространённые уязвимости на основе последней версии списка OWASP Top-10 от 2017 года. В первую очередь нам необходимо обратить внимание на уязвимость инъекции и межсайтовое выполнение скриптов.

# 2 РАЗРАБОТКА РАСШИРЕНИЯ ДЛЯ ПРОГНОЗИРОВАНИЯ ПОВЕДЕНИЯ ПОЛЬЗОВАТЕЛЯ НА WEB-СТРАНИЦЕ

Структурно расширение делится на три модуля: модуль сбора данных, модуль выявления закономерностей, модуль автоматизации действий. Каждый из модулей обращается к базе данных. Модуль сбора данных и модуль автоматизации взаимодействуют с web-страницами, посещёнными пользователем. Структурная схема расширения представлена на рисунке (рисунок 2.1). Далее будет описан процесс проектирования базы данных и разработка каждого из модулей.

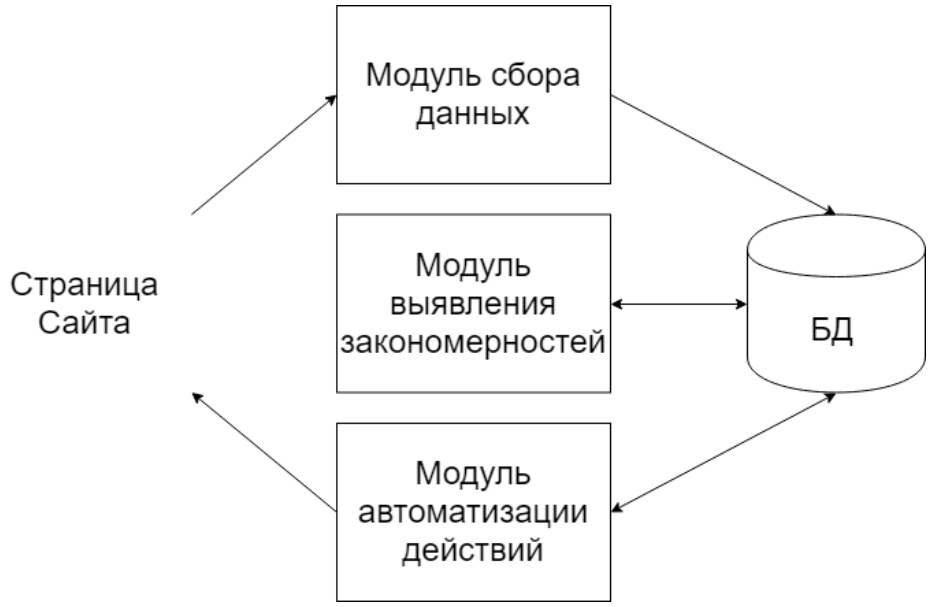


Рисунок 2.1 – Структурная схема расширения

## 2.1 Проектирование базы данных

В качестве технологии для хранения данных была выбрана NoSQL база данных IndexedDB. Аналогом таблицы в IndexedDB выступает объект ObjectStore, в котором мы можем задавать поля, указывая их параметры: уникальность, автоматическая инкрементация, индекс и другие.

Проектирование базы данных мы начали с концептуального проектирования. Оно производится на описании предметной области:

– у расширения может быть несколько пользователей, которые авторизируются посредством логина и пароля;

– для каждого пользователя собираются информации о посещённых страницах и время посещения;

– у каждого пользователя есть список игнорируемых сайтов, с которых данные не собираются;

– с каждой страницы собираются данные, отправленные с форм;

– повторяющиеся действия конкретного пользователя, выполнение которых будет автоматизироваться, сохраняются для дальнейшего использования.

На основе описания предметной области мы выявили следующие сущности: пользователь (user), посещённая страница (page), действия, выполненные на странице (action), список игнорируемых страниц (ignore), цепочка действий (chain), первая страница в цепочке (chain). На основе этого была построена ER-диаграмма (рисунок 2.2).


Рисунок 2.2 – ER-диаграмма базы данных

На основе ER-диаграммы была построена схема базы данных (рисунок 2.3). Поскольку в ER-диаграмме не были выявлены связи типа «многие ко многим» нам не пришлось проводить дополнительных преобразований.

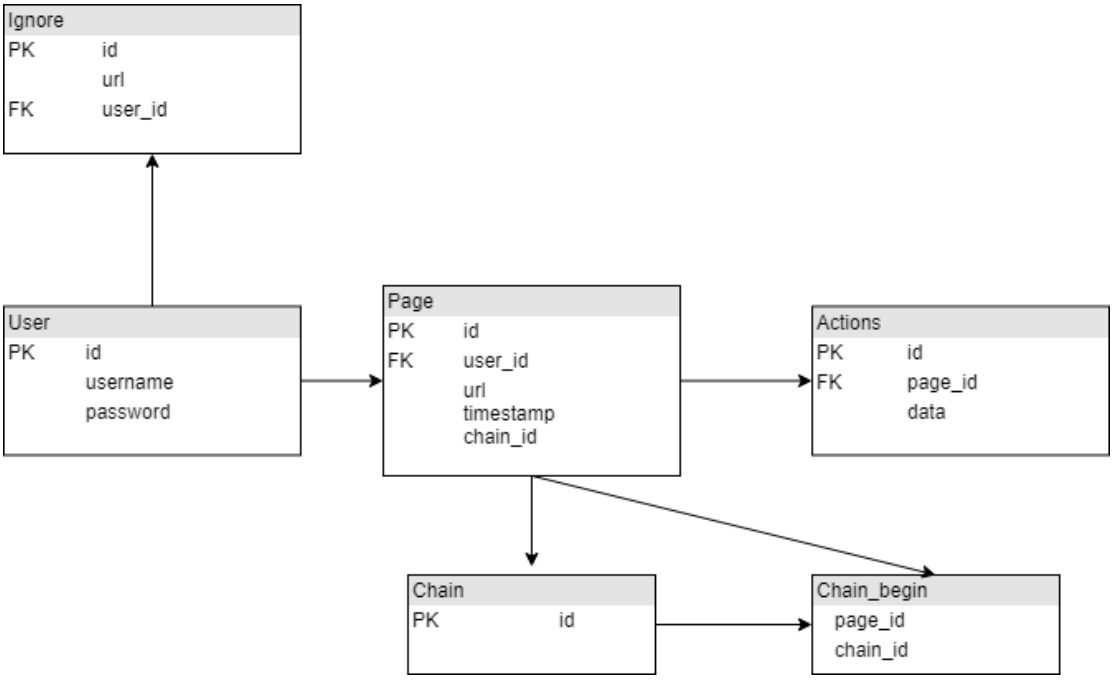


Рисунок 2.3 – Диаграмма базы данных

## 2.2 Модуль сбора данных

Каждый раз, когда пользователь заходит на страницу, создаётся соответствующая запись в таблице Page. Далее, если на странице происходит событие submit, то есть происходит отправка данных, данные заносятся в таблицу Actions.

Среди всех данных, отправляемых на сервер нас интересуют в первую очередь формы и все разновидности полей input: checkbox, text и другие. Данные в базу помещаются как JSON-объект вида «id элемента – значение».

При этом, пользователь может указать программе игнорировать некоторые сайты, на случай, если он не хочет автоматизировать взаимодействие с ними (например, если речь идёт о сайте банка и о совершении каких-либо финансовых операций). Список игнорируемых сайтов находится в

таблице Ignore, добавлять и удалять данные из списка пользователь может через интерфейс расширения.

### 2.3 Модуль прогнозирования и выявления закономерностей

Для автоматизации мы выбрали следующий сценарий:

– пользователь заходит на сайт;

– добавляет товары в корзину;

– переходит на страницу с оформлением заказа, где заполняет данные о себе и адресе доставки;

– попадает на страницу подтверждения.

Эти действия он совершает с определённой периодичностью. Исследования о поведения пользователя в интернете [16], говорят, что у пользователя сформировалась привычка поведения, когда он выполняет одинаковые не менее 3-х раз с интервалом не более 1 месяца между ними.

Назовём действия, которые мы будем автоматизировать, цепочкой. В реальности пользователь не будет идеально следовать описанному сценарию, он также может просматривать другие страницы сайта или между звеньями цепочки переходить на другие сайты. При этом цепочка должна состоять минимум из 2-х элементов, так как с заполнением форм в рамках одной страницы справится функция автоматического заполнения форм браузера.

Пусть у нас есть следующие данные (таблица 3).

Таблица 3 – Пример набора данных

| Id | URL | DATA | TimeStamp |
|---|---|---|---|
| 1 | my-shop.com/index | -- | 25.05.19 16:20 |
| 2 | my-shop.com/cart | Item_id=1, qnt=2 | 25.05.19 16:25 |
| 3 | my-shop.com/order | Mail=maks@mail.ru, Address=lenina80 | 25.05.19 16:30 |

Продолжение таблицы 3

| Id | URL | DATA | TimeStamp |
|---|---|---|---|
| 4 | youtube.com | -- | 26.05.19 15:30 |
| 5 | my-shop.com/index | -- | 27.05.19 10:03 |
| 6 | my-shop.com/cart | Item_id=1, qnt=2 | 27.05.19 10:05 |
| 7 | Mail.ru | -- | 27.05.19 10:07 |
| 8 | my-shop.com/order | Mail=maks@mail.ru, Address=lenina80 | 27.05.19 10:08 |
| 9 | my-shop.com/help | -- | 28.05.19 11:03 |
| 10 | my-shop.com/index | -- | 28.05.19 11:03 |
| 11 | my-shop.com/cart | Item_id=1, qnt=2 | 28.05.19 11:03 |
| 12 | my-shop.com/order | Mail=maks@mail.ru, Address=lenina80 | 28.05.19 11:03 |
| 13 | my-shop.com/index | -- | 29.05.19 15:05:13 |

Для того, чтобы выявить цепочку, сначала необходимо разбить записи на группы, по следующим признакам:

− записи должны принадлежать одному домену;

− записи вносятся в хронологическом порядке;

− запись не принадлежит группе, если от прошлой записи группы прошло более получаса.

На основе этих правил из данных (таблица 3) формируются следующие группы, элементы – номер записи из таблицы.

,

,

,

.

Искомая цепочка, которая нас интересует, это последовательность my-shop.com/index, my-shop.com/cart, my-shop.com/order с одинаковыми наборами данных. Чтобы её обнаружить, мы выполним следующие действия:

– каждая группа – это множество, элементы которого являются страницами;

– далее мы попарно начинаем искать пересечения множеств, нас интересуют пересечения мощностью больше 2. Порядок следования элементов можно не учитывать, потому что страница / cart не может появиться раньше, чем /order согласно бизнес-логике сайта;

– для найденных страниц мы сравниваем данные и смотрим, чтобы они полностью совпадали. В случае полного совпадения, это пересечение – предполагаемая цепочка, запоминаем её;

– после того, как мы нашли предполагаемую цепочку, мы продолжим пересекать множества, если мы нашли ещё два аналогичных вхождения, то цепочка обнаружена;

– после обнаружения цепочки, входящие в неё страницы необходимо внести в таблицу Chain, первую страницу в цепочке занести в таблицу Chain_begin.

## 2.4 Модуль автоматизации

Как только пользователь переходит на какую-либо страницу сайта, модуль автоматизации проверяет, не является ли данная страница начальной для какой-либо цепочки. Для того чтобы сократить время перебора таблицы

Chain, мы создали отдельную таблицу Chain_begin, которая содержит id страницы, с которой начинается цепочка и id самой цепочки.

Если текущая страница является начальной для какой-либо из цепочек, модуль автоматизации предлагает пользователю автоматизировать выполнение действий. Если пользователь соглашается, то модуль начинает извлекать данные из базы, преобразовывать их в строку с помощью метода JSON.stringify и посылать их на сервер.

### 2.5 Файл manifest.json

Особенностью разработки расширения для Google Chrome является необходимость написания специального файла manifest.json. Этот файл сообщает браузеру информацию о расширении (версия, названия, ссылки на используемые графические элементы и другие) и разрешениях, которые необходимо предоставить расширению для корректной работы [17].

В разрабатываемом расширении мы используем следующие разрешения:

– activeTab, предоставляет разрешения в соответствии со спецификацией активной вкладки;

– declarativeContent, даёт доступ к chrome.declarativeContent API;

– storage, предоставляет доступ к chrome.storage API;

– unlimitedStorage, снимает ограничение на размер 5Мб для хранилища;

– tabs, предоставляет расширению доступ к привилегированным полям объектов Tab, используемых несколькими API, включая chrome.tabs и chrome.windows;

– alarms, даёт доступ к chrome.alarms API;

– notifications, даёт доступ к chrome.notifications API;

– webRequests, даёт доступ к chrome.webRequests API;

## 2.6 Защита приложения

При разработке информационной безопасности расширения нашей главной целью было защитить пользовательские данные, которые помещаются в IndexedDB. Основным недостатком IndexedDB является то, что данные хранятся ровно в том виде, в каком они вносятся в базу. То есть если злоумышленник получит доступ к ПК, он сможет скопировать файл базы (обычно располагается в папке AppData\Local\Google\Chrome\User Data\ Default\IndexedDB для ОС Windows) и получить доступ ко всей пользовательской информации.

Чтобы обезопасить базу, нам необходимо шифровать данные при записи и дешифровать при чтении. Ранее мы уже определились, что нам не подходят ассиметричные алгоритмы шифрования, поэтому в работе мы будем использовать симметричный алгоритм «Кузнечик». Пользовательский пароль хранится в IndexedDB в таблице user в хэшированном виде, выбранный алгоритм хэширования – SHA-256.

Однако, при таком шифровании, пользователю придётся каждый раз подтверждать паролем любое чтение и запись в БД, что не соответсвует цели расширения облегчить и упростить работу пользователя в интернете. Чтобы этого избежать, мы решили хранить пароль в открытом виде в Session Storage и обязывать пользователя выполнять авторизацию в расширении каждый раз, когда он открывает браузер, и автоматически выходить из аккаунта соответственно. Объекты Session Storage очищаются каждый раз, когда пользователь закрывает браузер. Таким образом, пароль в открытом виде хранится в системе только в тот момент, когда пользователь находится за компьютером, и злоумышленник не может получить доступ к ПК физически, чтобы скопировать данные. Остаётся случай, когда пользователя во время работы пытаются атаковать через локальную сеть или другие общие сетевые

ресурсы, однако в этом случае информационная безопасность должна обеспечиваться грамотной настройкой доступа и выдачи разрешений в сети.

Для предотвращения внедрений и межсайтовых выполнений сценариев необходимо изолировать данные от команд и запросов, для этого реализовано экранирование спецсимволов при вводе логина и пароля.

Local Storage, Session Storage и IndexedDB придерживаются Same Origin Policy (SOP), которая даёт доступ к хранилищу только ресурсу, который его создал. Для примера, сайт my-site.com создаёт в браузере Local Storage, никакой другой сайт или поддомен my-site.com не сможет получить доступ к Local Storage. Этот механизм реализуется на уровне браузера и защищает от XSS и CSRF-атак.

## 2.7 Интерфейс расширения

После установки расширения в браузер, иконка расширения отображается на панели инструментов браузера. После клика на иконку открывается стартовое окно расширения (рисунок 2.4). В котором пользователю предлагается авторизироваться (кнопка «Войти») либо создать нового пользователя (кнопка «Добавить пользователя»).

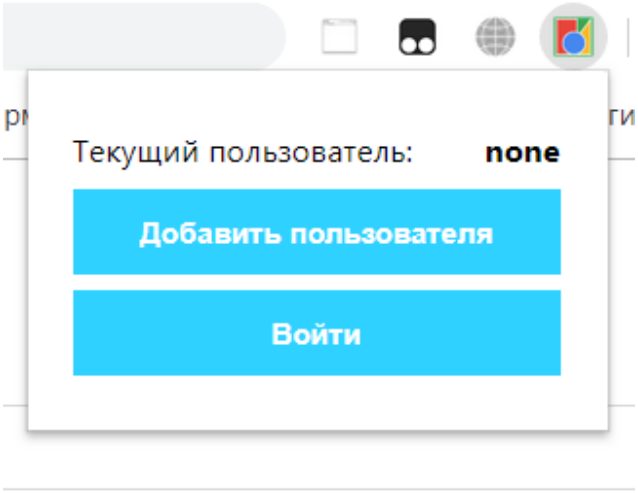


Рисунок 2.4 – Стартовое окно расширения

Окно создания пользователя представлено на рисунке (рисунок 2.5). Для регистрации пользователя достаточно указать будущий логин и пароль.

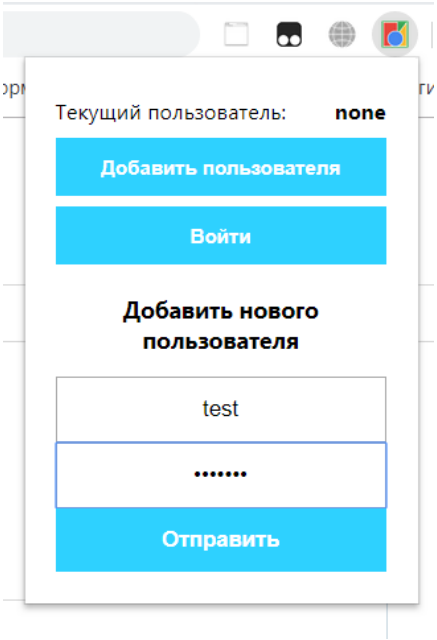


Рисунок 2.5 – Интерфейс добавления нового пользователя

Для начала работы с расширением необходимо в нём авторизироваться. Окно авторизации аналогично окну регистрации (рисунок 2.6).

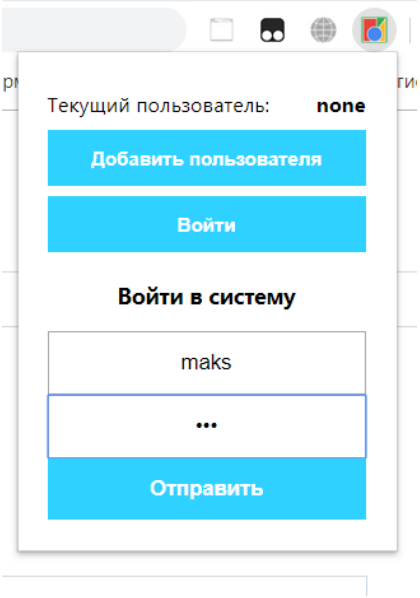


Рисунок 2.6 – Интерфейс авторизации

После авторизации приложение отображает основное окно расширения (рисунок 2.7): здесь отображается имя авторизировавшегося пользователя, кнопка «Выйти», которая возвращает пользователя на стартовое окно (рисунок 4), и кнопка «Настройки».

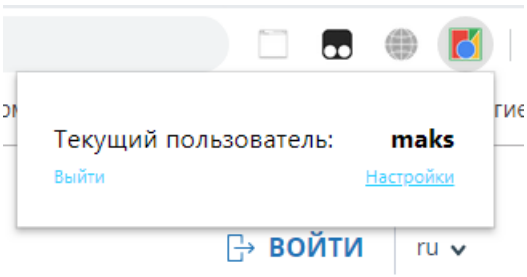


Рисунок 2.7 – Интерфейс расширения
после авторизации

Настройки расширения расположены на отдельной html-странице, которая открывается при нажатии кнопки «Настройки» (рисунок 2.8).

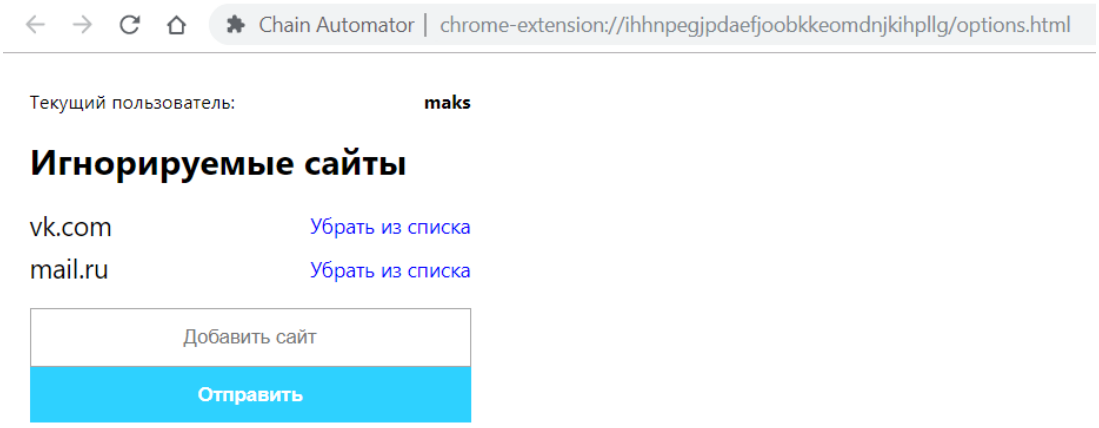


Рисунок 2.8 – Окно настроек расширения

На данной странице пользователь может редактировать список игнорируемых сайтов. Список хранится в IndexedDB в таблице Ignore и отображается на странице.

Когда модуль автоматического выполнения действий решает, что пользователь собирается начать цепочку действий, пользователю предлагают выполнить действия автоматически (рисунок 2.9). Данное окно реализовано стандартной функцией браузера confirm.

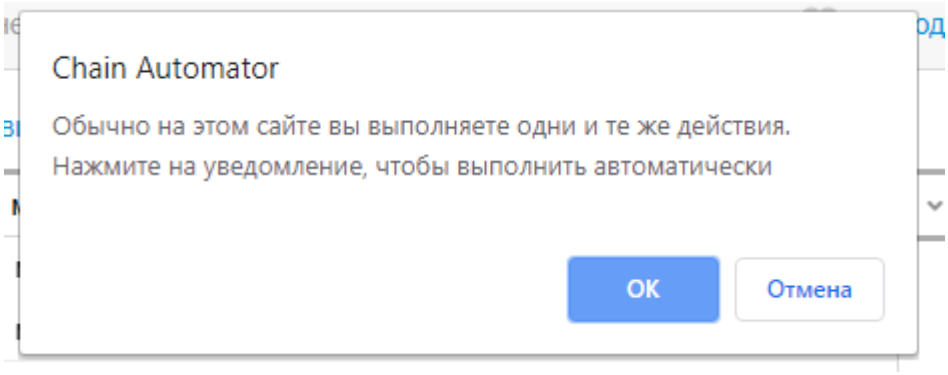


Рисунок 2.9 – Предложение об автоматизации действий

## 2.8 Выводы по разделу

В данном разделе была приведена структурная схема приложения, описан процесс разработки базы данных IndexedDB, описана работа модуля сбора данных, модуля выявления закономерностей, модуля автоматизации, а также рассмотрены способы защиты.

# 3 ТЕСТИРОВАНИЕ РАСШИРЕНИЯ

## 3.1 Проверка корректности работы

Для проверки работоспособности расширения и отладки были написаны три тестовые html-страницы, имитирующие работу интернет-магазина: страница с корзиной (рисунок 3.1), страница с вводом данных (рисунок 3.2) и страница подтверждения заказа (рисунок 3.3).

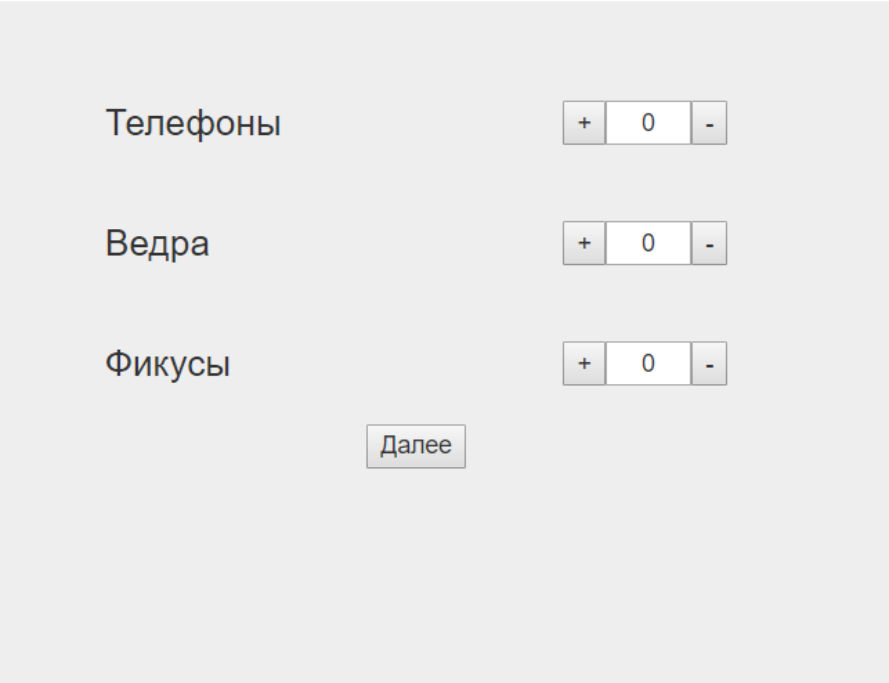


Рисунок 3.1 – Тестовая страница с корзиной

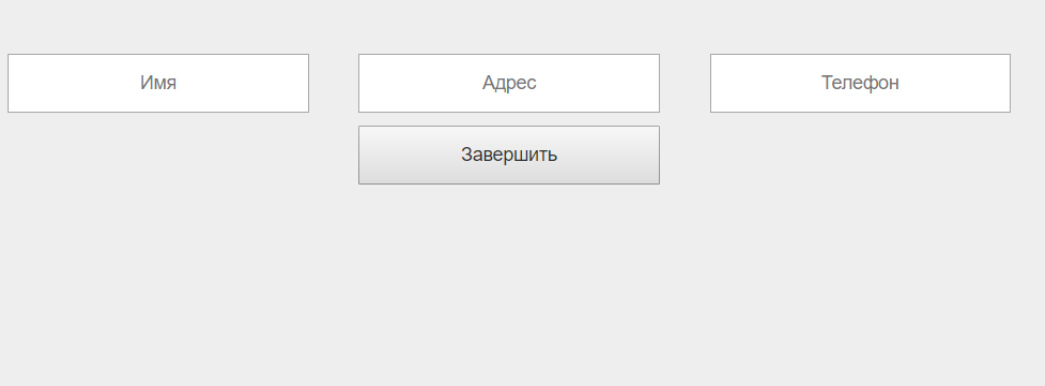


Рисунок 3.2 – Страница с вводом данных


Рисунок 3.3 – Страница с подтверждением заказа

После отладки на тестовых страницах, расширение было опробовано в реальных условиях. Всего в течение 3-х дней было собрано 1000 записей. Действие, которое необходимо автоматизировать – заказ н сайте компании «Люкс Вода» https://l-w.ru/. Цепочка, которую нам необходимо найти, состоит из следующих звеньев:

– https://l-w.ru/#screen-1, главная страница сайта;

– https://l-w.ru/catalog/voda/, каталог сайта;

– https://l-w.ru/personal/cart/, корзина;

– https://l-w.ru/personal/order/make/, страница с оформлением заказа.

Каждый раз мы заказывали одно и то же количество воды, 3 бутылки по 9 литров, на один и тот же адрес. В результате работы расширения данная цепочка и сохранённые данные были выявлены корректно и при следующем посещении сайта, определилось вхождение в цепочку. Также была выявлена цепочка на сайте vk.com, однако автоматизация работы для соцсетей в данной работе не рассматривалась.

## 3.2 Тестирование защищённости

Для предотвращения внедрений и межсайтовых выполнений сценариев было реализовано экранирование спецсимволов при вводе логина и пароля.

Для предотвращения некорректной аутентификации при создании пароля была были введены условия минимальных требований безопасности для паролей:

– не мене 8 символов;

– минимум одна заглавная и одна прописная буква;

– пароль содержит минимум одну цифру.

При несоответствии этим условиям расширение не даёт возможности создать пользователя и выводит сообщение (рисунок 3.4).

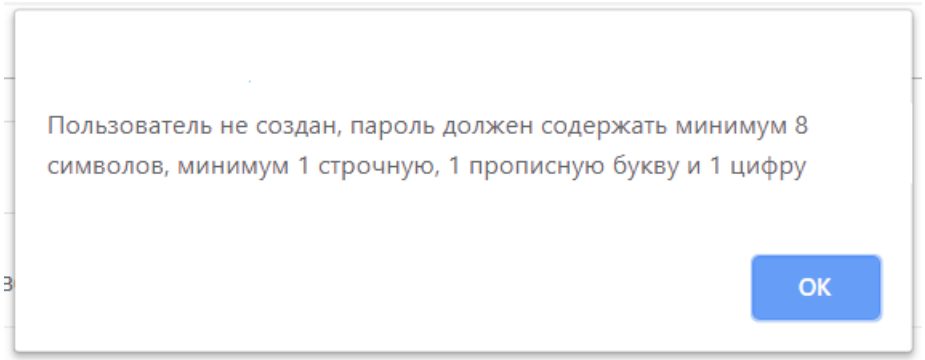


Рисунок 3.4 – Сообщение о несоответствии паролей

Для предотвращения разглашения конфиденциальных данных было реализовано шифрование данных. При попытке просмотреть её содержимое через браузер или в скопированном файле, мы не увидели там данных в открытом виде.

## 3.3 Выводы по разделу

В данном разделе проведено тестирование корректности работы расширения и тестирование защищённости. Для проверки работоспособности были написаны тестовые примеры, имитирующие поведение реального сайта. После расширение было опробовано на реальном примере. Во время тестирования защищённости были произведены попытки просмотреть зашифрованную базу данных, которые не привели к успеху.

# ЗАКЛЮЧЕНИЕ

В результате проделанной работы было разработано расширение для браузера Google Chrome, прогнозирующее поведение пользователя на web-странице. Были рассмотрены существующие системы и механизмы автоматизации действий на основе ранее использованных данных, способы хранения данных на стороне клиента. После сравнения характеристик и с учётом актуальности технологии хранения данных была выбрана база данных IndexedDB. В качестве алгоритма шифрования, наиболее подходящим для данной работы был выбран алгоритм «Кузнечик», для хэширования – алгоритм SHA-256.

Далее в работе была спроектирована база данных, разработаны 3 модуля: сбора данных, выявления закономерностей действий и автоматизации действий, разработана защита расширения. Для выявления закономерностей нами был разработан собственный алгоритм. При проектировании защиты мы опирались на топ-10 уязвимостей OWASP, опираясь на актуальную версию от 2017 года. На последней стадии работы было проведено тестирование корректности работы расширения и тестирование защищённости.

# БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1 The HTML autocomplete attribute [Электронный ресурс]. URL: https://developer.mozilla.org/en-US/docs/Web/HTML/Attributes/autocomplete (дата обращения 25.05.19).

2 Автозаполнение форм [Электронный ресурс]. URL: https://support.google.com/chrome/answer/142893?co=GENIE.Platform%3DDesktop&hl=ru (дата обращения 25.05.19).

3 Manage saved passwords [Электронный ресурс]. URL: https://support.google.com/chrome/answer/95606?co=GENIE.Platform%3DDesktop&hl=en (дата обращения 25.05.19).

4 Global browser usage statistics [Электронный ресурс]. URL: http://gs.statcounter.com/ (дата обращения 25.05.19).

5 Web Storage Concepts and Usage [Электронный ресурс]. URL: https://developer.mozilla.org/en-US/docs/Web/API/Web_Storage_API (дата обращения 25.05.19).

6 Tools for web-developers [Электронный ресурс]. URL: https://developers.google.com/web/tools/chrome-devtools/storage/cookies (дата обращения 25.05.19).

7 WebSQLDatabase [Электронный ресурс]. URL: https://www.w3.org/TR/webdatabase/ (дата обращения 25.05.19).

8 Web Storage Concepts and Usage [Электронный ресурс]. URL: https://developer.mozilla.org/en-US/docs/Web/API/Web_Storage_API (дата обращения 25.05.19).

9 IndexedDB documentation [Электронный ресурс]. URL: https://developer.mozilla.org/ru/docs/IndexedDB (дата обращения 25.05.19).

10 Kimak, S. HTML5 IndexedDB Encryption: Prevention against Potential Attacks / S. Kimak, J. Ellman // International Journal of Intelligent Computing Research (IJICR). – 2016. V.65, №2. – P. 117-123.

11 Kimak, S. Some Potential Issues with the Security of HTML5 IndexedDB / S. Kimak, J. Ellman, C. Laing // International Journal of Intelligent Computing Research (IJICR). – 2017. V.65, №10. – P. 64-75.

12 Federal Information Processing Standards Publication 197 [Электронный ресурс]. URL: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf (дата обращения 25.05.19).

13 ГОСТ Р 34.12–2015 [Электронный ресурс]. URL: http://docs.cntd.ru/document/1200121984/ (дата обращения 25.05.19).

14 OWASP Top 10 – 2017: The Ten Most Critical Web Application Security Risks [Электронный ресурс]. URL: https://www.owasp.org/images/7/72/OWASP_Top_10-2017_%28en%29.pdf.pdf (дата обращения 25.05.19).

15 Cross-OriginXMLHTTP-Request [Электронный ресурс]. URL: https://developer.chrome.com/extensions/xhr (дата обращения, 25.05.19).

16 Recognizing user behavior patterns [Электронный ресурс]. URL: http://ui-patterns.com/patterns/Recognize-patterns (дата обращения 25.05.19).

17 Declare Permissions [Электронный ресурс]. URl: https://developers.chrome.com/extensions/declare_permissions (дата обращения 25.05.19).

18 Same-origin policy – веб-безопасность [Электронный ресурс]. https://developer.mozilla.org/ru/docs/Web/Security/Same-origin_policy (дата обращения 25.05.19)

# ПРИЛОЖЕНИЯ

## ПРИЛОЖЕНИЕ 1

### Текст программы

**Manifest.json**

```
{
  "name": "Chain Automator",
  "version": "1.0",
  "description": "",
  "permissions":      ["activeTab",      "declarativeContent",
"storage", "unlimitedStorage", "tabs", "alarms", "notifications"],
  "background": {
    "scripts": ["background.js"],
    "persistent": false
  },
  "page_action": {
    "default_popup": "popup.html",
    "default_icon": {
      "16": "images/get_started16.png",
      "32": "images/get_started32.png",
      "48": "images/get_started48.png",
      "128": "images/get_started128.png"
    }
  },
  "icons": {
    "16": "images/get_started16.png",
    "32": "images/get_started32.png",
    "48": "images/get_started48.png",
    "128": "images/get_started128.png"
  },
  "options_page": "options.html",
  "manifest_version": 2
}
```

**Popup.html**

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>Google Extension</title>
    <link rel="stylesheet" href="style.css">
</head>
<body>

    <div class="info"></div>
```

```
        <a id="get-test" href="#"></a>

        <div class="current-user">
            <div                class="current-user-info">Текущий
пользователь:</div>
            <div id="user"></div>
        </div>
        <div class="user-bar">
            <div class="logout">Выйти</div>
            <div class="settings">Настройки</div>
        </div>

        <div class="windows-container">
            <div class="buttons" id="buttons">
                <button                  class="addUser">Добавить
пользователя</button>
                <button class="logIn">Войти</button>
            </div>


            <form id="addUser" action="#">
                <h3         class="title">Добавить         нового
пользователя</h3>
                <input  type="text"  name="name"  placeholder="Имя
пользователя">
                <input       type="password"       name="password"
placeholder="Пароль">
                <!--<input   type="text"   name="confirm-password"
placeholder="Подтвердите пароль">-->
                <input type="submit">
            </form>

            <form action="#" id="logIn">
                <h3 class="title">Войти в систему</h3>
                <input  type="text"  name="name"  placeholder="Имя
пользователя">
                <input       type="password"       name="password"
placeholder="Пароль">
                <input type="submit">
            </form>
        </div>



        <button id="changeColor"></button>
        <script src="jquery.min.js"></script>
        <script src="sha256.js"></script>
        <script src="popup.js"></script>
        <script src="background.js"></script>
    </body>
    </html>
```

**Popup.js**

```
$(document).ready(function () {
    $('button.addUser').on('click', function () {
        $('form#addUser').css('display', 'flex');
        $('form#logIn').hide();
    })

    $('button.logIn').on('click', function () {
        $('form#addUser').hide();
        $('form#logIn').css('display', 'flex');
    })

});

let userNameDiv = document.getElementById('user');
let buttonsContainer = document.getElementById('buttons');
var addUserFrom = document.getElementById('addUser');
var logInFrom = document.getElementById('logIn');

$('.settings').on('click', function () {
    chrome.runtime.openOptionsPage();
})

$('.logout').on('click', function () {
    chrome.storage.sync.set({userName: 'none'});
    userNameDiv.innerHTML = 'none';
    $(this).parent().hide();
    buttonsContainer.style.display = 'flex';
})

chrome.storage.sync.get('userName', function(data) {
    if(data.userName != 'none'){
        window.userName=data.userName;
        buttonsContainer.style.display = 'none';
        document.querySelector('.user-bar').style.display    =
'flex';
    }
    userNameDiv.innerHTML = data.userName;
});

addUserFrom.addEventListener('submit', function (e) {
    e.preventDefault();
    var request = window.indexedDB.open("extensions");

    request.onerror = function(err) {
```

```
            console.log(err);
        };

        request.onsuccess = function(event) {
            var db = event.target.result;
            var     transaction     =     db.transaction(["users"],
"readwrite");
            var objectStore = transaction.objectStore("users");
            objectStore.add({                                    name:
addUserFrom.elements['name'].value,                          password:
sha256(addUserFrom.elements['password'].value)})
            addUserFrom.elements['name'].value = '';
            addUserFrom.elements['password'].value = '';
            alert('Новый пользователь успешно добавлен');
            addUserFrom.style.display = "none";
        };
    });

    logInFrom.addEventListener('submit', function (e) {
        e.preventDefault();

        var request = window.indexedDB.open("extensions");
        request.onerror = function(err) {
            console.log(err);
        };
        request.onsuccess = function(event) {
            var db = event.target.result;
            var     transaction     =     db.transaction(["users"],
"readwrite");
            var objectStore = transaction.objectStore("users");
            var index = objectStore.index('name');
            var userName = logInFrom.elements['name'].value
            index.get(userName).onsuccess  =  function(indexEvent)
{
                if      (indexEvent.target.result.password      ==
sha256(logInFrom.elements['password'].value)){
                    userNameDiv.innerHTML                        =
logInFrom.elements['name'].value;
                    chrome.storage.sync.set({userName:
logInFrom.elements['name'].value});

document.querySelector('#logIn').style.display = 'none';
                    document.querySelector('.user-
bar').style.display = 'flex';
                    logInFrom.elements['name'].value = ''
                    logInFrom.elements['password'].value = ''
                    buttonsContainer.style.display = 'none';
                };
            };
            console.log('Успех');
        };
```

```
    });

    let getList = document.getElementById('get-test');

    getList.addEventListener('click', function (e) {
        e.preventDefault();

        chrome.runtime.sendMessage({greeting:           "hello"},
function(response) {});

        var request = window.indexedDB.open("extensions");

        request.onerror = function(err) {
            console.log(err);
        };

        request.onsuccess = function(event) {
            var db = event.target.result;
            var    transaction    =    db.transaction(["actions"],
"readwrite");
            var objectStore = transaction.objectStore("actions");
            var index = objectStore.index('user, site');
            var arrays = [];
            var indexes = [];

            index.getAll([window.userName,
"www.avito.ru"]).onsuccess = function (event) {

                var objects = event.target.result;

                objects.forEach(function (element, i) {
                    arrays.push(Object.keys(element.object));
                    indexes.push(i);
                });

            };
        };
        console.log('Успех');
    });

    //Пересечение массивов
    function intersectionNew() {
        var result = {};
        for (arr in arguments) {
            for (i = 0; i < arguments[arr].length; i++) {
                var values = arguments[arr][i]
                if (result[values] === undefined) {
                    result[values] = 1
                } else result[values] += 1
            }
        }
```

```
        let answer = []
        for(key in result) {
            if       (result[key]       >=       arguments.length)
answer.push(Number(key))
        }
        return answer;
    };

    function intersection() {
        var result = [];
        var lists;

        if(arguments.length === 1) {
            lists = arguments[0];
        } else {
            lists = arguments;
        }

        for(var i = 0; i < lists.length; i++) {
            var currentList = lists[i];
            for(var y = 0; y < currentList.length; y++) {
                var currentValue = currentList[y];
                if(result.indexOf(currentValue) === -1) {
                    var existsInAll = true;
                    for(var x = 0; x < lists.length; x++) {
                        if(lists[x].indexOf(currentValue)  ===  -1)
{
                            existsInAll = false;
                            break;
                        }
                    }
                    if(existsInAll) {
                        result.push(currentValue);
                    }
                }
            }
        }
        return result;
    };
```

**Background.js**

```
    const dbName = "extensions";

    chrome.runtime.onInstalled.addListener(function() {
        chrome.storage.sync.set({color: '#3aa757'});
        chrome.storage.sync.set({userName: 'none'});

        if (!window.indexedDB) {
            window.alert("Your browser doesn't support a stable
version of IndexedDB. Such and such feature will not be
available.");
```

```
        }

        const dbName = "extensions";
        var request = indexedDB.open(dbName);

        request.onerror = function(err) {
            // Handle errors.
            console.log(err);
        };

        request.onupgradeneeded = function(event) {
            var db = event.target.result;
            var objectStoreUsers = db.createObjectStore("users",
{ keyPath: "id", autoIncrement:true });
            var objectStoreActions =
db.createObjectStore("actions", { keyPath: "id",
autoIncrement:true });
            var objectStoreIgnorePages =
db.createObjectStore("ignore_pages", { keyPath: "id",
autoIncrement:true });
            objectStoreUsers.createIndex("name", "name", {
unique: true });
            objectStoreActions.createIndex('user, site', ['user',
'site']);
            objectStoreIgnorePages.createIndex('user', 'user');
            objectStoreIgnorePages.createIndex('ignore_page',
'ignore_page');

        };


chrome.declarativeContent.onPageChanged.removeRules(undefined,
function() {
            chrome.declarativeContent.onPageChanged.addRules([{
                conditions: [new
chrome.declarativeContent.PageStateMatcher({
                    // pageUrl: {hostEquals: 'www.avito.ru'},
                })
                ],
                actions: [new
chrome.declarativeContent.ShowPageAction()]
            }]);
        });
    });

    chrome.tabs.onUpdated.addListener( function (tabId,
changeInfo, tab) {

        if (changeInfo.status == 'complete' && tab.active) {

            console.log(getSite(tab.url));
```

```
            chrome.storage.sync.get('userName', function(data) {
                if(data.userName != 'none'){

                    if(getSite(tab.url) == 'www.avito.ru'){
                        confirm("Обычно на этом сайте вы
выполняете одни и те же действия. Нажмите на уведомление, чтобы
выполнить автоматически");
                        console.log('all done');
                    }

                    var params = getParams(tab.url);
                    var outputData = {}
                    outputData.user = data.userName;
                    outputData.site = tab.url.split('/')[2];
                    outputData.object = params;

                    var request = indexedDB.open(dbName);

                    request.onerror = function(event) {
                        console.log(err);
                    };

                    request.onsuccess = function(event) {
                        var db = event.target.result;
                        var             transaction             =
db.transaction(["actions"], "readwrite");
                        var             objectStore             =
transaction.objectStore("actions");
                        objectStore.add(outputData);
                    };

                }else{
                    console.log('Пожалуйста, авторизуйтесь');
                }
            });
        }
    })

    //Функция получения GET-параметров
    function getParams (url) {
        var params = {};
        var parser = document.createElement('a');
        parser.href = url;
        var query = parser.search.substring(1);
        var vars = query.split('&');
        for (var i = 0; i < vars.length; i++) {
            var pair = vars[i].split('=');
            params[pair[0]] = decodeURIComponent(pair[1]);
        }
        return params;
```

```
};

function getSite(url) {
    var hostname;
    if (url.indexOf("//") > -1) {
        hostname = url.split('/')[2];
    }
    else {
        hostname = url.split('/')[0];
    }
    hostname = hostname.split(':')[0];
    hostname = hostname.split('?')[0];
    return hostname;
}


Options.html
<!doctype html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport"
          content="width=device-width,       user-scalable=no,
initial-scale=1.0, maximum-scale=1.0, minimum-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Настройки расширения</title>
    <link rel="stylesheet" href="options.css">
</head>
<body>
    <div class="current-user">
        <div                class="current-user-info">Текущий
пользователь:</div>
        <div id="user"></div>
    </div>
    <div>
        <div class="ignore-sites-block">
            <h1>Игнорируемые сайты</h1>
            <ul class="ignore-sites">
            </ul>
        </div>
        <form action="#" id="ignore-form">
            <input  type="text"  placeholder="Добавить  сайт"
name="ignore">
            <input type="submit" value="Отправить">
        </form>
    </div>
    </body>
    <script src="jquery.min.js"></script>
    <script src="options.js"></script>
</html>
```

**Options.js**

```
let currentUser = document.getElementById('user');
var userName;
let ignoreForm = document.getElementById('ignore-form');

chrome.storage.sync.get('userName', function(data) {
    currentUser.innerHTML = data.userName;
    userName = data.userName;
});

var request = window.indexedDB.open("extensions");

request.onerror = function(err) {
    console.log(err);
};

request.onsuccess = function(event) {
    var db = event.target.result;
    var transaction = db.transaction(["ignore_pages"],
"readwrite");
    var objectStore =
transaction.objectStore("ignore_pages");
    var index = objectStore.index('user');

    index.getAll(userName).onsuccess = function(event) {
        var data = event.target.result;
        let sitesContainer = document.querySelector('.ignore-
sites');
        data.forEach(function (item) {
            var deleteItem = document.createElement('div');
            deleteItem.classList.add('delete-item');
            deleteItem.innerHTML = 'Убрать из списка';
            var elem = document.createElement('li');
            elem.innerHTML = item.ignore_page
            elem.appendChild(deleteItem);
            sitesContainer.appendChild(elem);
        })
    };
}

document.addEventListener('click', function(e) {
    e = e || window.event;
    var target = e.target || e.srcElement,
        text = target.textContent || target.innerText;
    if(target.classList.value == 'delete-item'){
        // target.parentElement.remove()
```

```
deleteItem(target.parentElement.firstChild.nodeValue);

console.log(target.parentElement.firstChild.nodeValue);
        }

    }, false);

    function deleteItem(site) {

        var request = window.indexedDB.open("extensions");
        request.onerror = function(err) {
            console.log(err);
        };

        request.onsuccess = function(event) {
            var db = event.target.result;
            var   transaction   =   db.transaction('ignore_pages',
"readwrite");
            var                    objectStore                    =
transaction.objectStore('ignore_pages');
            var index = objectStore.index('ignore_page');
            index.get(site).onsuccess = function(event){
                console.log(event.target.result.id);
                objectStore.delete(event.target.result.id);
                transaction.complete;
            }
            console.log('Успех');
        }
    };

    ignoreForm.addEventListener('submit', function (e) {
        e.preventDefault();

        var request = window.indexedDB.open("extensions");
        request.onerror = function(err) {
            console.log(err);
        };
        request.onsuccess = function (event) {
            var db = event.target.result;
            var  transaction  =  db.transaction(["ignore_pages"],
"readwrite");
            var                    objectStore                    =
transaction.objectStore("ignore_pages");
            objectStore.add({   user:   userName,   ignore_page:
ignoreForm.elements['ignore'].value});
            ignoreForm.elements['ignore'].value = '';
            alert('Игнорируемая страница успешно добавлена');
        };
    });
```

**Optinons.css**

```
body{
    padding: 10px;
    width: 300px;
}

form{
    display: none;
}

button#changeColor {
    height: 0px;
    width: 0px;
    outline: none;
    display: none;
}

#user{
    font-weight: bold;
    display: flex;
}

.buttons{
    display: flex;
    flex-direction: column;
}

.buttons button{
    margin: 3px 0;
    padding: 10px;
    font-size: 12px;
    font-weight: bold;
    background: #2ed1ff;
    border: 1px solid #2ed1ff;
    color: #fff;
}

form{
    flex-direction: column;
}

form input{
    padding: 10px;
    text-align: center;
}

form input[type='submit']{
    background: #2ed1ff;
    border: 1px solid #2ed1ff;
    padding: 10px;
    color: #fff;
```

```
    font-weight: bold;
}

.current-user{
    font-size: 13px;
    margin: 5px 0;
    display: flex;
    flex-direction: row;
    justify-content: space-between;
}


form#ignore-form{
    margin-top: 15px;
}

li{
    font-size: 18px;
    list-style: none;
    margin: 5px 0;
    display: flex;
    justify-content: space-between;
    align-items: center;
}

ul{
    padding: 0;
}

.delete-item{
    font-size: 14px;
    color: blue;
    cursor: pointer;
}

.delete-item:hover{
    text-decoration: underline;
}

form#ignore-form{
    display: flex;
}
```

ПРИЛОЖЕНИЕ 2

Руководство пользователя

После установки расширения, его иконка отобразится в интерфейсе браузера рядом с другими установленными расширениями. При щелчке по иконке откроется главное окно расширения (рисунок 2.1)

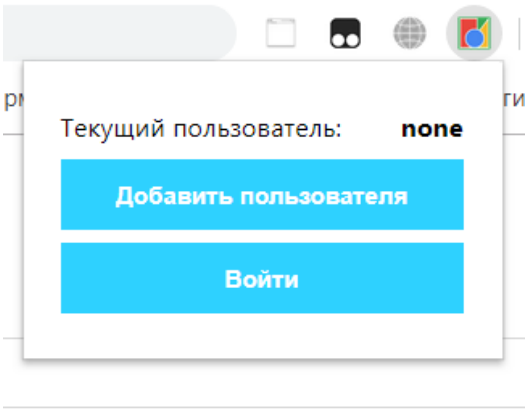


Рисунок П2.1 – Главное окно расширения

Если вы до этого не работали с расширением, вам необходимо зарегистрироваться. Для этого необходимо нажать кнопку «добавить пользователя». После этого появятся два поля для ввода данных нового пользователя: логин и пароль (рисунок 2.2)

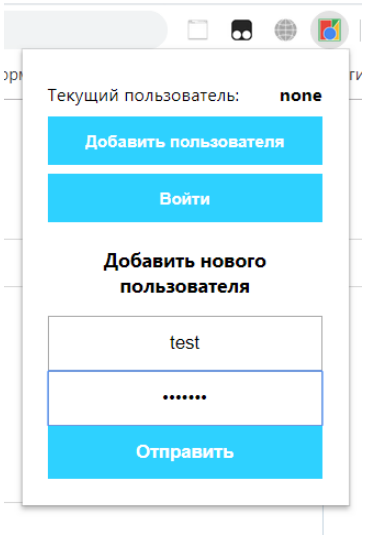


Рисунок П2.2 – Интерфейс регистрации пользователя

После создания пользователя (или в случае если вы регистрировались ранее) необходимо нажать кнопку «Войти» и пройти авторизацию (рисунок 2.3).

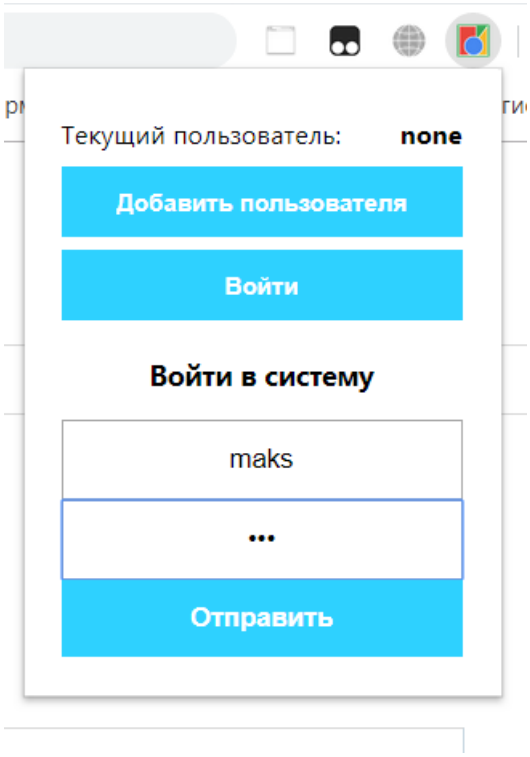


Рисунок П2.3 – Интерфейс авторизации
в расширении

После авторизации в расширении главное окно программы принимает следующий вид (рисунок 2.4).

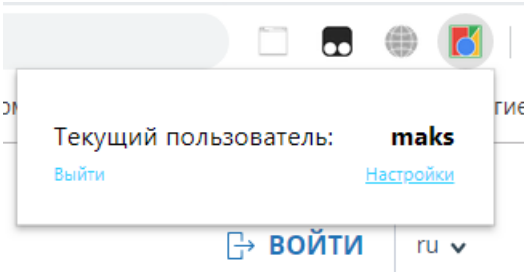


Рисунок П2.5 – Главное окно расширения
после авторизации

Если вы впервые начали работу с расширением, рекомендуется настроить список игнорируемых ресурсов в окне настройки (рисунок 2.6).

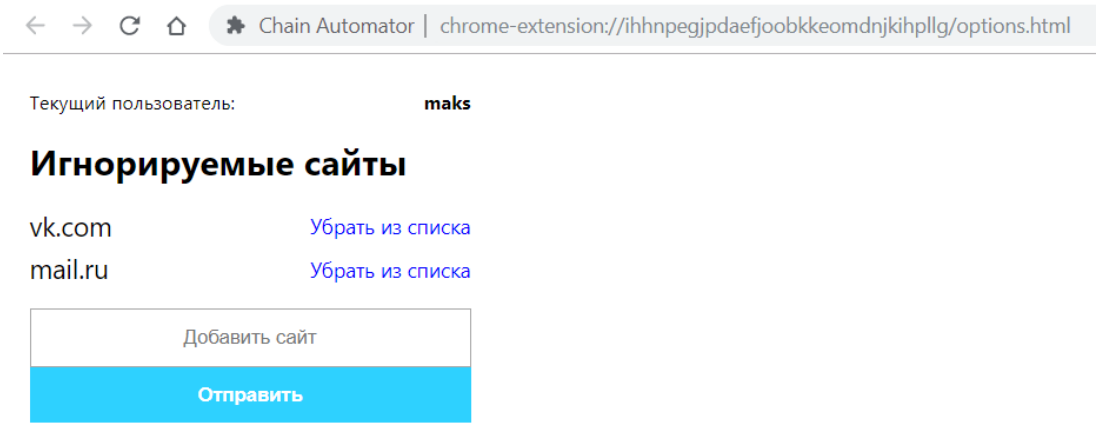


Рисунок П2.6 – Настройки расширения

После того как произведена настройка игнорируемых сайтов (к которой можно вернуться в любой момент). От пользователя не требуется никаких дополнительных действий. Программа продолжит собирать данные и анализировать их, выявлять цепочки для автоматизации. В случае если пользователь начнёт выполнять цепочку, известную расширению, на странице появится окно с предложением автоматизировать действия (рисунок 2.7).

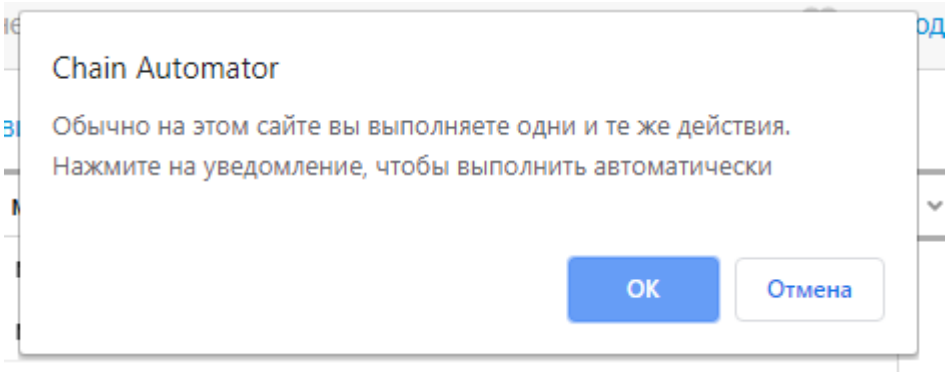


Рисунок П2.7 – Предложение об автоматизации действий