Оглавление

[Оглавление 2](#_Toc163315195)

[Введение 4](#_Toc163315196)

[ГЛАВА 1. Обзор темы ОТКРЫТОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ 7](#_Toc163315197)

[1.1. Понятие открытого программного обеспечения 7](#_Toc163315198)

[1.2. Особенности программного обеспечения в финансовой сфере 10](#_Toc163315199)

[1.3. Риски, связанные с уязвимостями в финансовом программном обеспечении 15](#_Toc163315200)

[1.4. Примеры уязвимостей в финансовом программном обеспечении 17](#_Toc163315201)

[1.4.1. Уязвимости в коде приложения 18](#_Toc163315202)

[1.4.2. Уязвимости в библиотеках с открытым исходным кодом 27](#_Toc163315203)

[1.4.3. Уязвимости окружения 29](#_Toc163315204)

[1.4.4. Уязвимости в онлайн-банкинге 30](#_Toc163315205)

[1.4.5. Уязвимости в платежных системах и торговых платформах 31](#_Toc163315206)

[1.4.6. Уязвимость в системах управления финансовыми активами 32](#_Toc163315207)

[1.4.7. Уязвимости в алгоритмах шифрования: 32](#_Toc163315208)

[1.4.8. Отказ в обслуживании (DoS) и распределенные атаки на отказ обслуживания (DDoS) 32](#_Toc163315209)

[1.5. Методы предотвращения уязвимостей 33](#_Toc163315210)

[Выводы по главе 1 41](#_Toc163315211)

[ГЛАВА 2. МЕТОДОЛОГИЯ ЗАЩИЩЕННОСТИ ОТКРЫТОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ДЛЯ ФИНАНСОВОЙ СФЕРЫ 42](#_Toc163315212)

[2.1. Цикл оценки защищенности открытого программного обеспечения для финансовой сферы 42](#_Toc163315213)

[2.2. Методика поиска и устранения уязвимостей в открытом программном обеспечении для финансовой сферы 47](#_Toc163315214)

[Выводы по главе 2 50](#_Toc163315215)

[ГЛАВА 3. Практическая часть: РЕАЛИЗАЦИЯ СКРИПТА АВТОМАТИЗИРОВАННОГО ПОИСКА УЯЗВИМОСТЕЙ в открытом программном обеспечении ФИНАНСОВОй сферы 51](#_Toc163315216)

[3.1 Описание среды 51](#_Toc163315217)

[3.1.1 Виртуальная машина - веб-приложение UnsafeBank 52](#_Toc163315218)

[3.1.2 Виртуальная машина – сканер приложения 59](#_Toc163315219)

[3.2 Разработка и тестирование скрипта поиска и устранения уязвимостей 64](#_Toc163315220)

[Выводы по главе 3 66](#_Toc163315221)

[ГЛАВА 4. Оценка экономической составляющей работы 67](#_Toc163315222)

[Выводы по главе 4 71](#_Toc163315223)

[ЗАКЛЮЧЕНИЕ 72](#_Toc163315224)

[ПРИЛОЖЕНИЕ 1. Программа сканирования На Уязвимости 73](#_Toc163315225)

[Список литературы 79](#_Toc163315226)

# Введение

В современном мире финансовая сфера занимает важное место в экономике, и ее надежное функционирование является критически важным для обеспечения стабильности и безопасности мировой финансовой системы. Однако, в условиях всеобщей цифровизации и широкого использования информационных технологий, финансовые организации сталкиваются с рядом угроз и вызовов, связанных с безопасностью информации и программного обеспечения.

Банки стремятся использовать передовые технологии, чтобы дать своим клиентам возможность удаленно управлять банковскими продуктами. Для получения большинства банковских услуг достаточно оформить карту банка и заключить договор о дистанционном банковском обслуживании (ДБО). Все чаще появляются банки, которые вовсе не предлагают клиентам привычное обслуживание в территориально распределенных филиалах, а оказывают полный спектр услуг удаленно. У такого подхода множество преимуществ: для пользователей это, прежде всего, удобство и скорость получения услуги; для банка - сокращение затрат на содержание филиалов, сокращение штата. Решается проблема очередей, бюрократических проволочек - и, как следствие, сокращается количество недовольных клиентов.

Однако существует и обратная сторона медали - необходимость обеспечивать безопасность и минимизировать риски, связанные с использованием систем ДБО. Это общедоступные веб и мобильные приложения, и для них характерны все соответствующие уязвимости и угрозы информационной безопасности (в частности, рассматриваемые в классификации Web Application Security Consortium Threat Classification - WASC TC v. 2). К примеру, наиболее опасной угрозой в случае реализации атак на системы ДБО можно назвать хищение денежных средств. Также среди специфических угроз можно выделить несанкционированный доступ к данным платежных карт, персональным данным пользователей и банковской тайне, отказ в обслуживании банковского приложения; есть и другие угрозы, реализация которых может привести к существенным финансовым и репутационным потерям.

Проблематика безопасности в финансовой сфере охватывает множество аспектов, начиная от защиты конфиденциальности клиентских данных и финансовых транзакций до обеспечения непрерывной доступности критически важным финансовым системам. Одним из основных элементов обеспечения безопасности информации является применение эффективных методов поиска и устранения уязвимостей в программном обеспечении, используемом финансовыми учреждениями.

Актуальность выбранной темы обусловлена рядом факторов. Во-первых, финансовая сфера постоянно подвергается атакам со стороны злоумышленников, которые стремятся получить доступ к чувствительным данным или пытаются нарушить нормальное функционирование финансовых систем. Во-вторых, использование открытого программного обеспечения (Open Source Software, OSS) в финансовой сфере становится все более распространенным из-за его гибкости, экономической эффективности и широким функционалом, но при этом может приводить к повышению уровня уязвимостей из-за возможности анализа его исходного кода злоумышленниками.

Безопасность финансового программного обеспечения становится ключевым фактором для обеспечения доверия клиентов, сохранения репутации финансовых институтов и предотвращения финансовых потерь. Поэтому разработка и применение эффективных методов поиска и устранения уязвимостей в открытом программном обеспечении для финансовой сферы является важной задачей современной информационной безопасности.

Целью данной выпускной квалификационной работы является исследование и разработка эффективного практического метода поиска и устранения уязвимостей в открытом программном обеспечении для финансовой сферы с целью повышения уровня безопасности финансовых систем и данных. Для достижения этой цели будут рассмотрены следующие вопросы:

- особенности открытого программного обеспечения,

- анализ рисков и угроз, связанных с уязвимостями в финансовом программном обеспечении,

- описание цикла оценки защищенности открытого программного обеспечения для финансовой сферы, включая этапы оценки текущего состояния безопасности, определения требований к безопасности, разработки методики поиска и устранения уязвимостей, обучения персонала, регулярного сканирования и аудита безопасности, применения исправлений и обновлений, мониторинга и отчетности,

- пример реализации методики/политики для поиска и устранения уязвимостей в открытом программном обеспечении финансовой сферы, включая требования, правила работы сотрудников и используемые для этого инструменты.

Индивидуальной частью данного выпускного проекта является практическая работа по реализации скрипта автоматизированного поиска уязвимостей в открытом программном обеспечении, используемом в финансовой сфере.

Результаты данной работы будут полезны как для специалистов по информационной безопасности, работающих в финансовой сфере, так и для руководителей таких учреждений, которые заинтересованы в обеспечении безопасности своих систем и данных. В конечном итоге, данное исследование поможет повысить уровень защиты данных и операционной стабильности финансовых организаций в условиях постоянно меняющегося информационного ландшафта и угроз.

# ГЛАВА 1. Обзор темы ОТКРЫТОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

## 1.1. Понятие открытого программного обеспечения

Открытое программное обеспечение (Open Source Software, OSS) - это программное обеспечение, распространяемое с открытым исходным кодом, который доступен для просмотра, изменения и распространения. Для обеспечения информационной безопасности это имеет ряд значимых последствий.

Во-первых, прозрачность и доступность исходного кода позволяют экспертам проводить аудит данного ПО на предмет уязвимостей и других проблем безопасности. Это означает, что сообщество разработчиков может быстро обнаруживать и исправлять уязвимости, улучшая безопасность программного обеспечения.

Во-вторых, наличие активного сообщества разработчиков и пользователей способствует быстрой реакции на обнаруженные уязвимости. Найденные уязвимости могут быть быстро переданы разработчикам, что позволяет им выпустить обновления и патчи для исправления проблем безопасности.

В-третьих, возможность независимой верификации безопасности программного обеспечения стимулирует конкуренцию между разработчиками и улучшает качество безопасности ПО.

В-четвертых, открытое ПО обеспечивает возможность аудита кода и анализа безопасности перед его использованием в критических системах, что повышает уровень доверия к программному обеспечению.

В-пятых, использование открытого ПО позволяет избегать зависимости от конкретных поставщиков и продуктов. Это означает, что управлять своими системами можно более гибко, а также имеется возможность внедрять изменения и исправления без ожидания реакции со стороны поставщика.

В-шестых, одной из ключевых особенностей открытого ПО является его бесплатность. Пользователи могут свободно скачивать, устанавливать и использовать открытое ПО без необходимости платить за лицензию. Это делает открытое ПО особенно привлекательным для небольших компаний, стартапов и некоммерческих организаций, которые часто сталкиваются с ограниченным бюджетом на приобретение лицензий на коммерческое ПО.

Также, открытое ПО можно адаптировать и модифицировать в соответствии с собственными потребностями, что обеспечивает высокий уровень контроля над безопасностью (например, веб-сервер Apache, который является одним из самых популярных серверов в мире, предоставляет открытый исходный код и широкие возможности для настройки и расширения функционала).

Как видим, использование открытого программного обеспечения очень привлекательно для применения в своей деятельности. Однако, помимо многочисленных преимуществ, открытое ПО сопряжено с определенными рисками и угрозами безопасности.

К минусам открытого программного обеспечения можно отнести:

* риск намеренной порчи кода;
* более низкая защищённость открытого ПО в отличие от закрытого;
* риск исчезновения модифицированной версии продукта при отказе конкретного разработчика её обслуживать;
* сопоставимая стоимость в обслуживании с проприетарным ПО.

Также имеются требования к лицензиям на открытое ПО:

1. Свободное распространение. Это значит, что лицензия не должна налагать ограничений на продажу и распространение ПО.

2. Доступные исходные тексты. Даже если ПО не поставляется с исходными текстами, эти тексты должны быть легко доступны.

3. Возможность модификации. Простая возможность читать исходные тексты не позволяет экспериментировать с ними и выпускать модификации

4. Даже в случае неприкосновенности авторского исходного текста, производные программы и их исходные тексты должны свободно распространяться.

5. Отсутствие дискриминации против людей и групп людей. Некоторые страны, например, США, имеют некоторые ограничения на экспорт ПО.

6. Отсутствие дискриминации по цели применения. Свободная лицензия должна разрешать все виды деятельности, включая генетические и ядерные исследования, коммерческое применение и т. д

7. Распространение лицензии. Права, связанные с Открытым ПО, должны быть применимы ко всем пользователям программы без заключения дополнительных соглашений, например, соглашения о неразглашении.

8. Лицензия не должна ограничивать другие программные продукты. За исключением банальной несовместимости, пользователь имеет право выбирать, чем пользоваться.

9. Лицензия должна быть технологически нейтральной. То есть, лицензия не должна требовать что-либо от интерфейса или технологий, применяемых в производной программе.

10. Лицензия не должна быть привязана к конкретному продукту. Права на программный код не должны зависеть от того, является ли программа частью какого-то продукта. Человек, распространяющий программу в отрыве от сборника или перенёсший часть кода в другой продукт, имеет такие же права, какие давал сборник.

В зоне ответственности IT-службы любой кредитно-финансовой организации - повышение надежности и доступности банковских систем при снижении стоимости владения средствами мониторинга, анализа, связи и инфраструктурных решений. Добиться этого можно тоже с помощью открытых продуктов, перечислим некоторые из них:

* мониторинг IT-инфраструктуры - решения на основе Zabbix, Prometheus;
* мониторинг прикладных систем - ElasticSearch, Logstach, Kibana - связки продуктов, ставшей уже классическим решением;
* шины взаимодействия между Enterprise-системами - RabbitMQ, Apache Kafka;
* NoSQL базы данных и кэш - Redis, Apache Cassandra;
* контейнерные платформы - Red Hat OpenShift;
* СУБД - PostgreSQL, MySQL;
* операционные системы - Linux.

Рассмотрим требования и особенности к программному обеспечению, используемому в финансовой сфере.

## 1.2. Особенности программного обеспечения в финансовой сфере

Программное обеспечение, используемое в финансовой сфере, имеет ряд особенностей из-за высокой степени конфиденциальности и чувствительности данных, с которыми оно работает. К основным особенностям относят высокую надежность и стабильность, строгие требования к безопасности, высокую скорость обработки данных и соответствие регулированиям и нормативным актам. Рассмотрим их более подробно.

*Высокие требования к надежности и стабильности*: программное обеспечение обязано обеспечивать надежность и стабильность работы в любых условиях. Даже минимальное отклонение или сбой может привести к серьезным последствиям. Поэтому разработчики стремятся к тщательному тестированию, включая функциональное и нагрузочное тестирование, чтобы гарантировать его надежность.

*Сложная архитектура и масштабы:* финансовые системы обычно имеют сложную архитектуру и масштабы из-за высокой степени интеграции с другими системами и сервисами. Это может привести к повышенной сложности разработки, обнаружения и устранения уязвимостей в программном обеспечении. Большое количество компонентов и их взаимодействие могут создавать новые уязвимости и сложности при их обнаружении и устранении.

*Строгие регуляторные требования и правила безопасности:* финансовые учреждения подчиняются строгим регуляторным требованиям и стандартам безопасности, установленным соответствующими органами и ассоциациями (например, PCI DSS (Payment Card Industry Data Security Standard) для обработки платежных карт), а также законодательство в области финансовой сферы часто устанавливает жесткие требования к защите конфиденциальных данных клиентов. Например, в России действует Федеральный закон «О персональных данных», который устанавливает обязанности по обеспечению безопасности персональных данных и наказание за их утечку или несанкционированный доступ.

Эти требования касаются защиты конфиденциальных данных клиентов предотвращения мошенничества и обеспечения безопасности транзакций. Разработка и использование программных средств в финансовой сфере должны строго соответствовать этим нормативам и стандартам.

*Необходимость интеграции с другими системами и сервисами:* финансовое ПО часто используется в комплексных системах, включающих в себя не только банковские системы, но и торговые платформы, платежные шлюзы, системы аналитики и многое другое. Интеграция с другими системами повышает поверхность атак и уязвимости, поэтому необходимо уделить особое внимание защите и безопасности передачи данных между системами.

*Сложность обновления и совместимости:* в финансовой сфере даже незначительные изменения в ПО могут иметь серьезные последствия. Обновления должны быть тщательно протестированы и совместимы с другими системами, чтобы избежать непредвиденных проблем и сбоев. Поэтому разработчики и администраторы финансовых систем должны тщательно планировать и координировать процессы обновления, чтобы минимизировать риски.

Рассмотрим на примере архитектуру автоматизированной банковской системы:

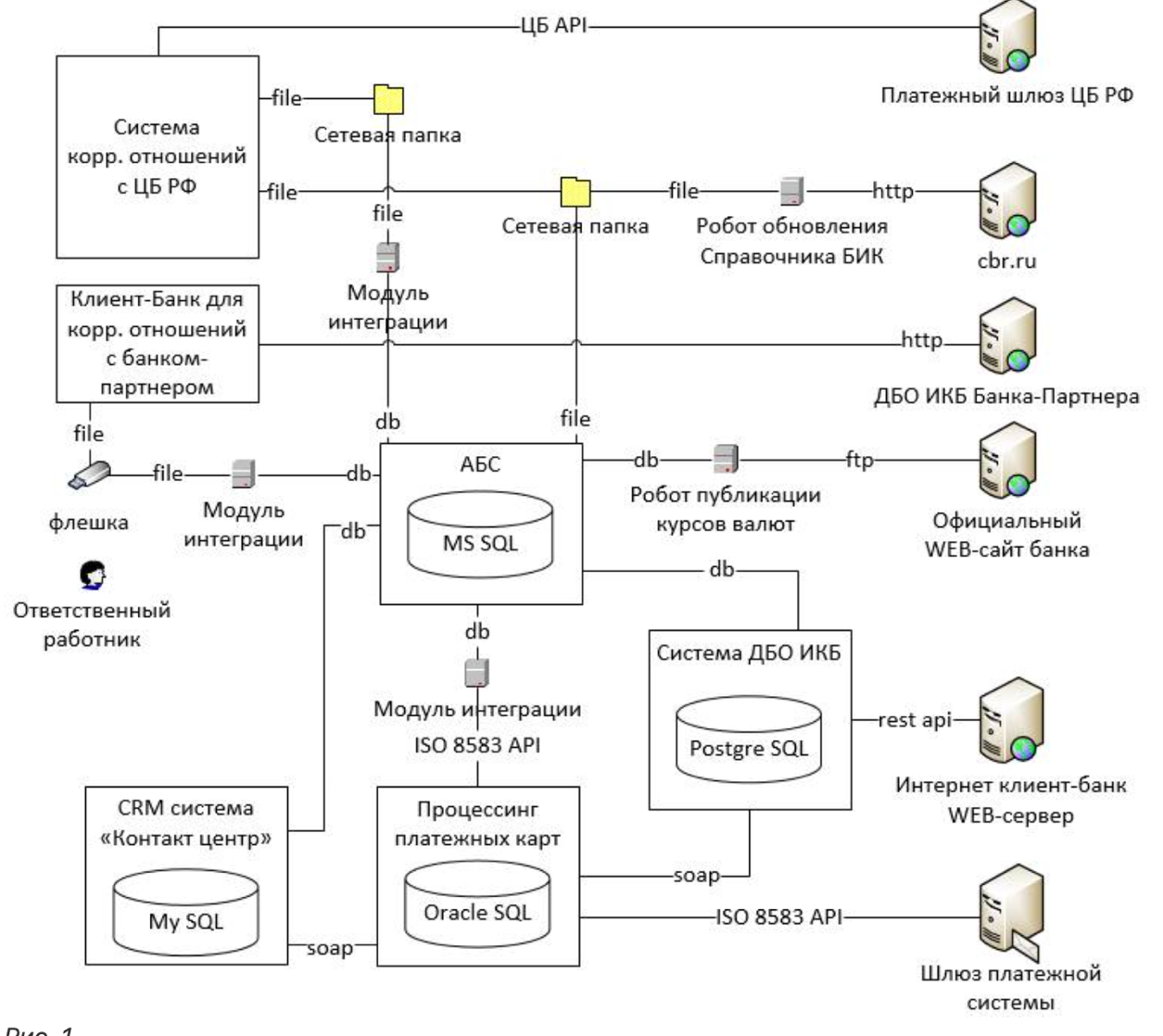


Рис. 1.1. Автоматизированная банковская система

Ядром информационной инфраструктуры любого банка является автоматизированная банковская система или сокращенно АБС.

Данное определение позволяет подвести под него практически любую IT-систему в банке. В то же время обычные банковские служащие называют АБСту систему, которая занимается учетом банковских счетов, проводок между ними (движением денежных средств) и остатков. Второе определение не противоречит первому и более четко его детализирует, им и будем пользоваться дальше.

Если посмотреть на эту схему (Рис.1.1) с точки зрения осуществления безналичных расчетов, то можно увидеть, что банк реализует их при помощи  
прямых корреспондентских отношений с банком-партнером,

* международной платежной системы (МПС) (например, VISA, MasterCard).
* корреспондентских отношений с Банком России.  
  Технически прямые корреспондентские отношения с банками-партнерами могут быть организованы с помощью:
* обычных систем ДБО ИКБ (системы дистанционного банковского обслуживания Интернет Клиент-Банк), применяемых банками для обслуживания юридических лиц (в рассматриваемом примере Рис.1.1), используется именно этот способ);
* межбанковских платежных систем (например, [SWIFT](https://www.swift.com/));
* специализированных систем обмена платежными сообщениями (например, [REX400](http://www.rex400.ru/products/rex400_system/what_is/), [TELEX](http://www.rex400.ru/services/service/telex/));
* специализированного ПО, разработанного одним из взаимодействующих банков.  
   Подключение к платежным системам, обслуживающим пластиковые карты, осуществляется через стандартные модули, входящие в состав процессинговых систем.  
   Для успешного функционирования банк обязан обеспечивать у себя информационную безопасность всех перечисленных способов осуществления платежей. Рассмотреть их в рамках одного, даже крупного исследования весьма проблематично, и поэтому мы рассмотрим на примере наиболее критичном, с точки зрения возможных потерь, направлении - платежном взаимодействии банка с Банком России.

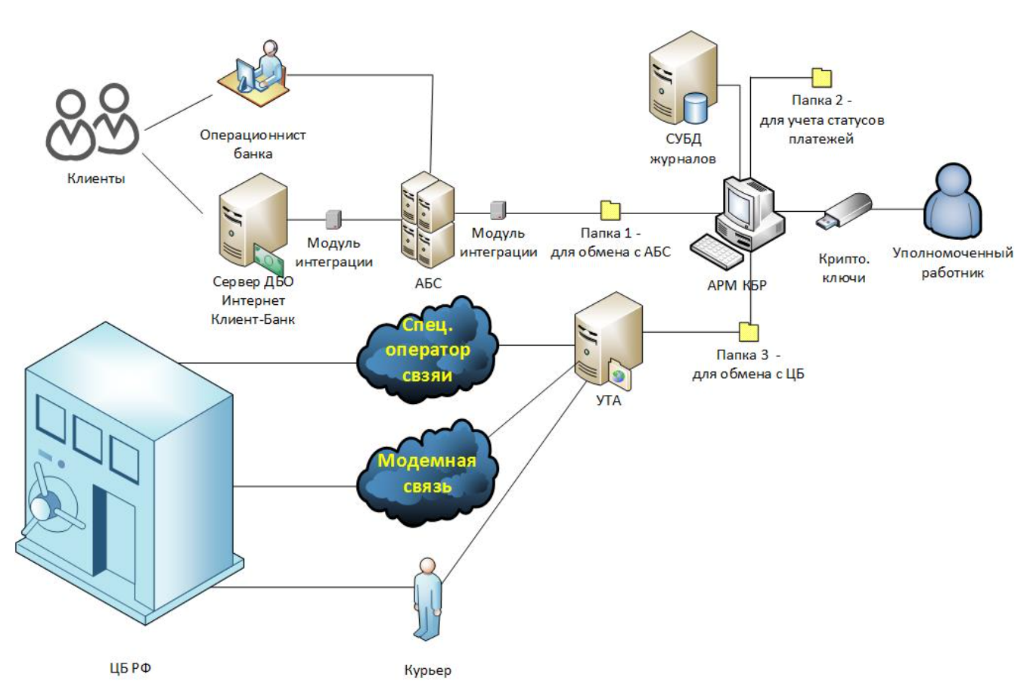


Рис. 1.2. Инфраструктура обеспечения платежного взаимодействия с Банком России

IT-инфраструктуру платежного взаимодействия банка с Банком России рассмотрим на примере исполнения платежа, отправляемого в адрес клиента другого банка.  
 В начале клиент должен передать в банк платежное поручение. Сделать это он может двумя способами:

1. Явиться лично в отделение банка и передать заверенное платежное распоряжение на бумажном носителе.
2. Направить платежное распоряжение через систему ДБО ИКБ.  
   Тут важно отметить, что системы ДБО ИКБ - это лишь системы, обеспечивающие юридически значимый электронный документооборот между клиентом и банком, самостоятельно они платежи не проводят. Именно поэтому, когда клиент открывает расчетный счет в банке, он обычно заключает два договора. Первый - договор обслуживания банковского счета, второй - договор на осуществление электронного документооборота с помощью системы ДБО ИКБ. Если второй договор заключен не будет, то клиент все равно сможет пользоваться своим счетом, но только при личном визите в отделение банка. Если клиент передал платежное поручение на бумажном носителе, то работник банка на его основании делает электронное платежное поручение в АБС. Если распоряжение было подано через ДБО ИКБ, то через модуль интеграции оно попадает в АБС автоматически.  
   Доказательством того, что именно клиент сделал распоряжение о переводе денежных средств, в первом случае является лично подписанный им бумажный документ, а во втором, электронный документ в ДБО ИКБ, заверенный в соответствии с договором. Обычно для заверения электронных документов клиентов - юридических лиц в ДБО ИКБ применяют криптографическую электронную подпись, а для документов клиентов – физических лиц коды SMS-подтверждений. Попав в АБС, платежное поручение в соответствии с внутренними регламентами банка проходит контроль и передается для исполнения в платежную систему Банка России.

Таким образом, использование открытого ПО в финансовой сфере также несет свои риски. Например, неправильное применение или настройка открытых решений может привести к уязвимостям и угрозам безопасности. Поэтому важно, чтобы специалисты по безопасности осуществляли постоянный мониторинг и аудит безопасности систем, использующих открытое ПО, а также следили за обновлениями и патчами, выпускаемыми сообществом разработчиков.

## 1.3. Риски, связанные с уязвимостями в финансовом программном обеспечении

Финансовые учреждения сталкиваются с различными угрозами и рисками, связанными с уязвимостями в используемом программном обеспечении. Они могут привести к серьезным последствиям, как для финансовых организаций, так и для их клиентов. Эти риски включают в себя:

*Несанкционированный доступ к финансовым данным и средствам:*

Уязвимости в финансовом ПО могут стать причиной несанкционированного доступа к финансовым данным и средствам. Это может привести к краже средств, мошенничеству и утечке конфиденциальной информации. Злоумышленники могут получить доступ к учетным записям клиентов, паролям, банковским картам и другой чувствительной информации, что приводит к финансовым потерям как для клиентов, так и для финансового учреждения.

*Атаки на доступность систем:*

Финансовые системы, такие как онлайн-банкинг и платежные системы, подвержены риску атак на доступность. Например, DDoS-атаки (атаки распределенного отказа в обслуживании) могут привести к простоям в работе финансовых услуг, недоступности веб-сайтов банков и платежных платформ. Это может вызвать недовольство клиентов, потерю доверия к финансовым учреждениям и существенные финансовые потери.

*Вредоносные программы и эксплойты:*

Финансовое ПО подвержено риску вредоносных программ, таких как вирусы, троянские программы, шпионские программы и эксплойты. Эти программы могут быть использованы для незаконного доступа к системам, кражи информации, манипулирования финансовыми данными и средствами, а также для проведения других кибератак.

*Финансовые потери и ущерб репутации:*

Успешная атака на финансовое ПО может привести к серьезным финансовым потерям для финансовых учреждений. Это включает в себя потерю средств, возможные штрафы и санкции со стороны регуляторных органов, а также ущерб репутации учреждения. Клиенты могут потерять доверие к банку или другому финансовому учреждению из-за недостаточной безопасности и защиты их финансовых данных.

*Юридические последствия:*

Финансовые учреждения могут столкнуться с юридическими последствиями в результате утечки данных или неблагоприятных событий, связанных с безопасностью. Это может включать в себя гражданские исковые требования со стороны клиентов, штрафы и санкции от регуляторных органов, а также возможные уголовные преследования в случае умышленных или небрежных действий.

Кроме значительного экономического ущерба, кибератаки приводят к изменениям в геополитических отношениях и снижению уровня доверия к сети Интернет и в конечном счете могут спровоцировать финансовый кризис.

К ключевым рискам в кредитно-финансовой сфере эксперты Банка России относят: финансовые потери клиентов (потребителей финансовых услуг), которые подрывают доверие к современным финансовым технологиям; финансовые потери отдельных финансовых организаций, которые могут отрицательно воздействовать на их финансовое положение; нарушение надежности операционной деятельности и непрерывности предоставления финансовых услуг, что может нанести ущерб репутации финансовых организаций и способствовать усилению социальной напряженности в обществе; возникновение системного кризиса из-за значимых для финансового рынка инцидентов информационной безопасности.

В следующем пункте мы рассмотрим примеры уязвимостей, характерные для финансовой сферы, а также последствия, к которым может привести их эксплуатация.

## 1.4. Примеры уязвимостей в финансовом программном обеспечении

Проведем краткий обзор примеров уязвимостей, наиболее встречающихся в финансовом ПО. Данный обзор поможет лучше понять характер угроз и способы их предотвращения.

Для начала стоит отметить, что каждое приложение состоит из нескольких основных частей: самого приложения, код которого реализует бизнес-логику, библиотек с открытым исходных кодом, очень часто и очень много используемых в современном ПО и окружения, где запускается и работает приложение.

В каждой из этих частей могут быть самые различные уязвимости.



Рис. 1.3. Уязвимости библиотек с открытым исходным кодом

### 1.4.1. Уязвимости в коде приложения

Код приложений пишут люди, пусть и обладающие техническими знаниями, но допускающие ошибки по неосторожности или по незнанию. Нередко именно поэтому в код попадают уязвимости. К чему может привести их наличие в коде веб-приложений? Приведём самые распространённые проблемы. Согласно статистике, уязвимости делают возможным несанкционированный доступ к приложению (39 % сайтов), в 68 % веб-приложений из-за них присутствует угроза утечки важных данных. При этом в третьи руки попадают и персональные данные (47 % утечек), и в целом учётные данные (31 %). Уязвимости, которые приводят к таким последствиям, могут быть самыми разными, начиная от незначительных ошибок и заканчивая самыми страшными из них, о которых бы хотелось рассказать подробнее.

На сегодняшний день, как бы это ни было странно, до сих пор в число самых распространённых уязвимостей входят XSS (53 %) и различного вида инъекции (29 %).

1. XSS (англ. Cross-Site Scripting - «межсайтовый скриптинг») - это очень распространённая уязвимость в веб-приложениях, суть которой состоит во внедрении JavaScript-кода в страницу, загружаемую пользователем. Цель атаки - получить данные пользователя и дальше совершать различные действия от его имени (например, войти в систему и вывести деньги или веб-приложение для онлайн-платежей может быть уязвимо к XSS-атакам, если не проводится достаточная фильтрация ввода перед выводом пользовательских данных на веб-страницы. Злоумышленник может внедрить вредоносный JavaScript-код, который будет выполнен в браузере пользователя, что позволит злоумышленнику получить доступ к сессионным файлам или украсть данные аутентификации).

Рассмотрим фрагмент уязвимого HTML-кода:

<form action="https://example.com/search" method="get">

<input type="text" name="query" value="">

<input type="submit" value="Search">

</form>

Если данные, введенные пользователем в поле поиска, не фильтруются должным образом, злоумышленник может вставить вредоносный код в параметр запроса. Например,

https://example.com/search?query=<script>alert('XSS')</script>

В этом случае, если веб-приложение не защищено от XSS, скрипт <script>alert('XSS')</script> выполнится в контексте страницы поиска, показывая всплывающее окно с сообщением “XSS”.

1. Инъекции - одна из самых распространённых атак на серверную часть приложения. При её реализации злоумышленники применяют различные технологии. Чаще всего они атакуют базы данных (SQL-инъекции), операционную систему (Command-инъекции), внутренние сервисы компании (LDAP-инъекции), задействуют обработку различных форматов (XML-инъекции или XXE). В конечном итоге целью эксплуатации таких уязвимостей является получение пользовательских данных, хранящихся на сервере, или захват полного контроля над сервером.

Например, веб-приложение для онлайн-банкинга может быть уязвимо к SQL-инъекциям, если не проводится достаточная фильтрация пользовательского ввода перед выполнением SQL-запросов к базе данных. Злоумышленник может использовать SQL-инъекцию, чтобы получить доступ к чувствительным финансовым данным, включая балансы счетов или личную информацию клиентов.

Ниже приведен фрагмент кода, содержащий уязвимость SQL-инъекции:

$query = "SELECT \* FROM users WHERE username = '" . $\_POST['username'] . "' AND password = '" . $\_POST['password'] . "'";

В этом фрагменте кода данные, полученные из формы ввода пользователя, непосредственно вставляются в SQL-запрос без какой-либо проверки или фильтрации. Злоумышленник может использовать это для выполнения вредоносных запросов, например, введя следующее значение в поле для пароля:

' OR '1'='1

Это приведет к изменению запроса следующим образом:

SELECT \* FROM users WHERE username = 'значение' AND password = '' OR '1'='1';

И, таким образом, атакующий сможет получить доступ к данным любого пользователя.

1. 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 передаваться 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. 1.4).

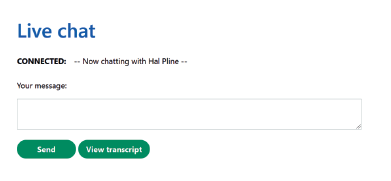


Рис. 1.4. Пример чата на сайте банка (условный сайт)

По нажатию на кнопку «View transcript» происходит скачивание истории текущего чата. Проанализируем запрос на получение такого файла помощью инструмента Proxy в BurpSuite (Рис. 1.5).

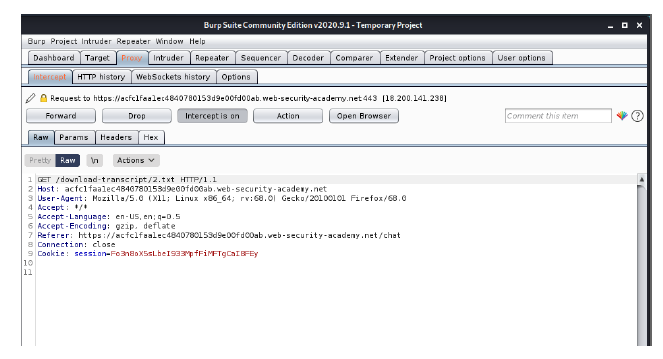


Рис. 1.5. Запрос к сайту на скачивание файла

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. 1.6, 1.7).

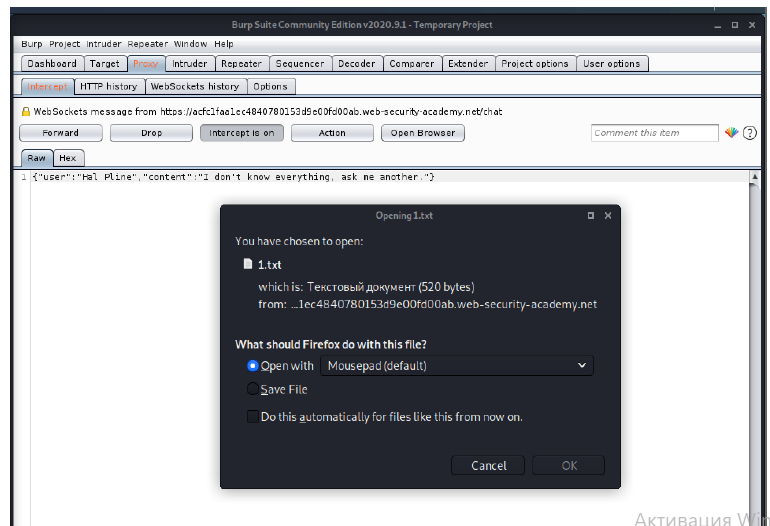


Рис. 1.6. Скачивание файла

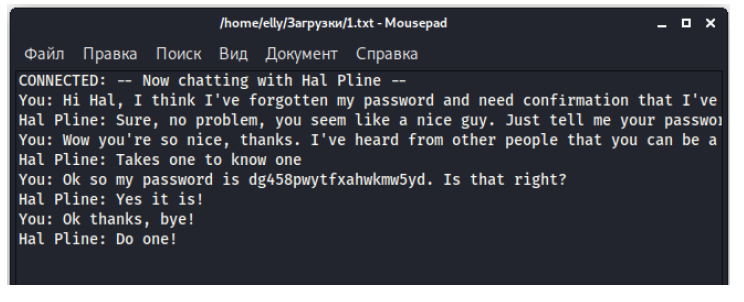


Рис. 1.7. Полученный файл с паролем пользователя

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.

1. 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 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. 1.8).



Рис. 1.8. Форма перевода денежных средств на сайте банка

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. 1.9).

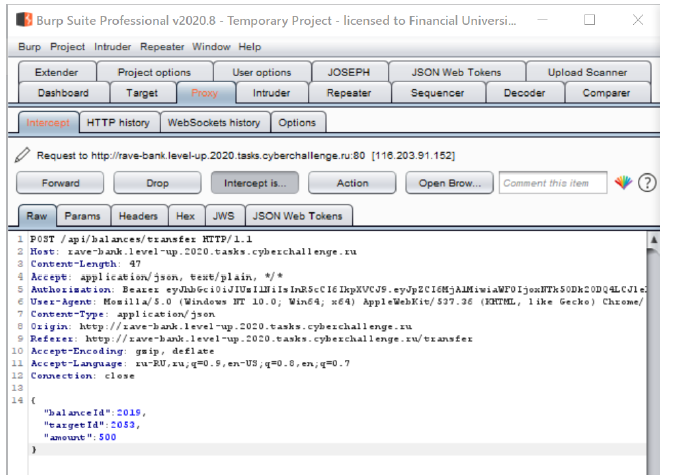


Рис. 1.9. Форма перевода денежных средств на сайте банка

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. 1.10).

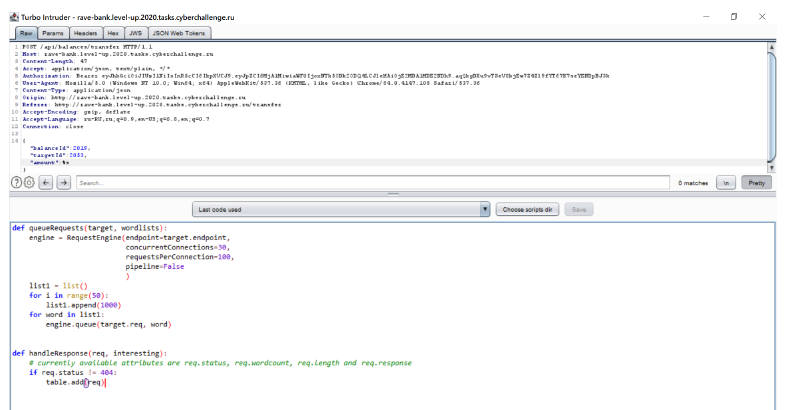


Рис. 1.10. Подготовка к проведению атаки Race Condition

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):

# 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. 1.11).

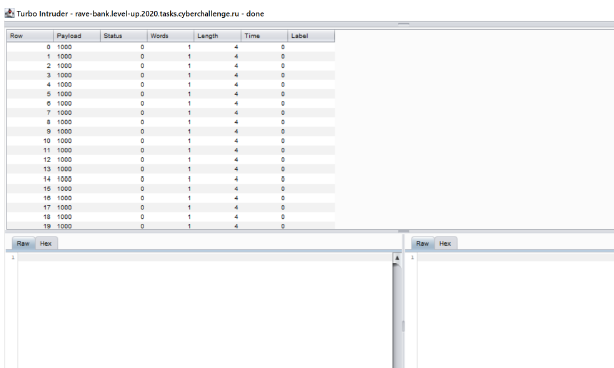


Рис. 1.11. Проведение атаки Race Condition

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 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.

Итак, через использование самых популярных проблем безопасности веб-приложений финансовой сферы могут быть атакованы как непосредственно клиенты, так и серверная часть (иначе говоря, инфраструктура организации целиком).

Это лишь несколько уязвимостей, которые могут быть обнаружены и использованы злоумышленниками в финансовом программном обеспечении. Важно осуществлять регулярные аудиты безопасности, обнаруживать подобные уязвимости и принимать меры для их устранения, чтобы обеспечить безопасность финансовых данных и систем, а также обучать разработчиков вопросам из области информационной безопасности.

### 1.4.2. Уязвимости в библиотеках с открытым исходным кодом

Однако уязвимости могут быть не только в коде, который пишут разработчики компании, но и в библиотеках с открытым исходным кодом, применяемых для упрощения и ускорения разработки. Согласно статистике, использование библиотек с открытым исходным кодом в больших проектах может достигать 30-40 % от всего кода приложения, а в некоторых случаях - и больше. Что и говорить, если за прошлый год появилось более 6 миллионов новых версий библиотек, а суммарное количество компонентов, которые можно использовать, превысило отметку в 37 миллионов. При этом количество загрузок различных библиотек составило более 2,2 триллиона за 2021 год. Все эти библиотеки тоже пишут люди, которые могут допускать ошибки (и сейчас мы даже не говорим о случаях, когда туда специально встраивается вредоносная функциональность, что также нередко случается). В исследованиях отмечают, что более 29 % популярных библиотек содержат уязвимости, а количество атак через них увеличилось на 650 % по сравнению с прошлым годом. И действительно, за последнее время атаки на приложения через библиотеки с открытым исходным кодом стали намного популярнее, некоторым уязвимостям даже стали давать имена. Вспомнить хотя бы нашумевшую недавно историю с Log4Shell, которая позволяла выполнить любой код на сервере удалённо. Если посмотреть на то, как её пытались использовать, то цифры получаются очень интересными. Во-первых, для реализации этой уязвимости было написано более 60 готовых эксплойтов и в отдельные промежутки времени можно было наблюдать до 100 попыток атак в минуту. Во-вторых, хакерские группировки суммарно провели более 1,5 миллиона атак, направленных на эту уязвимость, в течение первых четырёх дней после публикации уязвимости.

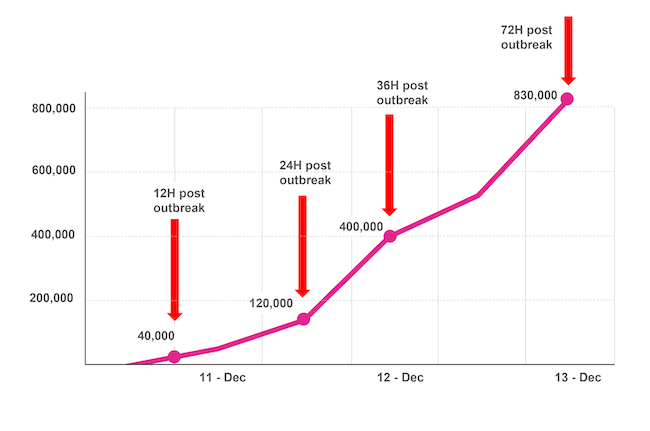


Рис. 1.12. Количество атак через Log4Shell

И таких примеров - множество. Уязвимости в openSSL (критическая уязвимости CVE-2014-0160 в популярной криптографической библиотеке OpenSSL. Уязвимость связана с отсутствием необходимой проверки границ в одной из процедур расширения Heartbeat (RFC6520) для протокола TLS/DTLS), который используется практически везде и который в 2014 году стал главной новостью и проблемой всех, кто занимается и разработкой, и безопасностью. Из-за этой маленькой ошибки одного программиста кто угодно получает прямой доступ к оперативной памяти компьютеров, чьи коммуникации «защищены» уязвимой версией OpenSSL. В том числе, злоумышленник получает доступ к секретным ключам, именам и паролям пользователей, и всему контенту, который должен передаваться в зашифрованном виде.

Уязвимость в библиотеке struts2, через которую произошёл самый массовый «слив» данных компании Equifax. Известно, что соответствующая информация была получена из документа, предоставленного в банковский комитет Сената США. В документе говорится о том, что киберпреступники получили доступ к личной информации, которая не упоминалась ранее в связи с этой учётной записи. Речь идет об идентификационных номерах налогоплательщиков, которые используются вместо номера социального страхования при его отсутствии. Также пострадали следующие данные: адреса электронной почты, информация о кредитных картах, а также дополнительные сведения о лицензии на драйверы.

Для того чтобы избежать подобных проблем и всегда понимать, какие версии библиотеки используются в ваших продуктах и насколько они безопасны, необходимо проводить постоянный анализ библиотек с открытым исходным кодом.

### 1.4.3. Уязвимости окружения

В дополнение ко всем уязвимостям, перечисленным выше, отметим ещё один распространённый тип проблем информационной безопасности. Речь - о неправильно настроенном окружении приложений, когда уязвимости находятся не в самом приложении, а где-то рядом, например, на некорректно сконфигурированном сервере, или появляются из-за неправильно настроенных параметров веб-сервера. Подобные баги, по статистике, встречаются в более чем 80 % приложений. Одним из примеров такой уязвимости является отсутствие важных заголовков безопасности, которые сильно упрощают атаки на пользователей. Другой случай - попадание директорий, связанных с системой хранения версий кода (git или svn), на сайт, когда открывается возможность получения всего исходного кода сайта. Примеры неправильной настройки веб-серверов и окружения можно перечислять очень долго. Они включают в себя и неправильно настроенный SSL, и возможность определения версий ПО, используемого на сервере (чтобы поискать в нём известные уязвимости), и незакрытые порты, и неправильную конфигурацию удалённого доступа к серверу, и учётные данные по умолчанию (кстати, это тоже одна из самых распространённых проблем), и многое другое. Обычно такого рода уязвимости не так опасны, как два серьёзных типа проблем, описанные выше, но они заметно упрощают способы эксплуатации и предоставляют возможность для построения вектора атаки. Конечно, их тоже нужно закрывать, а ещё лучше - не допускать их появления, всегда правильно настраивать все смежные системы и контролировать их конфигурацию. Помочь с этим могут, в частности, специализированные программы, проверяющие инструменты на соответствие лучшим практикам - к примеру, стандарту по безопасности настройки Docker-окружения CIS Benchmark.

### 1.4.4. Уязвимости в онлайн-банкинге

С развитием мобильных технологий, приложения мобильного банкинга становятся все более популярными, но они также подвержены угрозам безопасности. Уязвимости такого рода позволяют злоумышленникам получать доступ к учетным записям клиентов и совершать финансовые операции от их имени.

Недавно были обнаружены случаи проведенной полномасштабная фишинг-кампании, нацеленной на пользователей онлайн-банкинга JPMorgan Chase, осуществляющих операции через мобильные телефоны.

Злоумышленники создали несколько поддельных web-страниц, имитирующих законную страницу для проведения банковских операций онлайн. Единственным отличием было отсутствие в адресной строке HTTPS.

На первой фальшивой странице у пользователей запрашивали ID и пароль, на второй - адрес электронной почты и пароль, а на последней - отсканированную копию удостоверения личности, выданного правительством.

В результате злоумышленники смогли получить доступ к личным данным и финансовым средствам клиентов, совершая несанкционированные финансовые операции. Это привело к серьезным финансовым потерям для банка JPMorgan Chase и ухудшению его репутации, а также к потере доверия клиентов.

### 1.4.5. Уязвимости в платежных системах и торговых платформах

Уязвимости позволяющие атакующим манипулировать транзакциями и переводить денежные средства на свои счета, а уязвимости в торговых платформах также могут быть использованы злоумышленниками для изменения цен и условий сделок в свою пользу.

Торговые платформы также могут быть подвержены уязвимостям, например, уязвимостям в алгоритмах торговли, которые могут быть использованы для манипуляции ценами или выполнения незаконных сделок.

Например, обнаружена уязвимость в платежной системе PayPal, позволяющая злоумышленникам осуществлять мошеннические транзакции. Она заключалсь в том, что во время привязки учетной записи PayPal к учетной записи Google Pay PayPal создает виртуальную карту с собственным номером, сроком действия и CVC. Когда пользователь Google Pay решает воспользоваться функцией бесконтактных платежей PayPal, оплата производится с этой виртуальной карты.

Если бы виртуальная карта использовалась только для оплаты через PoS-терминалы, проблем бы не было. Однако виртуальная карта может использоваться и для online-платежей. Вероятно, злоумышленники нашли способ похищения данных виртуальных карт и с их помощью оплачивали покупки в американских магазинах.

Это привело к финансовым потерям для компании PayPal, а также к потере доверия клиентов и ухудшению ее репутации.

### 1.4.6. Уязвимость в системах управления финансовыми активами

Существует риск уязвимостей в системах управления финансовыми активами, которые могут привести к несанкционированному доступу к инвестиционным портфелям или изменению данных о финансовых активах.

Финансовые учреждения используют системы управления рисками для оценки и управления рисками в своей деятельности. Уязвимости в таких системах могут привести к искажению данных о рисках или неправильному принятию решений, что может иметь серьезные финансовые последствия.

### 1.4.7. Уязвимости в алгоритмах шифрования:

Неправильно реализованные или устаревшие алгоритмы шифрования могут привести к утечке конфиденциальных финансовых данных. Например, если приложение использует слабые методы шифрования или хранит ключи шифрования в ненадежном месте, злоумышленник может легко расшифровать или перехватить финансовую информацию.

### 1.4.8. Отказ в обслуживании (DoS) и распределенные атаки на отказ обслуживания (DDoS)

Эти атаки направлены на перегрузку серверов или сетей, что приводит к недоступности сервисов для легитимных пользователей. В финансовой сфере это может иметь серьезные последствия, такие как недоступность онлайн-банкинга или задержки в обработке финансовых транзакций.

Все приведенные примеры демонстрируют разнообразие уязвимостей, которые могут существовать в финансовом ПО, а также необходимость принятия мер для их обнаружения и устранения.

Анализ таких случаев позволяет выявить общие уязвимости и разработать эффективные методы их обнаружения и устранения.

## 1.5. Методы предотвращения уязвимостей

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

**Каналы связи:**

1. Между пользователем и приложением:

A. Настройка SSL/TLS-сертификатов. Протокол TLS (transport layer security) основан на протоколе SSL (Secure Sockets Layer), изначально 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. 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.

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: maxage=<expire-time> - устанавливает временной промежуток действия HSTS для определенного сайта;

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

3) Strict-Transport-Security: maxage=<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.\_\_

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.

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, 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. 1.13. Применяется хеширование, «соль» и передача зашифрованных данных с использованием RSA-шифрования.

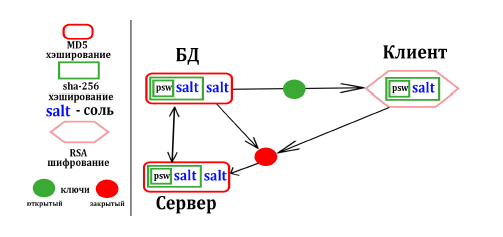


Рис. 1.13. Процесс хеширования данных

**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.

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. 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:

- Logstash - сервиса для сбора логов и отправки их в Elasticsearch;

- Elasticsearch - системы хранения, анализа, поиска по логам;

- Kibana - веб-панели для визуализации выходных данных; Beat s - агентов для отправки логов в Logstash.

**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.

4. Использование нестандартных паролей для администраторов сервисов.

5. Перемещение сервисов на нестандартные порты.

6. Настройка использования технологии portknocking 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.

Безопасность приложения - комплексное понятие, и при её обеспечении нужно учитывать не только код, который пишется разработчиками, но и безопасность компонентов с открытым исходным кодом, используемых в приложении, правильную конфигурацию сервера и окружения, обучение команды разработки и многие другие немаловажные вещи. Делать это максимально эффективно, не отвлекаясь на сроки выхода новой функциональности на рынок, поможет правильно выстроенный процесс разработки защищённых программных продуктов, с помощью которого можно бесшовно интегрировать ИБ в инженерный процесс. Пользователям систем и услуг, в свою очередь, можно рекомендовать максимально задействовать все возможности, которые предоставляет система обеспечения безопасности данных. Как минимум, стоит следить за базовыми вещами: Если есть возможность, установить двухфакторную аутентификацию в приложении; да, это не всегда удобно, но даже если ваши учётные данные попадут в руки к злоумышленникам, им будет намного труднее завладеть вашим аккаунтом. Не использовать одинаковые пароли для различных сервисов. Чтобы помнить их все, можно воспользоваться удобными программами для хранения паролей. Всегда тщательно проверять, где и куда вы вводите свои данные, и быть внимательными при звонках с неизвестных номеров.

## Выводы по главе 1

Разработка безопасного финансового программного обеспечения является необходимостью для обеспечения стабильной и надежной работы финансовых учреждений. В главе 1 были рассмотрены особенности открытого программного обеспечения, характерные уязвимости, встречающиеся в финансовой сфере, а также последствия и методы обнаружения и устранения уязвимостей.

# ГЛАВА 2. МЕТОДОЛОГИЯ ЗАЩИЩЕННОСТИ ОТКРЫТОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ДЛЯ ФИНАНСОВОЙ СФЕРЫ

В предыдущем разделе были рассмотрены характерные уязвимости, которые встречаются в программном обеспечении, приведены примеры и последствия их деятельности.

Для того, чтобы финансовые учреждения могли безопасно эксплуатировать и предоставлять наиболее современные технологии для пользователей, необходимо уметь выявлять, предотвращать и по возможности не допускать эксплуатации уязвимостей в программном обеспечении.

Современная финансовая сфера стоит перед непрерывными вызовами в области кибербезопасности. С увеличением числа цифровых транзакций и развитием технологий, финансовые учреждения становятся более уязвимыми к различным видам киберугроз. В этом контексте безопасность открытого программного обеспечения (ОПО) приобретает особую важность, поскольку множество финансовых институтов используют его в своей деятельности. В данном разделе рассмотрим методологию обеспечения защищенности открытого программного обеспечения для финансовой сферы, включая цикл оценки безопасности и методику поиска и устранения уязвимостей. Понимание и эффективное применение этих методов является ключевым для обеспечения безопасности и надежности финансовых систем в условиях постоянно меняющейся киберугрозовой среды.

## 2.1. Цикл оценки защищенности открытого программного обеспечения для финансовой сферы

Цикл оценки защищенности открытого программного обеспечения для финансовой сферы представляет собой комплексный процесс, направленный на выявление и управление уязвимостями, а также обеспечение надежной защиты финансовых данных и систем. Этот цикл состоит из нескольких важных этапов, каждый из которых играет свою роль в обеспечении безопасности и защиты финансовых интересов. Давайте подробно рассмотрим каждый из этих этапов.

1. Идентификация и инвентаризация ПО

Первый этап цикла оценки защищенности заключается в идентификации и инвентаризации всего открытого программного обеспечения, используемого в финансовой организации. Это включает в себя как официально приобретенное ПО, так и программное обеспечение, которое может быть установлено или использовано сотрудниками без разрешения IT-отдела. Для эффективной инвентаризации ПО необходимо использовать специализированные инструменты для сканирования сети и обнаружения всех установленных программ.

Примером этого этапа может служить сценарий, когда информационный отдел финансовой компании применяет специализированные инструменты для сканирования сети и обнаружения всех установленных программ. Например, используя утилиты сканирования сети, IT-специалисты могут выявить все программные продукты и версии, которые используются в компании, начиная от операционных систем и заканчивая библиотеками и фреймворками. Это позволяет создать полный список программного обеспечения, который используется в финансовой организации.

- Nmap: Это инструмент сканирования сети, который может использоваться для обнаружения и инвентаризации устройств и программного обеспечения в сети.

- Lansweeper: Это программное обеспечение для инвентаризации, которое автоматически сканирует сеть и составляет список установленного программного обеспечения на компьютерах и серверах.

2. Анализ уязвимостей и рисков

На втором этапе происходит анализ уязвимостей и рисков, связанных с использованием открытого программного обеспечения в финансовой организации. Это включает в себя сканирование программного обеспечения на наличие известных уязвимостей, а также оценку уровня риска для каждой уязвимости в контексте финансовых операций. Для анализа уязвимостей часто используются специализированные утилиты и сервисы, такие как утилиты сканирования уязвимостей и базы данных уязвимостей.

Примером анализа уязвимостей и рисков может быть обнаружение уязвимости в финансовом ПО, такой как SQL-инъекция. Например, при проведении тестирования на проникновение выявляется, что финансовое приложение подвержено SQL-инъекции из-за недостаточной фильтрации пользовательского ввода. Это уязвимость может быть классифицирована как критическая, поскольку злоумышленник может получить доступ к конфиденциальным данным, таким как данные банковских счетов или персональная информация клиентов.

- OpenVAS (Open Vulnerability Assessment System): Это инструмент сканирования уязвимостей, который может использоваться для автоматического обнаружения и анализа уязвимостей в открытом программном обеспечении.

- Nessus: Это еще один инструмент сканирования уязвимостей, который предлагает широкий набор функций для обнаружения и классификации уязвимостей.

3. Разработка и применение мер безопасности

На третьем этапе разрабатываются и применяются меры безопасности для уменьшения рисков, связанных с уязвимостями в открытом программном обеспечении. Это может включать в себя установку обновлений и патчей безопасности, настройку прав доступа, реализацию механизмов мониторинга и обнаружения аномалий, а также обучение сотрудников в области информационной безопасности. Каждая мера безопасности должна быть адаптирована к специфике использования ПО в финансовой сфере и направлена на минимизацию рисков для финансовых операций.

После выявления уязвимости в ПО, финансовая компания разрабатывает и применяет меры безопасности для ее устранения. Например, для предотвращения атак SQL-инъекции может быть реализована проверка входных данных и использование параметризованных запросов. Кроме того, может быть установлено обновление или патч, предоставленное разработчиком ПО, чтобы закрыть уязвимость.

- Security Onion: Это интегрированное средство мониторинга безопасности с открытым исходным кодом, которое включает в себя множество инструментов для обнаружения и предотвращения угроз, таких как средства контроля доступа и межсетевой экран.

- Snort: Это система обнаружения вторжений (IDS), которая может использоваться для мониторинга сетевого трафика и обнаружения попыток эксплуатации уязвимостей в реальном времени.

4. Мониторинг и анализ

Четвертый этап цикла оценки защищенности заключается в мониторинге и анализе безопасности открытого программного обеспечения в режиме реального времени. Это включает в себя постоянное отслеживание активности и аномалий в сети и приложениях, а также анализ журналов событий и отчетов о безопасности. Мониторинг и анализ помогают выявлять потенциальные угрозы и реагировать на них своевременно, прежде чем они приведут к серьезным последствиям для финансовых операций.

На этом этапе финансовая компания внедряет системы мониторинга и анализа для постоянного отслеживания активности и аномалий в сети и приложениях. Например, могут быть установлены средства мониторинга сетевого трафика и обнаружения вторжений, которые позволяют выявлять подозрительную активность, связанную с попытками эксплуатации уязвимостей в ПО.

- Splunk: Это платформа анализа данных, которая позволяет собирать, анализировать и визуализировать данные о безопасности из различных источников, включая журналы событий и данные сенсоров безопасности.

- Elasticsearch с Kibana: Эти инструменты предоставляют мощный механизм сбора, анализа и визуализации данных безопасности в реальном времени.

5. Аудит и обновление

Последний этап цикла оценки защищенности состоит в проведении аудита и обновлении мер безопасности на основе полученных результатов. Это включает в себя оценку эффективности примененных мер безопасности, выявление слабых мест и уязвимостей, а также внедрение обновлений и улучшений. Аудит и обновление помогают обеспечить постоянную защиту финансовой информации и систем от новых угроз и атак.

- OpenSCAP: Это открытая система аудита соответствия, которая позволяет автоматизировать процессы аудита безопасности и обеспечивать соответствие стандартам безопасности.

- WSUS (Windows Server Update Services): Это программное обеспечение, которое позволяет администраторам управлять обновлениями и патчами безопасности для операционных систем и другого программного обеспечения на компьютерах в сети.

Последний этап цикла заключается в проведении аудита и обновлении мер безопасности на основе полученных результатов. Например, после проведения аудита безопасности может быть выявлено, что уязвимость в ПО была успешно устранена, но были обнаружены другие уязвимости, требующие дополнительных мер безопасности. В таком случае компания разрабатывает и применяет обновленные меры безопасности для обеспечения надежной защиты финансовых данных и систем.

В данном пункте был представлен цикл оценки защищенности открытого программного обеспечения для финансовой сферы. Этот цикл состоит из пяти важных этапов: идентификации и инвентаризации ПО, анализа уязвимостей и рисков, разработки и применения мер безопасности, мониторинга и анализа, а также аудита и обновления.

На каждом этапе цикла выполняются определенные действия, направленные на обеспечение безопасности и защиты финансовых данных и систем от потенциальных угроз. Это включает в себя сканирование ПО на наличие уязвимостей, разработку и применение мер безопасности, мониторинг активности и аномалий, а также проведение аудита и обновление мер безопасности на основе полученных результатов.

Важно отметить, что цикл оценки защищенности является непрерывным процессом, который требует постоянного внимания и улучшения. Только путем постоянного мониторинга, анализа и обновления мер безопасности можно обеспечить эффективную защиту от угроз в сфере финансовых технологий.

Таким образом, данный цикл представляет собой важный инструмент для финансовых организаций, позволяющий обеспечить безопасность и защиту финансовых данных и систем от современных киберугроз.

## 2.2. Методика поиска и устранения уязвимостей в открытом программном обеспечении для финансовой сферы

Давайте представим подробный теоретический пример реализации методики информационной безопасности для финансовой организации, учитывая анализ методик таких организаций, а также требования нормативной базы ФСТЭК, Центрального Банка России и критически важной инфраструктуры.

Необходимо определить требования:

Установить регулярный график сканирования открытого программного обеспечения на наличие уязвимостей. Минимальная частота сканирования - ежеквартально, с возможностью более частого сканирования при выходе критических обновлений или обнаружении новых уязвимостей (специализированные системы сканирования OpenVAS, Nessus, для обнаружения уязвимостей в открытом ПО).

Установить политику автоматического обновления открытого программного обеспечения, чтобы минимизировать время эксплуатации уязвимостей.

Перед внедрением новых версий ПО проводить аудит кода на наличие уязвимостей (SonarQube, Veracode, для проверки наличия уязвимостей в исходном коде).

Проводить обучение сотрудников по правилам безопасности, процедурам реагирования на инциденты и корректному использованию инструментов поиска и устранения уязвимостей. Выделим требования, которые относятся к разным категориям организации.

Правила работы для рядовых сотрудников:

- проверка целостности данных: сотрудникам следует регулярно проверять целостность данных, с которыми они работают. Это может включать контроль за цифровой подписью документов, проверку чек-сумм файлов и другие методы подтверждения целостности;

- соблюдение правил безопасности: должны строго соблюдать правила безопасности, такие как не передавать пароли или доступ к системам другим сотрудникам, не открывать подозрительные письма или файлы;

- cообщение об аномалиях: в случае обнаружения подозрительной активности или аномалий в работе системы или программного обеспечения, следует немедленно сообщать своему руководству или подразделению информационной безопасности.

Правила работы для технических специалистов:

- проводить сканирование уязвимостей: рекомендуется регулярно сканировать сетевые ресурсы и системы на предмет уязвимостей с использованием специализированных инструментов (например, SIEM);

- анализ журналов событий: регулярно проводить анализ журналов событий с целью выявления подозрительной активности и аномалий, которые могут указывать на наличие уязвимостей или кибератак;

- устранение уязвимостей: после обнаружения уязвимостей следует немедленно принимать меры по их устранению, включая установку необходимых патчей, изменение конфигурации системы или другие действия, а также необходимо о выявленных угрозах сообщить руководителям организации.

Правила работы для руководителей:

- планирование и координация: руководители должны обеспечить планирование и координацию процессов поиска и устранения уязвимостей, включая выделение необходимых ресурсов и времени для выполнения этих задач;

- мониторинг выполнения: регулярно мониторить выполнение процессов поиска и устранения уязвимостей, а также контролировать соблюдение установленных сроков и требований безопасности;

- принятие стратегических решений: руководители также должны принимать стратегические решения о внедрении новых технологий и методов защиты информации, направленных на улучшение общей безопасности организации.

В случае выявления уязвимостей:

- оценить серьезность и приоритетность уязвимостей в соответствии с их потенциальными последствиями и вероятностью их эксплуатации;

- создать план действий для устранения обнаруженных уязвимостей, включая установку патчей, обновление ПО или изменение конфигурации системы;

- следить за процессом устранения уязвимостей, регулярно обновлять заинтересованные стороны о текущем статусе и предоставлять отчеты о выполненных действиях и результатах.

Вместе эти действия обеспечивают комплексный подход к поиску и устранению уязвимостей, который минимизирует риски для финансовой организации и обеспечивает безопасность информации в соответствии с действующим законодательством.

## Выводы по главе 2

В главе 2 рассмотрен цикл оценки защищенности и рассмотрено применение методики поиска и устранения уязвимостей в открытом программном обеспечении для финансовой сферы, которая является крайне важной для обеспечения безопасности и надежности информационных систем финансовых организаций. Правильное применение методики, а также проведение оценки защищенности позволяет предотвратить потенциальные угрозы безопасности, минимизировать риски и обеспечить защиту конфиденциальных данных клиентов. Такой подход играет ключевую роль в современном цифровом мире, где безопасность информации становится все более важной для успешной работы финансовых учреждений.

# ГЛАВА 3. Практическая часть: РЕАЛИЗАЦИЯ СКРИПТА АВТОМАТИЗИРОВАННОГО ПОИСКА УЯЗВИМОСТЕЙ в открытом программном обеспечении ФИНАНСОВОй сферы

## 3.1 Описание среды

Для выполнения практической части выпускной квалификационной работы была подготовлена среда разработки и анализа приложения финансовой сферы.

В лабораторном стенде было развернуто несколько виртуальных машин: машины с выбранным веб-приложением, и машина для анализа кода, а также машина с приложением для контроля и управления уязвимостями (рис. 3.1).

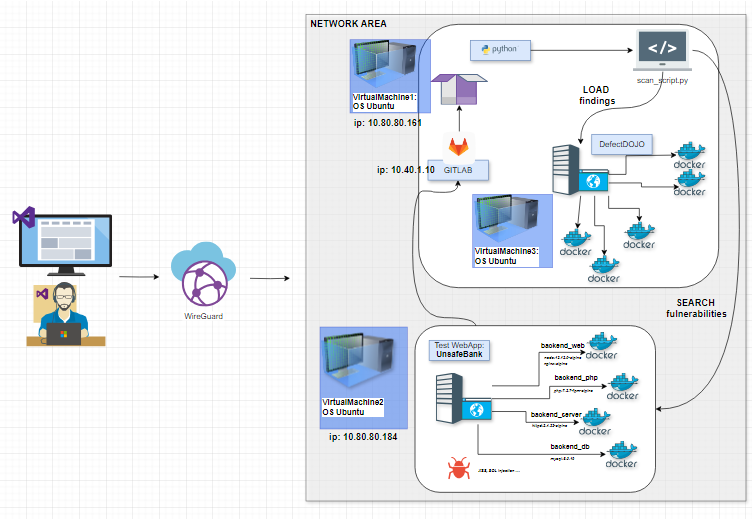


Рис. 3.1. Схема рабочего стенда

Вот краткий алгоритм действий при работе с нашим лабораторным стендом:

1. Устанавливаем соединение по шифрованному туннелю (конфиг Wire Guard) через SSH.

2. Получаем доступ к сети из 2-ух машин 10.80.80.161 и 10.80.80.184.

3. На 10.80.80.161 у нас крутится гитлаб с исходным кодом UnSAFE банка, на 10.80.80.184 запускаем сканер с локальной базой уязвимостей, которая регулярно обновляется с сайта https://nvd.nist.gov/vuln/data-feeds.

4. Отдаем сканеру (скрипт на python) ip адрес машины, на которой располагается код UnSAFE банка, логин/пароль для подключения по SSH и путь к директории.

5. Полученные в результате сканирования файндинги уязвимостей отправляем в менеджер уязвимостей defectDojo, на основании которых репортим отделу разработки о создании необходимых патчей.

6. Настраиваем менеджер задач в системе на автоматическое регулярное сканирование с нужным интервалом.

Рассмотрим более детально процесс.

### 3.1.1 Виртуальная машина - веб-приложение UnsafeBank

В качестве приложения финансовой сферы для демонстрации поиска уязвимостей выбор был остановлено на приложении UnsafeBank.

UnsafeBank - это учебное веб-приложение, разработанное для тестирования безопасности и обучения специалистов по информационной безопасности (рис. 3.2). Приложение имитирует базовые функции банковского веб-приложения и содержит различные типы уязвимостей, что позволяет проводить практические учебные исследования и тренировки по обнаружению и эксплуатации уязвимостей веб-приложений.

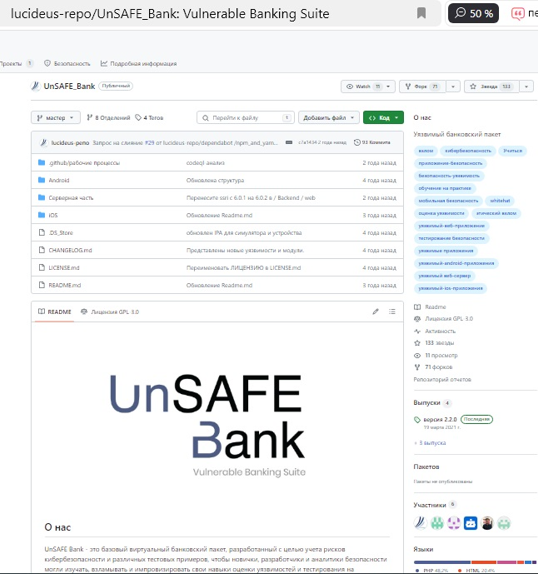
**

Рис. 3.2. Директория с открытым кодом на Github

Основная цель - демонстрация типичных уязвимостей, которые могут существовать в реальных банковских приложениях, а также оказание помощи специалистам по информационной безопасности в их изучении и понимании. Для наглядности отображения функционала приложение было установлено на виртуальной машине с графическим интерфейсом (было сделано дополнительно). Приложение для разворачивания загружаем из репозитория Github. Ниже представлен краткий процесс установки и запуска нескольких docker-контейнеров, необходимых для функционирования приложения (рис. 3.3, рис. 3.4, рис. 3.5, рис. 3.6).

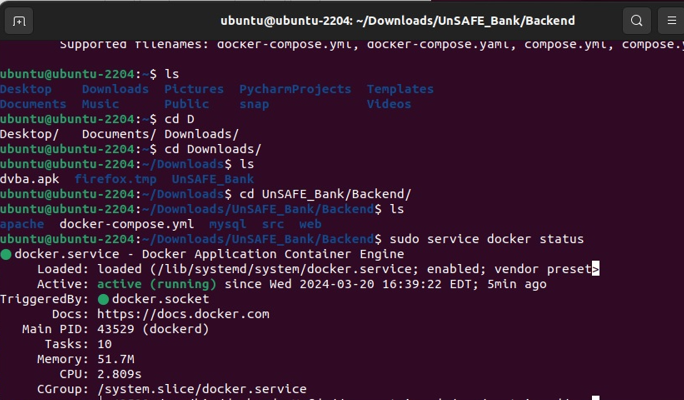
**

Рис. 3.3. Загрузка приложения на виртуальную машину

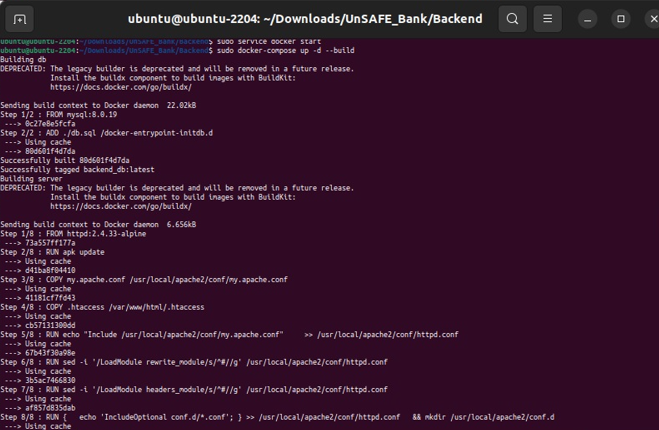
**

Рис. 3.4. Запуск docker-контейнеров для разворачивания

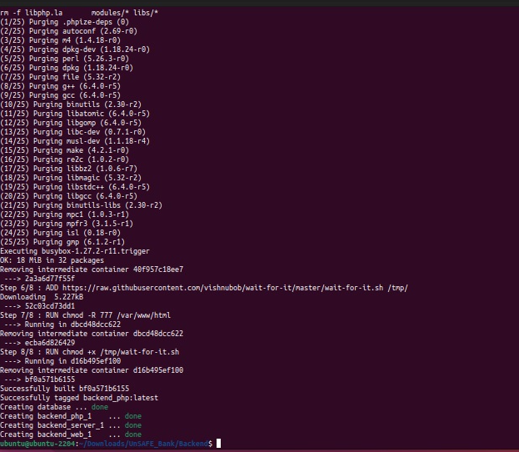
**

Рис. 3.5. Успешное создание образов приложения

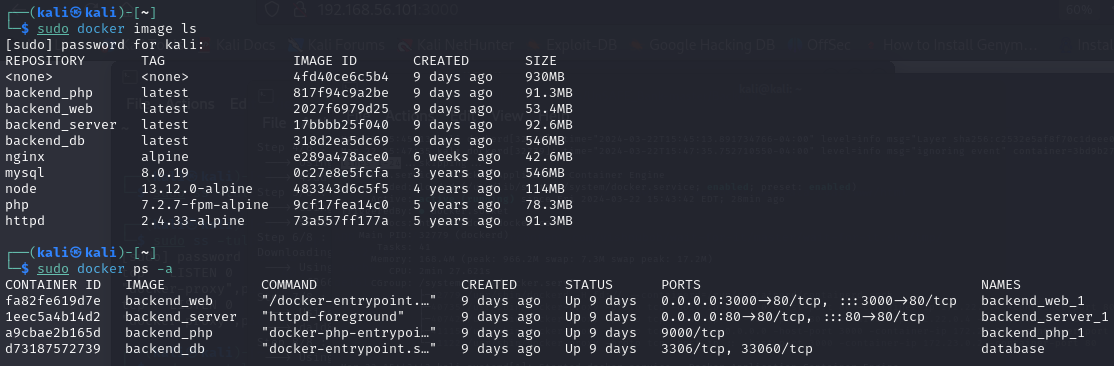
**

Рис. 3.6. Загруженные образы и запущенные docker-контейнеры

Приложение позволяет пользователям проводить следующие действия:

- регистрация пользователей: пользователи могут создать учетную запись в приложении, указав свои персональные данные, такие как имя, адрес электронной почты и пароль (рис. 3.7, рис. 3.8);

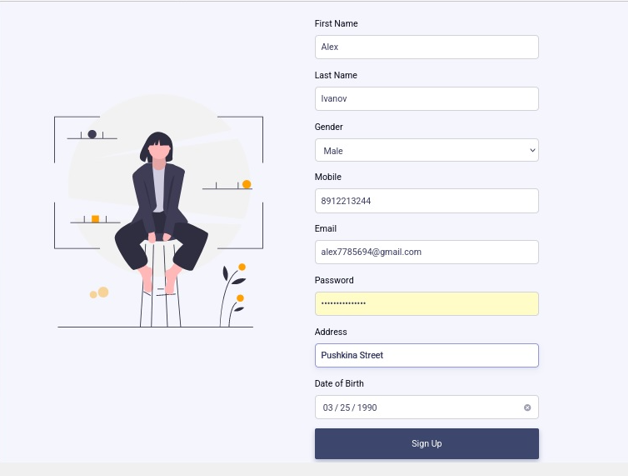
**

Рис. 3.7. Создание нового пользователя

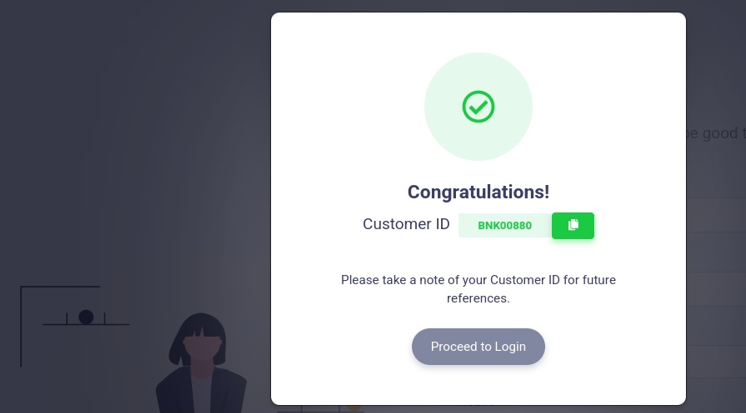
**

Рис. 3.8. Успешное создание ID пользователя для входа

* аутентификация: зарегистрированные пользователи могут войти в систему, указав свои учетные данные (имя пользователя и пароль), чтобы получить доступ к функциональности приложения (рис. 3.9);

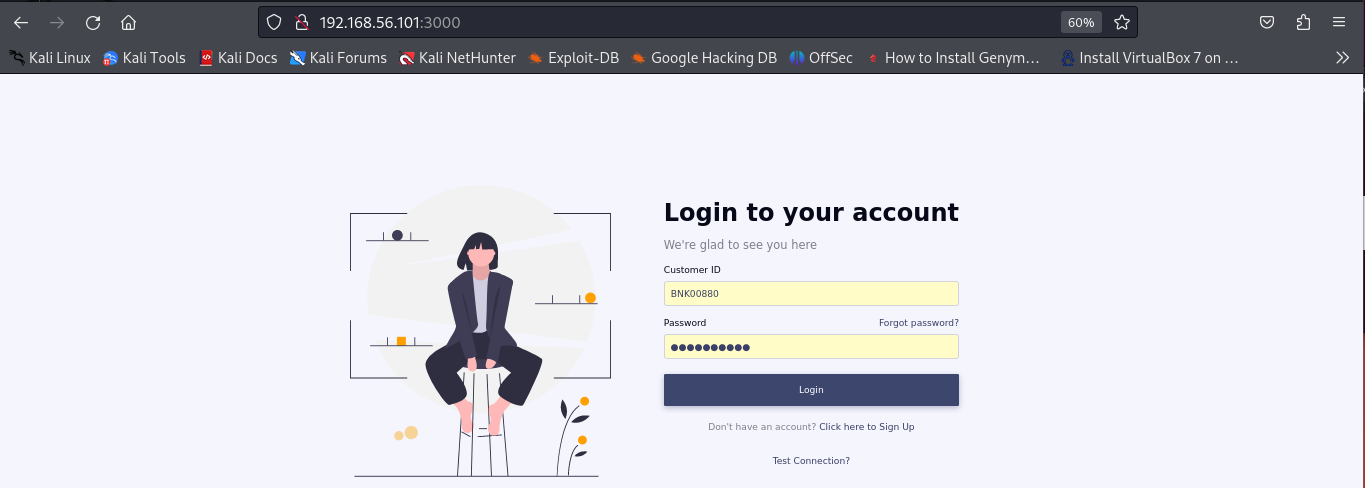
**

Рис. 3.9. Аутентификация пользователя

* просмотр баланса счета: после успешной аутентификации пользователи могут просмотреть текущий баланс своего банковского счета (рис. 3.10);

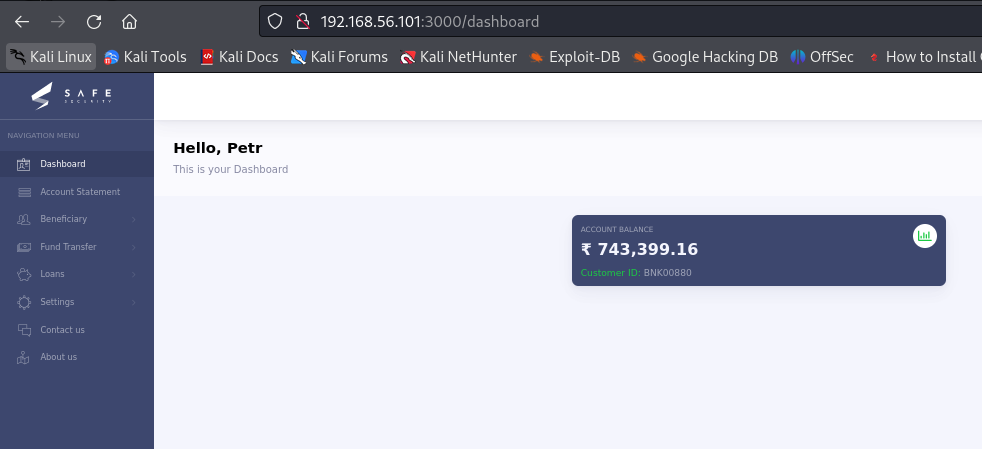
**

Рис. 3.10. Отображение баланса при входе в личный кабинет

- перевод средств: пользователи могут осуществлять перевод средств со своего банковского счета на другие счета, указав необходимые реквизиты получателя и сумму перевода (рис. 3.11, рис. 3.12, рис. 3.13, рис. 3.14).

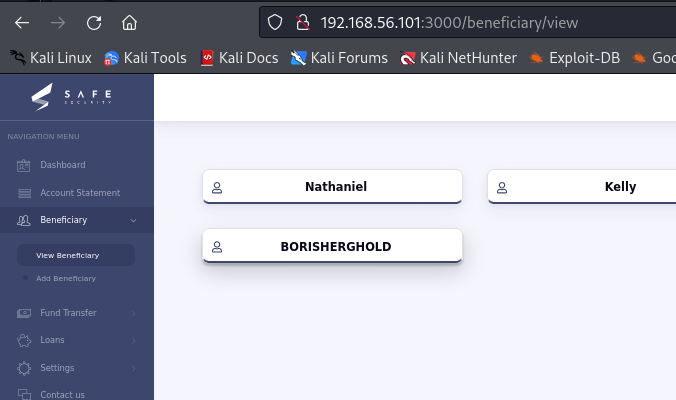


Рис. 3.11. Бенефициары, присутствующие для проведения транзакций

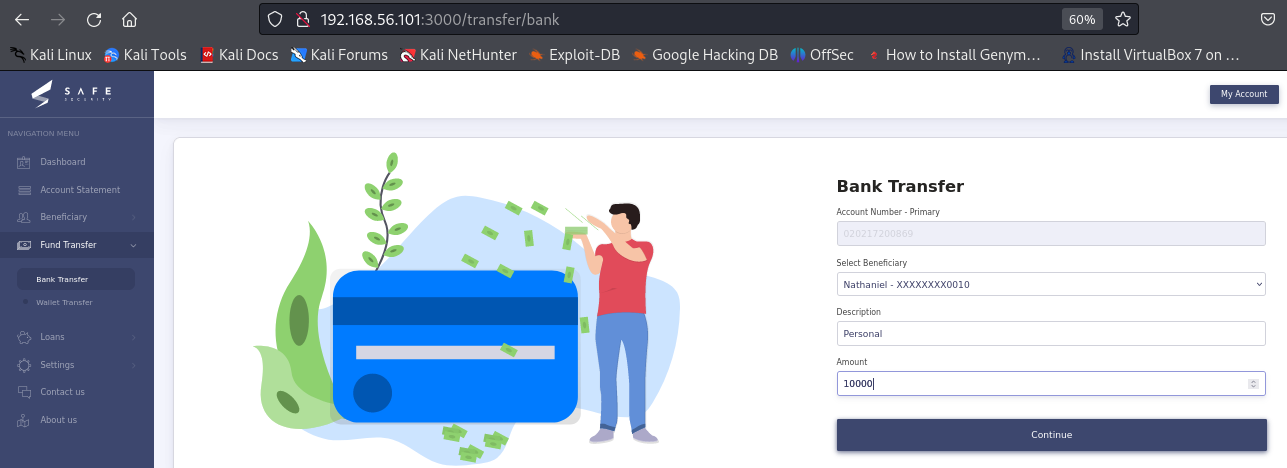


Рис. 3.12. Создание банковской транзакции

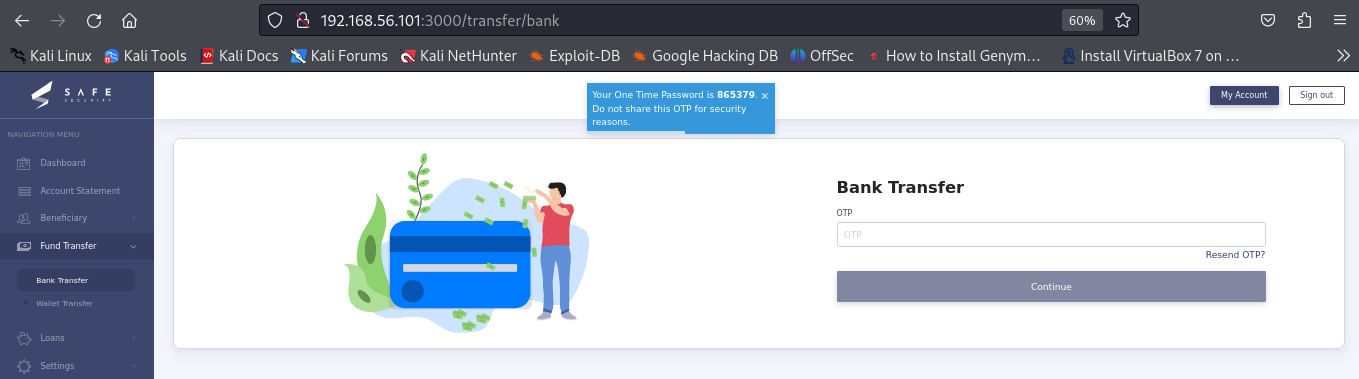


Рис. 3.13. Поддержание функции подтверждения операции по   
One Time Password

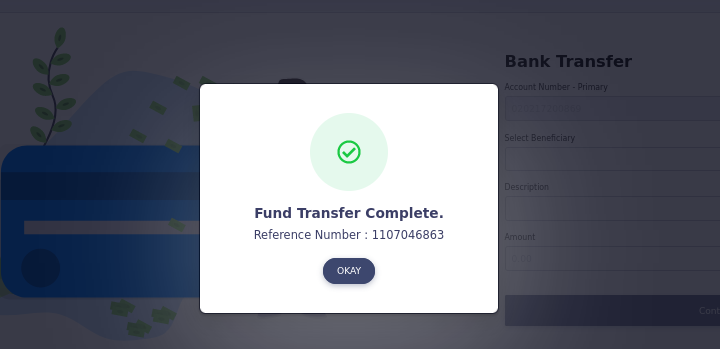


Рис. 3.14. Завершение банковской транзакции

Как уже отмечалось выше в работе, UnsafeBank содержит различные уязвимости в своей структуре. Повторим их:

* + - 1. XSS (межсайтовый скриптинг):

Эта уязвимость возникает, когда веб-приложение неправильно фильтрует или экранирует пользовательский ввод, который затем отображается на странице. Злоумышленники могут использовать XSS для внедрения вредоносных скриптов, которые выполняются в контексте пользователя, что может привести к краже сессионных cookie, перенаправлению на фишинговые сайты и другим атакам.

1. SQL-инъекции:

Эта уязвимость возникает, когда входные данные, передаваемые в SQL-запросы, не адекватно экранируются или валидируются, что позволяет злоумышленникам внедрять и выполнять зловредный SQL-код. С помощью SQL-инъекций атакующие могут получать доступ к конфиденциальной информации, изменять или удалять данные в базе данных, и даже получать удаленное управление сервером.

1. Неправильная аутентификация:

Уязвимость связана с недостаточно безопасными механизмами аутентификации. Это может включать в себя недостаточно длинные или слабые пароли, отсутствие механизмов защиты от атак подбора паролей, небезопасное хранение паролей и другие недочеты в процессе аутентификации пользователей.

1. CSRF (межсайтовое подделывание запросов):

Эта уязвимость возникает, когда злоумышленник вынуждает аутентифицированного пользователя выполнить нежелательное действие на веб-сайте без его согласия. Например, злоумышленник может создать поддельный запрос, который отправляется на сервер от имени аутентифицированного пользователя, и, таким образом, выполнить нежелательное действие, такое как изменение настроек аккаунта или отправка сообщения с фишинговым содержанием.

1. Отсутствие защиты от перечисления пользователей:

Эта уязвимость возникает, когда веб-приложение позволяет злоумышленнику определить наличие или отсутствие учетных записей пользователей. Это может быть использовано злоумышленниками для перебора или перечисления пользователей, что затем может быть использовано для проведения атак, таких как подбор паролей или фишинг.

Дальше рассмотрим разработанный скрипт для сканирования приложения, создания отчета о найденных уязвимостях и подготовки загрузки его в среду контроля и управления уязвимостями Defect Dojo.

### 3.1.2 Виртуальная машина – сканер приложения

На данной машине были установлены необходимые вспомогательные приложения, необходимые для проведения поиска уязвимостей, а также среда для разработки скрипта на языке Python.

Для управления уязвимостями и тестирования безопасности было развернуто открытое приложение DefectDojo (для наглядности отображения его способности на виртуальной машине с GUI), разработанное для помощи организациям в управлении и устранении уязвимостей, обнаруженных в их приложениях и инфраструктуре. Приложение, как и наш UnSAFE\_Bank является приложением с открытым исходным кодом и расположено на Github. Ниже приведен краткий обзор процесса разворачивания (рис. 3.15, рис. 3.16, рис. 3.17, рис. 3.18, рис. 3.19)

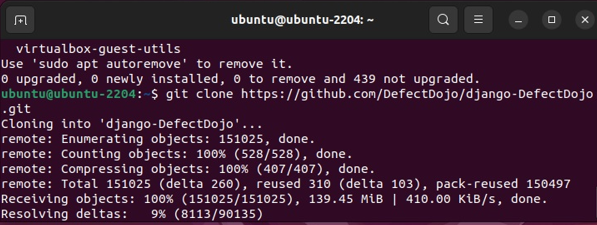
**

Рис. 3.15. Клонирование репозитория из Github на виртуальную машину

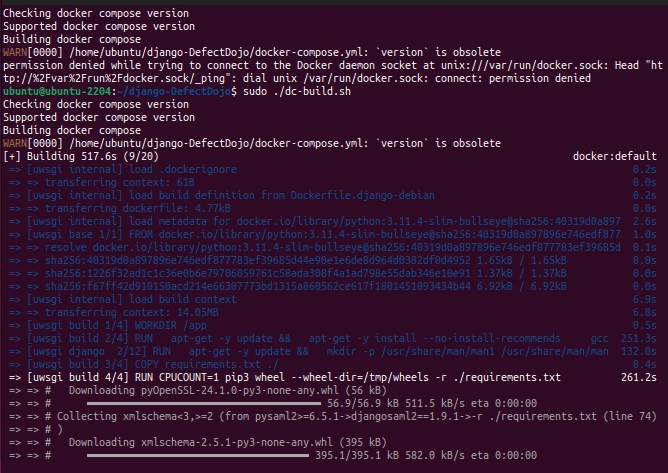
**

Рис. 3.16. Процесс сборки docker-контейнеров

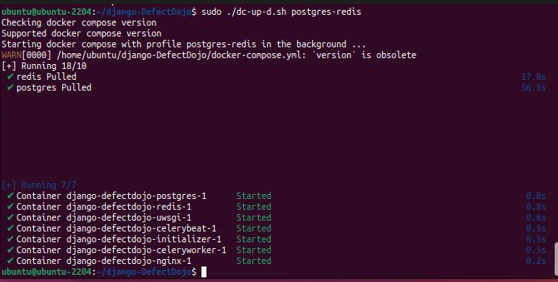
**

Рис. 3.17. Установка базы данных

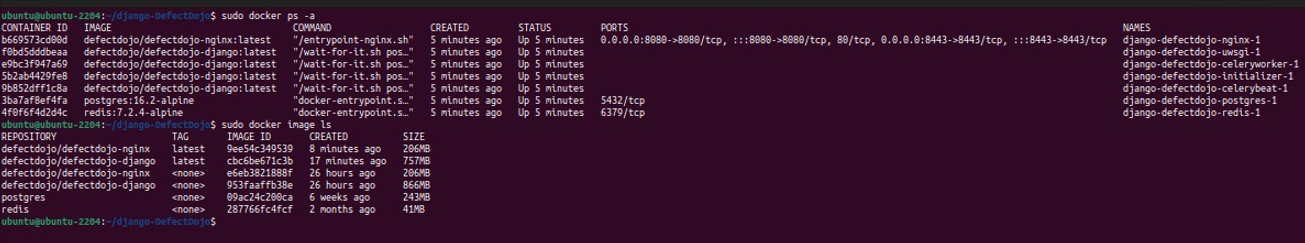
**

Рис. 3.18. Загруженные образы и запущенные docker-контейнеры

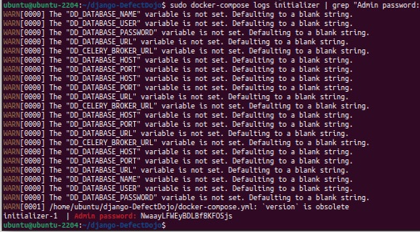
**

Рис. 3.19. Получения пароля к учетной записи администратора

Основная цель DefectDojo - предоставить инструменты для эффективного управления процессом обнаружения, отслеживания и устранения уязвимостей. Также, приложение:

- позволяет организациям отслеживать и управлять уязвимостями, обнаруженными в их приложениях и инфраструктуре. Пользователи могут просматривать список уязвимостей, классифицировать их по типам и критичности, а также назначать ответственных за их устранение;

- поддерживает импорт результатов сканирования уязвимостей из различных инструментов, таких как OWASP ZAP, Burp Suite, Nessus, Qualys и другие. Это позволяет пользователям централизованно хранить результаты сканирования и управлять ими;

- после импорта уязвимостей в DefectDojo, пользователи могут устанавливать приоритеты, назначать ответственных и отслеживать процесс устранения уязвимостей. Это помогает организациям эффективно планировать и реализовывать меры по устранению уязвимостей;

- позволяет генерировать различные отчеты о состоянии безопасности приложений и инфраструктуры. Пользователи могут создавать отчеты о количестве и типах обнаруженных уязвимостей, статусе их устранения, а также о мероприятиях, предпринятых для улучшения безопасности;

- обладает возможностью интеграции с другими инструментами и сервисами, такими как JIRA, GitHub, GitLab и др. Это облегчает совместную работу команд по безопасности и интеграцию процессов управления уязвимостями с другими инструментами разработки и управления проектами;

- предоставляет ряд дополнительных функций, таких как управление проектами, обзор кода, отслеживание изменений и многое другое, что делает его полезным инструментом для организаций, стремящихся к повышению безопасности своих приложений и инфраструктуры.

На рис. 3.20 приведен главный экран установленного приложения.

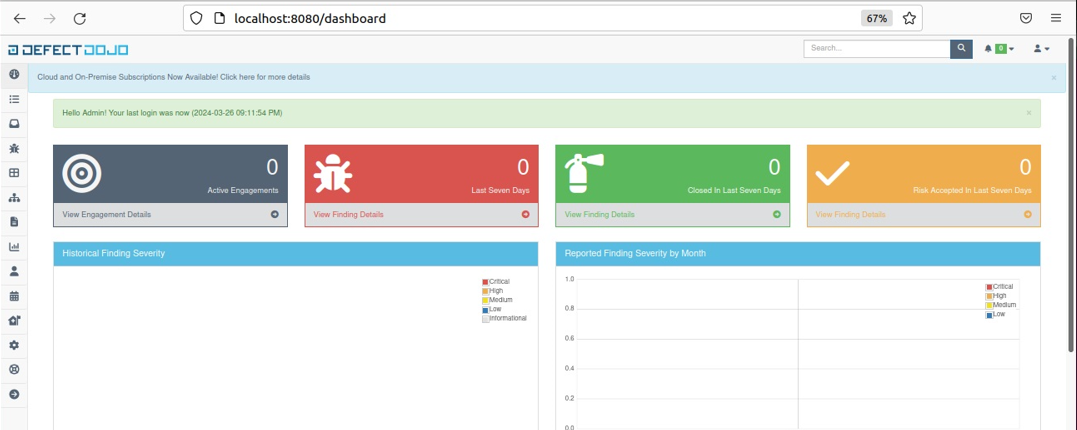
**

Рис. 3.20. Главный экран приложения DefectDojo

Как уже было предложено выше, для управления уязвимостями, найденными в нашем приложении UnSAFE\_Bank, будет использовано приложение DefectDojo.

Для организации интеграции и загрузки отчетов в результате работы нашего сканера, нам необходимо соблюсти несколько условий:

* + - 1. Получение токена для аутентификации (его можно найти в приложении, как отображено на рисунке 3.21).

Вот пример кода, как должна выглядеть реализация подключения:

import requests

url = 'http://127.0.0.1:8000/api/v2/users'

headers = {'content-type': 'application/json',

'Authorization':'Token c8572a5adf107a693aa6c72584da31f4d1f1dcff'}

r = requests.get(url, headers=headers, verify=True) # set verify to False if ssl cert is self-signed

for key, value in r.\_\_dict\_\_.items():

print(f"'{key}': '{value}'")

print('------------------')

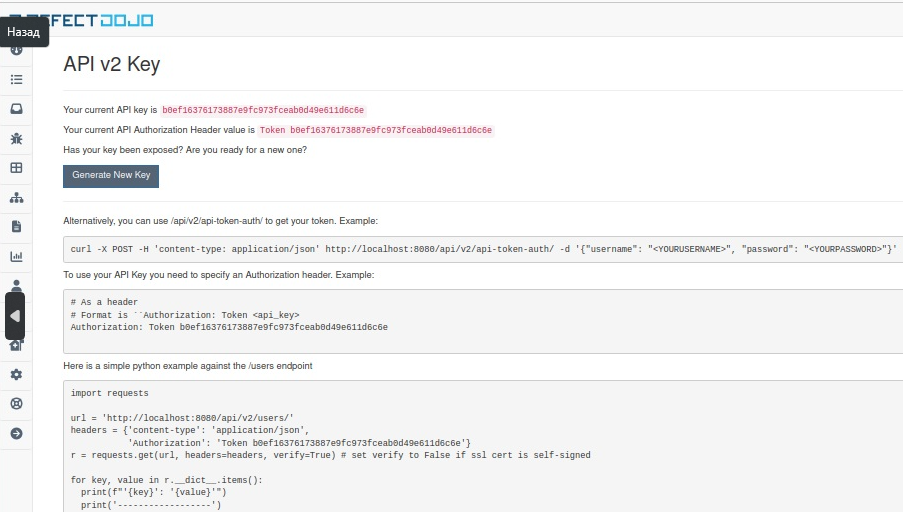
**

Рис. 3.21. Генерация токена для интеграции и подключения к DefectDojo

* + - 1. Формат загружаемых данных - JSON.

Программа поддерживает достаточно большое количество приложений, из которых возможна загрузка отчетов в формате .json.

Не будем подробно погружаться в документацию DefectDojo (более подробно можно познакомиться здесь https://defectdojo.github.io/django-DefectDojo/integrations/api-v2-docs/), просто сделаем акцент на данных моментах и учтем их при написании нашего сканера.

В следующем пункте приведено описание разработанного сканера.

## 3.2 Разработка и тестирование скрипта поиска и устранения уязвимостей

Скрипт разработан для автоматизации процесса обнаружения уязвимостей в исходном коде приложения и создания отчета для дальнейшего анализа. Опишем из чего состоит разработанный сканер.

Для разработки скрипта были установлены библиотеки:

os, time,

requests (для выполнения HTTP-запросов),

json (для работы с данными в формате JSON),

re (для работы с регулярными выражениями),

paramiko (для подключения к удаленному хосту по ssh).

Скрипт находится на удаленном хосте, для начала процесса сканирования необходимо указать данные для входа на удаленный хост, а также указать путь к директории, в которой находится исходный код нашего приложения (в нашем случае, приложение написано на языке программирования PHP, файлы которого и будут сканироваться, рис. 3.22).

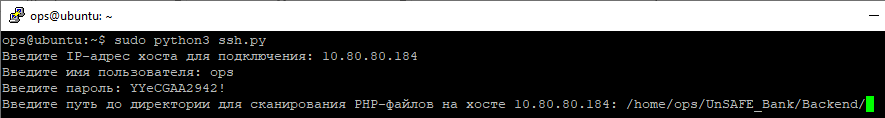


Рис. 3.22. Вход на удаленный хост

При запуске скрипт сначала загружает локальную базу уязвимостей из файла nvdcve-1.1-modified.json. Эта база содержит информацию о существующих уязвимостях, которая используется для сопоставления с данными из исходного кода.

Далее скрипт начинает сканирование исходного кода приложения, перебирая каждый файл в указанной папке. Он ищет совпадения с известными уязвимостями из загруженной базы и записывает найденные уязвимости в список «findings.json». Если локальная база уязвимостей не содержит информации о какой-либо уязвимости, скрипт запускает сканирование на типовые уязвимости, такие как SQL Injection, XSS и CSRF.

Затем скрипт приводится к необходимому формату данных, который в дальнейшем может быть загружен в DefectDojo через API. Отчет содержит информацию о найденных уязвимостях и метаданные, такие как заголовок и описание (полученные результаты представлены на рис. 3.23).

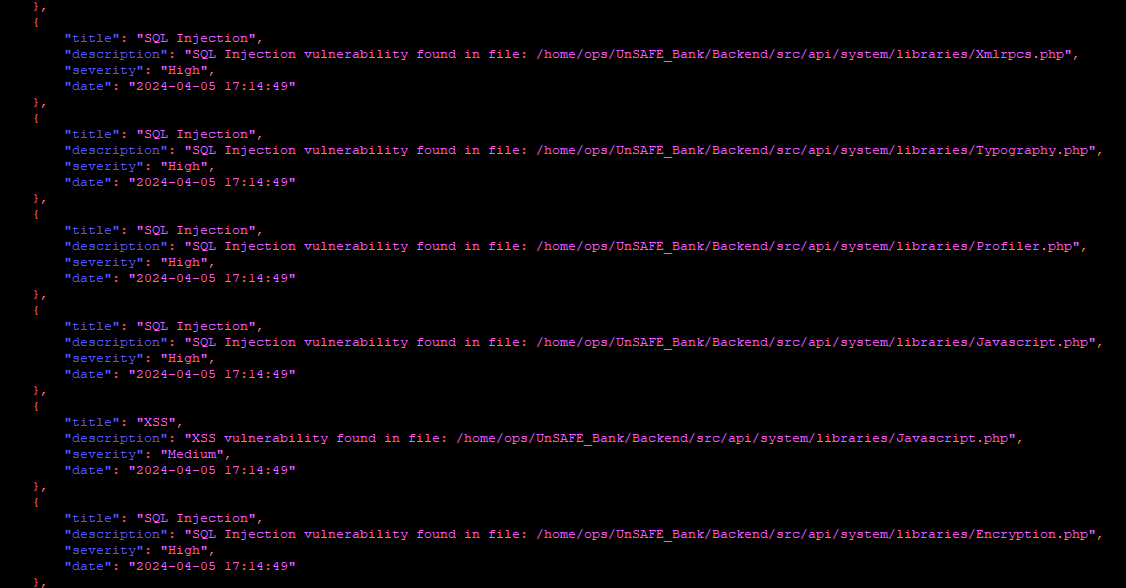


Рис. 3.23. Результаты сканирования

В процессе выполнения скрипт выводит сообщения о текущих действиях, такие как обнаружение уязвимостей и результаты загрузки отчета в DefectDojo. Если уязвимости не обнаружены, выводится соответствующее сообщение. Таким образом, скрипт обеспечивает автоматизированный процесс обнаружения уязвимостей и создания отчета для дальнейшего анализа и управления ими.

Листинг 3.1 скрипта представлен в приложении 1, вывод найденных уязвимостей представлен в приложении в файле findings.json (прилагается в директории артефактов).

## Выводы по главе 3

В главе 3 мы подробно разобрали лабораторный стенд, на котором проводилось тестирование, провели процесс создания скрипта на языке Python, позволяющего реализовать автоматизированный поиск наиболее известных уязвимостей в программном обеспечении, используемом в финансовой сфере.

# ГЛАВА 4. Оценка экономической составляющей работы

Большинство OSS инструментов доступны бесплатно или по оплате подписки, что снижает первоначальные затраты на их приобретение. Предварительная настройка и обучение персонала может потребовать некоторого времени и усилий, но это обычно компенсируется гибкостью и долгосрочной экономией.

После успешного завершения настройки и обучения, персонал будет более подготовленным к применению инструментов анализа исходного кода в различных сценариях. Обученные сотрудники смогут адаптироваться к новым требованиям и использовать их более эффективно. Это в свою очередь может привести к сокращению времени на выполнение работ, снижению риска возникновения ошибок и увеличению качества разработки.

Разработка и доработка собственного ПО - это серьезный процесс, который обычно включает следующие финансовые и временные затраты:

- заработная плата сотрудникам (программистам, инженерам, дизайнерам и другим участникам команды разработки);

- обучение персонала (в том числе повышение их квалификации);

- формирование необходимой инфраструктуры (технические мощности, серверы, облака для хранения и обработки данных и прочее);

- покупка лицензий на различные инструменты разработки и компоненты ПО, а также затраты на регистрацию прав интеллектуальной собственности;

- тестирование и обеспечение качества программного продукта также требует соответствующих финансовых затрат и временных ресурсов;

Чем более сложные и объемные функциональные требования к ПО, тем больше времени потребуется для его разработки и тестирования. Необходимость внутреннего согласования и обратной связи между различными отделами компании также может повлиять на временные затраты.

В таблице 4.1 приведен краткий обзор программных продуктов, имеющихся на рынке услуг.

Таблица 4.1

Обзор программных продуктов.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | Название программного продукта | Вендор | Краткое описание функционала | Сведения о стоимости (платное/бесплатное) |
| 1. | Nessus | Tenable | представляет собой мощный инструмент для сканирования уязвимостей, который обеспечивает охват как внутренних, так и внешних атак на сеть и веб-приложения, имеет обширную базу данных уязвимостей и предоставляет детальные отчеты о найденных проблемах | предлагает различные планы ценообразования, начиная от базового бесплатного варианта для небольших сетей до корпоративных лицензий с расширенным функционалом и поддержкой, стоимость которых может достигать нескольких тысяч долларов в год |
| 2. | Qualys Vulnerability Management | Qualys | предлагает облачное решение для управления уязвимостями, включая сканирование сети и веб-приложений, обладает высокой производительностью и поддерживает широкий спектр устройств и платформ |  |
| 3. | Acunetix | Acunetix | специализируется на обнаружении уязвимостей в веб-приложениях, инструмент предоставляет мощные возможности сканирования и анализа, включая проверку на SQL-инъекции, XSS-атаки и другие распространенные уязвимости | зависит от выбранного плана лицензирования и объема использования.  Цены начинаются от нескольких тысяч долларов в год для базовых версий и могут значительно увеличиваться для корпоративных планов с расширенным функционалом и поддержкой |
| 4. | MaxPatrol | Positive Technologies  (Россия) | предоставляет возможности сканирования сетей и приложений на наличие уязвимостей, а также анализ безопасности приложений, включает в себя функции аудита безопасности и управления рисками | цены обычно формируются по индивидуальным запросам |
| 5. | PT Application Inspector | Positive Technologies  (Россия) | для обнаружения уязвимостей в веб-приложениях и сетевой инфраструктуру, осуществляет сканирование на наличие уязвимостей и предоставляет детальные отчеты о найденных проблемах, специализируясь на обеспечении безопасности в финансовой сфере | для получения информации о стоимости и лицензировании следует обращаться к производителю |
| 6. | Acronis Cyber Protect | Acronis  (производство продукции в России) | предлагает не только поиск уязвимостей, но и решения для защиты данных, резервного копирования, обнаружения вторжений и многие другие функции, включая защиту от угроз в реальном времени | цены могут изменяться в зависимости от выбранного плана и обстоятельств |
| 7. | InfoWatch Traffic Monitor | InfoWatch  (Россия) | предназначен для мониторинга и анализа трафика в корпоративных сетях с целью обнаружения угроз безопасности и утечек конфиденциальной информации | цены обычно предоставляются по запросу |
| 8. | Group-IB Threat Intelligence | Group-IB  (Россия) | решения для обеспечения безопасности финансовых учреждений, включая продукты по обнаружению уязвимостей в сетевой инфраструктуре и веб-приложениях, обеспечивает мониторинг и анализ угроз, а также поиск уязвимостей в реальном времени | информацию о стоимости следует уточнять у поставщика |
| 9. | AppChecker | ЗАО «НПО Эшелон»  (Россия) | статический анализатор исходного кода | лицензия на использование ПО стоит от 400 000 рублей в год |
| 10. | Solar appScreener | Ростелеко-Солар  (Россия) | предназначен для анализа приложений с целью выявления уязвимостей информационной безопасности и недекларированных возможностей, можно восстанавливать исходный код приложений из рабочего файла с использованием технологии декомпиляции (обратной инженерии) | лицензия стоит от  2 340 000 рублей в год |
| 11. | WhiteSource Software | Mend.io |  | цены начинаются от 4000 до 148 тысяч долларов в месяц |
| 12. | Semgrep |  | инструмент статического анализа для поиска ошибок в коде | цена начального уровня от 40 долларов в месяц |
| 13. | Kiuwan Code Security & Insights | Kiuwan | комплексная платформа безопасности приложений (SAST, SCA и QA для выявления уязвимости в приложениях) | цены начинаются от 600 долларов |
| 14. | Визуальный эксперт | Novalys | [инструмент статического анализа кода](https://en.wikipedia.org/wiki/Static_program_analysis), извлекающий проектную и техническую информацию из [исходного кода](https://en.wikipedia.org/wiki/Source_code) программного обеспечения путем [обратного инжиниринга](https://en.wikipedia.org/wiki/Reverse_engineering), используемый программистами для обслуживания, модернизации или оптимизации программного обеспечения | от 500 до 8500 тысяч долларов (автономная или плавающая лицензии) |
| 15. | SonarQube | Sonar | комплексный инструмент анализа кода | есть как бесплатная версия, так и для разработчиков, цены варьируются от 160 долларов до 136 тысяч долларов в год |
| 16. | OpenVAS | Greenbone | инструмент для сканирования уязвимостей в сети, который предоставляет возможность обнаруживать уязвимости в операционных системах, сетевых устройствах и веб-приложениях. Он обладает гибкой настройкой и широким функционалом, несмотря на свою бесплатность | Бесплатное ПО |
| 17. | Nexpose Community Edition | Rapid7 | инструмент обеспечивает сканирование сети и поиск уязвимостей | Бесплатное ПО |
| 18. | OWASP ZAP (Zed Attack Proxy) |  | для тестирования безопасности веб-приложений. Он предоставляет широкий набор функций, включая сканирование на наличие уязвимостей, интерцептирование и изменение трафика, а также тестирование на возможность эксплуатации | Бесплатное ПО |

Как видно из таблицы, имеется достаточно большое количество инструментов как платных, так и бесплатных.

Использование платных продуктов требуют ежегодной или единовременной платы, их стоимость может варьироваться в зависимости от функциональности и лицензии. Порядок денежного вознаграждения за пользование платными продуктами обычно осуществляется на основе лицензионной модели. Вот несколько типичных вариантов:

1. Подписка на базовый пакет: пользователи могут приобрести ежемесячную или ежегодную подписку на базовый набор функций продукта.

2. Подписка на расширенные функции: некоторые продукты предлагают дополнительные функции или возможности, которые доступны только при дополнительной плате. Пользователи могут выбирать из различных планов подписки в зависимости от своих потребностей.

3. Оплата за количество активных устройств/пользователей: для продуктов, ориентированных на управление уязвимостями в сети или веб-приложениях, размер оплаты может зависеть от количества активных устройств или пользователей, подключенных к системе.

4. Лицензирование по использованию ресурсов: некоторые продукты могут предлагать модель лицензирования, основанную на объеме сканирования, обработанных запросов или других ресурсов, используемых в процессе анализа уязвимостей.

5. Индивидуальное ценообразование: для крупных организаций или проектов с особыми требованиями по функциональности и масштабу использования, поставщики могут предложить индивидуальные планы ценообразования и условия лицензирования.

Помимо этого, цены могут также варьироваться в зависимости от региона, длительности подписки, объема скидок и дополнительных услуг, предоставляемых поставщиком. Рекомендуется обращаться к конкретным поставщикам для получения подробной информации о порядке денежного вознаграждения за использование их продуктов.

Интеграция платных продуктов обычно требует меньше времени на начальную настройку и обучение по сравнению с разработкой собственного ПО.

## Выводы по главе 4

В главе 4 рассмотрены экономические затраты на применение открытых и платных решений. Выбор между открытым ПО, разработкой собственного или использованием платных продуктов требует внимательного анализа и оценки различных факторов, таких как цена, функциональность, интеграция и требования проекта. Важно провести сравнительный анализ и выбрать вариант, который наилучшим образом соответствует потребностям компании и позволит достичь поставленных целей более эффективно.

# ЗАКЛЮЧЕНИЕ

В ходе выполнения выпускной квалификационной работы рассмотрены особенности открытого программного обеспечения, изучены встречающиеся уязвимости в программном обеспечении, в том числе, используемом в финансовой сфере. На основании выявленных требований был сформулирован цикл по оценке защищенности программных средств, а также разработана методика поиска уязвимостей.

В качестве собственного продукта был разработан скрипт на языке Python для автоматизированного поиска уязвимостей в открытом программном обеспечении «UnSAFE\_Bank». Тестирование продукта проводилось на лабораторном стенде, предоставленном руководителем работы. По результатам тестирования подготовлены рекомендации по защите от выявленных уязвимостей.

Произведен анализ экономических показателей для использования платных продуктов по выявлению уязвимостей в программном обеспечении.

Дальнейшая работа по тематике, затронутой в выпускной квалификационной работе, заключается в практическом развертывании сценария по выявлению угроз и его плавной интеграции в конвейеры непрерывного развертывания (CI/CD). Данное встраивание гарантирует, что базы кода автоматически сканируются на наличие уязвимостей на ключевых этапах жизненного цикла разработки программного обеспечения, например, во время запросов на включение или перед развертыванием.

# ПРИЛОЖЕНИЕ 1. Программа сканирования На Уязвимости

Листинг 3.1

import os *# Модуль для работы с операционной системой*

import json  *# Модуль для работы с JSON-файлами*

import time *# Модуль для работы со временем*

import requests *# Модуль для отправки HTTP-запросов*

import re *# Модуль для работы с регулярными выражениями*

import paramiko *# Модуль для работы с SSH-подключениями*

*# Загрузка базы уязвимостей из файла JSON*

def load\_vulnerability\_database(vulnerability\_database):

*"""*

*Загружает базу уязвимостей из файла JSON.*

*Parameters:*

*vulnerability\_database (str): Путь к файлу с базой уязвимостей.*

*Returns:*

*list: Список уязвимостей.*

*"""*

*# Проверка существования файла с базой уязвимостей*

if os.path.isfile(vulnerability\_database):

*# Если файл существует, открываем его и загружаем данные*

with open(vulnerability\_database, 'r') as f:

return json.load(f)

else:

*# Если файла не существует, выводим ошибку*

print("Ошибка: Некорректный путь к файлу с базой уязвимостей")

return []

*# Запись результатов сканирования в файл JSON*

def write\_findings\_to\_file(findings, output\_file):

*"""*

*Записывает найденные уязвимости в файл JSON.*

*Parameters:*

*findings (list): Список найденных уязвимостей.*

*output\_file (str): Имя файла для записи результатов.*

*"""*

*# Открытие файла для записи*

with open(output\_file, 'w') as f:

*# Запись результатов в файл с отступами для удобного чтения*

json.dump(findings, f, indent=4)

*# Загрузка отчета о найденных уязвимостях в DefectDojo*

def upload\_report\_to\_defectdojo(report\_data, url, api\_key, product\_name, engagement\_name):

*"""*

*Загружает отчет о найденных уязвимостях в DefectDojo.*

*Parameters:*

*report\_data (dict): Данные отчета.*

*url (str): URL DefectDojo.*

*api\_key (str): API ключ.*

*product\_name (str): Название продукта.*

*engagement\_name (str): Название проекта.*

*"""*

*# Формирование заголовков запроса*

headers = {

'Authorization': f'Token {api\_key}',

'Content-Type': 'application/json'

}

try:

*# Отправка POST-запроса на указанный URL*

response = requests.post(url, headers=headers, json=report\_data)

*# Проверка статуса ответа*

if response.status\_code == 201:

print("Отчет успешно загружен в DefectDojo.")

else:

*# Вывод ошибки при неудачной загрузке отчета*

print(f"Ошибка при загрузке отчета в DefectDojo: {response.status\_code} - {response.text}")

except Exception as e:

*# Вывод ошибки при возникновении исключения*

print(f"Ошибка при отправке запроса: {e}")

*# Подключение к удаленному хосту по SSH*

def ssh\_connect(host, username, password):

*"""*

*Устанавливает SSH-соединение с удаленным хостом.*

*Parameters:*

*host (str): IP-адрес хоста.*

*username (str): Имя пользователя.*

*password (str): Пароль.*

*Returns:*

*paramiko.SSHClient: Объект клиента SSH.*

*"""*

try:

*# Создание объекта SSH-клиента*

client = paramiko.SSHClient()

*# Установка политики добавления новых хостов в список доверенных*

client.set\_missing\_host\_key\_policy(paramiko.AutoAddPolicy())

*# Подключение к удаленному хосту*

client.connect(host, username=username, password=password)

return client

except paramiko.AuthenticationException:

*# Обработка ошибки аутентификации*

print("Ошибка аутентификации. Проверьте правильность имени пользователя и пароля.")

return None

except paramiko.SSHException as e:

*# Обработка ошибки SSH*

print("Ошибка SSH:", str(e))

return None

*# Сканирование PHP-файла на уязвимости*

def scan\_php\_file(file\_path, ssh\_client, vulnerabilities):

*"""*

*Сканирует PHP-файл на наличие уязвимостей.*

*Parameters:*

*file\_path (str): Путь к PHP-файлу.*

*ssh\_client (paramiko.SSHClient): Объект клиента SSH.*

*vulnerabilities (list): Список уязвимостей.*

*Returns:*

*list: Список найденных уязвимостей.*

*"""*

findings = []

*# Получение содержимого PHP-файла через SSH*

stdin, stdout, stderr = ssh\_client.exec\_command("cat " + file\_path)

php\_code = stdout.read().decode('utf-8')

*# Поиск уязвимостей в коде PHP*

for vuln in vulnerabilities:

if vuln in php\_code:

*# Если уязвимость найдена, добавляем информацию о ней в результаты*

findings.append({

"title": vuln,

"description": f"Уязвимость {vuln} обнаружена в файле: {file\_path}",

"date": time.strftime("%Y-%m-%d %H:%M:%S")

})

*# Поиск типовых уязвимостей (SQL Injection, XSS, CSRF)*

sql\_pattern = re.compile(r'\b(select|insert|update|delete|drop|truncate|create|alter)\b', re.IGNORECASE)

xss\_pattern = re.compile(r'<script>', re.IGNORECASE)

csrf\_pattern = re.compile(r'<form\b[^<]\*(?:(?!<\/form>)<[^<]\*)\*<\/form>', re.IGNORECASE)

if sql\_pattern.search(php\_code):

findings.append({

"title": "SQL Injection",

"description": f"SQL Injection vulnerability found in file: {file\_path}",

"date": time.strftime("%Y-%m-%d %H:%M:%S")

})

if xss\_pattern.search(php\_code):

findings.append({

"title": "XSS",

"description": f"XSS vulnerability found in file: {file\_path}",

"date": time.strftime("%Y-%m-%d %H:%M:%S")

})

if csrf\_pattern.search(php\_code):

findings.append({

"title": "CSRF",

"description": f"CSRF vulnerability found in file: {file\_path}",

"date": time.strftime("%Y-%m-%d %H:%M:%S")

})

return findings

*# Преобразование результатов сканирования в формат DefectDojo*

def convert\_to\_defectdojo(input\_json):

*"""*

*Преобразует список уязвимостей в формат для загрузки в DefectDojo.*

*Parameters:*

*input\_json (list): Список уязвимостей.*

*Returns:*

*dict: Данные в формате DefectDojo.*

*"""*

defectdojo\_json = {

"product": {

"name": "Название продукта",

"description": "Описание продукта"

},

"engagement": {

"name": "Название проекта",

"description": "Описание проекта",

"target\_start": "YYYY-MM-DD",

"target\_end": "YYYY-MM-DD"

},

"test": {

"test\_type": "Тип тестирования",

"environment": "Окружение тестирования"

},

"findings": []

}

*# Преобразование каждого элемента JSON в объект finding для DefectDojo*

for item in input\_json:

finding = {

"title": item["title"],

"file\_path": item["file"],

"severity": "Low",

"description": "Описание уязвимости"

}

defectdojo\_json["findings"].append(finding)

return defectdojo\_json

*# Основная функция*

def main():

*# Получение данных для подключения к удаленному хосту*

host = input("Введите IP-адрес хоста для подключения: ")

username = input("Введите имя пользователя: ")

password = input("Введите пароль: ")

*# Установка SSH-соединения*

ssh\_client = ssh\_connect(host, username, password)

if ssh\_client:

try:

*# Получение пути к директории для сканирования PHP-файлов*

directory\_path = input("Введите путь до директории для сканирования PHP-файлов на хосте {}: ".format(host))

vulnerability\_database = "nvdcve-1.1-modified.json"

*# Загрузка базы уязвимостей*

vulnerabilities = load\_vulnerability\_database(vulnerability\_database)

findings = []

*# Поиск PHP-файлов в указанной директории*

stdin, stdout, stderr = ssh\_client.exec\_command("find " + directory\_path + " -name '\*.php'")

php\_files = stdout.read().decode('utf-8').split('\n')

php\_files = php\_files[:-1]

*# Сканирование каждого PHP-файла на уязвимост*и

for file\_path in php\_files:

findings.extend(scan\_php\_file(file\_path, ssh\_client, vulnerabilities))

if findings:

*# Если найдены уязвимости, записываем результаты в файл и выводим отчет в формате DefectDojo*

print("Найдены уязвимости:")

for finding in findings:

print(finding)

output\_file = "finding.json"

write\_findings\_to\_file(findings, output\_file)

*# Преобразование результатов в формат DefectDojo*

defectdojo\_data = convert\_to\_defectdojo(findings)

print("Отчет в формате DefectDojo:")

print(json.dumps(defectdojo\_data, indent=4))

else:

*# Если уязвимостей не найдено, выводим соответствующее сообщение*

print("Уязвимостей не найдено.")

except Exception as e:

*# Обработка общих ошибок выполнения программы*

print("Произошла ошибка:", str(e))

finally:

*# Закрытие SSH-соединения при завершении программы*

ssh\_client.close()

if \_\_name\_\_ == "\_\_main\_\_":

main()

# Список литературы

1. Федеральный закон от 27 июля 2006 г. № 149-ФЗ «Об информации, информационных технологиях и о защите информации».
2. Постановление Правительства РФ от 1 августа 2008 г. № 583 «О требованиях к защите информации, содержащейся в государственных информационных системах».
3. Федеральный закон от 26 июля 2006 г. № 152-ФЗ «О персональных данных».
4. Приказ ФСТЭК России от 18 февраля 2013 г. № 21 «Об утверждении методических рекомендаций по обеспечению информационной безопасности при использовании информационных систем обработки персональных данных».
5. Приказ ФСТЭК России от 26 октября 2015 г. № 93 «Об утверждении Правил проведения аттестации защищенности информации».
6. Постановление Правительства РФ от 20 марта 2018 г. № 319 «Об утверждении требований к системам защиты информации, используемым при обработке персональных данных».
7. «Методические рекомендации по организации защиты информации в информационных системах персональных данных» (утверждены приказом ФСТЭК России от 17 мая 2013 г. № 64).
8. Федеральный закон от 26 июля 2017 г. № 187-ФЗ «О безопасности критической информационной инфраструктуры Российской Федерации».
9. Постановление Правительства РФ от 26 марта 2019 г. № 363 «О требованиях к обеспечению безопасности критической информационной инфраструктуры Российской Федерации».
10. Федеральный закон от 21 июля 1997 г. № 116-ФЗ «О промышленной безопасности опасных производственных объектов».
11. Федеральный закон от 9 июля 1999 г. № 160-ФЗ «О безопасности объектов транспортной инфраструктуры».
12. Федеральный закон от 4 декабря 2006 г. № 190-ФЗ «О техническом регулировании».
13. Руководство ФСТЭК России по обеспечению безопасности критической информационной инфраструктуры (Безопасность критической информационной инфраструктуры. Методические рекомендации).
14. ГОСТ Р ИСО/МЭК 27001-2013 «Информационная технология. Системы менеджмента информационной безопасности. Требования».
15. ГОСТ Р ИСО/МЭК 27002-2007 «Информационная технология. Методы и средства обеспечения информационной безопасности. Практическое руководство».
16. Национальный стандарт ГОСТ Р 55728-2013 «Информационная технология. Методы и средства обеспечения безопасности. Критерии безопасности для критической информационной инфраструктуры».
17. ГОСТ Р 57580 Безопасность финансовых (банковских) операций   
    (устанавливает для стандартного и усиленного уровней защиты информации однократное тестирование на этапе «Ввод в эксплуатацию АС» и ежегодное тестирование на этапе «Эксплуатация (сопровождение) АС»).
18. Положение Банка России от 17.04.2019 № 683-П (ежегодное тестирование для кредитных организации).
19. Положение Банка России от 04.06.2020 № 719-П (ежегодное тестирование для операторов по переводу денежных средств, банковских платежных агентов (субагентов), операторов услуг информационного обмена, операторов услуг платежной).
20. Положение Банка России от 20.04.2021 № 757-П (ежегодное тестирование для некредитных финансовых организаций, реализующих усиленный и стандартный уровни защиты информации).
21. Положение Банка России от 8 апреля 2020 г. № 716-П (устанавливает ежегодное тестирование для кредитных организаций).
22. Статья «Какими уязвимостями в финансовых приложениях киберпреступники пользуются чаще всего». Режим доступа: <https://www.anti-malware.ru/analytics/Threats_Analysis/Popular-financial-apps-flaws>
23. Статья «Информационная безопасность в финансовом секторе: киберпреступность и стратегия противодействия», Семеко Г.В.
24. Статья «Актуальные веб-уязвимости сервисов финансово-банковского сектора и способы защиты веб-приложения», Жилина А.А.
25. Статья: https://www.securitylab.ru/news/443406.php
26. Статья «Уязвимости приложений финансовой отрасли». Режим доступа: https://vc.ru/u/1427864-elena-krosheva/745153-uyazvimosti-prilozheniy-finansovoy-otrasli