

ТАНЯ ЯНКА

ИСЧЕРПЫВАЮЩИЙ ГИД  
ДЛЯ НАЧИНАЮЩИХ  
РАЗРАБОТЧИКОВ

# БЕЗОПАСНОСТЬ WEB-ПРИЛОЖЕНИЙ

ПРОЕКТИРОВАНИЕ  
БЕЗОПАСНОЙ  
АРХИТЕКТУРЫ

МЕХАНИЗМЫ  
ЗАЩИТЫ  
ДАННЫХ

ТЕСТИРОВАНИЕ НА  
ПРОНИКНОВЕНИЕ

 **БОМБОРА**  
ИЗДАТЕЛЬСТВО

Таня Янка

**Безопасность веб-приложений**

*Искрывающий гид для начинающих  
разработчиков*

Alice and Bob Learn Application Security  
Tanya Janca © 2021 by John Wiley & Sons, Inc., Indianapolis, Indiana.  
All Rights Reserved. This translation published under license with the  
original publisher John Wiley & Sons, Inc

© Райтман М. А., перевод на русский язык, 2023  
© Оформление. ООО «Издательство «Эксмо», 2023

\* \* \*

# Отзывы о книге

## Тани Янка «Безопасность веб-приложений. Визуальный гид для начинающих разработчиков»

*Таня знает свое дело. Она обладает огромным объемом знаний и опыта в сфере безопасности приложений, DevSecOps и облачной безопасности. Мы все можем многое узнать от Тани, так что стоит прочитать ее книгу!*

*Дэвид Штуттارد, соавтор бестселлера *The Web Application Hacker's Handbook*, создатель приложения *Virp Suite**

*Я так много узнала из этой книги! Информационная безопасность действительно является работой каждого специалиста. Книга представляет собой потрясающее изложение обширных знаний, необходимых каждому: разработчику, специалисту по инфраструктуре, безопасности и многим другим. Благодарю госпожу Янку за написание такого познавательного и полезного учебника. Мне понравились правдоподобные истории с описанием реальных проблем, охватывающие всё, начиная с проектирования, миграции приложений из проблемных фреймворков, минимизации административных рисков и заканчивая вещами, которые должен знать каждый современный разработчик.*

*Джен Ким, автор бестселлера *The Unicorn Project*, соавтор *The Phoenix Project, DevOps Handbook* и *Accelerate**

*Практическое руководство для современной эпохи. Таня отлично рассказывает о современном представлении о безопасности приложений понятным для всех языком.*

**Трой Хант, создатель веб-сайта *Have I Been Pwned?***

Я посвящаю эту книгу моей неутомимой группе поддержки: Лекси, Матеусу, Эшу и Вейну. Постоянно поддерживая, ободряя и отмечая завершение каждого этапа создания книги, вы не позволили мне бросить начатое. Также спасибо, что не осуждаете меня за то, сколько мороженого я съела во время редактирования.

# Об авторе

**Таня Янка**, также известная под ником SheHacksPurple, является основательницей We Hack Purple, онлайн-академии, сообщества и канала подкастов, цель которых – обучение всех желающих созданию безопасного программного обеспечения. Она также является соучредителем компании WoSEC: Women of Security, руководит проектом OWASP DevSlop и отделением OWASP Victoria. Таня занимается программированием и работает в области IT более двадцати лет. За это время она завоевала множество наград, успела потрудиться везде, от стартапов до государственных организаций и технологических гигантов (Microsoft, Adobe и Nokia). Она занимала различные должности: была основателем стартапа, пентестером (тестировщиком, проверяющим уязвимость киберзащиты информационной системы), директором по информационной безопасности, инженером по безопасности приложений и разработчиком программного обеспечения. Будучи превосходным оратором, активным блогером и стримером, она провела сотни выступлений и тренингов на шести континентах. Она ценит разнообразие, вовлеченность и человеколюбие, что проявляется в ее бесчисленных инициативах.

# О технических редакторах

**Доминик Ригетто** начал свою карьеру в сфере разработки программного обеспечения, а восемь лет спустя перешел в область обеспечения безопасности, продолжая жить на границе двух миров. Доминику очень интересны наступательные и оборонительные аспекты безопасности приложений. В сфере безопасности (как и на всем протяжении профессиональной жизни) его главной целью было помогать командам разработчиков прагматически подходить к обеспечению безопасности своих проектов. С 2011 года Доминик является активным членом фонда OWASP, в рамках которого участвует в различных проектах, в основном касающихся его специализации в области доменов. Являясь приверженцем философии открытого исходного кода, в свободное время он участвует в различных проектах, соответствующих этой идеи. Его домашняя страница – [righettod.eu](http://righettod.eu).

**Эли Саад** – опытный специалист в области информационной безопасности, работающий в банковской сфере. Он участвует в различных инициативах OWASP по стандартизации и регулярно публикует статьи по этой теме. Его основная цель – дать разработчикам программного обеспечения рекомендации по обеспечению безопасности и защите. Он провел несколько лекций, в которых знакомил новичков с безопасностью, и был гостем подкаста на платформе Security Journey, где рассказывал о различных проектах, связанных с безопасностью приложений. Он является сторонником разрушения фрагментированной культуры в мире безопасности приложений. Кроме того, Эли с удовольствием находит время для более простых вещей в жизни, хороший передышки в горах и стакана вкусного виски (односолодового, конечно). Вы можете найти его в Twitter ([@7hunderSon](https://twitter.com/@7hunderSon)) и на GitHub ([thunderson](https://github.com/thunderson)). С ним можно связаться также по электронной почте [eliesaad7@gmail.com](mailto:eliesaad7@gmail.com).

## **Благодарности**

Я хотела бы поблагодарить моего издателя Джима Минателя за то, что он помог мне понять, что я готова написать книгу, и определиться с ее типом. Спасибо редактору Адаоби Оби Тултону за его бесконечное терпение, методическую помощь и эффективный контроль, которые помогли мне осуществить самый крупный проект в моей профессиональной жизни. Спасибо моим техническим редакторам Доминику Ригетто и Эли Сааду. Я никогда не смогу расплатиться с вами за вашу усердную работу по недопущению серьезных технических ошибок в книге. У меня нет слов, чтобы выразить мою признательность за ваше потраченное время, опыт и поддержку. Спасибо всем, кто отправился со мной в этот путь.

# Предисловие

За последние несколько лет в области безопасности приложений был достигнут значительный прогресс. Многие факторы заставляют организации заботиться о безопасности своего программного обеспечения: рост числа инцидентов, произошедших в результате использования небезопасного программного обеспечения; увеличение количества нормативных актов, которые предписывают компаниям заботиться об информационной безопасности, а также растущая зависимость от интернет-ориентированного программного обеспечения.

Организации любого размера и в любом секторе бизнеса и государственного управления сталкивались с утечками данных и связанными с ними потерями. Однако увеличение количества инцидентов в области информационной безопасности также способствует повышению осведомленности, которая помогает организациям и их разработчикам создавать более безопасное программное обеспечение.

Вот где Алиса и Боб вступают в игру.

В книге Тани тема безопасности приложений излагается в ясной и лаконичной форме, что позволяет применять полученные знания сразу, по мере прочтения. Главы изобилуют рассказами об Алисе и Бобе и о том, как принимаемые нами решения по безопасности влияют на жизни реальных людей. Книга начинается с объяснения важности этой темы, а затем излагаются все основные понятия безопасности, о которых мы все, кажется, откуда-то знаем, но никогда до конца не уверены в своих знаниях.

Для всего, начиная с требований безопасности для веб-приложений и заканчивая принципами проектирования безопасности, руководствами по написанию безопасного кода и распространенными «подводными камнями», в этой книге подобрано большое количество историй, примеров с Алисой или Бобом и схем. В ней также рассказывается о тестировании и развертывании программного обеспечения. Однако эта книга определенно не о «хакерстве». Она о том, как обеспечить прочность, надежность и безопасность

приложений. В ней описывается, как создать безопасную программу, как обезопасить современные технологии и системы, какие привычки помогут разработчикам (или кому угодно) защитить себя и свои системы, и даже приводится план обучения в конце! Данную книгу, наполненную советами, хитростями и даже шутками, нельзя назвать обычным учебником.

Я надеюсь, она вам понравится так же, как и мне, и вы присоединитесь к нам с Таней в борьбе за правое дело создания безопасного программного обеспечения.

*Джим Манико, основатель и преподаватель по написанию  
безопасного кода в школе Manicode Security*

# Введение

Почему именно безопасность приложений? Зачем нужна эта книга? Почему безопасность важна? Почему данная тема вызывает большие трудности?

Если вы взяли в руки эту книгу, то, вероятно, уже знаете ответ на эти вопросы. Вы видели газетные заголовки о фирмах, которые были «хакнуты»: данные, в том числе личного характера, – украдены, компании и жизни – разрушены. Однако вы, возможно, не знаете, что причиной нарушения безопасности данных номер один является небезопасное программное обеспечение, из-за которого происходит от 26 до 40 % случаев утечек и краж (Verizon Breach Report, 2019)<sup>[1]</sup>. Однако если посмотреть на бюджеты большинства компаний, то сумма, выделяемая на защиту их программного обеспечения, обычно составляет очень малую часть.

Большинство организаций на данный момент обладают высоким уровнем защиты сетевого периметра (с помощью брандмауэров), безопасности предприятия (блокирование вредоносных программ и запрет прав администратора для большинства пользователей) и физической безопасности (пропуск на вход и выход из безопасных зон). Тем не менее создание безопасного программного обеспечения все еще остается труднодостижимой целью для большинства организаций. Почему?

Прямо сейчас университеты и колледжи обучают программированию, но не учат тому, как обеспечить безопасность написанного кода, и даже основам информационной безопасности. Большинство программ послешкольного образования, в которых рассматриваются вопросы безопасности, лишь едва касаются защиты приложений, концентрируясь вместо этого на идентификации, сетевой безопасности и инфраструктуре.

Представьте, что кто-то учился на электрика, но так и не узнал о технике безопасности. Дома периодически бы загорались из-за того, что электрики не знали, как обеспечить безопасность выполняемой ими работы. Позволять студентам, изучающим инженерные науки и информатику, получить высшее образование с недостаточной

подготовкой в области безопасности не менее опасно, поскольку они создают программное обеспечение для кардиостимуляторов, для защиты государственной тайны и многоного другого, от чего зависит наше общество.

Это одна часть проблемы.

Другая ее часть заключается в том, что обучение (на английском языке) обычно стоит больших денег, что делает его недоступным для многих людей. Также не существует четкого карьерного пути или учебной программы, позволяющей человеку стать разработчиком безопасного кода, архитектором информационной безопасности, специалистом по реагированию на инциденты или инженером по безопасности приложений. Большинство людей в конечном итоге проходят обучение на рабочем месте, а это означает, что каждый из нас имеет совершенно разные представления о том, какие действия необходимо осуществлять, и получает разные результаты.

Следует также учесть, что преступления, совершенные в интернете, приносят большую выгоду, а поскольку провести атрибуцию (выявить того, кто совершил преступление) очень сложно, существует огромное количество угроз для любого интернет-приложения. Чем ценнее система или данные в ней, тем большему числу угроз она подвергается.

Последняя часть проблемы заключается в том, что обеспечить безопасность приложений довольно сложно. В отличие от безопасности инфраструктуры, где все версии Microsoft Windows Server 2008 R2 PS2 абсолютно одинаковы, каждая часть пользовательского программного обеспечения – уникальная снежинка. При строительстве деревянной террасы на заднем дворе вы идете в хозяйственный магазин, чтобы купить древесину размером два на четыре дюйма длиной восемь футов. В какой бы магазин вы ни пошли, древесина будет везде одинаковой, а значит, вы можете делать предположения и расчеты без существенных рисков. С программным обеспечением так не получится. Никогда нельзя делать никаких предположений, нужно проверять каждый факт. Следовательно, запоминание методом «грубой силы», автоматизированные инструменты и другие универсальные решения работают редко, что делает обеспечение безопасности приложений очень сложной задачей.

## Сдвиг влево

Если посмотреть на жизненный цикл разработки системы (англ. System Development Life Cycle, SDLC) на рис. В.1, можно увидеть, что все этапы сменяют друг друга слева направо. Требования предшествуют проектированию, за которым идет кодирование. Независимо от того, используете ли вы Agile, Waterfall, DevOps или любую другую методологию разработки программного обеспечения, всегда нужно узнать, какое ПО вы создаете (требования), составить план (проектирование), разработать его (кодирование), проверить, что оно выполняет все необходимые функции и ничего больше (тестирование), затем выпустить готовое ПО и поддерживать его работу (релиз).



Рис. В.1. Жизненный цикл разработки системы

Часто деятельность по обеспечению безопасности начинается на этапах выпуска или тестирования ПО – далеко справа и на довольно поздней стадии проекта. Проблема состоит в том, что чем позже исправлять недостаток (проблема проектирования) или ошибку (проблема реализации), тем дороже это обойдется и тем сложнее будет это сделать.

Позвольте мне объяснить мою мысль по-другому. Представьте, что Алиса и Боб строят дом. Они копили на него годами, и вот подрядчики завершают работу: клеят обои и прикручивают ручки на шкафчики. Вдруг Алиса поворачивается к Бобу и говорит: «Дорогой, у нас двое детей, а ванная комната только одна! Как мы станем делить ее?» Если сказать подрядчикам прекратить работу, дом не будет закончен вовремя. Если попросить пристроить вторую ванную комнату, где она должна находиться? Сколько это будет стоить? Обнаружение проблемы на столь поздней стадии проекта может привести к катастрофе. Однако если бы Алиса и Боб узнали о ней на этапе

разработки требований или на этапе проектирования, то можно было бы легко добавить больше ванных комнат за очень небольшие деньги. То же самое справедливо и для решения проблем безопасности.

Именно здесь вступает в игру «сдвиг влево»: чем раньше вы начнете заниматься обеспечением безопасности во время проекта по разработке программного обеспечения, тем лучше будут результаты. Стрелки на рис. В.2 показывают последовательность действий по обеспечению безопасности, которые следует начинать как можно раньше в проекте. Позже мы обсудим, что это за действия.



Рис. В.2. Сдвиг влево

## О книге

Эта книга научит вас основам безопасности приложений (сокращенно AppSec, от англ. Application Security), то есть тому, как создавать безопасное программное обеспечение. Она предназначена для разработчиков, специалистов по информационной безопасности, желающих узнать больше о безопасности программного обеспечения, и всех, кто хочет работать в этой области (которая включает в себя тестирование на проникновение, также известное как «этический взлом»).

Если вы разработчик, ваша работа заключается в создании наиболее безопасного программного обеспечения, которое вы способны сделать. Вашу ответственность здесь нельзя недооценивать. На каждого инженера безопасности в этой области приходятся сотни программистов, и без вас им не справиться. Эта книга – первый шаг на правильном пути: после ее прочтения вы будете иметь достаточно знаний для создания безопасного программного обеспечения, а также знать, где искать ответы в случаях, вызывающих затруднения.

Примечания к формату: в книге будут приведены примеры того, как проблемы безопасности могут повлиять на реальных пользователей. На всем ее протяжении будут периодически появляться персонажи Алиса и Боб. Их можно увидеть в различных примерах, касающихся безопасности. Они используются для упрощения сложных тем в нашей отрасли с момента появления криптографии и шифрования.

## **Темы, выходящие за рамки книги**

Хотелось бы сделать небольшое замечание по темам, которые выходят за рамки данной книги: реагирование на инциденты (IR), сетевой мониторинг и оповещение, облачная безопасность, безопасность инфраструктуры, сетевая безопасность, операции по обеспечению безопасности, управление идентификацией и доступом (IAM), безопасность предприятия, поддержка, антифишинг, обратная разработка, обfuscация кода и другие передовые методы защиты, а также все остальные типы безопасности, не перечисленные здесь. О некоторых из них мы поговорим в книге, но данную информацию ни в коем случае нельзя считать исчерпывающей. Чтобы узнать больше об этих важных темах, обратитесь к дополнительным ресурсам.

## Ответы

В конце каждой главы приведены задания, которые помогут вам усвоить материал и проверить знания. В конце книги есть раздел с ответами, однако не на все задания. Многие вопросы требуют написания эссе, проведения исследовательской работы или онлайн-дискуссии, в то время как другие носят личностный характер (только вы можете ответить, с какими препятствиями можете столкнуться на работе). Таким образом, раздел состоит из ответов (когда это возможно), примеров (когда это уместно) и некоторых пропущенных вопросов, оставленных для обсуждения в интернете.

В течение нескольких месяцев после выхода этой книги на сайте [youtube.com/shehackspurple](https://youtube.com/shehackspurple) в плейлисте «Алиса и Боб изучают безопасность приложений» будут появляться видео, где разбираются ответы на все вопросы. Вы можете подписаться на канал, чтобы не пропустить новые видео, посмотреть предыдущие и ознакомиться с другими бесплатными материалами.

Вы можете принять *живое* участие в обсуждениях, подписавшись на сайте [newsletter.shehackspurple.ca](https://newsletter.shehackspurple.ca) на рассылку SheHacksPurple, чтобы получать приглашения на стримы (а также много другого бесплатного контента).

Участие в дискуссиях или их последующий просмотр бесплатны. Из услышанных мнений, идей, историй успехов и неудач других людей вы можете многое для себя почерпнуть. Пожалуйста, присоединяйтесь к нам.

# **Часть I**

## **Все, что нужно знать о коде, безопасном для публикации в интернете**

**Глава 1.** Основы безопасности

**Глава 2.** Требования безопасности

**Глава 3.** Безопасность при проектировании ПО

**Глава 4.** Безопасность кода ПО

**Глава 5.** Часто встречающиеся подводные камни

# Глава 1

## Основы безопасности

Прежде чем учиться создавать безопасное программное обеспечение, необходимо разобрать несколько ключевых концепций, касающихся безопасности. Нет смысла запоминать, как реализовать ту или иную концепцию, не понимая, когда и зачем она нужна. Знание основ позволит вам принимать безопасные проектные решения и аргументировать необходимость повышения уровня безопасности в случае возникновения возражений. Кроме того, понимание того, на чем базируются правила безопасности, заметно облегчает работу с ними.

### **Обязательство по обеспечению безопасности: «CIA»**

Обязательство и цель каждой команды ИТ-безопасности заключается в защите *конфиденциальности, целостности и доступности* систем и данных компаний, правительства или организации, на которую команда работает. Вот почему служба безопасности беспокоит вас по поводу наличия ненужных прав администратора на вашем рабочем устройстве, не позволяет подключать неизвестные устройства к сети и требует выполнения всех остальных, как кажется, слишком сложных действий. Она хочет защитить эти три аспекта, которые для краткости называются «триадой CIA» (confidentiality, integrity, availability – конфиденциальность, целостность и доступность) (рис. 1.1).



Рис. 1.1. Триада CIA – причина существования команд IT-безопасности

Рассмотрим данную триаду на примере наших друзей Алисы и Боба. Алиса страдает диабетом I типа и несколько раз в день использует имплантированное в руку крошечное устройство для проверки уровня инсулина. У Боба есть «умный» кардиостимулятор, регулирующий работу сердца, доступ к которому он получает через мобильное приложение на телефоне. Оба этих устройства в нашей отрасли называются имплантируемыми медицинскими устройствами IoT.

**ПРИМЕЧАНИЕ.** IoT значит Internet of Things – интернет вещей. Это физические продукты, подключенные к интернету. Умный тостер или холодильник, подключенный к интернету, является устройством IoT.

## Конфиденциальность

Алиса – генеральный директор крупной компании, входящей в список Fortune 500. Она не стыдится своей болезни, но не хочет, чтобы эта информация стала достоянием общественности. Алиса часто дает интервью СМИ и выступает с публичными заявлениями, став примером для подражания для многих других женщин, работающих в ее отрасли. Она прилагает все усилия, чтобы сохранить в тайне сведения о личной жизни, в том числе и информацию о состоянии здоровья. Алиса считает, что некоторые люди в организации хотят занять ее место и пойдут на все, чтобы попытаться подорвать ее авторитет, представив ее «слабой». Случайно произошедшая утечка

информации с устройства в сеть или взлом аккаунта поставили бы ее в затруднительное положение и могли бы навредить карьере. Для Алисы важно сохранить свою личную жизнь в тайне.



Рис. 1.2. Конфиденциальность: обеспечение сохранности информации

Боб, напротив, открыто говорит о своем заболевании сердца и с радостью рассказывает всем о кардиостимуляторе. У него отличная страховка от федерального правительства, и он высоко ценит возможность продолжать пользоваться ею после выхода на пенсию, несмотря на то что болезнь была обнаружена еще до начала работы на организацию. В данном случае конфиденциальность не является для Боба приоритетом (рис. 1.2).

**ПРИМЕЧАНИЕ.** Мы часто недооцениваем роль конфиденциальности в нашей жизни. Многие люди уверяют меня, что им «нечего скрывать». Тогда я спрашиваю: «У вас дома есть занавески на окнах? Да? Почему? Вам же нечего скрывать». Я просто жгу на вечеринках.

## Целостность

Целостность (рис. 1.3) означает, что данные являются актуальными, правильными и точными. Целостность также подразумевает, что данные не были изменены во время передачи: необходимо сохранять правильность значения. Мы говорим о целостности компьютерной системы, когда результаты, которые она выдает, точны и правдивы.



Рис. 1.3. Целостность означает корректность

Для Боба и Алисы целостность, возможно, является самым важным из элементов триады CIA: некорректная работа одной из систем может привести к смерти. Для человека (в отличие от компании или государства) не существует более серьезного последствия, чем конец жизни. Таким образом, решающее значение для Боба и Алисы имеет целостность систем, связанных с их здоровьем.

Модель CIA – это центральное ядро всей нашей отрасли. Создание безопасного программного обеспечения невозможно без понимания сущности этой модели и ее влияния на членов команды, на программное обеспечение и, самое главное, на пользователей.

## Доступность

Если бы прибор для измерения инсулина вышел из строя из-за неисправности, взлома или севших батареек, у Алисы не было бы к нему доступа. Обычно она проверяет уровень инсулина несколько раз в день, но в случае необходимости может провести тестирование вручную (уколов палец и воспользовавшись соответствующим медицинским набором), поэтому доступность прибора для нее важна лишь отчасти. Отсутствие доступа к этой системе было бы для нее весьма неудобным, но не опасным для жизни обстоятельством.

У Боба же время от времени сбивается ритм сердцебиения, и он никогда не знает, в какой момент наступит аритмия. Если бы в периоды нестабильной работы сердца у Боба отсутствовал доступ к кардиостимулятору, то по прошествии времени это могло бы привести к критической для его жизни ситуации. Очень важно, чтобы при возникновении чрезвычайной ситуации кардиостимулятор был доступен и реагировал в режиме реального времени (немедленно).

Уже много лет Боб работает клерком в федеральном правительстве и занимается секретными и сверхсекретными документами. Будучи счастливым дедушкой, с момента установки кардиостимулятора он старательно ведет здоровый образ жизни.

**ПРИМЕЧАНИЕ.** Медицинские приборы, как правило, являются системами ПО, работающими в режиме реального времени. «Режим реального времени» означает, что система должна реагировать на изменения в кратчайшие сроки, обычно исчисляющиеся в миллисекундах. Она не может работать с задержками – отклик должен быть максимально быстрым или немедленным. Когда у Боба начинается аритмия, кардиостимулятор должен сработать немедленно – задержки быть не может. В таком режиме функционируют только немногие приложения: 10-миллисекундная задержка при покупке новых кроссовок или при прогнозировании изменений дорожного трафика не приведет к *действительно* большой проблеме.



Рис. 1.4. Устойчивость повышает доступность

**ПРИМЕЧАНИЕ.** Многие пользователи переводят информацию в облако по той единственной причине, что оно чрезвычайно надежно (почти всегда доступно) по сравнению с более традиционными уровнями обслуживания во внутренних центрах обработки данных. Как показано на рис. 1.4, устойчивость повышает доступность, что делает публичное облако привлекательным вариантом с точки зрения безопасности.

Ниже перечислены подходы к обеспечению безопасности, которые хорошо известны в индустрии информационной безопасности. Важно хорошо усвоить их, чтобы понять, как к ним относятся остальные темы данной книги. Если вы уже работаете специалистом по обеспечению безопасности, возможно, вам не нужно читать эту часть.

## «Предполагать взлом»

«Есть два типа компаний: те, которые взломаны, и те, которые еще не знают об этом»<sup>[2]</sup>. Это настолько известное высказывание в индустрии информационной безопасности, что мы уже не знаем, кому его приписывать. Возможно, звучит пессимистично, но те из нас, кто работает в области реагирования на инциденты, в судебно-медицинской экспертизе или в других сферах, связанных с расследованиями, знают, что оно более чем правдиво.

Принцип «предполагать взлом» означает выполнить подготовку и принять конструктивные решения, гарантирующие, что получение несанкционированного доступа к сети, приложению, данным или другим системам окажется сложным, трудоемким, дорогостоящим и рискованным делом, а обнаружить взлом и отреагировать на него можно будет быстро. Данный подход также подразумевает мониторинг и протоколирование систем: если вы не заметите взлом до того, как он произойдет, то по крайней мере сможете узнать, что случилось. Многие системы отслеживают изменения в поведении или аномалии системы для выявления потенциальных взломов. Задача данного подхода – заблаговременная подготовка к худшему, чтобы минимизировать ущерб, время обнаружения и усилия по устраниению последствий взлома.

Рассмотрим два примера применения этого принципа: пример с потребителем и пример с профессионалом.

Как потребитель Алиса использует для обмена документами аккаунт в интернете. Если бы она «предполагала взлом», то не стала бы загружать в него ничего личного или ценного (например, незарегистрированную интеллектуальную собственность, фотографии личного характера, которые могут повредить профессиональной или частной жизни, коммерческие и государственные тайны и т. д.). Она также установила бы мониторинг учетной записи и разработала бы план на случай кражи, изменения, удаления, публичного распространения информации или другого несанкционированного доступа к документам. Наконец, она периодически проверяла бы весь интернет на предмет утечки данных. Все вышеперечисленные действия представляли бы собой нереальный уровень

ответственности, который не ожидается от обычного потребителя. Эта книга не советует «предполагать взлом» в повседневной жизни, хотя периодически проводить поиск информации в интернете – хорошая идея, а также определенно рекомендуется не загружать в сеть конфиденциальные документы.

Как профессионал Боб работает с секретными и сверхсекретными документами. В его отделе никто никогда бы не подумал об использовании онлайнового файлообменного сервиса для обмена документами. Сотрудники контролируют каждый аспект хранения и передачи ценной информации. Когда они создавали сеть и программный комплекс, управляющие документами, они проектировали их и связанные с ними процессы, *предполагая взлом*. Они охотятся за угрозами в своей сети, разрабатывают ее с использованием нулевого доверия, проверяют интернет на предмет признаков утечки данных, аутентифицируют API перед подключением, проверяют данные из базы данных и из внутренних API, проводят учения «красной команды» (тестирование безопасности на производстве) и внимательно следят за своей сетью и приложениями на предмет аномалий или других признаков взлома. Сотрудники отдела написали автоматизированные ответы на распространенные модели атак, имеют процессы, разработанные и готовые к нестандартным атакам, и анализируют поведенческие модели для выявления признаков нарушения. В своих действиях они исходят из того, что хранилища данных уже взломаны или могут оказаться взломаны в любой момент.

Другим примером может быть запуск процесса реагирования на инцидент, когда серьезная ошибка раскрывается через «координированное раскрытие» или программу вознаграждения за найденные ошибки (Bug Bounty), исходя из предположения, что кто-то другой потенциально уже нашел и использовал эту ошибку в системе.

Согласно «Википедии», «координированное раскрытие» – это модель раскрытия уязвимостей, при которой уязвимость или проблема раскрывается только по истечении периода времени, позволяющего устраниить либо исправить уязвимость или проблему.

Многие организации проводят программы Bug Bounty. Они выплачивают награду исследователям в области безопасности за

сообщения об ошибках в системе. Особо ценятся те ошибки, которые связаны с уязвимостями.

## **Внутренние угрозы**

Под внутренней угрозой подразумевается ситуация, когда человек, имеющий разрешение на доступ к системам, сети и данным (обычно это сотрудник или консультант компании), негативно влияет на один или несколько аспектов триады CIA. Угроза может быть умышленной (преднамеренной) или случайной.

### **Примеры умышленных угроз и затрагиваемых ими аспектов триады CIA**

- Сотрудник загружает интеллектуальную собственность на переносной диск, покидает здание, а затем продает информацию конкурентам (конфиденциальность).
- Сотрудник удаляет базу данных и ее резервную копию в свой последний рабочий день, так как злится по поводу увольнения (доступность).
- Сотрудник встраивает бэкдор (тайный вход) в систему для последующей кражи у компании (целостность и конфиденциальность).
- Сотрудник скачивает конфиденциальные файлы с компьютера другого сотрудника и использует их для шантажа (конфиденциальность).
- Сотрудник случайно удаляет файлы, а затем меняет логи, чтобы скрыть свою ошибку (целостность и доступность).
- Сотрудник не сообщает руководству об уязвимости, чтобы избежать работы по ее устранению (потенциально все три аспекта CIA, в зависимости от типа уязвимости).

### **Примеры случайных угроз и затрагиваемых ими аспектов триады CIA**

- Сотрудники ненадлежащим образом используют программное обеспечение, в результате чего оно переходит в неизвестное состояние (потенциально все три аспекта).
  - Сотрудник случайно удаляет ценные данные, файлы или даже целые системы (доступность).
  - Сотрудник случайно неправильно настраивает сеть или другое программное обеспечение, в результате чего появляются уязвимости в безопасности (возможно, все три аспекта).
  - Неопытный сотрудник направляет веб-прокси или инструмент, выполняющий динамическое тестирование безопасности приложений (Dynamic Application Security Testing, DAST), на одно из ваших внутренних приложений, что приводит к сбою приложения (доступность) или «засорению» базы данных (целостность). В последующих главах мы расскажем о том, как избежать такой ситуации и обеспечить успешное проведение всех тестирований безопасности.

**ВНИМАНИЕ.** Обычно в профессиональных рабочих сетях веб-прокси и DAST запрещены для использования. Известные также как «сканеры веб-приложений», веб-прокси являются инструментами хакеров и могут нанести большой ущерб. Никогда не направляйте сканер веб-приложений на веб-сайт или приложение и не проводите активное сканирование или другое интерактивное тестирование без письменного разрешения от уполномоченного лица. Использование инструмента DAST для взаимодействия с сайтом в интернете (без разрешения) является уголовно наказуемым деянием во многих странах. Будьте осторожны и при наличии сомнений всегда спрашивайте перед началом каких-либо действий.

## Глубокая защита

«Глубокая защита» предполагает наличие нескольких уровней безопасности на случай, если одного окажется недостаточно (рис. 1.5). Хотя данная идея может показаться очевидной при столь простом объяснении, иногда непросто определить, сколько уровней необходимо иметь и какие это должны быть уровни (особенно если выделенный на обеспечение безопасности бюджет ограничен).

«Уровнями» безопасности могут быть процессы (проверка удостоверения личности человека перед выдачей ему почты, прохождение тестирования безопасности перед релизом программного обеспечения), физические, программные или аппаратные системы (замок на двери, сетевой брандмауэр, аппаратное шифрование), встроенные варианты проектирования (написание отдельных функций для кода, выполняющего более важные задачи в приложении, обеспечение входа в здание через одну дверь) и т. д.

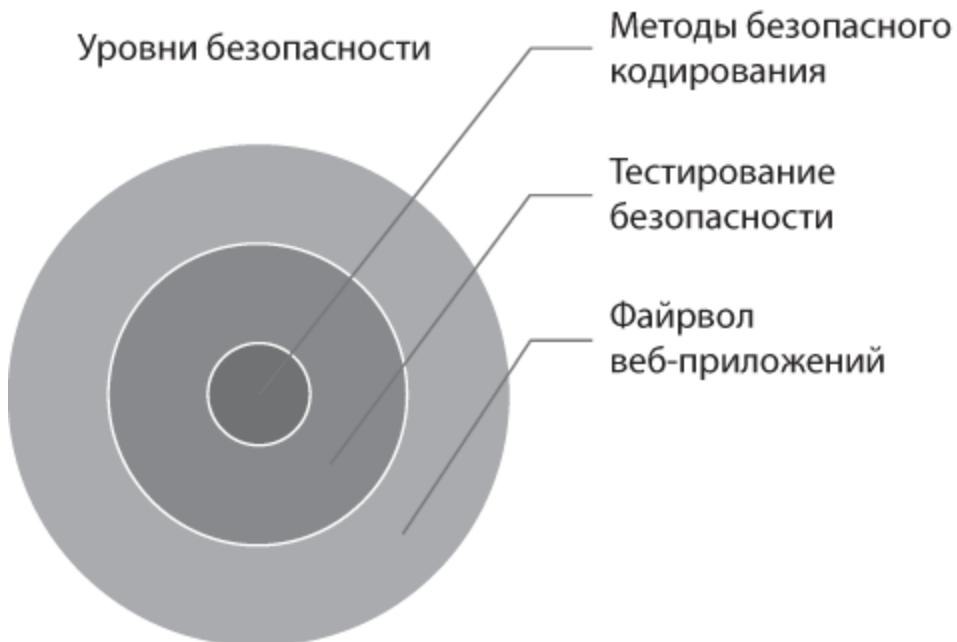


Рис. 1.5. Три уровня безопасности приложения; пример глубокой защиты

Примеры использования многочисленных уровней приведены ниже.

- **При создании программного обеспечения:** наличие требований безопасности, выполнение моделирования угроз, обеспечение использования концепций безопасного проектирования, обеспечение использования тактики безопасного кодирования, тестирование безопасности, тестирование несколькими способами посредством различных инструментов и т. д. Каждый из этих элементов представляет собой очередную форму защиты, что повышает уровень безопасности приложения.

- **Сетевая безопасность:** включение мониторинга, наличие системы SIEM (Security information and event management – «Управление информацией о безопасности и событиями безопасности», панель инструментов для просмотра возможных событий безопасности в режиме реального времени), наличие системы IPS/IDS (Intrusion prevention/detection system – «Система предотвращения и обнаружения вторжений», инструмент для поиска и препятствования действиям злоумышленников в сети), брандмауэры и многое другое. Каждый новый элемент усиливает защиту приложения.

- **Физическая безопасность:** замки, колючая проволока, заборы, ворота, видеокамеры, охранники, сторожевые собаки, датчики движения, сигнализация и т. д.

Довольно часто самое сложное при отстаивании интересов безопасности – это убедить в том, что одного уровня защиты недостаточно. В таких случаях подчеркивайте значимость того, что защищается (репутация, денежная ценность, национальная безопасность и т. д.). Хотя с экономической точки зрения не имеет смысла тратить миллион долларов на защиту чего-то стоимостью в одну тысячу долларов, в нашей отрасли очень часто наблюдаются примеры обратного.

**ПРИМЕЧАНИЕ.** Моделирование угроз – определение угроз, которым могут подвергаться приложения, и разработка планов по снижению их возникновения. Подробнее об этом см. в главе 3.

Система SIEM – мониторинг сети и приложений, панель инструментов для работы с возможными проблемами.

Система предотвращения и обнаружения вторжений (IPS/IDS) – программное обеспечение, установленное в сеть с целью обнаружения или предотвращения сетевых атак.

## Принцип наименьших привилегий

Принцип *наименьших привилегий* – это предоставление пользователям ровно такого уровня доступа и контроля, который им необходим для выполнения работы. Смысл этого подхода заключается в том, что если злоумышленнику удастся завладеть вашей учетной записью (или несколькими учетными записями), он получит доступ лишь к малой части данных. Например, разработчик программного обеспечения имеет доступ только к своему коду и доступ для чтения либо записи к единственной базе данных, над которой он работает. Следовательно, если кто-то взломает учетную запись разработчика, то получит только ту часть данных, к которой у него был доступ: к базе данных, коду, электронной почте и т. д. Однако, если бы злоумышленник получил доступ к учетной записи *владельца* всех баз данных, он мог бы уничтожить все. Возможно, это неприятно, но, отказываясь от своих суперсил на компьютере, в сети или других системах, вы значительно снижаете риск взлома этих систем.

Примеры реализации принципа наименьших привилегий таковы.

- Необходимость получения дополнительного разрешения от службы безопасности для доступа в лабораторию или часть здания с повышенным уровнем безопасности.
- Отсутствие привилегий администратора на рабочем компьютере.
- Наличие доступа для записи к своим проектам, доступа только для чтения ко всему коду своей команды и полное отсутствие доступа к хранилищам кода других команд.
- Создание сервисной учетной записи в приложении для входа в базу данных и предоставление разрешения на чтение и запись, но не доступ владельца базы данных (Database owner, DBO). Если приложению требуется доступ только для чтения, дайте ему не более того, что необходимо для нормальной работы. Сервисная учетная запись с доступом к базе данных только для чтения не может быть использована для изменения или удаления каких-либо данных, даже если через нее можно украсть копии данных. Все вышеперечисленное значительно снижает риски.

**ПРИМЕЧАНИЕ.** Разработчики программного обеспечения и системные администраторы являются привлекательными целями

для большинства злоумышленников, поскольку обладают наибольшими привилегиями. Отказавшись от некоторых из них, вы защитите свою компанию больше, чем можете предположить, и одновременно заслужите уважение команды безопасности.

## Безопасность цепи поставок

Каждый элемент, используемый для создания продукта, считается частью «цепи поставок». Цепь включает в себя участника (поставщика) от каждого элемента (производителя, магазина, фермы, человека и т. д.). Она называется цепью, так как при создании конечного продукта каждая ее часть зависит от предыдущей. Звеньями цепи могут быть люди, компании, природные или промышленные ресурсы, информация, лицензии и все, что необходимо для создания конечного продукта (который не обязательно должен быть физическим по своей природе).

Объясним цепь поставок на примере. Если Боб хочет смастерить кукольный домик для своих внуков, он может купить изготовленный на фабрике набор. Фабрике требуются древесина, бумага, краска, клей, машины для резки, люди для управления и обслуживания техники, а также энергия для ее питания. Древесину фабрика заказывает у лесозаготовительной компании, вырубающей деревья в лесу. Компания владеет этим лесом либо имеет лицензионный доступ на вырубку в нем. Бумага, краска и клей, скорее всего, производятся на разных заводах. Возможно, люди работают непосредственно на фабрике или являются временными подрядчиками. Энергия, скорее всего, поступает от энергетической компании, но нельзя исключать использование альтернативных источников (солнце или ветер) или генератора в случае чрезвычайной ситуации. На рис. 1.6 показана (гипотетическая) цепь поставок для приобретенного Бобом набора, из которого он сделает кукольный домик для внуков на Рождество.



Рис. 1.6. Вероятная цепь поставок для кукольного домика Боба

Какие угрозы безопасности могут возникнуть в данной ситуации? Клей, входящий в набор, может оказаться ядовитым, а краска, используемая для украшения деталей, – токсичной. Кукольный домик может изготавливаться на предприятии, где также обрабатываются орехи. В результате произойдет перекрестное загрязнение коробок, которое способно вызвать аллергическую реакцию у некоторых детей. В набор могут попасть неправильные элементы, например острые детали, которые не подходят для маленького ребенка. Все эти ситуации, вероятнее всего, возникают на фабрике непреднамеренно.

При разработке программного обеспечения мы также используем цепь поставок: платформу – для написания кода, библиотеки – для записи на экране, выполнения сложных математических вычислений или отрисовки кнопки, интерфейсы программирования приложений (API) – для выполнения действий от имени приложений и т. д. Более того, каждый из этих фрагментов программного обеспечения обычно зависит от других фрагментов, и все они чаще всего поддерживаются различными группами, компаниями или людьми. Как правило, современные приложения на 20–40 % состоят из оригинального кода<sup>[3]</sup> (того, что написали вы и ваши коллеги), в остальном – из сторонних компонентов, часто называемых зависимостями. При подключении

зависимостей к приложению вы принимаете на себя риски кода, который они содержат. Например, если вы добавите в приложение модуль для обработки изображений, вместо того чтобы самому написать часть соответствующего кода, и в этом модуле будет серьезный недостаток безопасности, то уязвимость приложения тоже значительно повысится.

Однако нельзя утверждать, что необходимо писать каждую строчку кода самостоятельно: такое решение крайне неэффективно, а вероятность допущения ошибок, приводящих к проблемам безопасности, все равно высока. Один из способов снизить риск – использовать меньше зависимостей и тщательно проверять те, которые вы решили включить в свое программное обеспечение. Многие инструменты (некоторые из них даже бесплатные) могут проверить наличие каких-либо известных проблем безопасности в используемых вами зависимостях. Их следует применять не только при каждом запуске нового кода, но и для регулярной проверки вашего репозитория кода.

---

## ПРИМЕР АТАКИ НА ЦЕПЬ ПОСТАВОК

В 2018 году модуль Node.js с открытым исходным кодом под названием `event-stream` был передан новому разработчику, который добавил в него вредоносный код. Он дождался, пока миллионы людей скачают его через NPM (Node Package Manager – систему управления пакетами для Node.js), а затем использовал эту уязвимость для кражи биткоинов с кошельков Copay, в которых применялась библиотека `event-stream`<sup>[4]</sup>.

---

Еще одной тактикой защиты от использования небезопасной цепи поставок программного обеспечения является применение платформ и других сторонних компонентов, которые были созданы известными компаниями или признаны группами, работающими с открытым исходным кодом, подобно тому как повар использует только самые

лучшие ингредиенты для приготовления своих блюд. Вы можете (и должны) проявлять осторожность при выборе компонентов, которые войдут в окончательную версию ваших продуктов.

Известно о небольшом количестве атак на цепи поставок, произошедших в последние несколько лет, когда злоумышленники внедряли уязвимости в программные библиотеки, микропрограммы (низкоуровневое программное обеспечение, являющееся частью оборудования) и даже в само оборудование. Эта угроза реальна, и принятие мер предосторожности против нее сослужит хорошую службу любому разработчику.

## **Безопасность через неясность**

Концепция *безопасности через неясность* подразумевает, что если какой-то фрагмент системы скрыт, то он находится в «большой безопасности», поскольку не попадает в поле зрения потенциальных злоумышленников. Наиболее распространенная реализация этой концепции: компании – разработчики программного обеспечения скрывают свой исходный код, а не выкладывают его в открытый доступ (с целью защиты интеллектуальной собственности и в качестве меры безопасности). Некоторые доходят до обfuscации своего кода, изменяя его таким образом, чтобы его было гораздо сложнее или вообще невозможно понять тому, кто попытается провести обратную разработку продукта.

**ПРИМЕЧАНИЕ.** *Обfuscация* – это усложнение чего-то для понимания или чтения. Распространенной тактикой является кодирование всего исходного кода в ASCII, Base64 или Hex, но ее довольно легко видят профессиональные реверс-инженеры. Некоторые компании дважды или трижды шифруют свой код. Другая тактика – применение операции XOR (команды ассемблера) или создание собственной схемы кодирования и добавление ее программным путем. Также можно купить продукты, выполняющие более сложную обfuscацию.

Другим примером «безопасности через неясность» является использование беспроводного маршрутизатора, скрывающего имя

SSID/Wi-Fi (то есть при подключении к сети необходимо вручную указать ее имя), или размещение веб-сервера без доменного имени в надежде, что никто его не найдет. Существуют инструменты для обхода данных мер, но риск атаки на беспроводной маршрутизатор или веб-сервер снижается.

Существует обратная концепция – «безопасность через открытость», которая подразумевает, что за программным обеспечением с открытым исходным кодом наблюдает больше глаз, а значит, эти глаза найдут уязвимости в безопасности и сообщат о них. На практике исследователи безопасности редко просматривают открытый код и сообщают об ошибках бесплатно. В проектах с открытым исходным кодом разработчики не всегда устраняют выявленные недостатки безопасности, а если уязвимости найдены, нашедший может решить продать их на черном рынке (преступникам, собственному либо иностранному правительству и т. д.), вместо того чтобы сообщить о них владельцу репозитория кода надежным способом.

Хотя сам по себе подход «безопасность через неясность» вряд ли является превосходным методом защиты, он, безусловно, полезен в качестве одного из уровней стратегии обеспечения безопасности – «защиты в глубину».

## Уменьшение поверхности атаки

Атаке подвержен каждый элемент программного обеспечения: любая функция, вход, страница или кнопка. Чем меньше приложение, тем меньше поверхность атаки. Четыре страницы с 10 функциями представляют гораздо меньшую поверхность атаки, чем 20 страниц со 100 функциями. Каждый элемент приложения, который может быть подвержен действию злоумышленника, считается поверхностью атаки.

Для уменьшения поверхности атаки нужно удалить из приложения все, что не является необходимым. Например, не до конца реализованная функция, уже имеющая неактивную кнопку, будет идеальным местом для начала атаки злоумышленника, поскольку она еще не полностью протестирована или усиlena. Перед публикацией в рабочей среде следует удалить этот код и дождаться готового варианта. Скрыть его недостаточно – необходимо уменьшить поверхность атаки, удалив эту часть кода.

**СОВЕТ.** Устаревшее программное обеспечение часто имеет очень большой объем неиспользуемой функциональности. Удаление ненужных функций – отличный способ уменьшить поверхность атаки.

Как вы помните, у Алисы и Боба есть медицинские имплантаты: устройство для измерения инсулина у Алисы и кардиостимулятор у Боба. Оба являются «смарт-устройствами», то есть к ним можно подключиться через смартфон. Устройство Алисы работает через Bluetooth, а кардиостимулятор Боба – через Wi-Fi. Одним из самых очевидных способов уменьшить поверхность атаки было бы не приобретать «смарт-версии» таких устройств. Однако в данном случае делать это уже поздно. Тем не менее Алиса могла бы отключить «видимость» Bluetooth у своего прибора для измерения инсулина, а Боб – скрыть SSID своего кардиостимулятора.

## Жесткое кодирование

Жесткое кодирование означает программирование значений в коде вместо получения их естественным путем (от пользователя, из базы данных, API и т. д.). Например, если в разработанном вами приложении-калькуляторе пользователь вводит  $4 + 4$ , нажимает **Enter** и на экране появляется 8, вы, скорее всего, решите, что калькулятор работает. Однако если пользователь вводит  $5 + 5$  и нажимает **Enter**, а на экране все равно отображается 8, это может говорить о возникновении ситуации жесткого кодирования.

Почему жесткое кодирование является потенциальной проблемой безопасности? Причина двоякая: вы не можете доверять выходным данным приложения, а значения, которые были жестко закодированы, часто имеют конфиденциальный характер (пароли, ключи от API, хеши и т. д.). Любой, кто зайдет в исходный код, будет иметь доступ к этим жестко закодированным значениям. Жесткое кодирование в исходном коде секретов, требующих постоянной сохранности, – далеко не безопасное решение.

Жесткое кодирование обычно считается симптомом плохой разработки программного обеспечения (есть и исключения). Если вы столкнулись с ним в одном месте приложения, следует проверить все приложение на его наличие, поскольку маловероятно, что найденный вами экземпляр – единственный.

## «Никогда не доверяй, всегда проверяй»

Если вы вынесете из этой книги только один урок, то он должен быть следующим: никогда не доверяйте никому за пределами вашего собственного приложения. Если приложение обращается к API, убедитесь, что это правильный API и что у него есть полномочия на то, что он пытается делать. Если приложение принимает данные из *любого источника*, выполняйте проверку данных (необходимо убедиться, что полученные данные правильные. Если же это не так, блокируйте их). Даже данные из вашей собственной базы могут содержать вредоносный ввод или другие средства заражения. Если пользователь пытается получить доступ к той части приложения, которая требует специального разрешения, перепроверяйте, есть ли у него разрешение на каждую используемую им страницу или функцию. Если пользователь прошел аутентификацию (доказал, что он тот, за

кого себя выдает) и переходит между страницами, необходимо все время удостоверяться, что это тот же самый пользователь (это называется управлением сессиями). Нельзя полагать, что одной проверки достаточно. Нужно постоянно проверять и перепроверять.

**ПРИМЕЧАНИЕ.** Мы проверяем данные из нашей базы, поскольку они могут содержать *хранимый межсайтовый скрипting* (*Cross-Site Scripting*, XSS) или другие значения, способные навредить программе. Хранимый XSS появляется в тех случаях, когда программа не выполняет надлежащую проверку вводных данных и случайно сохраняет XSS-атаку в своей базе. Когда пользователь в приложении выполняет действие, вызывающее вредоносный скрипт, начинается атака в браузере жертвы. Пользователь не в состоянии защититься от этой атаки, и, как правило, она считается критической опасностью, если обнаруживается во время тестирования безопасности.

Довольно часто разработчики забывают об этом уроке и полагаются на доверие из-за сложившихся условий. Например, к выпущенному вами интернет-приложению применяются чрезвычайно строгие меры безопасности. Внутри вашей сети (в обход брандмауэра) веб-приложение постоянно вызывает API (#1), который затем вызывает другой API (#2), изменяющий данные в соответствующей базе. Часто разработчики не утружддают себя аутентификацией (подтверждением личности) в первом API или проверкой API (#1) на наличие у приложения права на вызов той части API, к которой оно обращается. А если они и выполняют такую проверку, то часто применяют меры безопасности только для API #1, но не для API #2. В результате кто или что угодно в вашей сети может вызвать API #2, включая злоумышленников, которых там быть не должно, внутренние угрозы или даже случайных пользователей (рис. 1.7).

Приложение,зывающее API; при необходимости аутентификации

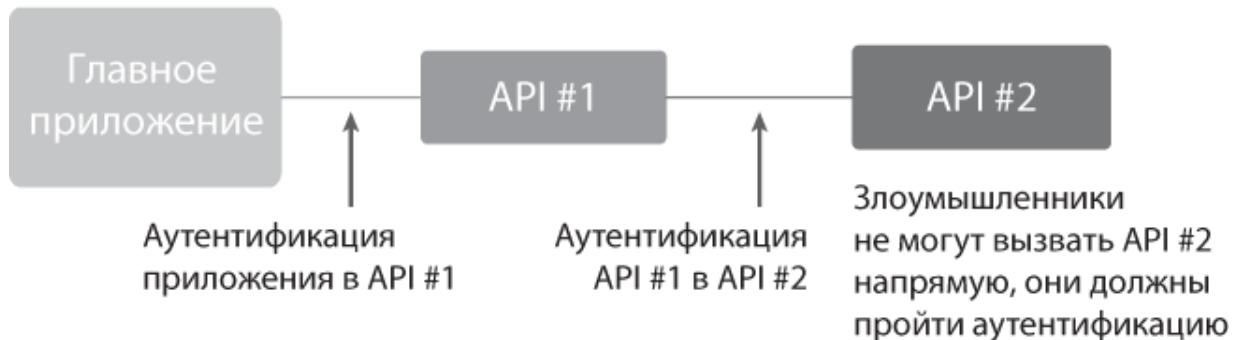


Рис. 1.7. Пример вызова API приложением и при необходимости аутентификации

Вот несколько примеров.

- Веб-сайт заражен хранимым межсайтовым скрипtingом, и злоумышленник использует его для хранения атаки в базе данных. Если веб-приложение проверяет данные, поступающие из базы данных, запуск сохраненной атаки будет безуспешным.
- Веб-сайт взимает плату за доступ к определенным данным, получаемым от API. Если пользователь знает, что API открыт для доступа в интернете, и не проверяет разрешение на его использование (аутентификация и авторизация), он может вызвать API напрямую и получить данные без оплаты (что было бы злонамеренным использованием сайта), то есть совершить кражу.
- Обычный пользователь приложения расстроен и многократно стучит по клавиатуре, случайно вводя гораздо больше данных, чем следовало. Если приложение правильно проверяет вводимые данные, оно отклонит их, так как количество символов превышает допустимый предел. Однако, если приложение не проверит эти данные, возможно, они перегрузят переменные или будут переданы в базу данных, следствием чего станет сбой. Без проверки соответствия получаемых данных ожиданиям (число в числовом поле, дата в поле даты, соответствующее количество вводимых символов и т. д.) приложение может перейти в неизвестное состояние со множеством ошибок безопасности. Приложение ни в коем случае не должно переходить в неизвестное состояние.

## Удобство и безопасность

Если элементы обеспечения безопасности делают ваше приложение сложным в использовании, пользователи найдут способ обойти защиту или уйдут к конкуренту. В интернете можно найти бесчисленное количество примеров того, как пользователи творчески обходят неудобные защитные меры, применяемые приложением. Люди хорошо умеют решать проблемы, и безопасность не должна становиться одной из них.

Решение заключается в создании *удобных* средств защиты. Хотя очевидно, что без доступа в интернет все наши приложения стали бы безопаснее, такая защита от угроз в интернете была бы непродуктивной. Необходимо проявить творческий подход и найти самый простой способ, но при этом и самый безопасный.

Вот примеры удобства и безопасности.

- Позволить использовать отпечаток пальца, распознавание лица или графический ключ для разблокировки персонального устройства вместо длинного и замысловатого пароля.

- Вместо установки правил сложности (обязательное использование специальных символов, цифр, строчных и прописных букв и т. д.) научить пользователей создавать парольные фразы (предложение или фразу, которую легко запомнить и набрать). Парольная фраза увеличит энтропию, которая затруднит злоумышленникам взлом, но в то же время упростит обеспечение защиты пользователям.

- Научить пользователей применять менеджеры паролей, а не ожидать, что они создадут и запомнят 100+ уникальных паролей для всех своих учетных записей.

А вот примеры обхода мер безопасности пользователями.

- Вход в охраняемое здание вместе с другим человеком (человек, который идет вплотную за другим, входящим в здание, может не проводить картой, чтобы попасть внутрь).

- Отключение телефона перед тем, как пройти через сканер, обнаруживающий передающие устройства. Затем, оказавшись в безопасной зоне, где мобильные телефоны запрещены, человек снова включает его.

- Использование прокси-сервиса для посещения веб-сайтов, которые заблокированы рабочей сетью.
- Фотосъемка экрана с целью получения изображения, защищенного авторским правом, или конфиденциальных данных.
- Использование одного и того же пароля раз за разом, но с увеличением последней цифры для удобства запоминания. Если ваша компания заставляет пользователей сбрасывать пароль каждые 90 дней, велика вероятность того, что в вашей организации есть немало паролей, соответствующих формату `ТекущееВремяГода_ТекущийГод`.

## Факторы аутентификации

Аутентификация – это предоставление компьютеру доказательства того, что вы – действительно настоящий, *подлинный* вы. «Фактор» аутентификации – метод доказательства компьютеру того, кто вы есть. В настоящее время существует только три фактора: *что у вас есть*, *что является частью вас*, и *что вы знаете*.

- *То, что у вас есть*, может быть телефоном, компьютером, ключом или рабочим бейджем. То, что должны иметь только *вы*.
- *То, что является частью вас*, может быть отпечатком пальца, радужной оболочкой глаза, походкой или ДНК. *Ваша* уникальная физическая особенность.
- *То, что вы знаете*, может быть паролем, парольной фразой, графическим ключом или комбинацией нескольких частей информации (часто называемых контрольными вопросами, например девичья фамилия матери, дата рождения и номер социального страхования). Идея фактора заключается в том, что эту информацию должны знать только *вы*.

Мы используем только один «фактор» аутентификации, когда входим в учетную запись в интернете посредством имени пользователя и пароля. Однако лучшим решением касательно безопасности является использование двух и более факторов. Взлом учетных записей или кража данных часто происходит из-за использования только одного фактора аутентификации. Использование более одного фактора обычно называют многофакторной аутентификацией (multi-factor authentication, MFA), двухфакторной аутентификацией (two-factor

authentication, 2FA) или двухэтапным входом в систему. Мы будем использовать аббревиатуру MFA.

**СОВЕТ.** Контрольные вопросы уже устарели. Ответы на большинство из них легко найти в интернете, выполнив сбор данных из открытых источников (OSINT, Open source intelligence – Разведка на основе открытых источников). Не используйте в своем программном обеспечении контрольные вопросы в качестве фактора аутентификации: злоумышленники слишком легко их обходят.

Учетные записи пользователей, использующих второй фактор аутентификации, защищены от взлома, производимого посредством украденных учетных данных (имени пользователя и пароля). Злоумышленник не сможет войти в систему, не пройдя второй фактор. Если кто-то пытается взломать систему или учетную запись с MFA методом грубой силы (используя сценарий быстрого автоматического перебора всех возможных вариантов), то, даже получив пароль, он не сможет войти в систему. Использование второго фактора значительно затрудняет взлом учетных записей в интернете.

Вот примеры MFA.

- **Многофакторный:** ввод имени пользователя и пароля, а затем использование второго устройства или физического ключа для получения кода аутентификации. Имя пользователя и пароль – первый фактор (то, что вы знаете), а использование второго устройства – второй фактор (то, что вы имеете).

- **Не многофакторный:** имя пользователя *и* пароль. Это два примера одного и того же фактора: они оба являются чем-то, что вы знаете. Многофакторная аутентификация подразумевает использование нескольких различных типов факторов аутентификации.

- **Не многофакторный:** использование имени пользователя и пароля, а затем ответ на контрольный вопрос. Два элемента *одного* фактора: что-то известное вам.

- **Многофакторный:** имя пользователя, пароль *и* отпечаток большого пальца.

**ПРИМЕЧАНИЕ.** Многие специалисты в области информационной безопасности расходятся во мнении, является ли использование телефона для получения SMS (текстового сообщения) с пин-кодом «хорошой» реализацией MFA, поскольку известны недостатки протокола SMS и некоторых его реализаций. Я считаю, что лучше иметь «достаточно хороший второй фактор», чем только один. Однако по возможности просите пользователей применять в качестве второго фактора приложение для аутентификации, а не SMS-сообщения.

## Упражнения

Цель данных упражнений – помочь понять концепции, изложенные в этой главе. Запишите ответы и посмотрите, какие вопросы вызвали затруднения: возможно, вам стоит перечитать главу. Такие упражнения будут в конце каждой главы. Незнакомый термин можно посмотреть в глоссарии в конце книги, что также позволяет облегчить понимание материала.

Если у вас есть коллега или профессиональный наставник, с которым вы можете обсудить ответы, это будет лучшим способом выяснить, правы вы или нет и почему. Некоторые из вопросов не являются логическими выражениями (не требуют ответов «истина/ложь»), а просто заставляют вас задуматься над проблемой.

1. Боб установил в настройках Wi-Fi на кардиостимуляторе запрет на передачу имени своего Wi-Fi. Как называется эта стратегия обеспечения безопасности?
2. Назовите пример значения, которое может быть жестко закодировано, и объясните почему. (Почему программист сделал бы это?)
3. Является ли капча удобной мерой безопасности? Почему?
4. Приведите один пример хорошей реализации удобства и безопасности.
5. Необходимо ли подтверждать данные, полученные при использовании параметров URL? Почему?
6. Если на работе сотрудник узнает коммерческую тайну, а затем продаст ее конкуренту, какую часть (или какие части) триады CIA он нарушит?
7. Вы купили смарт-холодильник и подсоединили его к домашней сети. К нему подключился злоумышленник и в настройках повысил температуру, в результате чего ваше молоко испортилось. Какую часть (или какие части) триады CIA он нарушил?
8. Если злоумышленник взломает ваш умный термостат и отключит отопление, какую часть (или какие части) триады CIA он нарушил?
9. Считается ли инсайдерской угрозой добавленная программистом «пасхалка» (дополнительный код, выполняющий

незарегистрированные функции, в качестве «сюрприза» для пользователей, о котором неизвестно руководству и команде безопасности)? Почему?

10. Какие меры предосторожности можно предпринять при подключении к общественному Wi-Fi, чтобы обеспечить «глубокую защиту»?

11. Если вы живете в квартире с несколькими соседями и у каждого из вас есть ключ от двери, считается ли один из таких ключей «фактором аутентификации»?

## Глава 2

# Требования безопасности

Независимо от используемой методологии разработки (Waterfall, Agile, DevOps), языка, платформы или аудитории требования к любому приложению, к любому новому проекту должны быть определены. Без плана нельзя создать что-то стоящее.

Если вы изучали информатику или компьютерную инженерию, то схема, изображенная на рис. 2.1, скорее всего, уже отпечаталась в вашем сознании. Это известный жизненный цикл разработки системы, который состоит из пяти фаз: требования, проектирование, кодирование, тестирование и релиз. Мы будем периодически возвращаться к данной схеме, чтобы объяснить, когда каждый упоминаемый нами вид деятельности может или должен происходить. Эта глава будет посвящена этапу требований.



Рис. 2.1. Жизненный цикл разработки системы

**СОВЕТ.** Жизненный цикл разработки системы называется также жизненным циклом *программного обеспечения* (ПО). Можно заметить, что во втором определении основное внимание уделено программному обеспечению, а не системе. Два этих определения взаимозаменяемы.

На первом собрании по проекту (часто называемом «установочной встречей») должен присутствовать человек из службы безопасности, который будет принимать участие в проекте с самого начала разработки. Даже не занимаясь проектом полный рабочий день, он должен быть полноценным членом команды и постоянно оказывать активную помощь в своевременном решении всех вопросов и проблем, касающихся обеспечения безопасности. Назначение сотрудника службы безопасности в проектную команду иногда называют *моделью партнерства*, а самого человека – «встроенным в матрицу команды».

Независимо от того, как его называют, данный сотрудник должен представлять интересы службы безопасности (то есть триады CIA) на протяжении всей разработки проекта.

**ПРИМЕЧАНИЕ.** Точное происхождение термина «Модель партнерства» неизвестно. Впервые я узнала о нем от команды Netflix, занимающейся вопросами безопасности. Выражение «встроенный в матрицу команды» я впервые услышала в секретariate Казначейства Канады, и оно также имеет неизвестное происхождение.

В этой главе предполагается, что у вас есть базовое понимание работы IT-проектов и процессов разработки программного обеспечения.

**СОВЕТ.** Определить разумные сроки ожидания ответа от службы безопасности можно с помощью соглашения об уровне поддержки (SLA, Support Level Agreement) между службой безопасности и другими командами. Часто во время работы над проектом взаимодействие с представителями службы безопасности становится практически невозможным. При наличии SLA этой проблемы можно избежать. Для достижения наилучших результатов следует установить скромные начальные цели и постепенно увеличивать их масштаб.

## Требования

Требования к проекту ПО всегда должны включать вопросы, касающиеся обеспечения безопасности. Ниже перечислены типы вопросов, которые должны задавать специалисты по безопасности при оказании помощи в сборе и анализе требований.

- Содержит ли система конфиденциальные, чувствительные или персональные данные (PII, Personally Identifiable Information – идентифицируемая личная информация) или контактирует с ними?
- Где и как будут храниться данные? Будет ли приложение доступно для общественности (находиться в интернете) или только для внутреннего пользования (находиться в интранете)?

- Выполняет ли приложение конфиденциальные или важные задачи (например, перевод денег, отпирание дверей или доставку лекарств)?
  - Выполняет ли приложение какие-либо программные действия, сопряженные с риском (например, позволяет ли пользователям загружать файлы)?
    - Какой уровень доступности необходим?
    - Требуется ли 99,999 % времени непрерывной работы? (Примечание: практически ни одна система не требует такого уровня доступности.)

В идеале представитель службы безопасности должен задать необходимые вопросы и на основании ответов к ним добавить соответствующие пункты в общий список требований к проекту. Например, «Позволит ли приложение загружать файлы пользователям? Да? Хорошо, тогда добавим вот такие требования безопасности в спецификацию проекта, чтобы разработка с самого начала велась с учетом обеспечения безопасности».

Как человек, создающий ПО, вы обязаны обеспечивать безопасность, сохранность и конфиденциальность своих пользователей. Требования безопасности помогут вам в этом.

В следующих разделах будут подробно рассмотрены определения и объяснения требований по обеспечению безопасности. В конце главы находится контрольный список требований, которые можно добавить в спецификацию проекта любого веб-приложения.

## Шифрование

*Криптография* – это раздел математики, который применяется к информации для того, чтобы сделать ее значение непонятным. Она используется для сокрытия секретов и обеспечения конфиденциальности связи. *Шифрование* представляет собой двусторонний процесс, в котором можно сделать информацию нечитаемой, а затем «расшифровать» ее обратно в исходную форму. *Хешированием* называется односторонний процесс, то есть когда исходное значение восстановить невозможно.

Шифрование довольно часто используется для защиты секретов или для передачи данных, поскольку система потом требует эти данные обратно. Ценность представляют сами данные. К хешированию чаще всего прибегают для подтверждения личности, аутентификации в системе, проверки целостности данных, а также для решения некоторых задач или контроля. Фактически никому не интересно, какой у вас пароль, – важно его наличие: программе нужно знать, пускать вас в систему или нет. В данном случае ценностью являются не данные, а подтверждение личности (которое выполняется с помощью знания исходного значения, пароля). Хеширование значения также подразумевает, что в случае кражи использовать значение невозможно. Украденные пароли (в измененной и хешированной форме) не принесут вору никакой пользы, поскольку при вводе в систему они не будут распознаны как пароли.

**ПРИМЕЧАНИЕ.** В настоящее время существует опасение, что из-за квантовых вычислений наши нынешние формы криптографии и шифрования устареют, однако в данной книге предполагается, что этого еще не произошло. Неизвестно, когда это опасение станет реальностью, а на момент написания этой книги ни у кого, включая меня, нет доказанного рабочего алгоритма или стратегии шифрования, устойчивых к квантовым вычислениям. Таким образом, в книге мы не будем затрагивать эту тему.

Для обеспечения конфиденциальности данные должны быть зашифрованы при передаче (на пути к пользователю, базе данных, API и т. д.) и в состоянии покоя (в базе данных). Следует отметить, что таким образом сохранность секретов гарантирована. Если кто-то получит несанкционированный доступ к данным или перехватит трафик с помощью анализатора трафика (снiffeра), он не сможет понять, что он нашел. Однако шифрование не защищает доступность или целостность данных. Кто-то все равно может удалить или изменить их в базе данных (изменение или удаление можно будет легко увидеть, но при неидеальной работе резервного копирования и восстановления это доставит ряд неудобств в ходе устранения последствий). Злоумышленник может перехватить трафик и изменить или заблокировать сообщения, что опять же приведет к проблемам. Тем не менее защита секретов (конфиденциальность в триаде CIA) представляет большую важность, поэтому, независимо от разрабатываемой системы, необходимо обеспечить шифрование (а не хеширование) данных при передаче и в состоянии покоя.

Кто-то может возразить, что данные следует шифровать даже во время их использования (в памяти), однако при работе с чрезвычайно чувствительными данными это, как правило, не является обязательным требованием к проекту. Для защиты особо важных данных рекомендуется чистить память перед выходом из программы, из системы или перед другим способом завершения работы.

## **Никогда нельзя доверять входному потоку системы**

Любые данные из входного потока системы с большой вероятностью могут быть взломаны или же привести к сбою или аварийному завершению работы приложения. Независимо от того, намеренно ли входным данным придали вредоносный характер или нет, из-за них приложение может перейти в неизвестное состояние (состояние, для которого не разрабатывался план действий), то есть попадает в большую опасность. С момента перехода приложения в неизвестное состояние злоумышленники могут управлять им как угодно, в том числе заставить его нарушить один или несколько аспектов CIA. Ваша программа должна уметь *быстро и эффективно*

обрабатывать любой тип ввода, даже тот, который имеет вредоносный характер.

Под входными данными (вводом) подразумевается буквально все и вся, что не является частью приложения или что могло быть подвергнуто манипуляциям вне его.

**ПРИМЕЧАНИЕ.** Одним из основных рисков для компьютерного программного обеспечения является ситуация, когда данные (значения в переменных, из API или из базы данных) обрабатывают так, как будто они являются частью кода приложения. Запуск стороннего кода обычно называется «инъекционной» уязвимостью, которая признана многими специалистами угрозой номер один для безопасности программного обеспечения<sup>[5]</sup> с самого начала существования нашей индустрии. Она является обоснованием для многих требований к проекту из этой главы.

Ниже приведены примеры входных данных приложения.

- Пользовательский ввод на экране (например, ввод поисковых фраз в поле).
- Информация из базы данных (даже специально разработанной для приложения).
- Информация из API (даже написанного самостоятельно).
- Информация, поступающая от другого приложения, с которым ваше приложение интегрируется или от которого принимает входной поток данных (сюда входят бессерверные приложения и скрипты).
- Значения в параметрах URL, значения cookie, файлы конфигурации.
- Данные или команды из облачных рабочих процессов.
- Изображения, взятые с других сайтов (с разрешением или без него).
- Значения из онлайн-хранилища.

Это не окончательный список примеров. Пожалуйста, имейте в виду: нанести вред может *все*, что не является частью вашей программы.

**ПРИМЕЧАНИЕ.** *Облачные рабочие процессы* обычно используются для вызова бессерверных приложений, но с их помощью можно запустить действие внутри приложения.

*Бессерверные приложения* – приложения или сценарии, которые запускаются в облаке без необходимости постоянного функционирования сервера. Другими словами, они не используют ресурсы инфраструктуры до момента своего запуска. При вызове бессерверного приложения создается контейнер, в котором приложение или сценарий выполняют свои функции, а затем он самоуничтожается, освобождая ресурсы инфраструктуры.

Элементы приложения, которыми можно манипулировать вне программы:

- параметры URL (их может изменить пользователь);
- информация в cookie, для которой не установлены флаги «безопасные» и «HTTPS only»;
- скрытые поля (они не защищены от злоумышленников);
- заголовки запросов HTTPS;
- введенные на экране значения, которыми можно манипулировать после пройденной проверки JavaScript с использованием веб-прокси (подробнее об этом позже);
- фронтенд-фреймворки, которые не являются частью приложения, а размещаются в интернете и вызываются в режиме реального времени;
- код сторонних разработчиков, который включается в состав приложения при его компиляции (библиотеки, вставки, платформы и т. д.);
- изображения, которые включаются в состав приложения и развертываются в другом месте в интернете;
- неуправляемые файлы конфигурации;
- API или любой другой сервис, к которому обращается приложение;
- неконтролируемые скрипты.

Иногда разработчики забывают, что даже пользующиеся доверием, уважением и поддержкой платформы и онлайн-сервисы все равно остаются возможными векторами атак.

Для эффективного применения концепции «никогда нельзя доверять входному потоку системы» необходимо проверять входные данные перед каждым их использованием. Входные данные считаются

ненадежными до тех пор, пока не пройдут проверку. Под проверкой подразумевается выполнение тестов, подтверждающих, что входные данные соответствуют ожиданиям. Если данные не проходят проверку, их следует заблокировать. В особых случаях необходимо провести санитизацию ввода (удалить все, что может нанести вред системе), о чем будет рассказано позже. В этом разделе мы обсудим проверку входных данных приложения.

Вот примеры проверки данных из входного потока.

- Вы ожидаете ввод даты рождения, поэтому проверяете, что полученное значение действительно имеет формат даты или преобразовывается в формат даты, а также не выходит за рамки предыдущих 100 лет (например, `age > current year - 100 && age < current year`). Если значение не соответствует формату даты (например, «aaaaaaaaa»), показывает, что человеку 5000 лет или что он еще не родился, то оно не принимается. При этом должны появиться соответствующие сообщения об ошибке: в случае неправильного формата необходимо сообщить о некорректном виде данных и указать ожидаемый формат, в случае выхода значения за пределы заданного диапазона в 100 лет – о неверном возрасте.

- Поле с лимитом в 80 символов предназначено для ввода имени человека. Необходимо убедиться, что количество символов ввода равно или не превышает 80, а символы соответствуют формату имени. Например, если значение содержит символы %, [, {, < или ], то вряд ли оно является настоящим именем, поэтому его нужно отклонить с соответствующим сообщением об ошибке. Однако, поскольку многие имена и фамилии содержат апострофы (‘), например О'Коннор, необходимо допускать такой ввод, но обрабатывать его осторожно (то есть закодировать его, о чем подробнее будет рассказано ниже). Также необходимо допускать диакритические знаки (é, å и т. д.), буквы из других алфавитов помимо латинского, дефисы и т. д.

- Поле предназначено для ввода адреса электронной почты. В интернете можно найти регулярные выражения для проверки адресов электронной почты, а также валидаторы в фреймворке. Я предлагаю использовать проверенные и испытанные функции валидации в фреймворке. Они имеют довольно сложную структуру, но работают отлично.

- Ваша программа выполняет поиск в базе данных и выводит на экран строку из найденной записи. Эти входные данные следует проверить на соответствие ожиданиям точно так же, как если бы они были получены от пользователя. Они не должны содержать межсайтового скрипtingа (cross-site scripting, XSS) или чего-то еще потенциально вредоносного. Всегда кодируйте выходные данные<sup>[6]</sup> перед отображением на экране.

**ПРИМЕЧАНИЕ.** XSS – это внедренный в приложение JavaScript-код, выполняемый в браузере на устройстве, через которое пользователь просматривает веб-приложение. Если перед выводом на экран все выходные данные кодируются, XSS-атаки будут отображаться в виде текста и не выполняться, выдавая на экране нечто похожее на «<script>...» . Это некрасиво, зато безвредно.

- Вы вызываете API: отправляете индекс, и обратно возвращается остальная часть адреса. Следует убедиться, что адрес имеет правильный формат и соответствует ожиданиям. Если он состоит из одних цифр или содержит символы, которые часто встречаются в коде ([, {, <, / и т. д.), то, скорее всего, он неверный. Адрес также не должен быть очень длинным: 500 символов достаточно для вызова любого API. Принимать входные данные следует только в том случае, если они прошли проверку. Наконец, данные, передаваемые между приложением и API, должны быть зашифрованы для защиты конфиденциальности пользователя. Шифрование может происходить в самом приложении посредством сервисной сетки или с помощью любого другого надежного механизма.

- Стороннее приложение вызывает ваше приложение и передает URL-адрес в параметрах. Это рискованное с точки зрения безопасности действие обычно называется *открытым перенаправлением*. По возможности приложение должно принимать информацию из внешних источников только по защищенным каналам (TLS), а также проверять получаемые данные. Лучше найти другой способ передачи данных, поскольку злоумышленник может изменить URL и отправить пользователей на опасный сайт. Если же это –

единственный доступный вариант, следует перейти к главе 4 («Безопасный код»), где объясняется, как с ним работать.

- При написании приложения на небезопасном для памяти языке (например, C/C++) необходимо выполнять так называемую *проверку границ*<sup>[7]</sup>, которая следит за тем, чтобы вводимые данные не переполняли ваши типы переменных. В C/C++ можно ввести больше максимальной суммы для целого числа, что приведет к его откату<sup>[8]</sup> в отрицательные значения и, очевидно, вызовет проблемы. Также возможно переполнение строк, что приводит к известной уязвимости под названием *переполнение буфера*<sup>[9]</sup>, когда злоумышленник может перезаписать часть памяти. В лучшем случае при переполнении буфера происходит сбой приложения, активирующий сигнал тревоги, и команда реагирования на инциденты блокирует действия злоумышленника. В худшем – злоумышленник использует информацию о сбое для корректировки и улучшения своей атаки, захватывает веб-сервер и проникает в сеть. Нельзя недооценивать риски при обсуждении возможностей переполнения буфера – данная категория уязвимостей требует серьезного отношения.

Важно, чтобы приложение сначала проверяло входные данные, а затем использовало их. Бессмысленно выполнять проверку данных *после* работы с ними. Проверка должна быть первым действием после поступления входных данных в приложение.

**ПРИМЕЧАНИЕ.** В случае вывода сообщения об ошибке на экран для отклонения пользовательского ввода, если вы решите показать пользовательский ввод, имейте в виду, что он может иметь вредоносный характер и, следовательно, вызвать сбой в работе программы. *Всегда* кодируйте вывод с помощью HTML-кодировки (эта функция доступна во всех современных системах программирования), если вы в контексте HTML.

**СОВЕТ.** Тип кодировки вывода зависит от контекста данных. Например, написанный в JavaScript-строках текст необходимо экранировать с применением Юникода. Однако, если вы встраиваете пользовательский ввод в обработчик событий, кодировка вывода будет состоять из двух уровней (JavaScript и

затем HTML). По возможности избегайте подобных ситуаций либо изучите памятку от OWASP по профилактике XSS<sup>[10]</sup>.

**ПРИМЕЧАНИЕ.** Если вы пишете или переписываете низкоуровневое приложение с нуля, всегда выбирайте язык Rust вместо C или C++. Rust – это новый язык программирования, который может выполнять низкоуровневые задачи так же хорошо, как C и C++, но, в отличие от них, Rust безопасен для памяти. Таким образом, при использовании этого языка проверка границ больше не требуется, а переполнение переменных для создания потенциальных уязвимостей безопасности становится невозможным. Безопасность памяти – это не шутка. По мнению создателя браузера Mozilla (Firefox), 73 % уязвимостей только в стилевом компоненте браузера никогда бы не возникли, если бы он был написан на Rust, а не на C/C++<sup>[11]</sup>. Даже одно это проектное решение может очень сильно сократить поверхность атаки, что сводит на нет все приемлемые деловые аргументы, которые могли бы оправдать написание новых приложений на C, когда доступен Rust. «Но мы уже умеем программировать на C» является недопустимой причиной не изучать и не использовать Rust.

## Кодирование и экранирование

Наиболее известная уязвимость безопасности в веб-приложениях – это *межсайтовый скрипting* (XSS). По оценкам, на данный момент он присутствует в более чем двух третях<sup>[12]</sup> веб-приложений в интернете. Существует несколько способов снижения данного риска: через заголовок политики безопасности контента (CSP), проверку ввода и кодирование вывода. Требование кодировки вывода существует только для предотвращения XSS-атаки, а из-за большой распространенности данной уязвимости, несомненно, стоит добавить в приложение все три перечисленных средства защиты.

**СОВЕТ.** Добавление в приложение всех трех способов борьбы с XSS – отличный пример применения принципа «защиты в глубину».

Экранирование символа или значения означает удаление специальных полномочий, которые он имел бы, если бы выполнялся как код, а не рассматривался как часть данных (как и должно быть). Обычно экранирование осуществляется путем добавления обратной косой черты (\) перед соответствующим специальным символом.

Кодированием называется преобразование исходного формата значения согласно используемому стандарту (кодирование URL, кодирование base64, кодирование HTML). Кодирование можно легко обратить, поэтому его не следует путать с шифрованием или использовать вместо него. Целью кодирования является не защита кодируемого значения, а изменение его оригинального формата на нужный для использования. Например, если вы собираетесь вывести какую-нибудь информацию на экран, вам необходимо *закодировать ее*, используя соответствующую функцию (все современные платформы имеют такую функцию). Если значение, выведенное на экран, содержит вредоносный код (например, атаку межсайтового скрипtingа), то при произведенной кодировке выходных данных будет выведено только текстовое значение, а не интерпретация (выполнение) кода в браузере в качестве JavaScript. Кодирование выходных данных обезвреживает код XSS-атаки.

Заповедь здесь, скорее всего, очевидна: кодируйте (и, если нужно, экранируйте) все выходные данные.

**ПРИМЕЧАНИЕ.** XSS – это особый вид инъекционных уязвимостей, поскольку при успешной атаке код (JavaScript) выполняется на стороне клиента (в браузере), тогда как большинство других уязвимостей данного типа выполняются на стороне сервера (на уровне интерпретатора, операционной системы и т. д.). Защита от XSS подразумевает сочетание проверки входных значений и кодирования либо экранирования выходных значений, в то время как все остальные инъекционные уязвимости зависят в основном от проверки ввода и настроек конфигурации на стороне сервера. Кроме того, XSS встречается чаще других серьезных типов уязвимостей в интернете. Таким образом, несмотря на то что XSS является одним из видов инъекционных уязвимостей, его всегда выделяют в отдельный класс уязвимостей.

## Сторонние компоненты

Как уже говорилось ранее, каждая строка кода, взятая из библиотеки, платформы, плагина или другого компонента стороннего производителя, подвергает риску ваше приложение. Если приложение обращается, вызывает или использует часть небезопасного компонента, то оно само становится уязвимым. В зависимости от ситуации уязвимость может повысить даже неиспользуемая незащищенная строка кода в компоненте. Проверка сторонних компонентов на уязвимости – залог быстрого и легкого успеха в понимании уровня безопасности приложения. Устранение найденных проблем может потребовать времени и сил, но в значительной степени обеспечит защиту приложения, а потому этот пункт всегда включается в список требований к проекту и обслуживанию.

---

## ИЗВЕСТНАЯ УЯЗВИМОСТЬ

Что значит «известная уязвимость»? По своему определению пользовательское программное обеспечение уникально. Каждая его часть – снежинка, и, следовательно, каждая новая уязвимость, которую можно в ней найти, неизвестна общественности. Программное обеспечение, которое не является пользовательским, часто называют просто программным обеспечением, как, например, операционная система или COTS. (COTS расшифровывается как Commercial/Customizable/Configurable Off The Shelf и буквально означает «Коммерческий/пользовательский/конфигурируемый готовый программный компонент».) Такое программное обеспечение может приобрести и установить любой желающий, и при этом оно обладает широкими возможностями конфигурирования. В качестве примера можно привести такие программы, как SharePoint, WordPress, Microsoft Office и Adobe Illustrator. В идеальном мире, когда тестировщик на проникновения, исследователь в области безопасности или другой «хакер» находит уязвимость в каком-либо программном обеспечении (пользовательском или нет), он сообщает о ней разработчику (это называется координированным раскрытием) или отмечает ее в программе Bug Bounty и получает вознаграждение. Если это не пользовательское программное обеспечение, то после сообщения об ошибке и выпуска исправления (обычно вместе с обновлением программного обеспечения) ошибка публикуется в интернете (например, в базе данных CVE Mitre<sup>[13]</sup>) для всеобщего обозрения и таким образом становится «известной» уязвимостью. Версия программного обеспечения с неисправленной уязвимостью, обычно обнаруживающейся сканирующими автоматическими инструментами, называется «имеющей известную уязвимость».

После обнаружения, но до выпуска исправления ошибки в программном обеспечении, операционной системе или COTS-компоненте называется «нулевым днем», или «0-day»: тем самым отмечается, что ошибка еще не исправлена. Часто команды безопасности говорят об исправлениях, используя понятие количества дней с момента обнаружения ошибки, например «этой

уязвимости 90+ дней, а мы до сих пор не исправили ее!», поэтому был выбран вариант «ноль».

---

**ПРИМЕЧАНИЕ.** Если в подкасте или в новостной статье говорится, что кто-то «сбросил день», это значит, что была обнародована информация об уязвимости в системе безопасности, для которой не существует известного исправления. Обычно это делается для оказания социального давления, чтобы заставить компанию выпустить исправление, или для демонстрации своих навыков исследователя безопасности. По моему мнению, всем, кто располагает информацией об уязвимости в системе безопасности, следует всегда сообщать о ней разработчику программного обеспечения, прежде чем публиковать ее на общественных ресурсах. Не для защиты компании, которая производит ПО, а для защиты пользователей. Они ни в чем не виноваты, и подвергать их риску противоречит цели и обязательствам индустрии информационной безопасности.

Существует множество средств, позволяющих проверить наличие уязвимостей в сторонних компонентах. В этой книге мы не будем рекомендовать конкретные инструменты или поставщиков, но предложим стратегические варианты применения таких средств. Первая стратегия заключается в использовании двух инструментов, если вы можете себе это позволить. Они проверяют разные базы данных разными способами и поэтому могут уловить разные проблемы. Вторая стратегия заключается в регулярном (ежедневном или, по крайней мере, еженедельном) сканировании репозитория кода, а также сканировании при каждом запуске кода в производство. Сканировать репозиторий необходимо потому, что даже в редко обновляемых приложениях все равно постоянно обнаруживаются новые уязвимости. Чем старше компоненты, внедренные в код приложения, тем больше времени было у исследователей безопасности и злоумышленников для изучения и поиска в них уязвимостей. Следует проводить сканирование при каждой публикации кода,

поскольку вы (или ваши коллеги), не зная, могли добавить в него новый компонент или обновить уже имеющийся компонент до версии, в которой есть известная уязвимость. Включение проверки на наличие уязвимостей в CI/CD-конвейер (или любой другой процесс, используемый для публикации кода) защитит вас от непреднамеренного внедрения вредоносного стороннего кода.

Обратите внимание, что эти действия подразумевают под собой проверку кода сторонних разработчиков на *известные* уязвимости. Скорее всего, еще больше уязвимостей остаются неизвестными. Проверка на известные уязвимости является необходимым минимумом в создании безопасного программного обеспечения. Если разрабатывающееся программное обеспечение требует очень высокого уровня надежности, нужно тестировать и просматривать каждую строчку кода, от которого оно зависит. Лучшая практика – включать в приложение только необходимые сторонние компоненты, а не каждую новую крутую штуку, увиденную в интернете. Несмотря на сложности при продаже такого приложения (давайте признаем, что новые технологии – это интересно и весело), если вместо простого «нет» вы сможете объяснить существующие риски и предложить альтернативу или решение, то скорее всего добьетесь положительных результатов.

---

## ПРЕДУПРЕЖДЕНИЕ О НАРУШЕНИИ КОНФИДЕНЦИАЛЬНОСТИ

Алиса хотела прикрепить свой зашифрованный личный онлайн-календарь к программе настольного календаря, чтобы видеть рабочий и личный календари в одном месте. Она прочитала страницу справки для программы настольного календаря, в которой говорилось, что нужно перевести параметр «Общий доступ к календарю» в режим «Публичный». Алиса была потрясена! Ее календарь зашифрован, потому что она хочет сохранить *приватность* личной информации. Она оставила жалобу, в которой объяснила, что данная страница должна предупреждать пользователей о возможных проблемах с

конфиденциальностью при такой настройке. В результате Алиса получила обновленную программу настольного календаря.

---

## **Заголовки безопасности: ремни безопасности для веб-приложений**

Заголовки безопасности – это параметры, которые указывают браузеру и серверу, как обрабатывать различное содержимое вашего веб-приложения. Они применяются только к веб-ресурсам, которые используют браузер (веб-приложениям и программному обеспечению как услуга (SaaS) – продуктам, доступ к которым осуществляется через браузер). Они не относятся к устанавливаемому на компьютер программному обеспечению, операционным или встроенным системам, таким как микропрограмма. Заголовки безопасности подобны ремням безопасности: они не привлекательны, не сложны в использовании и не требуют много времени, но, если вы выработаете привычку их использовать, они могут спасти вас в чрезвычайной ситуации (например, в автомобильной аварии или при атаке на веб-приложение соответственно). Заголовки безопасности обычно могут применяться либо в веб-сервере, либо в коде, при этом требуется добавить одну строку кода или установить флагок в настройках веб-сервера – это несложно. Вы сможете это сделать, я в вас верю.

Теперь давайте поговорим о том, для чего нужен каждый заголовок, почему вы должны или не должны использовать каждый из них и какие настройки следует выбрать. Не стесняйтесь копировать и вставлять эти настройки непосредственно в приложения, если считаете, что они будут полезны, однако перед внедрением протестируйте их.

**СОВЕТ.** Узнать больше о заголовках безопасности можно на сайте OWASP.org или SecurityHeaders.com.

### **Заголовки безопасности на практике**

Для проекта OWASP DevSlop Project мы с моей подругой Франциской Бюлер создали несколько видеороликов и записей в блоге о добавлении заголовков безопасности на сайт. Вот пример кода, который можно было бы использовать для ASP.Net:

```
<! – Start ASP.Net Security Headers – >
```

```
<httpProtocol>
<customHeaders>
<add name="X-XSS-Protection" value="1; mode=block"/>
<add name="Content-Security-Policy" value="default-src
'self'"/>
<add name="X-frame-options" value="SAMEORIGIN"/>
<add name="X-Content-Type-Options" value="nosniff"/>
<add name="Referrer-Policy" value="strict-origin-when-cross-
origin"/>
<remove name="X-Powered-By"/>
</customHeaders>
</httpProtocol>
<!-- End Security Headers -->
```

## X-XSS-Protection

Данный заголовок устарел. Он не только не поддерживается современными браузерами, но и, по рекомендациям ряда экспертов, вообще не должен использоваться из-за уязвимостей, которые может создать. В *отдельных* случаях он может помочь *некоторым* очень старым браузерам, однако принесет больше вреда, чем пользы, и поэтому его не следует использовать. [\[14\]](#)

## Content-Security-Policy (CSP)

Первое, что делает злоумышленник, когда понимает, что сайт уязвим для XSS, – вызывает расположенный в интернете собственный скрипт. Обычно этот скрипт значительно длиннее допустимого, что не позволяет уязвимому приложению обнаружить и заблокировать его. Большинство приложений разрешают ввод только 20–100 символов для большинства полей, то есть злоумышленник не сможет установить полноценный вирус или нанести запланированный ущерб, поэтому он обращается к другому месту в интернете, где его ждет уже готовый вредоносный код.

Заголовок Content-Security-Policy заставляет перечислять все источники содержимого (скрипты, изображения, фреймы, шрифты и

т. д.), которые использует сайт и которые находятся за пределами домена, что не позволит уязвимому веб-приложению вызвать и запустить «вторую» ступень атаки. Таким образом значительно снижается риск и потенциальный ущерб от этого типа атак. Тем не менее разработчики склонны считать, что отслеживание источников отнимает много времени, поэтому данный заголовок не пользуется популярностью. По моему мнению, Content-Security-Policy использовался бы гораздо чаще, если бы разработчики понимали риски и осознавали, какую защиту он обеспечивает. Если разработчика трудно убедить применять этот заголовок, подумайте о том, чтобы одолжить ему эту книгу. Надеюсь, он отблагодарит вас в будущем.

**ПРИМЕЧАНИЕ.** *Никогда* не включайте CSP в код без согласия и помощи разработчика. Вообще не следует включать средства безопасности без согласования с командами, на работу которых они могут повлиять, но особенно это касается CSP. Этот заголовок почти наверняка нанесет ущерб внешнему виду сайта и некоторым его функциям, но, что более важно, его применение нанесет ущерб вашим доверительным отношениям с командой разработчиков. Не торопитесь: тщательно протестируйте его перед первым развертыванием.

Самая простая настройка – просто заблокировать все лишнее, если разрабатываемый сайт статичен или имеет примитивный вид (не обращается к стороннему контенту). Настройки в таком случае будут следующими:

```
Content-Security-Policy: default-src 'self'; block-all-mixed-content;
```

Но давайте будем честны: немногие современные сайты настолько просты. Ничего страшного, мы справимся.

OWASP любезно предоставил список различных источников, которые можно определить в политике безопасности.

- **default-src:** Как понятно из названия, это настройка по умолчанию, своего рода «поимка всего». При попытке загрузить то, что не имеет четкого определения в остальной части политики, будет применен этот параметр. Его часто устанавливают на `self`, чтобы запретить загрузку контента, не имеющего четко полученного разрешения в политике.

Всегда устанавливайте значение `self` в случае неуверенности в возможном содержимом сайта.

**ПРИМЕЧАНИЕ.** Исключением из этого правила являются опции `Frame Ancestors` и `Form Action`. Они не возвращаются к `default-src`.

- `script-src`: Список доменов (местоположений скриптов) или точный URL-адрес скрипта, которые разрешено запускать как часть сайта. Любой другой скрипт из любого другого места в интернете, кроме тех, что находятся в вашем домене и были включены в перечень разрешенных ресурсов, не будет выполняться. Так осуществляется защита от XSS-атак.

**ВНИМАНИЕ.** Ключевое слово `unsafe-inline` может использоваться как часть конфигурации для отмены всех блокировок, произведенных в политике безопасности контента. Оно позволяет запустить любой скрипт из любого места. Нельзя оставлять `unsafe-inline` в приложениях на постоянной основе: данная опция должна быть лишь временной мерой для проведения тестирования по мере продвижения к зрелой и полной реализации CSP. Кроме того, обязательно проверяйте наличие этой настройки в производстве во время оценки безопасности.

Каждая из следующих опций следует вышеизложенной схеме: если указанный тип ресурса включен в список, то он может быть использован или загружен как часть веб-приложения, наряду с ресурсами из вашего домена. Все остальные не перечисленные здесь типы ресурсов при использовании заголовка CSP будут заблокированы:

- `object-src` = плагины (надежные источники элементов `<object>`, `<embed>` и `<applet>`);
- `style-src` = стили (каскадные таблицы стилей, или CSS);
- `img-src` = изображения;
- `media-src` = видео и аудио;
- `frame-src` = фреймы;
- `font-src` = шрифты;

– `plugin-types` = ограничивает типы запускаемых плагинов.

• **script-nonce**: сложная, но заслуживающая внимания директива.

Атрибут `Nonce` – это созданная для однократного использования строка символов, с помощью которой верифицируется вызываемый скрипт. `script-nonce` является дополнительным уровнем безопасности в заголовке безопасности CSP, а ее использование означает, что для запуска сценария необходим `nonce`.

**СОВЕТ.** Строки `nonce` – сложная тема, так как реализация функции `nonce` в CSP менялась с течением времени. В связи с этим я рекомендую изучить памятку от OWASP по этой теме, которая содержит свежие сведения и более подробное объяснение:

[cheatsheetseries.owasp.org/cheatsheets/Content\\_Security\\_Policy\\_Cheat\\_Sheet.html](https://cheatsheetseries.owasp.org/cheatsheets/Content_Security_Policy_Cheat_Sheet.html).

• **report-uri**: CSP может составить отчет, содержащий данные о заблокированных элементах и другую полезную информацию. URI указывает, куда отправить отчет. На данный момент существует всего четыре заголовка безопасности с функцией отчета, и, честно говоря, это действительно здорово. Наличие метрик и информации о типах атак, которым подвергается сайт, – это просто подарок.

**ВНИМАНИЕ.** URL-адрес отчетов находится в открытом доступе, то есть злоумышленник может просмотреть отчеты, а также провести атаку «отказ в обслуживании» (DoS или DDoS), чтобы скрыть свои злодеяния.

Content-Security-Policy, безусловно, является самым сложным из всех заголовков безопасности. Получить дополнительную информацию можно по адресам [csp-evaluator.withgoogle.com](https://csp-evaluator.withgoogle.com) (Google) и [mscotthelme.co.uk](https://mscotthelme.co.uk).

Например,

```
Content-Security-Policy: default-src 'self'; img-src  
https://*.wehackpurple.com; media-src  
https://*.wehackpurple.com;
```

позволяет браузеру загружать изображения, видео и аудио из `*.wehackpurple.com`.

Content-Security-Policy: default-src 'self'; style-src https://\*.jquery.com; script-src https://\*.google.com; позволяет браузеру загружать стили из jquery.com и скрипты из google.com.

**СОВЕТ.** Заголовками безопасности, которые предоставляют отчеты, являются CSP, Expect-CT, Public-Key-Pins и XSS-Protection. Они очень полезны!

**ПРИМЕЧАНИЕ.** DoS или DDoS означает «отказ в обслуживании» (от англ. denial of service), а дополнительная буква «D» означает «distributed», или «распределенная атака». Цель атаки DoS – перегрузить ресурсы жертвы так, чтобы никто не мог ими воспользоваться. Она может привести к прекращению работы веб-сайта, потере продаж в интернет-магазинах и другим формам блокировки доступа к онлайн-ресурсам. В самом начале DoS-атаки часто исходили из одного-единственного источника, таким образом, IP-адрес можно было заблокировать и сорвать атаку. Со временем злоумышленники поняли, что наличие большого количества (сотен или даже тысяч) различных IP-адресов, запускающих атаку, наносит гораздо больший ущерб и от них трудно защититься. На данный момент большинство DoS-атак являются распределенными (DDoS), часто с использованием взломанных IoT-устройств в качестве составной части атаки.

## X-Frame-Options

Данный заголовок помогает защититься от кликджекинга<sup>[15]</sup>, когда вредоносный сайт «обрамляет» надежный сайт и тем самым может украсть информацию либо «клики». Бывают случаи, при которых сайт должен быть «обрамлен» одним или несколькими конкретными сайтами, однако в результате может сложиться ситуация, когда пользователь будет считать, что находится на вашем сайте, а на самом деле начнет кликать на невидимые элементы другого сайта, являющегося, скорее всего, вредоносным (вот что подразумевается под кликджекингом). Это может привести к перехвату нажатия клавиш клавиатуры и краже учетных данных пользователя. Для активации кликджекинга пользователь должен собственоручно нажать на фишинговую ссылку или ссылку на вредоносном сайте, то есть данная атака не происходит сама по себе, как некоторые другие, но потенциальный ущерб от нее очень велик.

**ВНИМАНИЕ.** Заголовок X-Frame-Options устарел, вместо него в современных браузерах используется Content-Security-Policy (CSP). X-Frame-Options нужен для обратной совместимости старых браузеров и, надеюсь, будет постепенно выводиться из активного использования.

Разрешить сайту использовать фреймы из вашего домена можно, установив для X-Frame-Options значение `sameorigin`:

`X-Frame-Options: SAMEORIGIN`

Запретить любые фреймы можно через значение `deny` в X-Frame-Options:

`X-Frame-Options: DENY`

## X-Content-Type-Options

Часть красоты написания программ заключается в творческом подходе и поэтической свободе в применении языка программирования, поиске новых, фантастических способов использования языка и среды. Однако иногда это приводит к

двумысленности, из-за чего возникают уязвимости в безопасности, когда приложение не уверено в своих дальнейших инструкциях, то есть «переходит в неизвестное состояние». Нам *ни в коем случае* не нужно, чтобы приложение переходило в неизвестное состояние, в отличие от тестировщика на проникновение или исследователя безопасности, которые любят его больше всего на свете, поскольку там часто можно найти уязвимости.

Заголовок безопасности `X-Content-Type-Options` инструктирует браузер не «обнюхивать» (определять или угадывать) тип содержимого данных, используемого в веб-приложении, а полагаться исключительно на тип, который был указан приложением. Браузеры любят думать, что могут предвидеть тип обслуживаемого содержимого, пытаясь быть полезными, но, к сожалению, данная функция превратилась в известную уязвимость, которую можно использовать во вред сайту. У данного заголовка безопасности есть только одна возможная настройка:

```
X-Content-Type-Options: nosniff
```

## Referrer-Policy

Когда пользователь переходит с сайта на сайт, каждый предыдущий сайт отправляет следующему значение `referrer` – ссылку на страницу, с которой был совершен переход. Это очень полезная функция для тех, кто анализирует свой трафик: с ее помощью можно узнать, откуда приходят люди. Однако, находясь на сайте личного характера (например, с заявкой на ипотеку или с подробным описанием конкретного медицинского заболевания), пользователь, вероятно, не захочет, чтобы эти данные были отправлены на следующую страницу, которую он посетит. Для защиты конфиденциальности пользователей разработчик сайта может установить значение `referrer` так, чтобы передавался только домен, а не конкретная страница, на которой находился пользователь (то есть только `wehackpurple.com`, а не `wehackpurple.com/embarrassing-blog-post-title`), или вообще не передавалось никакое значение. Можно также изменить значение `referrer` в зависимости от того, «переходит» ли пользователь с HTTPS на HTTP.

Чтобы передавать информацию только о протоколе и домене, установите для `referrer` значение `origin`. Никакое другое условие не сможет изменить эту настройку.

`Referrer-Policy: origin`

Например, документ по адресу <https://wehackpurple.com/page.html> отправит ссылку <https://wehackpurple.com>.

Если пользователь покинет ваш домен, дальше будут переданы только протокол и домен, а если он находится в пределах домена – весь путь, что подходит для нечувствительных веб-сайтов:

`Referrer-Policy: strict-origin-when-cross-origin`

При отсутствии значения в поле `referrer` условие не имеет значения:

`Referrer-Policy: no-referrer`

Эта настройка может быть довольно сложной, но в целом она подойдет для большинства бизнес-ситуаций и обеспечит достаточную защиту конфиденциальности пользователей. Если возникнут сомнения, вы всегда можете ничего не отправлять и тем самым гарантировать, что конфиденциальность вашего пользователя будет соблюдена.

Mozilla является лидером в данной области и предлагает замечательное техническое руководство: [developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Referrer-Policy](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Referrer-Policy).

Бонусный ресурс: получить больше информации по этой теме можно на сайте [scotthelme.co.uk](http://scotthelme.co.uk). Скотт Хельме – исследователь безопасности, который делится большим количеством сведений и инструментов по заголовкам безопасности.

## Strict-Transport-Security (HSTS)

Этот заголовок безопасности превращает соединение в HTTPS (шифрует соединение), даже если пользователь попытался подключиться к сайту через HTTP. Таким образом, передаваемые данные будут зашифрованы принудительно. Пользователи, включая злоумышленников, не смогут перейти на HTTP (незашифрованное соединение), так как браузер принудительно переключит его на HTTPS перед загрузкой любых данных.

---

## ОПРЕДЕЛЕНИЯ

**Платформа как услуга** (англ. Platform as a Service, PaaS) – услуга облачных вычислений, в рамках которой в облаке размещается программное обеспечение (обычно веб-приложение). Обслуживание PaaS (исправления или обновления версии) производится поставщиком облачных услуг.

**Центр сертификации, или ЦС** (англ. Certification authority, CA), – доверенная компания или организация, которая проверяет личность того, кто приобретает сертификат.

**Фонд электронных рубежей** (англ. Electronic Frontier Foundation, EFF) – международная некоммерческая организация, которая работает над защитой конфиденциальности и других прав в интернете.

**Let's Encrypt** (букв. «Давайте зашифруем») – проект под управлением Фонда электронных рубежей (EFF), предлагающий бесплатные сертификаты шифрования. На момент написания книги Let's Encrypt является единственным потребительским центром сертификации.

**Wildcard-сертификат** – сертификат, который распространяется на все поддомены, а не только на основной домен. При этом для всех поддоменов будет нужен один такой сертификат. Поддоменами является всё под знаком «\*» в формуле: \*.какой-угодно. домен.

Например, сертификат \*.wehackpurple.com будет включать newsletter.wehackpurple.com, store.wehackpurple.com и www.wehackpurple.com.

---

Для правильной работы Strict-Transport-Security (HSTS) необходимо иметь выданный центром сертификации (ЦС) сертификат, установленный на веб-сервере, PaaS, контейнере или любом другом месте, где размещается приложение. Этот сертификат будет использоваться в процессе шифрования, и без него невозможно включить HSTS. При наличии поддоменов лучше получить сертификат под названием Wildcard. Желательно иметь сертификат с максимально

возможным сроком действия (один год, а не три месяца), чтобы не тратить время на частую «ротацию сертификатов».

Необходимо также знать, сколько времени в секундах длится действие вашего сертификата. Нет, я не шучу, они решили измерять время в секундах. Подсказка: один год равняется 31 536 000 секундам:

```
Strict-Transport-Security: max-age=31536000;  
includeSubDomains
```

**СОВЕТ.** Вы можете отправить свой домен на [hstspreload.org](https://hstspreload.org) и добавить суффикс preload. Google будет предварительно загружать сайт, тем самым не давая никому подключиться к нему через незашифрованное соединение. Хотя основные браузеры объявили о своем намерении начать применять эту функциональную возможность, она не является частью официальной спецификации HSTS<sup>[16]</sup>.

Скорректированный синтаксис предыдущего примера станет таким:

```
Strict-Transport-Security: max-age=31536000; includeSubDomains;  
preload.
```

## Feature-Policy

На момент написания книги Feature-Policy является самым новым заголовком безопасности, поддерживаемым современными браузерами. Он разрешает или запрещает веб-приложению использовать HTML 5 и многие другие новые возможности в более современных браузерах.

Настроить данный заголовок можно с помощью следующих атрибутов:

- **none**: ничего не разрешать;
- **self**: разрешить функцию, но пользоваться ею может только собственный домен;
- **src** (только iframes): документ, загружаемый в iframe, должен иметь тот же источник, что и URL-адрес в атрибуте `src` для iframe<sup>[17]</sup>;
- **\***: любой домен может пользоваться функцией;
- **<origin(s)>**: функция разрешена для определенных URL<sup>[18]</sup>.

Вот пример, позволяющий запускать на сайте только свой динамик и полный экран:

```
Feature-Policy: camera 'none'; microphone 'none'; speaker 'self';  
vibrate 'none'; geolocation 'none'; accelerometer 'none';  
ambient-light-sensor 'none'; autoplay 'none'; encrypted-media 'none';  
gyroscope 'none'; magnetometer 'none'; midi 'none'; payment 'none';  
picture-in-picture 'none'; usb 'none'; vr 'none'; fullscreen *
```

Эти настройки использовались для сайта проекта OWASP DevSlop. Мы запретили почти все функции. Разрешили использовать динамик только при вызове с нашего сайта. Мы также разрешили любому домену переключать браузер в полноэкранный режим. В сомнительных моментах лучше быть более строгим. Пользователи отблагодарят за это.

## X-Permitted-Cross-Domain-Policies

Заголовок безопасности X-Permitted-Cross-Domain-Policies относится только к продуктам Adobe (Reader и Flash), являющимся частью приложения. Adobe Flash ужасно небезопасен и больше не поддерживается компанией Adobe, поэтому его не следует использовать в современных веб-сайтах или приложениях.

Цель данного заголовка – разрешить или запретить доступ файлам домена к продуктам Adobe с других сайтов. Если вы намерены разрешить Adobe Reader, размещенному вне вашего домена, доступ к документам на вашем сайте, нужно указать сторонние домены в данном заголовке. В противном случае, установив значение none, вы запретите любым сторонним доменам использовать продукты Adobe для доступа к вашим документам либо ресурсам:

X-Permitted-Cross-Domain-Policies: none

## Expect-CT

CT (от англ. Certification Transparency) обозначает прозрачность сертификатов, то есть это фреймворк с открытым исходным кодом, который обеспечивает надзор за центрами сертификации (ЦС). Время от времени центры сертификации случайно выдают сертификаты далеко не идеальным, а иногда даже откровенно вредоносным сайтам. Вся система центров сертификации была разработана для того, чтобы обеспечить доверие и убедиться в том, что имеющий сертификат сайт безопасен для браузеров и пользователей. Недопустимо, чтобы ЦС предоставлял сертификаты вредоносным сайтам, будь то по ошибке, небрежности или специально. Система Certificate Transparency Framework регистрирует данные о различных операциях, отслеживая случаи некорректной выдачи сертификатов центром сертификации.

Если при выдаче сертификатов ЦС допустил несколько «ошибок», браузер или организация могут перестать «доверять» выданным им сертификатам.

Вы можете задаться вопросом, какое отношение это имеет к вам как разработчику программного обеспечения, специалисту по поддержке приложений или обеспечению безопасности. Для того чтобы помочь сохранить целостность всей системы центров сертификации, мы должны регистрировать наши сертификаты в онлайн-реестре CT. Если сертификат сайта отсутствует в реестре, современные веб-браузеры

будут выдавать пользователям предупреждения о том, что ваш сайт не заслуживает доверия. Компаниям совершенно не нужно, чтобы браузеры сообщали пользователям, что их сайт небезопасен.

При включенном заголовке безопасности Expect-CT:

1) так или иначе браузер пользователя будет проверять журналы СТ на предмет наличия в нем вашего сертификата;

2) при установленном значении `enforce` браузер пользователя будет обеспечивать прозрачность сертификата: если сертификат отсутствует в реестре или «не одобрен СТ», соединение между сайтом и пользователем будет прервано. Если режим `enforce` не установлен, отчет отправляется на соответствующий URL-адрес.

Рекомендуется сначала установить режим `only reporting` («только отчетность»), а после подтверждения того, что сертификаты соответствуют требованиям и были правильно зарегистрированы, перейти в режим `enforce`.

Поле `max-age` (измеряется в секундах) – это время применения настройки (то есть она будет кэшироваться в браузере в течение определенного времени).

**ВНИМАНИЕ.** Как и в случае с отчетами CSP, URL-адреса отчетов Expect-CT носят открытый характер. Информация из них может быть доступна любому человеку, в том числе тем, у кого не самые честные намерения.

Ниже приведены два примера реализации заголовка Expect-CT:

#### Только отчетность

```
Expect-CT: max-age=86400, report-  
uri="https://wehackpurple.com/report"
```

#### Отчетность и блокировка

```
Expect-CT: max-age=86400, enforce, report-  
uri="https://wehackpurple.com/report"
```

Ниже приведен еще один пример из проекта OWASP DevSlop, на этот раз для приложений .Net CORE. Как указано в первой строке, в нем требуется добавление дополнительного пакета Nuget («Nwebsec.AspNetCore.Middleware») [\[19\]](#):

```
<PackageReference Include="Nwebsec.AspNetCore.Middleware"
```

```
Version="2.0.0"/>

//Security headers.Net CORE, not the same as ASP.net
app.UseHsts(hsts => hsts.MaxAge(365).IncludeSubdomains());
app.UseXContentTypeOptions();
app.UseReferrerPolicy(opts => opts.NoReferrer());
app.UseXXssProtection(options =>
options.EnabledWithBlockMode());
app.UseXfo(options => options.Deny());
app.UseCsp(opts => opts
BlockAllMixedContent()
StyleSources(s => s.Self())
StyleSources(s => s.UnsafeInline())
FontSources(s => s.Self())
FormActions(s => s.Self())
FrameAncestors(s => s.Self())
ImageSources(s => s.Self())
ScriptSources(s => s.Self())
);
//End Security Headers
```

## Public Key Pinning Extension for HTTP (HPKP)

Привязка открытого ключа (англ. *Public Key Pinning*) – это система, созданная для защиты от поддельных сертификатов. Ее идея заключалась в том, что доверять определенному URL-адресу или сайту можно было только при наличии сертификата от одного или двух центров сертификации. Сначала она реализовывалась с помощью статических привязок (встроенных в браузер непосредственно и вручную, в частности в Chrome и Firefox). Со временем владельцы других сайтов тоже захотели «привязать» сертификаты, в результате чего появились динамические привязки. Сертификат привязывался на определенный период времени (обычно на год) к определенному криптографическому идентификатору (ключам). Потеря ключей при этом означала потерю контроля над сайтом на срок до года, поскольку без них он не работал. URL сайта по сути оказывался «замурованным» (непригодным для использования), и в результате эта функция безопасности наносила катастрофический ущерб бизнесу. Такая ситуация называется «самоубийством с помощью HPKP».

Хотя использование данного заголовка безопасности потенциально дает большие преимущества, высокие риски делают его нежелательным средством защиты. Он применяется в тех случаях, когда требуется чрезвычайно высокий уровень безопасности и когда с ним работает команда, очень хорошо разбирающаяся в этой теме и готовая принять бизнес-риски.

**ВНИМАНИЕ.** Заголовок безопасности Public Key Pinning Extension for HTTP (HPKP) считается устаревшим и более не поддерживается.

## Обеспечение безопасности файлов cookies

Протокол HTTP никогда не был предназначен для обработки пользовательских сессий (то есть отслеживания входов в систему или наличия товаров в корзине. Отслеживание того, как пользователь переходит с одной страницы на другую внутри сайта, называется «состоянием»). Cookie используются для передачи информации о

сессии пользователя от браузера к веб-серверу и могут быть сохранены для усиления персонализации (например, сохранения языковых предпочтений) в будущем. Многие сайты используют cookie для хранения информации в целях осуществления маркетинговых действий и отслеживания поведения пользователей, а также для продажи информации другим компаниям. Мы не будем рассматривать этические аспекты конфиденциальности данных, хранящихся в cookie.

Для обеспечения безопасности данных в файлах cookie необходимо применить нижеизложенные параметры.

Обратите внимание, что иногда разработчики решают использовать вместо cookie локальное хранилище. Для его защиты применяются совершенно иные меры безопасности, и в данном разделе они рассматриваться не будут. Кроме того, передача сессии должна происходить не в постоянном, а только в сессионном файле cookie и никогда не храниться в локальном хранилище. *Всегда используйте сессионные файлы cookie.*

Также не забывайте каждый раз проверять ввод в файлах cookie. Если получаемые данные не имеют смысла, отклоните их и попробуйте снова. Данные в файле cookie очень ценные, поэтому необходимо позаботиться о постоянной защите сессии (подробнее об этом в следующих главах).

## Флаг Secure

Флаг `Secure` гарантирует, что файл cookie будет отправлен только по зашифрованным (HTTPS) каналам. Если злоумышленник попытается перевести сессию на HTTP, веб-приложение откажется отправлять cookie. Данная настройка должна быть включена на постоянной основе:

```
Set-Cookie: Secure; (плюс все остальные настройки)
```

## Флаг `HTTPOnly`

Так как данный флаг не имеет ничего общего с принудительным незашифрованным соединением (HTTP), его название вносит путаницу среди программистов. Когда он установлен, то это значит,

что к файлу cookie нельзя получить доступ через JavaScript. Его можно изменить только на стороне сервера. Причина использования этого параметра заключается в защите от XSS-атак, которые пытаются получить доступ к значениям в файле cookie. Необходимо всегда устанавливать данный флаг во всех cookie в качестве еще одного уровня защиты от XSS-атак, угрожающих конфиденциальности данных:

```
Set-Cookie: HttpOnly; (плюс все остальные настройки)
```

## Для постоянных cookies

При сборе конфиденциальных данных пользователя либо при управлении сессией файлы cookies не должны быть постоянными. Они должны самоуничтожаться в конце сессии, чтобы защитить данные. Файл cookie, который самоуничтожается в конце сессии, называется «сессионным cookie», в противном случае – «постоянным cookie» или «отслеживающим cookie». Решать, какой тип cookie вам подходит, необходимо с помощью команды по обеспечению конфиденциальности и бизнес-аналитиков.

Для постоянных файлов cookie можно с помощью атрибута `expires` установить дату окончания действия, а через параметр `max-age` – определить максимальный срок жизни cookie.

### Окончание действия: 1 января 2021

```
Set-Cookie: Expires=Mon, 1st Jan 2021 00:00:00 GMT; (плюс все остальные настройки)
```

### Максимальный срок жизни – 1 час

```
Set-Cookie: Max-Age=3600; (плюс все остальные настройки)
```

## Domain

Чтобы к файлам cookies могли получить доступ сторонние домены, нужно указать их как доверенные домены с помощью атрибута `domain`, иначе браузеры будут считать, что cookies предназначены «только для хоста», то есть доступны только вашему домену, а попытки других доменов получить доступ будут блокироваться. Этот тип встроенной

защиты считается «безопасным по умолчанию». Вот бы все настройки программного обеспечения работали таким образом!

Set-Cookie: Domain=app.NOTwehackpurple.com; (плюс все остальные настройки)

**ВНИМАНИЕ.** Если поддомен не задан (то есть вы указали NOTwehackpurple.com вместо app.NOTwehackpurple.com), каждое приложение и страница, размещенные в этом домене, смогут получить доступ к вашему файлу cookie.

## Path

Фактически большинство URL-адресов в интернете представляют собой множество отдельных приложений, имеющих различные пути и поддомены. Для пользователя они выглядят как одна огромная веб-страница, но на самом деле это тысячи различных приложений. В такой ситуации лучше всего ограничить доступ к cookie только определенной областью местоположения приложения – его «путем». Ограничение области действия cookie осуществляется с помощью атрибута `path`:

```
Set-Cookie: path=/YourApplicationsPath; (плюс все остальные настройки)
```

## Same-site

Атрибут `same-site` был создан компанией Google для борьбы с межсайтовой подделкой запроса (CSRF). CSRF, ласково называемый «си сёрф», представляет собой атаку на авторизованных на надежном сайте пользователей, при которой злоумышленник пытается совершить действия от имени пользователя без его согласия или ведома. Обычно атака происходит через фишинговое электронное письмо.

Представим, что Алиса собирается принять участие в телевизионном шоу, в котором будет рассказывать о своей компании. Чтобы выглядеть великолепно, она решила купить новый наряд через интернет. Алиса заходит на сайт любимого магазина одежды и, просматривая новинки, замечает, что ей пришло письмо. Оно фишинговое: в нем содержится ссылка на сайт, на котором она в данный момент авторизована, с закодированной инструкцией, позволяющей злоумышленнику купить какой-нибудь продукт и отправить его себе, а не Алисе. Если она нажмет на данную ссылку, а сайт не обеспечен надлежащей защитой, вся атака может произойти без ее участия и вовсе остаться незамеченной. При этом атака сработает только в том случае, если на момент ее совершения Алиса будет авторизована на сайте, указанном в фишинговом письме.

Данная ситуация может показаться надуманной, но вспомните, как вы работаете со своей учетной записью (аккаунтом). Всегда ли нажимаете кнопку «Выход»? Бывает ли такое, что сайт остается открытым в течение нескольких дней? Никто не совершенен, и наша работа – защищать пользователей наших сайтов, даже если они совершают такую ошибку, как нажатие на фишинговую ссылку.

Таким образом, атрибут `cookie same-site` обеспечивает соблюдение правила, согласно которому `cookies` могут поступать только с одного сайта. `Cookies` не могут передаваться между различными сайтами (то есть исходить не с вашего сайта). Возможны следующие варианты применения данного атрибута: `None` (файлы `cookies` могут исходить откуда угодно), `Strict` (только с собственного домена) и `Lax` (если нужно, чтобы `cookies` отправлялись при переходе по ссылке или с другой страницы, в них должна содержаться только нейтральная информация). Если ни одно из данных значений не установлено, то по умолчанию в современных браузерах используется `Lax:/`, а в старых версиях – `none`.<sup>[20]</sup>

`Lax` означает, что пользователи могут оставаться в учетной записи и посещать другие сайты, а по возвращении на сайт файлы `cookies` будут продолжать работать. `Lax` обычно используется для навигации и других функций, выполняемых приложением до того, как пользователь войдет в систему. Он блокирует очевидные CSRF-атаки (POST-запросы). `Lax` нельзя назвать надежным решением, однако это хороший компромисс, если не получается использовать значение `strict`.

`Set-Cookie: same-site=Strict;` (плюс все остальные настройки)

`Set-Cookie: same-site=Lax;` // блокирует только POST-запросы, разрешает переход по ссылке

## Cookie с префиксом

Префиксы для файлов `cookies` были введены относительно недавно и принимаются еще не всеми браузерами. Тем не менее они являются одним из средств «защиты в глубину» (дополнительным уровнем безопасности) для случаев ошибочной обработки файлов `cookies`. Например, если поддомен был взломан, а в настройках `cookies Path` указан весь домен, то взломанный домен может попытаться получить

доступ к файлу cookie. Префиксы расположены в имени файла cookie, и поэтому сервер увидит его, несмотря на то, зашифрован он или нет. Обеспечить доступ к cookie только в пределах определенного поддомена можно с помощью префикса `host`. Более подробную информацию по этой теме можно найти на веб-сайтах Mozilla и Chrome.

## **Политика конфиденциальности**

В настоящее время большинство веб-сайтов имеют политику конфиденциальности, согласно которой веб-сайты должны указывать, какие данные они собирают и как их используют. Если на вашей работе имеется такая политика, убедитесь, что вы следите ей. Если нет, возможно, ее необходимо установить. Есть над чем подумать.

## **Классификация данных**

Все собираемые, используемые или создаваемые приложением данные должны быть классифицированы и размечены, чтобы все участники проекта знали, как с ними работать. Используемая система классификации может отличаться в зависимости от того, в какой стране вы живете и где работаете (частная компания или государственное учреждение). За рекомендациями следует обратиться к специалистам по обеспечению конфиденциальности или безопасности.

Давайте рассмотрим несколько примеров того, зачем и как нужно классифицировать данные.

Боб работает в федеральном правительстве, имеющем свою систему классификации данных (рис. 2.2). В его обязанности входит работа с данными, которые являются государственной тайной, секретной и совершенно секретной информацией. Существует строгая система идентификации типа этих файлов и данных, которой Боб всегда следует. «Государственная тайна», «Секретно» и «Совершенно секретно» в Канаде (где живет Боб) означает, что в случае раскрытия данных может быть нанесен ущерб целой стране, а не конкретному лицу. На предыдущей должности Боб работал с данными, которые

были гораздо менее чувствительными и классифицировались как общедоступные (их мог видеть любой человек), под защитой уровня «А» (могли причинить вред или смутили человека), под защитой уровня «В» (причиняли вред человеку или группе людей) и под защитой уровня «С» (могли привести к смерти или иному непоправимому ущербу для одного или нескольких людей).

Классификация данных, используемая на работе Боба



Рис. 2.2. Классификация, используемая Бобом при работе с данными

Классифицируя (определяя уровень чувствительности) и размечая собираемые данные, Боб помогает всем членам команды понять, как правильно обращаться с этими данными. Работая с неразмеченными данными, люди могут совершить ошибку, что может привести к непреднамеренной утечке информации или незащищенности очень ценных или конфиденциальных данных. Многие системы баз данных (например, Microsoft SQL Server) позволяют добавлять к полям данных или таблицам уровни классификации. Если используемая вами программа не предоставляет такой возможности, вы сами можете разметить данные путем добавления дополнительного поля в таблицу (или нескольких, если данные имеют разные уровни чувствительности). Даже «несекретным» или «общедоступным» данным следует присвоить соответствующую метку.

Всегда следуйте правилам и положениям о классификации данных. Спросите о действующих в вашей организации правилах и нормах,

если не знаете их. В случае их отсутствия можно следовать либо стандарту, разработанному правительством вашей страны, либо «Руководству по сопоставлению типов информации и информационных систем с категориями безопасности» от Национального института стандартов и технологий (NIST).

**ПРИМЕЧАНИЕ.** Хотя это не является обязательным требованием проекта, но должна существовать техническая документация, объясняющая, как работать со всеми уровнями конфиденциальных данных, а также готовый к использованию код или библиотеки для работы с данными. Лучше всего, если этот код или библиотека централизуют управление и обработку данных для получения одинаковых результатов.

## Пароли, хранилище и другие важные решения, касающиеся обеспечения безопасности

У Алисы есть несколько различных аккаунтов для работы, дома и хобби. Боб находится в такой же ситуации: у него буквально сотни паролей, используемых на работе и в личной жизни. Чтобы запомнить все свои пароли, Алиса использует менеджер паролей, в то время как Боб просто использует одну и ту же пару паролей (что специалисты по безопасности называют «повторным использованием паролей»).

**СОВЕТ.** Этот раздел содержит требования, которые подходят не всем проектам, но могут быть более полезны в качестве стандарта для вашего ИТ-отдела в целом, например использование менеджера паролей. Делайте то, что имеет смысл для вашей организации.

*Менеджер паролей* – это программное обеспечение, которое надежно хранит пароли пользователей и помогает им создавать уникальные, длинные и сложные комбинации. Люди используют менеджеры паролей, чтобы не запоминать несколько паролей – им нужен лишь один для входа в менеджер (а также пароли, которые нельзя хранить там, например пароль от телефона или компьютера). Менеджер паролей обеспечивает создание случайных (не предсказуемых с точки зрения компьютера или человека) и уникальных (не используемых повторно, как в случае Боба) паролей.

Большинство менеджеров паролей имеют плагины для браузеров, позволяющих аутентифицироваться на посещаемом сайте простым нажатием кнопки. Менеджер сам скопирует имя пользователя и пароль в браузер, что помогает избежать ошибок при вводе. Качественные менеджеры паролей также сообщают, если в нескольких местах используется один и тот же пароль, имя пользователя и пароль стали частью взломанных данных, был выбран плохой пароль (возможность придумывать свои собственные пароли при использовании менеджера остается), а также если сайт предлагает функцию MFA (многофакторной аутентификации).

Ко всему прочему, в менеджерах паролей обычно есть защищенная область для заметок (в которой можно сохранить номер социального страхования, информацию о кредитной карте и т. д.). Я советую всем заядлым пользователям компьютеров применять менеджеры паролей не только для обеспечения дополнительной безопасности, но и для экономии времени, затрачиваемого на поиск или сброс забытых паролей. Также стоит отметить, что менеджеры паролей предназначены для пользования людьми, а не компьютерами.

При утечке данных на каком-нибудь сайте, когда имена пользователей и пароли (то есть «учетные данные») попадают в интернет или в руки злоумышленников, пользователь теряет лишь ту учетную запись, которая использовалась на взломанном сайте, что является одним из преимуществ менеджеров паролей и повсеместного применения уникальных паролей. Довольно часто украденные учетные данные используются в *атаке методом подстановки учетных данных* (англ. credential stuffing attack): злоумышленник использует их как часть автоматизированной атаки на один или несколько других сайтов, чтобы увидеть, какие из них были использованы повторно, что позволяет ему получить доступ к аккаунтам на других сайтах. Некоторые взломщики доходят до того, что пишут сценарии «корректировки» паролей, например изменения «Пароль1» на «Пароль2», «Пароль3» и так далее, и тем самым пробуют вариации украденных паролей. Такая атака называется *атакой методом подстановки учетных данных с использованием радужных таблиц* (англ. rainbow credential stuffing attack).

Если на сайте, которым пользовались и Алиса, и Боб, произойдет утечка данных, то, Алисе нужно будет сбросить только один пароль и беспокоиться только об одном аккаунте, поскольку она везде использует уникальные пароли. У Боба же возникнет много работы и забот! Также Алиса и Боб могут защититься от этого типа атаки с помощью включения MFA для каждой учетной записи, представляющей для них ценность (банковские счета, рабочие аккаунты, электронная почта, учетные записи, к которым привязаны кредитные карты, аккаунты очень личного характера, электронные медкарты и т. д.).

Менеджер паролей + повсеместное использование уникальных паролей + MFA = глубокая защита личной информации

По работе разработчикам и техническим специалистам необходимо использовать огромное количество паролей, следовательно, у них всегда должен быть доступ к менеджерам паролей.

*Тайники* похожи на менеджеры паролей, только они предназначены для использования компьютерными системами, а не людьми. Тайник – это программное хранилище, которое шифрует секреты, а затем позволяет приложению получить к ним программный доступ (через код приложения или процесс формирования CI/CD-конвейера). В нем можно хранить сертификаты, учетные данные (имя пользователя и пароль), строки подключения, хеши и все, что считается «секретом», используемым приложением.

*Сервисными учетными записями* называются аккаунты, используемые компьютерами, а не людьми. С их помощью происходит идентификация компьютерных систем в сети. Если разрабатываемому приложению требуется доступ к базе данных, разработчику следует создать сервисную учетную запись, а не использовать личную. Этому есть много причин, самая очевидная из которых заключается в необходимости, чтобы все программное обеспечение могло продолжить работу даже после ухода его разработчика. Другими причинами выступают мониторинг (если каждое приложение имеет собственную учетную запись, команда мониторинга может видеть, что делают приложения, а не разработчик), расследование инцидентов (необходимо, чтобы были видны действия всех членов команды, иначе возникнет представление, как будто все ошибки принадлежат разработчику, так как везде было использовано его имя пользователя), соблюдение принципа наименьших привилегий (учетная запись разработчика, скорее всего, имеет множество разрешений повсюду, и если кто-то получит доступ к его аккаунту, то получит доступ ко всем приложениям) и т. д. Необходимо обеспечить постоянное соблюдение принципа наименьших привилегий при создании и включении учетных записей служб.

**СОВЕТ.** Срок действия паролей от сервисных учетных записей никогда не должен истекать, если только не произошло

(предполагаемого) взлома. Пароли также должны быть чрезвычайно сложными и как можно более длинными. Сервисные учетные записи никогда не должны иметь права (привилегии) на вход на рабочие станции или доступ к любым другим ресурсам, кроме тех, для которых они изначально были созданы.

А как насчет паролей пользователей приложения? Где их следует хранить? Они должны храниться в базе данных (или другом централизованном месте управления, например у поставщика удостоверений), в *засоленном* и *хешированном* формате. Если вы помните, хешированием называется односторонний криптографический процесс, который нельзя отменить. Засаливание – это добавление уникальной длинной строки к значению перед хешированием с целью увеличения энтропии и усложнения для потенциального злоумышленника взлома или угадывания пароля. Когда пользователь аутентифицируется в приложении и вводит необходимые значения, запускается функция хеширования и соления, а затем результат сравнивается с тем, что хранится в базе данных. Таким образом, украденные имена пользователей и пароли бесполезны для злоумышленника, поскольку если бы он ввел в приложение хешированное и засоленное значение, то оно бы изменилось и в результате уже не совпадало бы со значением в базе данных.

**ПРИМЕЧАНИЕ.** Под *взломом* здесь подразумевается использование автоматизированного инструмента для угадывания пароля с помощью списка слов до тех пор, пока пароль не будет определен. Данный процесс также называется «методом грубой силы» или «полным перебором» (англ. *brute force*) и является автоматизированным угадыванием чего-то с целью получения доступа.

Соль должна состоять как минимум из 28 символов (но желательно намного длиннее), создаваться безопасным генератором случайных чисел и быть уникальной для каждого пользователя приложения. Следует хранить отдельное значение соли для каждого пользователя в базе данных вместе с хешированным значением «соль + пароль». Надо

отметить, что соль не является секретом, в отличие от перца (о котором мы расскажем чуть позже).

Более новыми методами обеспечения того, чтобы пароли было чрезвычайно трудно взломать, являются «рабочий фактор» и «перец».

*Рабочий фактор* означает повторение алгоритма хеширования  $X$  раз, где  $X$  – это количество итераций<sup>[21]</sup>. Значение рабочего фактора должно быть равно 2 или более (в зависимости от изменений в оборудовании в нашей отрасли, уровня чувствительности данных и учетных записей или чего-либо еще в изменяющемся характере угроз, которым подвергается приложение).

«Засыпка перцем», или по-криптографически «перец»<sup>[22]</sup>, имеет общие черты с солью: его тоже нужно добавлять к паролю до хеширования и создавать с помощью безопасного генератора случайных чисел. Однако, в отличие от соли, перец является секретом и не может храниться в базе данных, следовательно, его размещают в тайнике. Он должен быть довольно длинным: состоять как минимум из 32 символов (но желательно из 128). Ко всему прочему, перец уникален для каждого приложения, но одинаков для всех его пользователей.

Можно как солить, так и перчить свои пароли. Однако, как правило, для большинства систем требуется только соль. Решите этот вопрос со своей службой безопасности.

**СОВЕТ.** В идеале управление пользователями системы может осуществляться в рамках управления идентификационными записями. Таким образом можно переложить эту сложную работу по обеспечению безопасности на систему, созданную специально для таких целей.

**ВНИМАНИЕ.** В случае необходимости ротацию перца следует проводить осторожно, так как она может привести к сбросу всех паролей всех пользователей приложения. Это серьезный риск, который негативно отразится на ваших пользователях и клиентах. Если вы решите произвести ротацию перца, разработайте ее с учетом данного риска.

Пароли пользователей приложения должны быть длинными, но не обязательно сложными: хороший пароль имеет заглавные и строчные буквы, цифры и специальные символы, однако требования по количеству символов каждой категории раздражают пользователей. Достаточно будет включить в требования «сложности» пароля наличие одного символа из каждой категории. Разрешите пользователям вводить пароли длиной до 64 символов (чем длиннее, тем лучше), а также поощряйте использование парольных фраз. Не нужно заставлять пользователей менять пароли через определенное время, если только не подозревается взлом. Чтобы убедиться, что новые пароли пользователей не были ранее взломаны, можно сравнить хеши sha1 в диапазоне с помощью сервисов вроде HaveIBeenPwned.

**ВНИМАНИЕ.** При проверке того, не был ли пароль взломан ранее, убедитесь, что анонимность пользователей защищена. Существует несколько моделей обеспечения анонимности, например  $k$ -анонимность. Подробную информацию можно найти на [haveibeenpwned.com/API/v3#SearchingPwnedPasswordsByRange](https://haveibeenpwned.com/API/v3#SearchingPwnedPasswordsByRange).

Никогда не стоит проверять правильность имени пользователя или пароля, когда предупреждаете пользователя о том, что он не смог войти в систему. Предоставление такой информации позволяет злоумышленникам собирать имена пользователей (проверяя, использует ли пользователь вашу систему).

**СОВЕТ.** Как метод проверки контрольные вопросы устарели, поскольку большинство пользователей выбирают те из них, ответы на которые находятся в открытом доступе (например, в социальных сетях). По возможности откажитесь от их применения.

*Функции забытого пароля* проверяют личность пользователя перед отправкой ссылки на сброс пароля по электронной почте или SMS с использованием либо другой формы аутентификации, либо контрольного вопроса. При неудачной аутентификации ни в коем случае нельзя раскрывать, какая часть не прошла проверку. Также

нельзя допускать, чтобы пользователь смог сбросить пароль непосредственно в системе. Сброс пароля всегда должен происходить посредством «внеполосной» формы связи (через электронную почту или SMS) в качестве второй формы аутентификации. Ссылка на сброс должна быть одноразовой, а ее действие – не превышать один час, если она не была использована. Каждый сброс пароля (или неуспешная попытка) должен быть зарегистрирован. Для его подтверждения на адрес или номер, привязанный к учетной записи, следует высыпать электронное письмо или SMS. Если за последние 24 часа пользователь безуспешно пытался сбросить пароль 10 раз, нужно заблокировать его учетную запись на IP-адресе, с которого поступают запросы, при этом данная ситуация регистрируется и подается сигнал тревоги. Если IP-адрес пытается сбросить пароль для несуществующего аккаунта, следует обрабатывать его так, как будто такая учетная запись существует, чтобы предотвратить перечисление (сбор) настоящих имен пользователей злоумышленниками. На рис. 2.3 изображена блок-схема безопасной процедуры сброса забытого пароля.

### **Правила работы с паролями**

- Хешируйте и солите все пароли пользователей. Соль должна состоять не менее чем из 28 символов.
- Все секреты приложения должны храниться в тайнике.
- Все учетные записи, используемые внутри приложения, должны быть сервисными (не личными аккаунтами людей). Они должны быть уникальными для каждого приложения и соответствовать концепции наименьших привилегий.
- Попросите всех сотрудников команды использовать менеджеры паролей, управляемые организацией, и установите политику, согласно которой нельзя повторно использовать пароли, включая вариации одного и того же пароля.
- Включите MFA для всех важных аккаунтов на работе и дома. При любой возможности используйте приложение аутентификации или само устройство, а не SMS в качестве второго фактора аутентификации.
- Не заставляйте менять пароли по расписанию – только если произошел взлом или наблюдается подозрительная активность.

- Всегда используйте *современный* алгоритм хеширования.
- При наличии сомнений касательно хранения паролей следуйте политике вашего правительства, а если таковой нет – политике паролей от NIST. Она была разработана группой экспертов и проверена на деле, поэтому в ней представлены самые лучшие рекомендации.

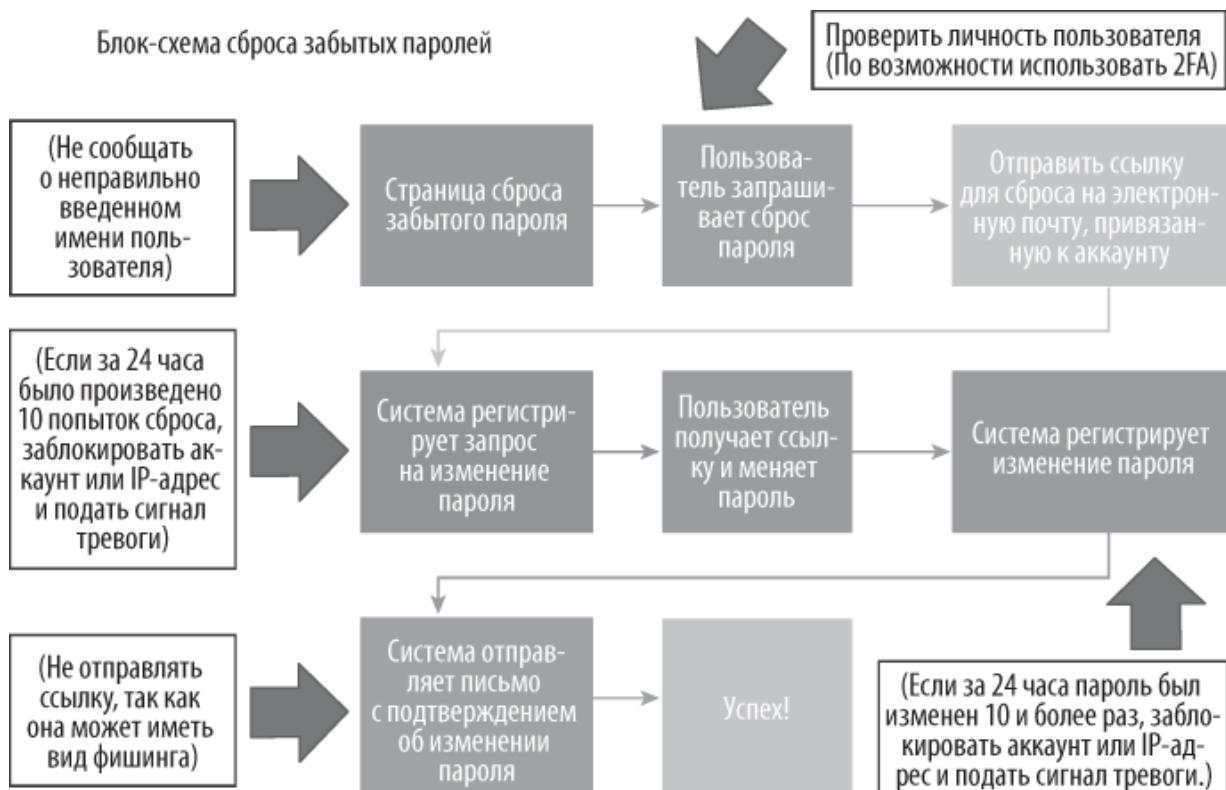


Рис. 2.3. Блок-схема сброса забытых паролей

**СОВЕТ.** Современные алгоритмы хеширования и дополнительные советы по хранению паролей можно найти на сайте [cheatsheetseries.owasp.org/cheatsheets/Password\\_Storage\\_Cheat\\_Sheet.html](https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html).

## HTTPS повсюду

Интернет – это триллионы сайтов и миллиарды ежедневных пользователей. Он является основным способом общения для большинства людей в мире. Однако, когда интернет только создавался, никто и не предполагал, что он станет таким, как сейчас. Безопасность, конфиденциальность, способы перевода средств и другие элементы не были заложены в протоколы, так как никто не думал, что они понадобятся. Изначально в интернете не было шифрования (использовался открытый текст), потому что разработчики и представить себе не могли нынешние инструменты для отслеживания и просмотра трафика в сетях и интернете («снiffeинг» сетевого трафика), которые можно использовать также для прерывания, изменения и последующего отправления трафика по назначению без ведома и участия получателя. Именно поэтому сейчас шифрование всего трафика в интернете – насущная необходимость. Тем не менее пока не все согласны с этой идеей, поэтому, прежде чем обсуждать «как», обратимся к «почему».

Когда пользователь заходит на ваш сайт через интернет, его местоположение неизвестно. Он может находиться в кафе с открытой точкой Wi-Fi, на конференции в отеле, дома в сети, которую он делит с соседней квартирой, или на хорошо защищенном военном объекте. К сожалению, необходимо разрабатывать приложение с учетом худшего сценария из этого списка, то есть обеспечить защиту пользователям, использующим небезопасную сеть.

Трафик может быть перехвачен в ситуациях, когда люди пользуются незащищенной сетью, например в кафе или когда соседи делятся Wi-Fi друг с другом. Неизвестно, кто еще находится в сети и может наблюдать за посещающими ваш сайт людьми.

Существует также множество ситуаций, когда можно с полной уверенностью сказать об отслеживании трафика, например при использовании интернета в отеле и на работе или при подключении к любой сети, где производится Deep Packet Inspection (глубокая проверка пакетов).

**СОВЕТ.** Deep Packet Inspection означает обработку данных, которая проверяет передаваемые по сети пакеты (данные), что приводит к таким действиям, как блокирование, перенаправление и протоколирование<sup>[23]</sup>.

Когда кто-то отслеживает сетевой трафик, он может не только видеть то, что видит пользователь (нарушая его конфиденциальность), но и изменять передаваемую ему информацию. В настоящее время в крупных отелях распространена практика замены всей рекламы на незашифрованных сайтах на свою собственную рекламу. Это один из возможных вариантов изменения информации, которую видят пользователи. Другим примером может быть внедрение вредоносных программ, скриптов или дезинформации на сайт, который они посещают. Возможный ущерб пользователям удивительно высок, даже несмотря на довольно низкую вероятность совершения серьезной атаки (если только речь не идет про очень большой либо популярный сайт). Если это все еще звучит недостаточно убедительно, то стоит напомнить: когда трафик идет по HTTP, а не по HTTPS, современные браузеры предупреждают пользователей о том, что посещаемый ими сайт небезопасен (либо с помощью графических элементов, либо с помощью текста). Вред для бизнеса очевиден.

Теперь, когда мы убедились, что всегда нужно использовать HTTPS, определим *Правило*:

Необходимо разрешать доступ к сайту только через HTTPS. Если кто-то пытается использовать HTTP-соединение, следует перенаправить соединение на HTTPS. Это можно сделать с помощью заголовков безопасности в коде или серверных настроек, о чем было рассказано ранее в разделе «Заголовки безопасности: ремни безопасности для веб-приложений».

## Настройки TLS

Следует убедиться, что для шифрования используется последняя версия TLS (в настоящее время 1.3, но если она не поддерживается провайдером, то на момент написания книги еще можно использовать 1.2). Поскольку версии часто обновляются, рекомендуем поискать в интернете информацию о текущих передовых практиках.

О других передовых практиках в этой области можно узнать в главе 3 «Проектирование защиты».

---

## **СОВЕТЫ ПО TLS**

**Рекомендации по применению TLS находятся на [bettercrypto.org/#webservers](https://bettercrypto.org/#webservers) и <https://ssl-config.mozilla.org/>.**

**Получить помощь в настройке совместимости можно на [wiki.mozilla.org/Security/Server\\_Side\\_TLS](https://wiki.mozilla.org/Security/Server_Side_TLS).**

**Проверить конфигурацию можно на `testssl.sh`.**

---

## **Комментарии**

Разработчики пишут комментарии (обычный невыполняемый текст по всему коду), чтобы помочь себе и другим понять, что происходит в программе. Однако иногда такие комментарии содержат то, чего там не должно быть: пароли баз данных или другие секреты, сведения о компании или пользователях, которыми не следует делиться, или иную конфиденциальную информацию. Комментарии, помещенные во внешний код (JavaScript, HTML, CSS), не всегда удаляются во время упаковки, и реверс-инженер легко может их найти. Поэтому никогда нельзя разглашать конфиденциальные сведения в комментариях. Никогда.

**СОВЕТ.** Существует множество инструментов, доступных для CI/CD-конвейера или для сканирования репозитория, которые ищут секретную информацию в коде. Всегда используйте один из них!

## **Резервное копирование и восстановление**

Следует регулярно производить резервное копирование всех данных, которые использует, создает или хранит приложение (в соответствии с политикой классификации данных организации),

причем для этой цели обычно выделяется второе, географически удаленное место. Система должна быть способна восстановить данные в случае внедрения вредоносного ПО, вируса-вымогателя или нарушения безопасности. Процедуры резервного копирования и восстановления должны быть протестированы и отработаны на практике.

## Элементы безопасности платформы

Платформы пишутся профессиональными командами экспертов и регулярно тестируются сотнями, тысячами разработчиков. Элементы безопасности платформы проверяют как этичные, так и неэтичные хакеры. Другими словами, за элементами безопасности в платформе стоит гораздо больше людей, опыта, времени и тестирования, чем за теми, которые были написаны непосредственно разработчиками. Таким образом, можно с почти полной уверенностью утверждать, что элементы безопасности платформы лучше защищены, чем код, написанный для выполнения действий по обеспечению безопасности. Из этого можно сделать вывод, что следует использовать элементы безопасности в платформе, а не писать специальный код собственноручно.

Опытные тестировщики на проникновение в веб-приложения и инженеры по безопасности приложений всегда могут рассказать парочку историй о разработчике программного обеспечения, который решил «создать собственную криптографию». Вместо того чтобы использовать функции криптографии платформы, он решил, что либо закодирует значение в base64 (возможно, несколько раз), либо напишет собственную функцию для перепутывания значений. К сожалению таких разработчиков, самые простые задачи в соревновании «Захват флага» (Capture the Flag, CTF) часто связаны с обнаружением подобных простых и неадекватных попыток защиты, то есть обычно такую «защиту» легко обнаружить и обойти.

В качестве других примеров ненормальной защиты можно привести написание собственной функции кодирования вывода, попытки управления сессиями вручную, сохранение данных сессий в локальном хранилище, а не в защищенном файле cookie и многое другое. Следует пользоваться элементами обеспечения безопасности, которые предлагает платформа. Если ваша платформа не предоставляет таких функций, рассмотрите возможность добавления в приложение дополнительной платформы или компонента, имеющего соответствующие средства для выполнения требуемых действий, и только в случае крайней необходимости пишите собственные.

## Технический долг = Долг безопасности

*Технический долг – это метафора программной инженерии, обозначающая накопленные в программном коде или архитектуре проблемы, связанные с пренебрежением к качеству при разработке программного обеспечения и вызывающие дополнительные затраты труда в будущем.*

*Википедия*[\[24\]](#)

Технический долг может включать в себя жесткое кодирование, отказ от обновления серверов или платформ в течение длительного периода времени, отсутствие исправлений, выбор коротких путей решения задач, чтобы «просто со всем разобраться», и т. д. Технический долг часто приводит к тому, что организация медленно отвечает на изменения, то есть не может своевременно реагировать на инциденты или другие угрозы безопасности. Обе эти ситуации приводят к ослаблению защиты и неспособности организации эффективно обезопасить себя.

Боб привык к большим объемам технического долга, исходящего от одного из правительственныеых отделов, в котором он раньше работал. В отделе было так много процессов и согласований, что иногда он даже не был уверен в том, как провести то или иное изменение. Требовалось получить разрешение от Совета по утверждению изменений и Группы технической архитектуры, но прежде всего – подпись начальника начальника его начальника, чтобы тот одобрил это изменение. Боб считал бессмысленным, что кто-то на четыре уровня выше него утверждает изменения. Как генеральный директор понимает, не будет ли какое-нибудь изменение противоречить другим уже имеющимися технологиям? Он же не пишет код. Боб часто расстраивался и чувствовал, что руководство просто не доверяет ему выполнение его работы, замедляя весь процесс без реальной пользы.

Когда Боб работал в этом отделе, у них произошел очень серьезный инцидент, связанный с безопасностью. Он подслушал, как разработчики рассказывали команде безопасности, что цикл выпуска новой функции равняется 16 месяцам, а экстренный релиз займет не менее четырех месяцев. Лицо сотрудника службы безопасности

сказало все: в случае возникновения чрезвычайной ситуации у отдела будет очень сложный день. Когда Бобу предложили работу в новом отделе, где применялись более современные методы разработки программного обеспечения, он ухватился за этот шанс. Изнуряющие правила в его техническом отделе очень мешали ему выполнять повседневные задачи, и он хотел работать там, где чувствовал бы доверие со стороны руководства.

Если организации не могут вносить изменения в кратчайшие сроки и минимальными усилиями, это говорит о существовании технического долга, мешающего их продуктивной работе. Если организация тратит большую часть своего времени на то, чтобы просто поддерживать свет, постоянно борясь с пожарами, у нее наверняка будут проблемы с обеспечением безопасности.

## Загрузка файлов

Если это вообще возможно, вместо написания собственного кода для выполнения загрузки файлов найдите хорошо зарекомендовавший себя компонент стороннего производителя, разработанный специально для этой цели. Таким образом вы снизите риски, связанные с обеспечением безопасности, поскольку он уже прошел всестороннее тестирование (выбирайте только такой компонент стороннего производителя). Предоставление разрешения на загрузку файлов обычным пользователям (а не авторизованным пользователям из вашей организации) – это самый рискованный функционал программного обеспечения, который только может быть у обычного приложения. Как правило, пользовательский ввод считается самой опасной частью приложения, однако разрешение пользователям загружать файл (который может оказаться вредоносным) выводит «опасный пользовательский ввод» на совершенно новый уровень потенциального риска.

Алиса помнит, как ей было стыдно, когда она случайно принесла на работу вирус. Дома она пользуется Mac, на работе – Windows. Однажды она принесла на работу USB-накопитель с любимой музыкой, которую скачала дома. После подключения флешки к рабочему компьютеру начался запуск EXE-файла, который Алиса не заметила... В мгновение ока команда безопасности была у ее стола.

Каждый может совершить ошибку. Алиса – чрезвычайно умный человек, но она не профессионал в области безопасности. Программы должны учитывать как злоумышленников, так и людей, которые по ошибке загружают в приложение неправильные типы файлов.

Принимая загруженный пользователем файл, необходимо предполагать худшее: проверьте его тип и размер, переименуйте файл, не позволяйте пользователю задавать путь к месту его расположения и храните его в безопасном месте вдали от остальных частей приложения и веб-сервера. Приняв файл, просканируйте его хотя бы одним инструментом. Если подразделение позволяет, разрешите загрузку только определенных, менее опасных типов файлов. Например, можно принимать JPG, TXT и PNG, но блокировать PDF или EXE.

Серия памяток OWASP, проект с открытым исходным кодом под эгидой Открытого проекта по безопасности веб-приложений (англ. Open Web Application Security Project), содержит обширный список мер предосторожности, которые необходимо предпринять при разрешении загрузки файлов. Список будет очень полезен тем, кто пишет эту функциональность с нуля: [cheatsheetseries.owasp.org/cheatsheets/Protect\\_FileUpload\\_Against\\_Malicious\\_File.html](https://cheatsheetseries.owasp.org/cheatsheets/Protect_FileUpload_Against_Malicious_File.html).

Загрузки вредоносных файлов являются настолько серьезной и важной темой, что канадское правительственные подразделение по кибербезопасности – Канадский центр безопасности коммуникаций (CSE) – создало и выложило в открытый доступ бесплатный инструмент для проверки загружаемых файлов под названием AssemblyLine ([cyber.gc.ca/en/assemblyline](https://cyber.gc.ca/en/assemblyline)). В отличие от других бесплатных онлайн-инструментов, он не передает информацию канадскому правительству.

## Ошибки и их регистрация

Все ошибки следует отлавливать и корректно обрабатывать. На экране не должны отображаться трассировки стека или ошибки базы данных. Эти требования необходимы не только для того, чтобы приложение выглядело профессионально, а у пользователей остались приятные впечатления от работы с ним, но и для того, чтобы злоумышленники не получили дополнительную информацию, которой они могли бы воспользоваться для выполнения своих атак. При возникновении ошибок приложение ни в коем случае не должно переходить в неизвестное состояние. Оно должно откатить все выполненные операции и «закрыть» все, что было открыто (то есть выполнить «аварийное завершение работы»). Такие ситуации всегда следует регистрировать, чтобы в случае необходимости специалистам по реагированию на инциденты было с чем работать при расследовании, а аудиторы могли убедиться в корректной работоспособности системы.

В большинстве случаев команда безопасности имеет SIEM, систему управления инцидентами и событиями безопасности, которая получает все лог-файлы (файлы регистрации) от сетевых инструментов, а также от приложения. Необходимо обеспечить получение файлов регистрации в желаемом командой формате и доступ SIEM к ним, когда приложение перейдет на стадию производства.

Нельзя допускать запись конфиденциальной информации: паролей, номеров социального страхования, полных номеров кредитных карт (будет достаточно последних четырех цифр), дат рождения, имен и фамилий и т. д. Не следует регистрировать никакую комбинацию данных, которая в совокупности может представлять собой персонально идентифицируемую информацию (ПИ). При наличии сомнений лучше посоветоваться с командой по защите конфиденциальности и с бизнес-аналитиками.

Если внутри приложения происходит активность, которую можно отнести к событию или происшествию, связанному с обеспечением безопасности, приложение должно не только зарегистрировать данную ситуацию, но и оповестить о ней. Оповещение может быть направлено в виде электронного письма команде безопасности, дежурному из

оперативного отдела либо тому, кому организация поручила реагировать на такие ситуации – решить этот вопрос необходимо еще на этапе разработки требований. Данное электронное письмо должно прийти на адрес команды или службы, а не отдельного человека, чтобы при его увольнении оповещения не отправлялись в никуда. Периодически нужно тестировать эту функциональность, чтобы убедиться в ее работе, ведь неработающие оповещения никому не помогают.

События, которые могут вызвать предупреждение о нарушении безопасности, включают в себя переход в неизвестное состояние, отказ в доступе к функции или документу, попытку обхода рабочего процесса, превышение квоты использования (например, людям не свойственно пытаться войти в систему 10 раз за минуту или 100 раз за час), аварийное завершение работы приложения, использование пользователем определенного (большого) объема трафика, оперативной памяти или ресурсов процессора, вызовы заблокированных HTTP-глаголов и т. д. Решение о том, что подходит для вашей организации и вашего приложения, должно быть принято совместно с бизнес-аналитиками и командой безопасности.

**СОВЕТ.** Дополнительную информацию о возможных оповещениях и регистрации можно посмотреть на сайте [cheatsheetseries.owasp.org/cheatsheets/Error\\_Handling\\_Cheat\\_Sheet.html](https://cheatsheetseries.owasp.org/cheatsheets/Error_Handling_Cheat_Sheet.html).

## Проверка и санитизация вводимых значений

JavaScript является особым типом языка программирования, потому что он работает в браузере, обычно называемом «стороной клиента», поскольку люди (клиенты), использующие веб-приложения, получают к ним доступ с помощью браузера, который находится на *их компьютере*, а не на *серверах*. Все остальные языки программирования работают на стороне сервера: на веб-сервере, PaaS («платформе как услуге» от облачного провайдера) или контейнере (подробнее о контейнерах позже).

Когда в веб-приложении используется веб-прокси, то он размещает себя *после выполнения JavaScript-приложения*, но *до того*, как запрос

будет отправлен по сети на веб-сервер. Тот, кто использует веб-прокси, может легко изменить HTTP-запрос до того, как он покинет компьютер, и таким образом обойти любую проверку ввода или санитизацию, записанную в JavaScript. Из рис. 2.4 становится ясно, почему проверка безопасности или санитизация должны выполняться на стороне сервера: слишком легко обойти защиту безопасности в JavaScript.

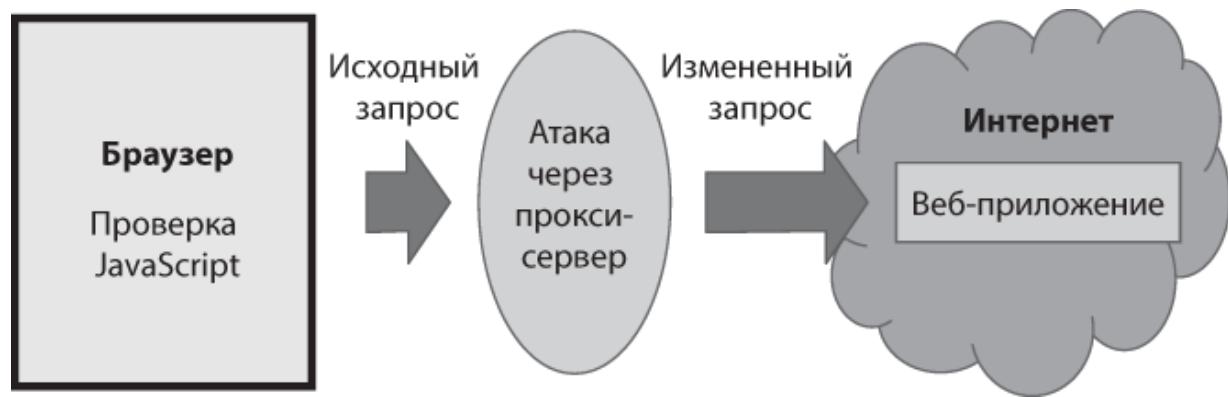


Рис. 2.4. Как прокси-сервер перехватывает веб-трафик

Второе, что необходимо обеспечить при формировании проверки, – это «список разрешенного» или «список одобренного» (белый список) вместо «списка блокировки» (черного списка). Злоумышленники и тестировщики на проникновения показали, что они способны регулярно обходить списки блокировки с относительной легкостью, творчески используя незаблокированные опции ввода. Количество различных способов, которыми кто-то может ввести одинарную кавычку (символ, наиболее часто используемый в атаках внедрения SQL-кода) в поле ввода, просто поражает воображение. Вместо того чтобы пытаться блокировать длинный список различных типов атак, можно составить список разрешенных элементов. «Белый список» пишется с помощью регулярных выражений (regex), и все, что им не соответствует, будет запрещено. Например, если вы хотите разрешить ввод только a – z и A – Z для имени пользователя, можно употребить следующее выражение: `^[a-zA-Z]{1,10}$`.

**ВНИМАНИЕ.** Regex иногда становится объектом атак типа «отказ в обслуживании» (DoS), поскольку его выполнение требует больших ресурсов. Поэтому приложение должно предупреждать о многократном вызове данной функции.

## Авторизация и аутентификация

Если приложение имеет различные типы доступа для различных типов пользователей, то есть когда речь идет об управлении доступом на основе ролей (Role Based Access Control, RBAC), необходимо определиться с ними на этапе разработки требований. Не следует менять их по мере разработки системы, так как это усложняет работу с ней. RBAC является частью авторизации (AuthZ).

Нужно также определить, какие методы либо системы аутентификации (AuthN) и идентификации следует использовать: как система будет проверять личность пользователей? Как будет устанавливаться личность? Нужно ли отслеживать ее в нескольких системах или только на сайте? Принимать решения по всем этим вопросам лучше всего на этапе разработки требований, но также допустимо и на этапе проектирования.

**СОВЕТ.** Внедрите автоматизированный набор тестов, чтобы проверять соответствие реализации AuthZ. Автоматизация тестов позволит чаще проводить проверки AuthZ, чтобы убедиться в отсутствии случайных изменений.

## Параметризованные запросы

При работе с базой данных приложение дает ей указание выполнить действия от его имени. Это может быть запрос на чтение, запись, обновление или удаление, но суть в том, что приложение обращается *непосредственно* к базе данных. Ни один человек или приложение не должны иметь возможность напрямую обращаться к *вашей* базе данных, кроме *вашего* приложения, *вашей* команды разработчиков программного обеспечения или *вашего* администратора базы данных. При атаке SQL или NoSQL злоумышленник пытается напрямую обратиться к базе данных, отправляя ей команды для выполнения нужных ему действий, а не запрограммированных инструкций. Такая атака называется инъекционной и является угрозой номер один по степени риска для любого веб-приложения (согласно Топ-10 OWASP, подробно описанному в главе 5).

При использовании параметризованных запросов (в SQL они называются хранимыми процедурами) мы отправляем параметры и имя запроса, который хотим выполнить, в базу данных, а не создаем код путем объединения пользовательского ввода для создания строки, которую затем отправляем в базу данных как команду. Разница в том, что в случае параметризованных запросов, если злоумышленник попытается добавить свой код через пользовательский ввод, приложение отправит его в одном из параметров и он не сработает. Так происходит потому, что параметры интерпретируются базой данных как данные, а не как код, что делает инъекционные атаки практически невозможными.

Когда приложение объединяет строки пользовательского ввода и затем отправляет их в базу данных непосредственно в виде команды, на языке SQL это называется «встраиваемым SQL». Написание встраиваемого SQL создает потенциальный риск атаки через внедрение SQL-кода. Лишь использование параметризованных запросов надежно снижает эту уязвимость. Всегда используйте параметризованные запросы.

## **Параметры URL**

Злоумышленники, как и пользователи, имеют доступ к адресной строке в браузерах. Не помещайте переменные, которые имеют значение для вашего приложения, в параметры URL. Можно увеличить ID-номер, и, если приложение не было запрограммировано на такую ситуацию, пользователь увидит чужую учетную запись. Содержащиеся в параметрах URL-адреса конфиденциальные значения регистрируются, что также приводит к проблемам безопасности. Очень просто манипулировать значениями в параметрах URL, и оправдание «никто не должен был этого делать» не сработает для вашей команды реагирования на инциденты. Передавайте в параметрах URL только значения нулевой важности, например на каком языке пользователь хочет просматривать страницу. Никогда не передавайте ничего важного или конфиденциального.

## **Принцип наименьших привилегий**

Необходимым условием работы приложения является соблюдение принципа наименьших привилегий, особенно в отношении доступа к базе данных или API. Приложение должно использовать только сервисную учетную запись для вызова API, параметризованных запросов или любых других вызовов, требующих наличия учетной записи.

### **Несколько примеров реализации принципа наименьших привилегий:**

Сервисная учетная запись, которая вызывает базу данных из приложения, должна иметь только права CRUD (CRUD – акроним, обозначающий четыре базовые функции, используемые при работе с базами данных: создание (англ. create), чтение (read), модификация (update), удаление (delete)), но никак не права владельца базы данных (DBO)).

Лучше всего, если одна сервисная учетная запись имеет доступ только для чтения и для вызова команды «select», а другая имеет CRUD-доступ, когда требуется создание, редактирование или удаление записей. Во всех возможных случаях лучше использовать учетную запись только для чтения.

Создание отдельной сервисной учетной записи для каждого используемого приложением API. Каждая из них должна иметь только те права, которые необходимы для выполняемых ею задач (например, только для команд «select» или «read», если предполагается, что она будет только возвращать данные, а не изменять их).

Предоставление исключительно членам команды доступа для чтения либо записи проекта в библиотеке кода, а всем остальным – только для чтения либо вообще оставить без доступа, если код, используемый в приложении, очень чувствителен или ценен по своей природе.

**СОВЕТ.** Реализацией принципа наименьших привилегий является использование во всех возможных случаях сервисной учетной записи с доступом только для чтения вместо учетной записи со всеми CRUD-правами.

## Чек-лист требований

Данный чек-лист можно использовать при работе над всеми проектами по разработке веб-приложений. Все перечисленные требования применимы к любому веб-приложению, и я предлагаю рассматривать их в качестве необходимого минимума, а также добавить те, которые соответствуют уникальным потребностям вашей организации.

Следует:

- шифровать все данные в состоянии покоя (пока они находятся в базе данных);
- шифровать все данные при передаче (на пути к пользователю, базе данных, API и т. д.);
- никому не доверять: проверять (при особых обстоятельствах санитизировать) все данные, даже из собственной базы данных;
- шифровать (и, если нужно, экранировать) все выходные значения;
- перед и регулярно после использования проверять все библиотеки и сторонние компоненты на наличие известных уязвимостей (так как постоянно появляются новые уязвимости и версии);
- использовать все подходящие заголовки безопасности;
- использовать необходимые безопасные параметры cookie;
- классифицировать и маркировать все хранимые, собираемые и создаваемые приложением данные;
- хешировать и солить все пароли пользователей. Соль должна состоять не менее чем из 28 символов;
- хранить все секреты приложения в тайнике;
- обеспечить повсеместное использование сервисных учетных записей (не личных аккаунтов) при работе внутри приложения;
- обеспечить использование менеджеров паролей всеми членами команды и запретить повторное использование паролей;
- избегать изменений паролей по расписанию или через определенный промежуток времени, кроме случаев взлома или подозрительной активности;
- разрешать доступ к интернет-сайтам общего пользования только через HTTPS. Перенаправлять с HTTP на HTTPS. Желательно

применять это требование как к внутренним, так и к внешним приложениям;

- обеспечить использование последней версии TLS для шифрования (в настоящее время 1.3);
- никогда не использовать жесткое кодирование. Никогда;
- никогда не размещать конфиденциальную информацию (в том числе строки соединения и пароли) в комментариях, ее необходимо хранить в тайнике;
- использовать функции безопасности, встроенные в платформу. Например, криптографию и шифрование, функции управления сессиями или функции санитизации ввода. Не следует писать собственные функции, если платформа предоставляет свои;
- использовать только последнюю (или предыдущую) версию платформы и поддерживать ее в актуальном состоянии. Технический долг = долг безопасности;
- при загрузке файлов следовать рекомендациям OWASP: сканировать все загружаемые файлы с помощью бесплатного инструмента AssemblyLine, предоставляемого Канадским центром безопасности коммуникаций (Communications Security Establishment, CSE);
  - обеспечить регистрацию всех ошибок (исключая конфиденциальную информацию) и включить оповещение о возникновении ошибок, связанных с обеспечением безопасности;
  - обеспечить выполнение всех проверок ввода (и его санитизацию) на стороне сервера с использованием «белого списка» (не «черного списка»);
  - обеспечить тестирование безопасности приложения перед релизом (подробнее об этом в следующих главах);
    - перед релизом приложения выполнить моделирование угроз, речь о котором будет идти в главе 3 в разделе «Моделирование угроз»;
    - выполнить проверку кода (в частности, функций безопасности) приложения перед релизом;
    - убедиться в том, что приложение отлавливает все ошибки и производит аварийное завершение работы правильно (никогда не переходит в неизвестное состояние);
    - обеспечить, чтобы все ошибки предоставляли пользователю только общую информацию, а не информацию из трассировки стека, ошибки

запроса или другую технически специфическую информацию;

- определить специфику доступа на основе ролей в требованиях к проекту;
- определить конкретные методы аутентификации и системы идентификации в требованиях к проекту;
- использовать только параметризованные запросы и никогда – встраиваемые SQL/NoSQL;
- избегать передачи имеющих какое-либо значение переменных в параметрах URL;
- обеспечить соблюдение принципа наименьших привилегий в приложении, особенно в отношении доступа к базе данных и API;
- минимизировать поверхность атаки, когда это возможно;
- разрешить функции вырезать-вставить в поле пароля, что позволит пользователям применять менеджеры паролей. Отключить функции автозаполнения полей, чтобы пользователи не сохраняли свои пароли в браузере;
- отключить кэширование на страницах, содержащих конфиденциальную информацию. Заголовок Cache HTTP технически не является заголовком безопасности, однако он подходит для выполнения этого требования;
- убедиться, что пароли пользователей приложения длинные, но не обязательно сложные. Чем длиннее пароль, тем лучше. Следует также поощрять использование кодовых фраз;
- с помощью специальных сервисов проверять, не были ли новые пароли пользователей ранее взломаны.

В зависимости от выполняемых приложением задач можно добавить или убрать некоторые требования. Смысл данной главы в том, чтобы заставить вас задуматься о безопасности во время составления списка требований к проекту.

Если разработчики с самого начала знают, что им необходимо придерживаться определенных требований, уровень безопасности создаваемого программного обеспечения вырастает.

## Упражнения

- Назовите два дополнительных возможных требования безопасности для веб-приложения (не упомянутых в списке).

- Назовите два дополнительных возможных требования безопасности для операционной системы автомобиля.
- Назовите два дополнительных возможных требования безопасности для «умного тостера».
- Назовите два дополнительных возможных требования безопасности для приложения, работающего с кредитными картами.
- Какое требование безопасности является наиболее важным для вас или вашей организации? Почему?
- Если бы вам нужно было убрать одно из упомянутых в этой главе требований из проекта веб-приложения, какое бы это было требование? Почему?

**ПРИМЕЧАНИЕ.** Напоминаем, что раздел «Ответы» в конце книги не предоставляет полную информацию. Более подробные ответы на вопросы можно получить во время их обсуждения с коллегой или другом. Присоединяйтесь к моему сообществу, чтобы принять участие в онлайн-дискуссии, или посмотрите видео с обсуждениями на [youtube.com/shehackspurple](https://youtube.com/shehackspurple).

## Глава 3

# Безопасность при проектировании ПО

В предыдущей главе мы говорили о требованиях безопасности. Они являются обязательным условием при создании любого продукта, а наличие требований по обеспечению безопасности с самого начала разработки – первый шаг к высокому качеству конечного продукта. В этой главе мы обсудим следующий этап жизненного цикла разработки системы – проектирование (рис. 3.1).

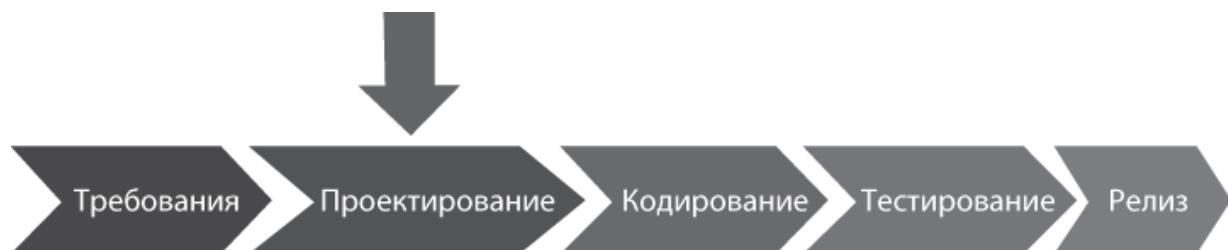


Рис. 3.1. Жизненный цикл разработки системы

При проектировании архитекторы программного обеспечения должны думать не только о бизнес-требованиях (о чем попросил заказчик) и функциональных требованиях (пользовательских требованиях, разработке, планировании работы, системных требованиях), но также о нефункциональных требованиях, которые часто рассматриваются как нечто само собой разумеющееся: удобстве использования, качестве, а также главной теме данной книги – безопасности.

*Безопасность через проектирование (англ. *Secure by design*) в программной инженерии означает, что безопасность программного обеспечения реализуется с самого начала его проектирования. При таком подходе вредоносные действия, направленные против приложения, воспринимаются как нечто само собой разумеющееся и внимание уделяется минимизации*

*воздействия обнаруженных уязвимостей безопасности или недопустимого пользовательского ввода.*

*Википедия*[\[25\]](#)

Проектирование программного обеспечения можно трактовать по-разному, поэтому дадим свое определение данному понятию.

Проектирование программного обеспечения – это планирование разработки ПО, способов его соединения или работы с необходимыми компонентами. Оно может представлять собой документ очень высокого уровня, показывающий, как все части приложения работают вместе: например, этот пользовательский интерфейс подключается к этой базе данных и вызывает этот API.

Проектные документы могут быть чрезвычайно формальными и содержать много подробностей, вплоть до того, какие классы будут определены и как, а могут просто состоять из нескольких рисунков и диаграмм потоков данных. Цель проектного документа – донести до команды, как ей нужно разрабатывать приложение.

Достаточность средств защиты следует *обсудить со службой безопасности*, однако перед этим проектный документ необходимо согласовать с командой разработчиков.

## **Ошибка проектирования и дефект безопасности**

*Ошибка проектирования* называется ошибка в конструкции приложения, позволяющая пользователю выполнять действия, на которые у него нет разрешения (то есть вредоносные или повреждающие приложение действия). Это ошибка, проблема проектирования. Использование концепции проектирования безопасного ПО, требования безопасности к проектам и моделирование угроз позволяют избежать или минимизировать возможности возникновения ошибок проектирования.

*Дефект безопасности* – это проблема реализации, проблема в коде, позволяющая пользователю использовать приложение вредоносным или иным некорректным способом. Для защиты от дефектов безопасности проводятся анализ кода, тестирование безопасности (многих типов, на разных этапах проекта), обучение написанию

безопасного кода, а также применяются концепции и рекомендации по безопасному кодированию. На рис. 3.2 видно, на каких этапах жизненного цикла разработки системы появляются ошибки проектирования и дефекты безопасности.



Рис. 3.2. Ошибки проектирования и дефекты безопасности

### Позднее обнаружение ошибки проектирования

Чем позже в жизненном цикле разработки системы будет устранена ошибка, тем дороже она обойдется. В одной статье онлайн-журнала *Slashdot* говорится, что ошибка, обнаруженная на этапе требований, может стоить \$1, на этапе проектирования – \$10, на этапе кодирования – \$100, на этапе тестирования – \$1000, а после релиза – еще больше<sup>[26]</sup>. В интернете можно найти множество различных оценок стоимости, но, вместо того чтобы использовать приблизительные числа в попытке объяснить данную идею, проконсультируемся с Бобом. На рис. 3.3 изображена растущая стоимость устранения проблем безопасности на каждом из этапов жизненного цикла разработки системы.

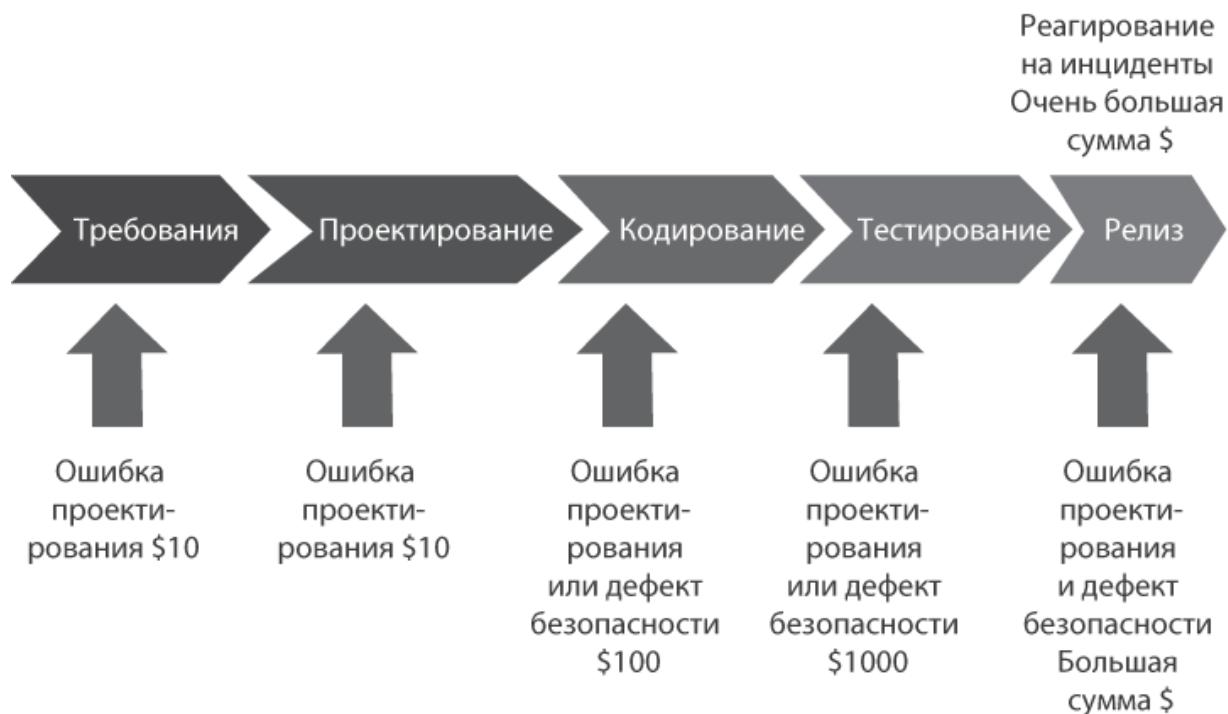


Рис. 3.3. Примерная стоимость исправления ошибок и дефектов безопасности в жизненном цикле разработки системы

Давным-давно Боб и его супруга построили дом своей мечты, на который они копили много лет. Строительство находилось на конечном этапе: уже устанавливались ручки на шкафы и раскатывались ковры, – когда Боб понял, что они совершили ошибку. Почему-то на протяжении всего планирования молодой семье не приходило в голову добавить вторую ванную комнату. Когда Боб и его жена начинали копить и планировать дом, у них был один ребенок, но к моменту окончания стройки детей стало уже двое и рассматривалась возможность рождения третьего – им определенно нужна была еще одна ванная комната.

Боб знал, что добавление ванной комнаты на таком позднем этапе будет довольно дорогим изменением и отодвинет срок завершения строительства, но он также понимал, что одной ванной комнаты будет определенно недостаточно. Строительная компания ему объяснила, что придется уменьшить жилую площадь в два раза, а переезд задержится на целый месяц. К тому же на 10 % увеличится стоимость проекта. Семья неохотно согласилась с этими условиями, но Боб

усвоил очень ценный урок: гораздо дешевле, проще и быстрее изменить проект на его ранних этапах.

Такая же ситуация и с программным обеспечением. Изменения, внесенные в конструкцию приложения в последнюю минуту, не всегда красивы, зачастую приводят к срыву сроков и повышают общую стоимость. К сожалению, из-за них также могут возникнуть новые уязвимости в конечном продукте.

## Сдвиг влево

Проблему «недостатка ванных комнат» можно обнаружить посредством моделирования угроз, о котором будет рассказано в конце главы. Она может стать явной на более ранних этапах жизненного цикла разработки системы также при следовании концепции проектирования безопасного ПО. Именно из-за таких проблем обеспечением безопасности нужно озабочиваться на начальных, а не конечных этапах работы над проектом.

На рис. 3.4 показан жизненный цикл разработки системы со стрелками, направленными в левую часть страницы: чем «левее» в данной схеме начинается деятельность по обеспечению безопасности, тем раньше в цикле это происходит. Такая схема, которую одни специалисты по безопасности называют «смещением влево», а другие – «сдвигом влево», представляет собой самый эффективный способ обеспечить создание безопасного программного обеспечения.



Рис. 3.4. Сдвиг влево

## Концепции проектирования безопасного ПО

Теперь поговорим о нескольких концепциях проектирования безопасного ПО, которые следует применять (или хотя бы учитывать) при разработке программных приложений.

### Защита конфиденциальных данных

В главе 2 «Требования безопасности» мы уже определили, что все создаваемые, собираемые, сохраняемые и обрабатываемые приложением данные должны классифицироваться и маркироваться. Замечательно, а что теперь?

При работе с любыми данными необходимо обеспечить их сохранность согласно триаде CIA. Независимо от того, конфиденциальные они или нет, в случае необходимости они должны быть доступными (доступность) и точными (целостность). Данные, которые не являются общедоступными либо несекретными, также должны быть защищены, чтобы секреты *оставались* секретами (конфиденциальность).

**ПРИМЕЧАНИЕ.** «Несекретные» – это не конфиденциальные по своей природе данные. «Общедоступные» данные могут быть опубликованы для общественности (им присвоен такой уровень чувствительности, согласно которому их могут увидеть все).

**ВНИМАНИЕ.** Если данные в приложении содержат государственную тайну, необходимо следовать правилам, установленным правительством, а не данному руководству.

Мы должны шифровать *все* данные при передаче и в состоянии покоя, использовать только протокол HTTPS, проверять входные данные и кодировать выходные, использовать заголовки безопасности, избегать размещения конфиденциальной информации в скрытых полях или параметрах URL, защищать файлы cookies, обеспечивать авторизацию пользователей перед предоставлением доступа к просмотру данных, а также использовать более продвинутые настройки веб-сервера и базы данных – это основы работы с данными. Но как насчет сверхчувствительных данных?

Прежде всего необходимо разработать процессы и политику обращения с конфиденциальными данными, следовать уже установленным правилам, а также добавить все возможные из обсуждаемых здесь пунктов для дальнейшего усиления защиты данных.

При работе с конфиденциальными данными следует установить время их хранения в приложении и создать план их утилизации по

окончании данного срока. Например, если данные должны храниться в течение семи лет, то по истечении этого времени система должна автоматически уничтожить каждую запись об этих данных. Бессмысленно ожидать, что пользователи не забудут удалить свои данные позже (это точно не так), поэтому стоит предусмотреть такую функцию в приложении. Помимо этого, следует обеспечить проверку успешного и полного уничтожения информации, а также спланировать удаление данных из всех возможных систем резервного копирования.

Многие компании переносят свои данные в публичное облако, поэтому очень важно решить, где будут размещаться различные типы и классы информации. В каждой стране существуют свои юридические указания и требования, которым необходимо следовать. Если для вашей ситуации нет юридических указаний, а вы решили использовать публичное облако, следует применить настройку «взять собственные ключи» для информации с высоким уровнем конфиденциальности, то есть вы сами будете управлять ключами для шифрования и дешифрования данных, а не разрешать поставщику услуг делать это от вашего имени.

Для территориально распределенных систем необходимо решить, каким регионам разрешить доступ к данным и где хранить резервные копии. Опять же, могут существовать юридические указания и требования правительства страны, которым необходимо следовать в этом вопросе.

При работе с чрезвычайно чувствительными данными можно использовать систему токенизации, которая обычно предоставляется третьей стороной. Данные заменяются токенами, а затем расшифровываются использующим их сторонним программным обеспечением. Токенизация является дополнительным фактором безопасности.

Если система или ее данные очень чувствительны, пароли пользователей следует хешировать, солить и перчить, как говорилось в главе 2. На момент написания данной книги большинство приложений солят, хешируют, но не перчат пароли.

Поскольку стандарты шифрования постоянно меняются (и совершенствуются), конкретные шифры, длина ключей и алгоритмы здесь не указываются. Поиските текущие стандарты в интернете и

обязательно проверяйте те, которые будут использоваться в новых проектах (они могли измениться).

По возможности следует установить оповещения об обнаружении утечки похожих на ваши данные в интернет. Вы же не хотите узнать такую информацию, прочитав газету, а хотите получить ее раньше, чем СМИ. Можно настроить такие оповещения вручную или воспользоваться услугами, предлагаемыми поставщиками систем безопасности. Будет полезно настроить оповещения в различных социальных сетях, поисковых системах и платформах обмена данными по ключевым словам (что приведет к ложным срабатываниям, поэтому будьте готовы к определенному «шуму»). Некоторые правительственные организации помогают компаниям – резидентам своих стран отследить утечки. Узнайте у своего правительства, какие виды услуг или помощи они могут предложить. В Канаде эта группа называется Канадский центр кибербезопасности (CCCS, англ. Canadian Center for Cyber Security) ([cyber.gc.ca](http://cyber.gc.ca)<sup>[27]</sup>), ранее известный как Канадский центр реагирования на киберинциденты (CCIRC).

В других странах действуют:

- Люксембург: [www.circl.lu](http://www.circl.lu) <sup>[28]</sup>;
- Япония: [www.jpcert.or.jp/english/at/2020.html](http://www.jpcert.or.jp/english/at/2020.html) <sup>[29]</sup>;
- Великобритания (Отчет об угрозах): [www.ncsc.gov.uk](http://www.ncsc.gov.uk) <sup>[30]</sup>;
- Соединенные Штаты Америки: [www.us-cert.gov/ncas/alerts](http://www.us-cert.gov/ncas/alerts) <sup>[31]</sup>;
- Новая Зеландия (Оповещения): [www.cert.govt.nz/it-specialists](http://www.cert.govt.nz/it-specialists) <sup>[32]</sup>.

Узнать о предложениях вашего правительства можно с помощью местного центра реагирования на киберинциденты (CIRT, англ. Cyber Incident Response Team).

**СОВЕТ.** Даже если данные, с которыми работает приложение, не являются конфиденциальными, следует обеспечить как их правильное хранение, так и безопасность самого приложения. Дополнительные меры предосторожности, упомянутые в этом разделе, требуются только в том случае, если заинтересованные стороны и служба безопасности установили, что данные или приложение имеют высокий уровень чувствительности. Остальные концепции, изложенные в книге, применимы к

данным и приложениям независимо от уровня их чувствительности (если не указано иное).

## «Никогда не доверяй, всегда проверяй» и «Предполагать взлом»

В главе 1 мы затронули идею нулевого доверия и концепцию предположения взлома, но как применять их при проектировании приложений? Помимо того что можно проанализировать и обсудить проект со специалистом по безопасности (что является обязательной частью проектирования), ниже приведены несколько непоколебимых правил.

- Использовать только надежные данные на стороне сервера (данные, которые были правильно проверены) для принятия решений по управлению доступом.
- Ставить запрет по умолчанию – перед началом работы все функции должны проверять авторизацию пользователя.
- В случае отказа приложение должно всегда закрываться или переходить в безопасный режим работы и никогда не переходить в открытое или неизвестное состояние. При каждом неправильном завершении какой-либо операции должен производиться откат.
- Выполнять проверку личности (аутентификацию) и лишь затем – авторизацию (управление доступом).
- Проверять доступ на каждой странице и при каждом обращении к какой-либо функции приложения, даже при перезагрузке страницы.
- Проверять аутентификацию (AuthN) и авторизацию (AuthZ) для API в обоих направлениях (API → приложение → API).
- Блокировать доступ к любому не используемому приложением протоколу, порту, HTTP-глаголу, методу и всему остальному на сервере, PaaS или контейнере.
- Одно приложение на сервер, PaaS или контейнер в зависимости от модели развертывания и бюджета.

**ПРИМЕЧАНИЕ.** Валидация ввода была рассмотрена в первой главе, но данную тему мы будем затрагивать на протяжении всей книги. Если бы каждый разработчик программного обеспечения овладел искусством валидации, исчезло бы значительное количество всех типов уязвимостей во всех веб-приложениях. Принцип «никогда не доверять, всегда проверять» является самым важным и ценным уроком, представленным в книге.

Стоит потратить время на анализ проекта со специалистом по безопасности или другим техническим экспертом, который не является членом команды (и поэтому непредвзят). Необходимо обнаружить все объекты в проекте, в отношении которых подразумевается доверие, и заменить это доверие на проверку. Такая работа обычно занимает не более часа. Ее можно выполнять на доске, а затем сфотографировать результат и сохранить для последующего использования.

## **Резервное копирование и откат**

Несколько месяцев назад компания Алисы подверглась атаке вымогательского ПО. Кто-то перешел по ссылке в фишинговом письме и случайно заразил всю сеть. К счастью, Алиса всегда хранит копии важных для нее файлов на личном ноутбуке, чтобы иметь возможность работать из дома. Это противоречит политике компании, но она так делает уже много лет. К несчастью для остальных сотрудников, программа-вымогатель зашифровала все файлы компании и тем самым остановила всю работу. В течение месяца сотрудники отвечали на электронные письма со своих телефонов, а некоторые файлы так и не были восстановлены. Алиса так и не узнала полную стоимость ущерба от происшествия, но была рада, что компания не закрылась, в отличие от других, о которых она слышала.

**ПРИМЕЧАНИЕ.** Алиса сохраняла копию своих файлов, чтобы работать из дома, то есть *обходила* политику безопасности, мешающую ей выполнять свои обязанности. В политике безопасности отсутствовало *удобство*. Поскольку Алисе нужно было работать из дома, она нарушила эту политику. Избежать данного обхода можно было бы посредством разработанного службой безопасности *удобного, но в то же время безопасного* решения (например, удаленный доступ в сеть компании через VPN).

*Резервные копии бесполезны, если их нельзя эффективно использовать для отката.* Процесс отката длиной в месяц – это месяц простоя. Чрезвычайно важно, чтобы частью конструкции приложения

было не только резервное копирование, но и быстрый откат данных и восстановление систем.

- Резервные копии должны храниться территориально в другой точке (не рядом с базой данных и веб-сервером).
- Резервные копии должны находиться в безопасном месте (как в физическом, так и в цифровом виде) и быть защищены так же, как и производственная база данных или веб-сервер.
- Следует проводить учебный откат не реже одного раза в год. Когда происходят значительные изменения в производственных системах, сценарии отката должны быть готовы и протестированы.
- Необходимо создавать резервные копии базы данных, конфигураций и кода приложения.
- Получение доступа к резервным копиям должно отслеживаться, регистрироваться и сопровождаться оповещением.
- Удаление резервных копий должно быть отложено по времени на случай, если их попытаются удалить в ходе атаки. Некоторые облачные провайдеры делают это по умолчанию. Предлагаемый график: удаление резервных копий всех производственных баз данных на четырнадцатый день после запроса.
- Проводите учебные откаты. Серьезно!

**СОВЕТ.** Симуляции инцидентов безопасности (также известные как учения по безопасности) – отличный способ потренироваться перед возникновением реальных происшествий. Симуляция атаки вымогающего ПО будет отличным поводом проверить, работают ли планы резервного копирования и отката. Намного дешевле и легче готовиться к событию, которое никогда не произойдет, чем оказаться неподготовленным к нему.

## Валидация на стороне сервера

При проведении тестирования безопасности веб-приложений (например, тестирования на проникновение) в большинстве случаев используется так называемый веб-прокси или прокси перехвата (иногда в сочетании со сканером веб-приложений, что позволяет проводить автоматизированное тестирование). Веб-прокси, как следует из названия, располагается между браузером и веб-сервером и перехватывает трафик. Этот тип инструмента используется для прямого взаимодействия с приложением посредством отправки запросов на веб-сервер и откликов после завершения работы JavaScript в интерфейсе пользователя (в компьютере либо браузере пользователя). Если вы впишете код проверки в JavaScript, чтобы убедиться в безопасности пользовательского ввода, любой пользователь с помощью веб-прокси может ввести достоверные данные, приостановить их передачу (перехватить запрос) после того, как JavaScript закончит проверку данных, и заменить на вредоносные данные (отредактировать запрос и отправить его дальше). Будет правильным, если приложение повторно проверит данные на сервере: можно выполнить очень простую проверку данных в JavaScript, которая сэкономит время. Если же данные на сервере не будут проверены, злоумышленники легко могут отправить приложению вредоносные запросы с помощью веб-прокси. На рис. 3.5 изображен поток HTTP-запросов при использовании веб-прокси.



Рис. 3.5. Использование веб-прокси для обхода проверки JavaScript

К великому сожалению проектировщиков веб-приложений, пользователи обладают большими возможностями. Они могут использовать небезопасный или устаревший браузер, заходить на сайты с компьютера, зараженного вредоносным ПО, изменять настройки браузера, редактировать параметры в URL-адресе и даже перехватывать запросы и отклики в наших приложениях. Люди также могут скрыть свое местоположение, IP-адрес, тип используемых ими операционной системы и браузера и даже веб-страницу, с которой они пришли. Плохо это или хорошо, но интернет был создан для того, чтобы дать пользователям большую власть.

Отправка результатов каждой проверки обратно на сторону сервера может показаться дорогостоящей, однако в действительности она обойдется дешевле, чем успешная эксплуатация веб-приложения злоумышленником.

## **Функции безопасности платформы**

В молодости Боб недолго проработал программистом, но понял, что кодирование не для него. Сначала он подумал, что может написать собственные функции, вместо того чтобы использовать элементы платформы. Таким образом он хотел улучшить качество работы и продемонстрировать руководству свою ценность. Начальник же объяснил Бобу, что писать собственные средства защиты опасно: «Боб, над этой платформой работали несколько профессионалов, люди, которые подробно изучили проблему и имеют многолетний опыт. Ты действительно думаешь, что у нас есть деньги, чтобы заплатить тебе за изобретение колеса, а в проекте выделено время на тестирование написанного тобой кода? Неужели ты считаешь, что один человек, не имеющий специальных знаний в данной области, может сделать работу лучше, чем целая команда экспертов?» Боб был подавлен, но, поразмыслив, признал правоту начальника: следует использовать функции безопасности платформы, а не писать собственные.

Это относится ко всем нам. Ни в коем случае нельзя писать собственные функции безопасности взамен существующих в используемой платформе. Если платформа таких функций не имеет, то,

прежде чем приступить к их созданию, необходимо тщательно оценить ситуацию или приобрести доступный и надежный вариант.

Например, существует множество различных тайников, которые можно получить бесплатно или по цене, доступной для большинства организаций. Придумать, как хранить секреты вне кода, – очень сложная задача, поэтому почти всегда имеет смысл использовать готовое решение. Тем не менее в очень редких случаях иногда нет другого выхода, кроме как написать собственный код. Но прежде, чем принять такое решение, следует провести анализ вариантов.

**СОВЕТ.** Тайник – это место, где хранятся секреты приложения. В отличие от менеджеров паролей, предназначенных для людей, тайниками пользуются машины. В них можно и нужно хранить строки подключения, учетные данные, ключи, хеши и другие секреты.

## Изоляция функций безопасности

Функции, касающиеся безопасности (это относится и к проектированию оборудования), по возможности следует изолировать от остальной функциональной части приложения. Благодаря этому нарушение функций, не связанных с обеспечением безопасности, не заденет защиту приложения и не станет причиной возникновения уязвимости. Данный совет упоминается в очень надежных руководствах по безопасности: ITSG-33<sup>[33]</sup> канадского правительства и Специальной публикации NIST 800-53 S-3<sup>[34]</sup>.

Как его реализовываем мы? Для каждого элемента управления безопасностью мы обособляем код или функцию. Мы выделяем их в отдельный объект либо класс, размещаем на другой странице либо панели внутри приложения или системы и/или вообще в отдельном приложении. Вероятнее всего, мы также заставим пользователя или приложение пройти повторную аутентификацию прежде, чем разрешить доступ к функциям безопасности. Вот примеры наших действий.

- Вынесение проверки ввода в отдельный объект или класс.
- Вынесение аутентификации и авторизации в отдельное приложение.

- Управление идентификацией, осуществляющееся облачным провайдером или другой внешней системой.

Уровень разделения функций зависит от чувствительности данных, с которыми будет работать разрабатываемое приложение, и мощности функционала, отвечающего за обеспечение безопасности.

Код функции безопасности может храниться на отдельном веб-сервере, PaaS, контейнере или файловой системе. Между двумя приложениями может быть брандмауэр или в крайнем случае совершенно отдельные сети.

Системы идентификации и аутентификации всегда должны находиться на отдельном сервере и следовать принципам по усилению защиты.

В число функций безопасности, которые всегда должны быть изолированы, входят: связанные с безопасностью протоколирование и мониторинг, предотвращение и обнаружение попыток несанкционированного доступа, файрвол веб-приложений (англ. Web application firewall, WAF), программное обеспечение для контроля приложений, мониторинг целостности файлов, оповещение или блокирование.

## Разделение приложения

По возможности функции, касающиеся управления системой (администрирования), должны быть изолированы от остальной функциональной части приложения. Таким образом, нарушение функций, не связанных с обеспечением безопасности, не повлияет на управление системой (контроль доступа). Данный совет упоминается в очень надежных руководствах по безопасности: ITSG-33<sup>[35]</sup> канадского правительства и Специальной публикации NIST 800-53 S-3<sup>[36]</sup>.

**ПРИМЕЧАНИЕ.** Определение из стандарта ITSG-33: информационная система отделяет функциональные возможности пользователя (включая услуги пользовательского интерфейса) от функций управления информационной системой.

Здесь также речь идет о разделении функций, но мы говорим уже об административных возможностях. Если в приложении есть административная часть, то соответствующие функции должны находиться в отдельном объекте либо классе, на другой странице либо панели или в отдельном приложении. Желательно, чтобы администраторы приложения входили в систему с отдельным набором учетных данных (с включенным MFA).

## Управление секретами приложения

Прежде всего стоит дать определение секретам применительно к программному обеспечению. Когда мы говорим о секретах и управлении секретами, мы не имеем в виду секреты, которые пользователи хранят внутри разрабатываемого программного обеспечения, мы имеем в виду скрытые элементы, *используемые программным обеспечением* для корректной работы. Имя пользователя и пароль (учетные данные), которые требуются для входа в приложение, являются *секретом пользователя*. Мы же говорим о секретах, которые *компьютерные системы используют* для взаимодействия друг с другом.

Когда приложение входит в базу данных, используемая им строка подключения является секретом. Когда веб-сервер включает протокол HTTPS (шифрование) в веб-приложении с помощью сертификата и пары ключей, закрытый ключ для этого сертификата является секретом. Закрытые ключи, API-ключи, хеши, пароли и все остальное, что является конфиденциальным и используется приложением (а не человеком или пользователем), считается *секретом*.

Ручное управление секретами – ужасная практика, которая отнимает много времени, имеет высокую сложность и чревата ошибками. Именно поэтому были созданы тайники – специальное программное обеспечение, которое управляет секретами и доступ к которому можно получить через приложение, конвейер CI/CD или сервисную учетную запись. Человек не должен иметь доступ к секретам, находящимся в тайнике, кроме тех случаев, когда их необходимо поместить в тайник или обновить либо когда произошла чрезвычайная ситуация. Доступ к тайнику должен тщательно охраняться, а все связанные с ним события – регистрироваться и сопровождаться оповещением.

Хороший тайник обязан предупреждать об истечении срока действия сертификатов и предоставлять подробную информацию о последних изменениях секретов. Некоторые тайники имеют еще больше возможностей. Лучше пользоваться всеми функциями тайника, а не управлять им вручную.

## **Повторная аутентификация при транзакционных операциях (предотвращение CSRF-атаки)**

Подделка межсайтовых запросов (CSRF)<sup>[37]</sup> – это уязвимость, при которой злоумышленник убеждает жертву перейти по ссылке или посетить вредоносный сайт. Ссылка или сайт запускает транзакцию в приложении (допустим, покупку нового модного телевизора, который будет отправлен злоумышленнику), и поскольку пользователь уже вошел в учетную запись (кто не оставляет браузер открытым несколько дней подряд?), уязвимое веб-приложение завершает транзакцию (покупку), о которой пользователь ничего не знает, пока не придет счет. Однако к тому времени уже слишком поздно реагировать. Лучший способ защиты от данной уязвимости – перед каждым важным действием приложения (покупка, деактивация аккаунта, смена

пароля и т. д.) запрашивать у пользователя что-то, что может предоставить только он сам: предлагать повторно ввести пароль, ввести капчу или секретный токен, который будет только у него. Наиболее распространенным подходом является использование секретного токена (часто называемого анти-CSRF-токеном). На момент написания книги этот метод защиты настолько распространен, что уже включен во многие современные платформы (такие как. Net, Java и Ruby on Rails). Таким образом, нет необходимости кодировать эту функцию самостоятельно, а следует лишь убедиться в том, что она включена.

**СОВЕТ.** Пользователи ненавидят капчу. По возможности используйте вместо них невидимые для пользователя анти-CSRF-токены. Применять анти-CSRF-токены вместо того, чтобы заставлять вводить капчу или (повторно) пароль, – значит соблюдать приоритет *удобной безопасности*.

## Разделение производственных данных

Производственные данные не должны быть задействованы в тестировании, разработке или любых других процессах, не связанных с бизнесом. В разработке и тестировании используется замаскированный (анонимизированный) набор данных, а в производстве находятся только «настоящие» данные.

Разделение производственных данных означает предоставление доступа к данным меньшему количеству систем (и, следовательно, меньшему количеству людей), что сокращает возможность возникновения внутренних угроз и тем самым уменьшает поверхность атаки.

Ко всему прочему, при разделении сотрудники будут реже подглядывать в личные данные. Представьте ваши ощущения, если бы вы узнали, что сотрудники компании, которая занималась разработкой используемого вами мессенджера, читают ваши сообщения. Это было бы нарушением конфиденциальности и, скорее всего, пользовательского соглашения. Разделение производственных данных устраняет большинство возможностей возникновения такого рода

угроз, а если подобная ситуация произойдет, то ее можно будет отследить.

## Защита исходного кода

Помимо материальной ценности, написанный код имеет другие виды стоимости (конкурентное преимущество, секретность государственного уровня и т. д.). Вполне вероятно, что организация хотела бы защитить свой код по причинам, связанным с интеллектуальной собственностью. Некоторые компании, наоборот, публикуют его в интернете для всеобщего обозрения (то есть делают исходный код открытым). В обоих решениях есть свои плюсы и минусы. Однако я придерживаюсь убеждения, что наиболее безопасным вариантом является отказ от публикации.

Многие спешат возразить, что «безопасность через неизвестность» – неэффективная тактика защиты, но я не согласна. Скрытие кода (то есть работа с закрытым, а не открытым исходным кодом) ни в коем случае не должно быть единственным средством обеспечения безопасности, но использовать этот подход в качестве одного из многих уровней защиты – разумное решение, если преимущества размещения кода в интернете не перевешивают защиту, которую предлагает закрытый исходный код. Многие компании не размещают свой код в открытых репозиториях, чтобы затруднить попытки конкурирующих компаний воспроизвести их продукты. Да, злоумышленник может попытаться провести реинжиниринг популярного программного обеспечения, но у кого есть столько времени?

Многие разработчики, выступающие за использование открытого кода, утверждают, что если код будет доступен для всеобщего обозрения, то люди станут быстрее и чаще замечать уязвимости в проектах. Однако цифры говорят о другом: исследования показали, что проприетарные (закрытые) приложения имеют в среднем такое же количество уязвимостей, как и проекты с открытым исходным кодом<sup>[38]</sup>. Лишь малая часть инженеров по безопасности проверяют открытый исходный код «просто ради интереса». Эта работа утомительна, требует высокого уровня профессионализма и отнимает много времени. При нынешнем дефиците квалифицированных специалистов<sup>[39]</sup> и очень высокой стоимости работы<sup>[40]</sup> найдется немногих желающих выполнять ее бесплатно. Таким образом, нехватка

людей не позволяет обеспечить безопасность всех проектов с открытым исходным кодом.

С другой стороны, если злоумышленник нацелился на конкретную компанию, использующую в своих продуктах открытый исходный код, значит, у него есть уйма времени, чтобы проверить его на наличие уязвимостей.

Закрытый исходный код является не пуленепробиваемой защитой, а просто одним из многих возможных ее уровней, которые можно использовать для обеспечения безопасности приложений и интеллектуальной собственности.

**ПРИМЕЧАНИЕ.** Исходный код представляет собой большую ценность, поэтому под защитой должны находиться не только его конфиденциальность, но и доступность и целостность. Необходимо обеспечить резервное копирование репозитория, в котором размещен код, с соблюдением тех же мер предосторожности, что и при резервном копировании данных, мониторинг и протоколирование репозитория, а также отслеживание всех пользовательских изменений с быстрым доступом к откату кода в случае необходимости.

## Моделирование угроз

К сожалению, при разработке программного обеспечения мы, как правило, сосредотачиваемся на том, чтобы оно выполняло все необходимые клиенту функции, а не на том, чтобы оно выполняло *только* эти функции. Именно здесь на помощь приходит моделирование угроз – процесс определения потенциальных рисков для организации и приложения, а также обеспечение надлежащих мер по снижению опасности их возникновения.

Моделирование угроз (ласково называемое «мозговым штурмом зла») в своей простейшей форме представляет собой мозговой штурм с целью выявления всех рисков, с которыми может столкнуться компания, приложение, система или продукт. Будут ли злоумышленники пытаться перехватить данные и продать их конкуренту? Какую стоимость будут иметь данные? Какой вред может быть нанесен, если это произойдет? Как можно защититься? Вот

некоторые из типов вопросов, которые могут быть заданы во время мозгового штурма. Последующие тестирование приложения и анализ его кода и дизайна покажут, правильно ли предотвращены угрозы и отвечают ли они требованиям безопасности проекта. Если в приложении обнаружится уязвимость, необходимо произвести в нем изменения, смягчающие угрозу, либо принять риск.

Принятие риска должно быть оформлено в письменном виде имеющими на это полномочия людьми (обычно руководством, руководителями высшего звена или заинтересованными сторонами проекта). В документе также должно быть четко обосновано решение о принятии риска.

**ПРИМЕЧАНИЕ.** Моделирование угроз – обширная тема, которой посвящено несколько книг. Здесь представлена лишь малая часть информации, так как цель данного раздела – рассказать в общем о существовании такого понятия, как моделирование угроз, и показать важность его использования при разработке новых приложений. Несмотря на то что моделирование угроз – достаточно новая процедура, ее проведение в любом случае лучше, чем отсутствие.

Для создания модели угрозы необходимо присутствие представителя от каждой группы заинтересованных сторон проекта: компании, заказчика, службы безопасности (человека с наступательным мышлением), а также технического архитектора или кого-то из оперативного управления и из команды разработчиков. Да, необходимо присутствие кого-то из технической команды – у него часто бывают самые пугающие идеи.

**СОВЕТ.** Если есть деньги, кто-то всегда хочет их украсть. Если есть информация в системе, всегда найдется тот, кто захочет узнать, изменить или удалить ее без разрешения. Если в приложении существуют различные уровни доступа, кто-то попытается эскалировать (повысить) свой доступ. Это те сценарии, которые всегда необходимо моделировать.

Собравшаяся группа обсуждает существующие для системы риски: «Что мешает спать по ночам?», «Если бы вы собирались атаковать

ваше приложение, как бы вы это сделали?», «О каких субъектах угроз мы должны знать? К чему следует готовиться?» и т. д. Необходимо взглянуть на систему с точки зрения злоумышленника: «Что может пойти не так?», «Как можно использовать систему не по назначению?», «Как обеспечить защиту пользователя (в том числе от нас)?», «Каков наихудший сценарий?» Эта сессия может быть невероятно формальной (например, создание деревьев атак) либо же вполне неформальной (именно так я бы посоветовала начинать данный процесс).

Отличным стартом будет превращение историй пользователей проекта в «истории злоупотреблений» или негативные примеры использования. Что произойдет, если приложение будет делать А вместо *предполагаемого* Б? Подборка таких историй даст вам много пищи для размышлений.

Существует несколько методов моделирования угроз, но наиболее популярными являются деревья атак<sup>[41]</sup>, STRIDE<sup>[42]</sup> и PASTA<sup>[43]</sup>. STRIDE фокусируется на вопросах, связанных с аутентификацией, авторизацией, триадой CIA и опровержением (то есть с возможностью доказать чьи-либо действия или бездействие). PASTA направлена на работу с бизнес-требованиями, пользовательскими историями, диаграммами данных и многим другим, что делает его немного более трудоемким. Деревья атак изображаются в виде «деревьев», где цель атаки (например, кража денег) находится на верхушке, а листья представляют собой возможные пути достижения этой цели (пример дерева атак приведен на рис. 3.6).



Рис. 3.6. Пример примитивного дерева атак на приложение для бега

Существенная часть всех методов помогает продумать важные вопросы и убедиться в наличии решений для всех возможных проблем. При разработке неформальной модели угроз необходимо обдумать, какие элементы приложения представляют ценность и как кто-то может украдь (деньги, данные и т. д.), манипулировать (смутить, запретить доступ, показать себя и т. д.), сломать и использовать их не по назначению. Записанные вопросы и ответы сформируют список проблем, еще не считающихся рисками.

**СОВЕТ.** В моем блоге ([wehackpurple.com](http://wehackpurple.com)) можно почитать о нескольких неформальных моделях угроз, разработанных мной для роботов, бессерверных систем, утечек данных при лабораторных испытаниях и т. д.

После составления списка проблем необходимо оценить каждый пункт с точки зрения вероятности возникновения, необходимости тестирования безопасности приложения (чтобы понять уровень угрозы) и важности: не все риски одинаковы. Те проблемы, которые кажутся невозможными или не принесут вреда компании или приложению, можно убирать из списка. Иногда в процессе мозгового штурма придумываются нереальные проблемы. Это нормально: они просто не войдут в окончательный список угроз.

Для всех остальных пунктов следует определить вероятность возникновения (высокая, средняя или низкая) и уровень ущерба, который они могут причинить. Теперь у вас есть список угроз (подтвержденный список опасений) и рисков (высокого, среднего или низкого уровней).

**ПРИМЕЧАНИЕ.** Можно использовать систему рейтинга Common Vulnerability Enumerator (CVE) либо 1–10 – все, что подходит команде и компании, при условии, что используется одна и та же система.

**ВНИМАНИЕ.** Обоснования опасности каждого риска могут удивить и даже испугать. Однажды мне пришлось сообщить одной компании, что возможный ущерб от угрозы будет «абсолютно катастрофическим». Я была уверена, что ее возникновение приведет к банкротству. Несмотря на то что сам риск был «довольно маловероятным», проектная команда немедленно изменила свой курс действий, как только поняла возможные последствия в долгосрочной перспективе.

Итак, вы составили список рисков и определили степень важности каждого из них. Затем следует составить план действий. Будете ли вы смягчать (исправлять либо устранять) некоторые из проблем? Будете ли вы управлять некоторыми рисками: регистрировать и отслеживать их, чтобы не пропустить увеличение серьезности? Примете ли вы какие-нибудь риски (документально подтвердив, что в отношении них не будут проводиться никакие действия)? Возможно, некоторые из них крайне маловероятны или представляют лишь незначительную угрозу? Зачем тратить средства на то, что не вызывает беспокойства? Скорее всего, принятые решения потребуют внесения дополнительных пунктов в список требований безопасности для проекта, что необходимо обсудить с представителями компании и с командой по управлению рисками.

**СОВЕТ.** Обсуждая риски со специалистами, не имеющими технических знаний, будьте очень внимательны и убедитесь, что при объяснении вы пользуетесь понятными терминами. Нельзя просто сказать: «Небо рушится». Формулировки должны быть

очень конкретными. Например: «Если кто-то сможет управлять роботом дистанционно и без разрешения, это может привести к авариям и повреждениям производимых нами автомобилей, стать причиной травмы или смерти сотрудника». Объяснять риски всегда нужно «на языке собеседника».

Весь процесс моделирования угроз должен быть задокументирован и подписан руководством, особенно окончательные решения, которые следует принимать совместно с руководством или с тем, кто имеет соответствующие полномочия. Разработчик программного обеспечения или вы – инженер по безопасности – не можете сами «принять риск». Вполне вероятно, что для принятия рисков, уровень опасности которых выше «низкого» или «среднего» уровня, потребуется руководитель высшего звена. Документ с подписью руководителя можно будет использовать в дальнейшем в случае возникновения инцидента безопасности или отказа других отделов выполнить запрошенную вами работу по смягчению проблем.

---

## **ИСТОРИЯ ОТ АВТОРА**

Однажды в процессе моделирования угроз я спросила двух разработчиков: «Если бы вы собирались взломать свое приложение, как бы вы это сделали?» Они посмотрели друг на друга, а затем один из них сказал: «Ну... мы написали один модуль администратора, чтобы управлять приложением из дома, может, это?» Модуль администратора не был включен ни в один проектный документ и оказался гигантской дырой в безопасности, о которой мы бы никогда не узнали, не будь этих программистов на встрече.

---

При итеративном проектировании может понадобиться провести несколько коротких сеансов моделирования угроз, когда появляются новые функции или производятся большие изменения существующих.

Для масштабного подхода в стиле Waterfall одного тщательного сеанса должно быть достаточно (при условии, что в дальнейшем не планируется больших изменений).

Инженеру по безопасности можно посоветовать пригласить на сеансы моделирования угроз специалистов по разработке и по информационно-технологическому обслуживанию (англ. Dev and Ops) и таким образом повышать их осведомленность о безопасности в будущем. К тому же обычно от них исходят самые хитрые идеи. Такие встречи помогут наладить отношения с другими людьми. Не заблуждайтесь: для качественного выполнения работы необходимы хорошие отношения.

Разработчику программного обеспечения, специалисту DevOps или оператору моделирование угроз откроет глаза на все потенциальные угрозы, которым может подвергаться приложение, и поднимет настроение. Проявлять свою креативность таким образом удивительно интересно, а участие в подобных встречах станет отличным познавательным опытом. Ко всему прочему, сеансы моделирования угроз могут побудить кого-нибудь присоединиться к отделу безопасности на более поздних этапах карьеры.

## Упражнения

- Когда следует шифровать данные? Выберите все подходящие варианты.
  - Когда один API отправляет данные другому API.
  - Когда данные расположены на выключеной виртуальной машине.
  - Когда данные хранятся в базе данных.
  - При отправке данных с сервера в браузер.
- Какими способами можно обеспечить безопасность используемых компонентов от сторонних производителей? Как минимизировать риск в этой области?
  - Где необходимо хранить секреты приложения? Каким образом приложение должно получать доступ к ним?
  - Назовите три типа «секретов».
  - С какими угрозами может столкнуться мобильное приложение банка? Назовите три угрозы и оцените вероятность их возникновения и степень опасности (низкая, средняя или высокая).
    - Назовите три возможные угрозы для «умного» автомобиля. Оцените вероятность их возникновения (низкая, средняя или высокая) и уровень потенциального ущерба.
    - Назовите пять различных типов функциональности для обеспечения безопасности, которые может предоставить современная платформа.

## Глава 4

# Безопасность кода ПО

Чтобы написать безопасный код, недостаточно просто помнить о безопасности. Она должна быть приоритетом для организации, официальной частью жизненного цикла разработки системы, а ее разработчикам должны быть предоставлены соответствующее обучение, время и ресурсы. В этой главе будет представлено руководство по безопасному кодированию, которое организация может использовать в качестве стандарта или рекомендации.

Очень важным условием использования информации из этой и предыдущих глав в работе является ее социализация. Проводите консультации, устраивайте обеды и тренинги, создавайте рекламные плакаты и инфографику, размещайте их на локальной веб-странице или в «Википедии», делайте рассылку по электронной почте и прежде всего отвечайте на все вопросы. Если вы хотите, чтобы люди начали создавать более безопасный продукт, необходимо обеспечить им всевозможную поддержку.

### Выбор используемой платформы и языка программирования

Довольно часто приходится работать в обстоятельствах, когда язык программирования и платформа уже выбраны за нас. Мы используем Dot Net Shop или Java Shop, и вряд ли в ближайшем будущем произойдут какие-нибудь радикальные изменения. Вместе с тем, возможно, у вас больше гибкости и влияния, чем вам кажется. Давайте посмотрим, что Алиса и Боб скажут по этому поводу.

Алиса не технарь, но она *является* руководителем высокого уровня. Когда ее IT-отдел объявил о желании постепенно перевести все свои приложения с Java Struts на Spring Framework for Java, она сразу же спросила: «Почему? Сколько это будет стоить?» Управление компании заботится о конечном результате, но IT-отдел был готов к подобным вопросам, ведь такую идею уже озвучили предыдущему

руководителю, и он ее не принял, потому что не получил внятных ответов.

Руководитель ИТ-отдела представил Алисе целую презентацию об окупаемости инвестиций (ROI, от англ. return on investment) в это решение и план безопасной миграции в течение 18 месяцев.

**ПРИМЕЧАНИЕ.** Struts и Spring – это две популярные платформы для языка программирования Java.

«Нулевым днем» называют найденную уязвимость в продукте или компоненте программного обеспечения, для которой в настоящее время нет доступного исправления от поставщика ПО.

Сначала ИТ-менеджер рассказал, что в 2016 году в фреймворке Struts были обнаружены три серьезные уязвимости нулевого дня различного характера<sup>[44]</sup>, которые очень дорого обошлись компании. Одна из них стала причиной крупного инцидента безопасности, который стоил более \$250 000 и сильно ударили по репутации. Решение данной проблемы потребовало большого количества времени: в течение нескольких недель ресурсы проекта перенаправлялись на наведение порядка в системе. Инцидент негативно повлиял и на клиентов, которые почти на неделю остались без доступа к сайту, пока команда по ликвидации последствий оценивала ущерб. Две другие уязвимости Struts не привели к нарушению системы или какому-либо другому типу инцидента безопасности. Тем не менее расследование и создание новых пользовательских сигнатур для брандмауэра веб-приложения (Web Application Firewall, сокращенно – WAF) обошлись в круглую сумму, а работу над другими проектами снова пришлось отложить. Согласно подсчетам, только в 2016 году Struts принес компании убытки в размере не менее миллиона долларов и подорвал доверие клиентов к бренду. ИТ-менеджер затруднился подсчитать стоимость репутационного ущерба.

Все три уязвимости были связаны именно с фреймворком Struts, а не с языком Java. После тщательного анализа других вариантов ИТ-отдел решил, что Spring лучше функционирует, более безопасен и современен, чем Struts, поэтому перевести на него 100 % своих приложений будет выгодно с финансовой точки зрения.

IT-менеджер изложил Алисе свой план: старые приложения будут переводиться на Spring вместе с добавлением новых функций, а время работы над обновлением увеличено на три недели. Все новые разработки будут вестись на Spring. Сам процесс начнется с привлечения инструктора, специализирующегося на новой платформе и безопасности, что обеспечит хороший старт. Также создадут стандарт безопасного кодирования для Spring и проведут проверку кода на соответствие ему.

План обойдется в \$550 000, но, по оценкам, окупится в первый год после внедрения за счет снижения количества инцидентов и событий, связанных с безопасностью. Данный переход представляет собой отличную инвестицию, которая позволит значительно сократить технический долг.

Что Алиса могла ответить на все эти исследования и наглядные аргументы? Она одобрила проект сразу же.

При выборе языка программирования или платформы нельзя опираться только на обеспечение безопасности. При этом я считаю, что служба безопасности всегда должна иметь возможность при необходимости наложить вето на сделанный выбор (с предоставлением альтернативных решений). Давайте вместе с нашим другом Бобом рассмотрим еще несколько примеров, связанных с языками программирования и платформами.

## Пример 1

За все время службы в канадском правительстве Боб успел поработать во многих ИТ-отделах. Некоторые из них были достаточно современными, а другие, казалось, отставали и постепенно накапливали все больше и больше технических долгов. Свою первую должность руководителя проекта Боб занял в месте, где использовался язык программирования, о котором он никогда не слышал. Даже в интернете он не нашел какой-либо внятной информации. В отделе также использовали платформу Net, и вся работа была разделена между двумя языками.

Когда появился проект по созданию нового программного продукта, Боб предположил, что они будут использовать новейшую версию платформы Net, а в качестве языка программирования – либо VB.net, либо C#. В разговоре с техническим руководителем проекта ему сообщили о намерении использовать странный древний язык программирования, так как «нужно обеспечить наших программистов работой». Ему сказали, что часть программистов не уверены в своей работе, потому что не знают Net-языка.

Боб был потрясен. Зачем намеренно создавать новые технические долги? Тот язык уже даже не поддерживался. Он поговорил с высшим руководством, и они пришли к компромиссу: чтобы избежать серьезных ошибок во время проекта, для программистов добавят обучение языку Net, и они будут работать в паре с теми, кто уже знает этот язык. Таким образом он обновил навыки своей команды и завершил проект без нового технического долга! Программисты были в восторге, узнав, что им оплатят обучение и наставничество на рабочем месте. На начальном этапе такое решение потребовало немного больше затрат, но в долгосрочной перспективе оно стало отличной инвестицией в своих сотрудников. Боб особенно гордился этим проектом.

## Пример 2

Управляя проектами в другом ИТ-отделе, Боб обнаружил, что все приложения используют разные версии платформы .Net. Некоторые были на версии 4.5, а другие – только на версии 2. Приходилось управлять таким количеством различных SDK, что специалисты по эксплуатации были очень недовольны разработчиками, из-за чего часто возникали конфликты при попытках развернуть два приложения на одном веб-сервере. Боб нанял консультанта, чтобы тот проверил безопасность одного из приложений. Тот просканировал весь репозиторий кода с помощью инструмента под названием SCA и обнаружил, что несколько старых версий .Net имеют критические уязвимости в системе безопасности. Служба безопасности была очень расстроена, узнав об этом. В итоге Боб создал сетку всех небезопасных версий платформы и запустил новый проект по обновлению до версии, в которой нет обнаруженных критических уязвимостей, а также разработал постоянный план по обеспечению того, чтобы они оставались в пределах трех незначительных вариаций текущей версии платформы. К концу проекта служба безопасности провозгласила Боба защитником безопасности и вручила небольшой сертификат, который он повесил у себя над столом. Он был так горд!

**СОВЕТ.** SDK расшифровывается как «комплект для разработки программного обеспечения» (от англ. software development kit), однако это название не является интуитивно понятным. SDK – это компоненты, которые должны быть установлены на локальной машине или на веб-сервере для правильной работы программ. Некоторые типы SDK могут иметь разные версии, установленные одновременно на одной машине, а некоторые – нет. Помните, что решение обновить одно из своих приложений до новейшей версии SDK может вызвать проблемы для других пользователей в офисе.

**ПРИМЕЧАНИЕ.** SCA расшифровывается как «анализ состава программного обеспечения» (от англ. Software Composition Analysis). Это процесс (обычно выполняемый автоматизированным инструментом) проверки всех включенных в приложение сторонних компонентов, библиотек или платформ

на отсутствие известных уязвимостей. С помощью SCA также отслеживается наличие лицензий. Подробнее об этом дальше.

### Пример 3

Боб недолгое время работал над сверхсекретным проектом, возглавляя команду, которая создавала инструмент для изучения вредоносных программ штатными аналитиками. Команда хотела написать его на языке C, а не Rust. Боб слышал о том, что Rust безопаснее для памяти, и поэтому твердо выступал за использование именно этого языка. Команда объяснила ему, что они, следуя принципу «Buy, Borrow, Build» (букв. «Покупай, заимствуй, строй»), используют инструмент, разработанный другим правительством, с которым у них доверительные отношения, и он уже написан на языке C. При переходе на Rust всю работу пришлось бы начинать с нуля. Поскольку данный инструмент будет использоваться только внутри компании и своими экспертами, он будет защищен от внешних злоумышленников. Последним аргументом в пользу C стало отсутствие опыта работы с Rust, из-за чего потребовалась бы организация соответствующего обучения. Взвесив все вышеперечисленные факторы, Боб согласился с тем, что следует продолжать использовать язык C, и оформил это решение как часть дизайна проекта.

## Языки программирования и платформы: правило

Необходимо выбирать для работы язык программирования и платформу, которые поддерживаются в течение длительного времени, и использовать только последнюю или предпоследнюю версии. Стоит ориентироваться на вариант, имеющий все функции по обеспечению безопасности, необходимые для разрабатываемого продукта. Нужно обходить стороной платформы, о которых известно, что они небезопасны, больше не поддерживаются или имеют другие проблемы.

**СОВЕТ.** Разумеется, никогда не следует выбирать платформу только потому, что она новая, крутая или интересная, однако стоит также подчеркнуть, что наличие долгосрочного видения стека технологий имеет решающее значение. Меньшее

количество технологий, платформ, языков, библиотек, компонентов, движущихся частей обеспечивает более легкий в обслуживании набор систем и, следовательно, меньший технический долг.

С помощью SCA выбранная версия проверяется на отсутствие серьезных известных уязвимостей. Этот инструмент можно использовать для еженедельного сканирования репозитория и конвейера при каждой сборке, чтобы убедиться, что в продукт случайно не попали известные уязвимости.

## Сомнительные данные

В предыдущих главах мы обсудили различные типы пользовательского ввода и причины постоянного недоверия к нему. Теперь рассмотрим варианты, которые можно использовать как план работы с каждым типом ввода.

1. Получаемые из любого источника данные проверяются на соответствие ожиданиям (тип, размер, формат, источник) и «правильность». Под правильностью подразумевается, что данные являются приемлемыми в рамках бизнес-контекста. Только владелец компании и другие заинтересованные стороны могут решать, является ли ввод данных правильным.

**А.** Проверка выполняется на стороне сервера, а не внутри JavaScript или иным образом на стороне клиента.

**В.** Если ввод не соответствует установленным требованиям, необходимо отклонить его и выдать сообщение об ошибке, четко объясняющее пользователю, что от него ожидается (а не то, что он сделал неправильно).

Пример: формат номера социального страхования: 999 999 999.

Ввод любых символов, кроме цифр, должен быть отклонен.

Ввод слишком большого количества цифр должен быть отклонен.

Ввод недостаточного количества цифр должен быть отклонен.

Ввод специальных символов должен быть отклонен и зарегистрирован в журнале как потенциальная проблема безопасности.

**С.** В тех редких случаях, когда необходимо принимать специальные символы, они должны быть экранированы. Следует соблюдать крайнюю осторожность и по возможности использовать функцию в собственной среде программирования или специальный компонент стороннего производителя, например OWASP Java Encoder ([github.com/OWASP/owasp-java-encoder](https://github.com/OWASP/owasp-java-encoder)). Писать свою функцию экранирования следует только в том случае, если в платформе она недоступна. При этом она должна быть тщательно и внимательно протестирована.

2. Если ввод вызывает какую-либо транзакцию (добавление, удаление, обновление, покупку и т. д.), следует удостовериться в том, что под его прикрытием не проводится CSRF-атака: необходимо

проверить, был ли передан соответствующий токен, была ли заполнена капча или была ли произведена повторная аутентификация пользователя другим способом.

**СОВЕТ.** Межсайтовая подделка запросов (CSRF, от англ. Cross-Site Request Forgery) – атака, направленная на активные сессии пользователей, – подробно описана в главе 5.

3. Если ввод будет выводиться на экран, необходимо выполнять кодирование вывода, что подробно описано в главе 3. По возможности следует убедиться, что применяется соответствующий контекст кодирования вывода.

4. Если ввод будет использоваться для обращения к базе данных, следует использовать параметризованные запросы (которые в MS SQL Server называются хранимыми процедурами). Его ни в коем случае нельзя связывать с другим текстом и отправлять в базу данных как одну команду (иногда называется «встроенным SQL»). Следует поместить входные данные в соответствующие параметры и затем вызвать хранимую процедуру. Такой подход нейтрализует большинство инъекционных атак, связанных с базами данных.

5. Если пользовательский ввод направляет приложение на другой сайт, это называется перенаправлением или переадресацией. Обычно перенаправление или переадресация на непроверенные сайты считается небезопасным действием. При проверке ссылки понадобится список предварительно одобренных ссылок. Маловероятно, что целью приложения является перенаправление пользователей на небезопасные ссылки, и поэтому следует запретить приложению их использовать.

Если впредь вы будете следовать представленной блок-схеме для всех входных данных в ваших приложениях, то устраните очень большой процент потенциальных уязвимостей, включая (но не ограничиваясь) инъекциями, XSS, CSRF, повышением привилегий и многими типами уязвимостей бизнес-логики. На рис. 4.1 показана блок-схема проверки сомнительных данных.

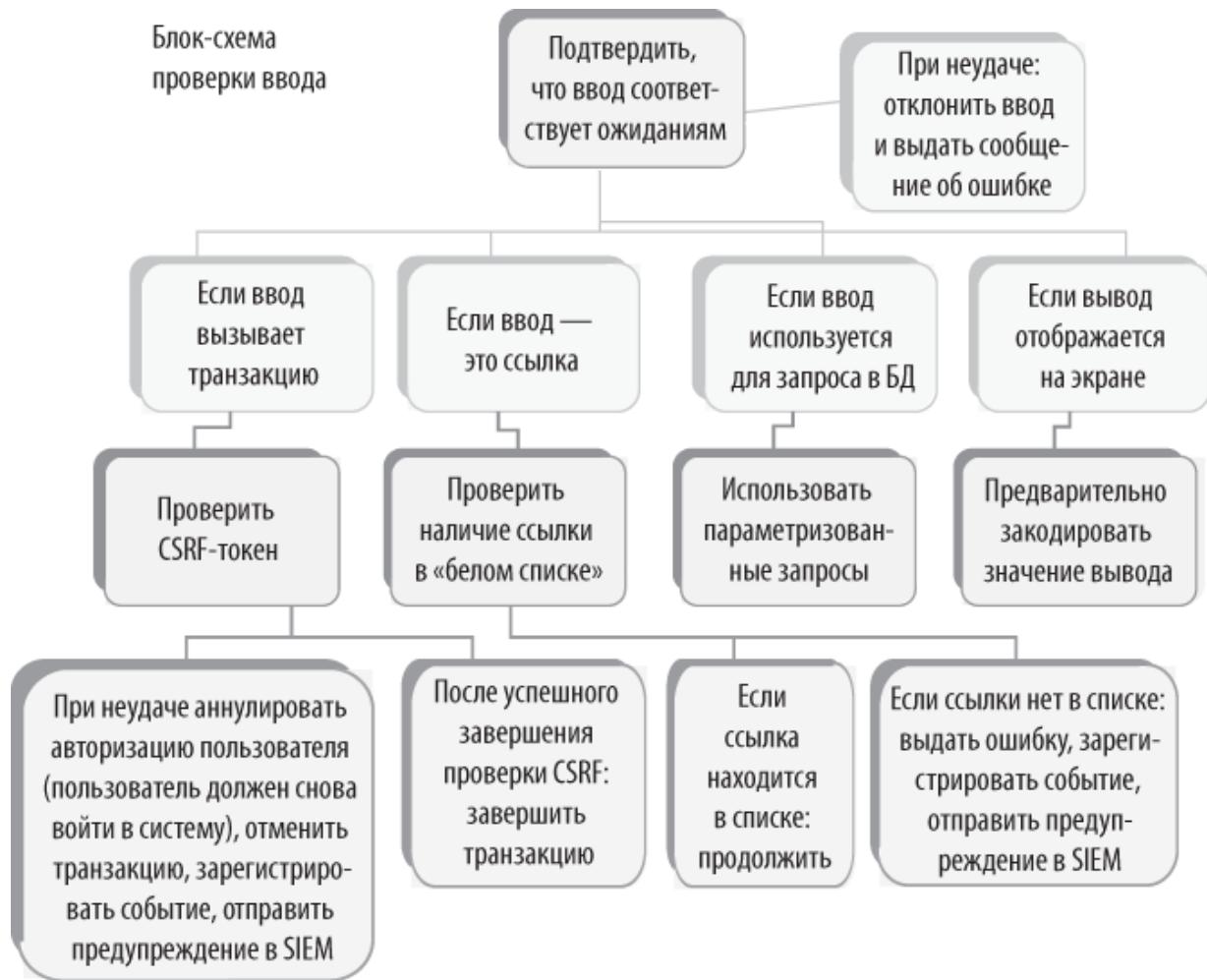


Рис. 4.1. Блок-схема проверки ввода для сомнительных данных

## HTTP-глаголы

Протокол HTTP – способ взаимодействия приложений через интернет – имеет несколько методов, которые обычно называют глаголами, поскольку большинство из них являются таковыми с лингвистической точки зрения. Они используются для выполнения действий на веб-сервере от имени приложения. Например, можно удалить что-то с помощью (как нетрудно догадаться) глагола `DELETE`. Глаголы применяются для общения с веб-приложением, расположенным на веб-сервере. Зачастую большинство разработчиков программного обеспечения, операторы и специалисты DevOps рассматривают только те глаголы, которые *они используют*, и

забывают о *неиспользуемых*, а потому оставляют все глаголы активными на веб-сервере, независимо от степени их необходимости. Если все глаголы включены, но не применяются, это означает, что они вряд ли были должным образом протестированы и нет гарантий, что они не делают того, чего не должны делать. Иногда злоумышленники используют неправильный глагол, пытаясь получить информацию или выполнить действия (атаки), направленные против приложения или веб-сервера. В идеале любой неиспользуемый HTTP-глагол должен быть отключен или заблокирован на веб-сервере.

Большинство веб-приложений используют только GET, POST, OPTIONS и HEAD, то есть остальные HTTP-глаголы не требуются. *Все неиспользуемые HTTP-глаголы должны быть отключены, чтобы уменьшить площадь атаки.*

**ВНИМАНИЕ.** WEBDAV расширяет HTTP для включения еще большего количества методов, *поэтому применять его в производстве следует, только если вы уверены в его необходимости и выполняете соответствующие инструкции по укреплению безопасности.* По возможности избегайте использования WEBDAV.

Ниже приведен пример отключения этих глаголов на Apache Tomcat Web Server. Обратите внимание, что это всего лишь пример. Внесите в него необходимые изменения, чтобы он правильно сработал в ваших приложениях. (Источник: [narayanatutorial.com/tutorial-blog/owasp/disable-dangerous-http-methods-apache-tomcat-server](http://narayanatutorial.com/tutorial-blog/owasp/disable-dangerous-http-methods-apache-tomcat-server).)

```
<security-constraint>
<web-resource-collection>
<web-resource-name>All JSP direct access</web-resource-name>
<url-pattern>*</url-pattern>
<http-method>PUT</http-method>
<http-method>DELETE</http-method>
<http-method>DEBUG</http-method>
<http-method>HEAD</http-method>
<http-method>CATS</http-method>
<http-method>JEFF</http-method>
<http-method>OPTIONS</http-method>
```

```
<http-method>TRACE</http-method>
<http-method>MKCOL</http-method>
<http-method>LOCK</http-method>
</web-resource-collection>
<auth-constraint>
<description>
No Access
</description>
</auth-constraint>
</security-constraint>
```

## Идентификация

Как уже говорилось ранее, под идентификацией в компьютерной системе или сети мы имеем в виду способ, которым компьютер распознает пользователя. Проверка компьютером подлинности личности называется аутентификацией (AuthN), а использование личности для проверки разрешения на просмотр и действия в системе – авторизацией (AuthZ). Система предоставления или запрета функциональных возможностей и информации в приложении или сети называется *управлением доступом*.

Если в системе есть пользователи, придется найти способ их идентифицировать. Если система находится внутри сети, *нет* необходимости самостоятельно писать функцию идентификации. Лучше использовать уже существующую (если только нет особых бизнес-требований). Наиболее распространенной сетевой системой идентификации является Active Directory компании Microsoft. Тем не менее некоторые провайдеры публичного облака предлагают собственные системы идентификации, а на рынке существует множество других систем, которые также могут выполнять подобные функции. Я не буду предлагать конкретную систему – необходимо выбрать ту, которая лучше всего подходит для вашей организации и ее технических требований. Любая из них будет значительно лучше написанной собственноручно (если только вы не занимаетесь именно этим бизнесом, то есть не являетесь компанией, предоставляющей услуги идентификации или облачного провайдера).

Про отличия каждой системы идентификации, ее плюсы и минусы легко можно было бы написать отдельную книгу. Цель данного раздела – донести мысль о том, что не следует создавать собственную систему идентификации. Всегда покупайте готовое решение, если только нет особых бизнес-требований, вынуждающих разрабатывать собственную систему: в этом случае можно использовать хорошо зарекомендовавший себя протокол OAuth.

## Управление сессиями

Как уже говорилось ранее, изначально функция управления сессиями не была заложена в дизайн HTTP-протокола и веб-страницы. Планировалось, что страницы будут статичными (неизменными): люди будут заходить на них, читать информацию и уходить. Однако, как всем известно, они стали совсем другими. Вход пользователей в приложение и переход между страницами называется сессией, и ее необходимо отслеживать. В этом разделе мы обсудим стратегии управления сессиями.

**СОВЕТ.** Помните правило всегда использовать функции платформы, а не писать свои? Оно применимо и в данном случае! Если в платформе есть функции управления сессиями, используйте именно их, а не пишите собственные с нуля.

Управление сессиями осуществляется путем передачи чего-либо (обычно называемого токенами сессии) между браузером и веб-сервером для подтверждения того, что пользователь и сессия не были изменены. Сессия отслеживается, и когда пользователь вошел в учетную запись, и когда не вошел. Однако, когда пользователь входит или выходит из учетной записи, предыдущая сессия уничтожается и создается новая. Сессия отслеживается посредством двусторонней передачи идентификатора сессии или токена сессии при каждом запросе (рис. 4.2).

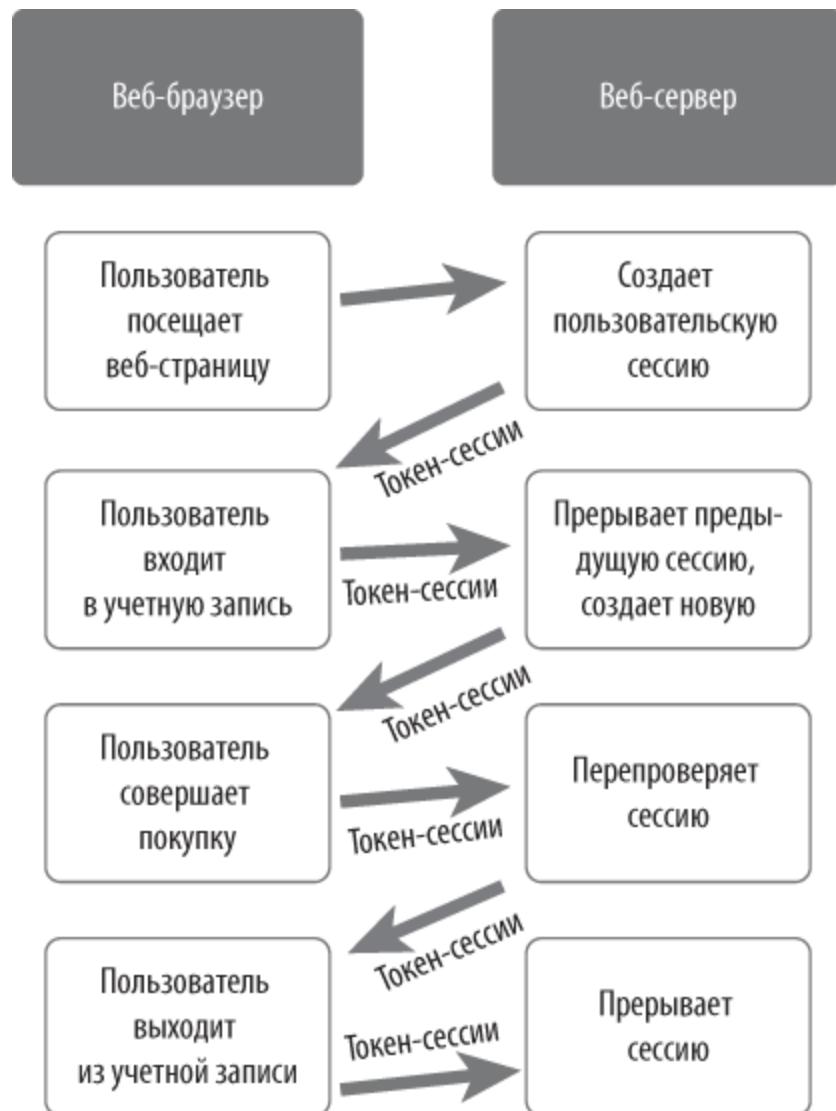


Рис. 4.2. Пример потока управления сессией

Идентификатор сессии и токен сессии используются как взаимозаменяемые понятия. Идентификаторы сессии передаются в защищенном файле cookie (с использованием настроек, описанных в главе 3) и никогда – в параметрах URL. Они предназначены только для управления сессиями, и использовать их для передачи сторонней информации (например, номера счета) в целях повышения эффективности не рекомендуется. Код бесплатен – просто пишите больше.

Ниже приведены дополнительные рекомендации по управлению сессиями от команды, разработавшей памятки OWASP<sup>[45]</sup>.

- Идентификаторы сессии должны иметь длину не менее 128 символов.
  - Идентификатор сессии должен быть непредсказуемым (рандомизированным) для предотвращения атак через угадывание. Используйте для этого общепризнанный генератор случайных чисел, а не пишите отдельный код. При каждом входе в учетную запись пользователь должен получать новый идентификатор сессии.
  - Используйте встроенную функцию управления сессиями в платформе, если такая функция существует.
  - Идентификатор сессии должен передаваться только по зашифрованным каналам.
  - Сессия должна быть прервана после выхода пользователя из системы.
  - Веб-приложения ни в коем случае не должны принимать не сгенерированный ими идентификатор сессии. Получение такого идентификатора обычно фиксируется и регистрируется как подозрительная активность, при этом создается соответствующее предупреждение, а IP-адрес, с которого он был отправлен, блокируется. *Данная ситуация является угрозой безопасности.*
  - При аутентификации, повторной аутентификации или любом другом событии, которое изменяет уровень привилегий связанного с ним пользователя, создается новый идентификатор.

**ПРИМЕЧАНИЕ.** Если после прочтения этой книги вы захотите получить больше информации о безопасности приложений, я советую присоединиться к местному отделению OWASP.

## Проверка границ

В 1996 году компания Aleph One опубликовала потрясшую мир программного обеспечения статью «Разбиение стека ради удовольствия и прибыли» (англ. Smashing the Stack for Fun and Profit) [\[46\]](#). В статье подробно описано, как пользователь может ввести чрезмерное количество данных в программное приложение, разработанное на языке C, перегрузить переменную в памяти, *перезаписав стек* своими данными. В ней объяснялось, как

злоумышленник может ввести shell-код (машиночитаемые компьютерные инструкции, не требующие компилятора) и таким образом получить контроль над компьютером, на котором было размещено приложение. Эта уязвимость стала возможной из-за того, что язык программирования С не является безопасным для памяти, то есть если вы объявили переменную типа `char` и отведете ей 20 символов пространства, а затем пользователь введет 30 символов, то язык не сможет обработать выход за установленные пределы. Вы будете сами по себе. Программа сохранит первые 20 символов в месте, выделенном для них в памяти при объявлении переменной, а остальные 10 символов будут записаны в следующие 10 мест в памяти, которые могут предназначаться для дальнейших инструкций программы или другой переменной либо быть неиспользуемым пространством. Независимо от того, что хранится в этих 10 местах в памяти, работа приложения будет некорректной. Продвинутый злоумышленник может воспользоваться такой ситуацией и внедрить собственный код для выполнения, то есть произвести удаленное выполнение кода (RCE, Remote Code Execution). Данная уязвимость перезаписи памяти для выполнения эксплойта называется *переполнением буфера* и считается критической уязвимостью безопасности системы.

Шло время, языки программирования развивались, и новые создавались уже с учетом требований безопасности для памяти (с обеспечением защиты от подобных ситуаций). Тем не менее многие небезопасные для памяти языки все еще используются в разработке приложений. Может возникнуть вопрос: почему?

Многие современные системы имеют гигантские масштабы, их структуры чрезвычайно сложны и хрупки. В таких случаях мы говорим о наличии технического долга, однако нельзя заранее судить о каждой ситуации без более подробной информации. Замена такой сложной системы может оказаться финансово невыгодной, а связанный с этим процессом риск будет довольно высоким, поэтому иногда организация решает сознательно «принять риск» сложившейся ситуации. Но порой компания не вполне осознает значимость такого риска. В таком случае необходимо, чтобы служба безопасности четко донесла необходимую информацию до высшего руководства (как в предыдущем примере с решением Алисы перевести все приложения на

другую платформу для Java). Каждая ситуация индивидуальна, и только сама компания может решить, какой путь правильный.

Тем не менее при работе с небезопасным для памяти языком есть несколько твердых правил, применяемых независимо от ситуации. Необходимо следующее.

1. Выполнять проверку границ на каждом вводе, тщательно и многократно тестировать эту функцию.
2. Использовать доступный в языке платформы оверлей или специальное приложение, которые можно использовать для тестирования границ.
3. Выполнять проверку типа на каждом вводе, а также тщательно тестировать эту функцию.
4. По возможности разработать модульные тесты для проверки границ и таким образом сформировать регрессивную систему тестирования, запускающуюся при каждой новой проверке кода.
5. По возможности проверять код на наличие надлежащего тестирования для каждого ввода.
6. По возможности нанять тестировщика на проникновение и попросить тщательно и скрупулезно проверить границы ввода системы.
7. По возможности добавить варианты компиляции для обнаружения проблем данного типа.

Более продвинутым, дополнительным уровнем защиты будет добавление защиты в реальном времени, например рандомизация компоновки адресного пространства (ASLR, Address Space Layout Randomization), предотвращение выполнения данных (DEP, Data Execution Prevention), защита Stack Canary и так далее. Эти средства защиты меняют способ использования памяти с целью предотвращения связанных с ней атак.

**ПРИМЕЧАНИЕ.** В качестве дополнительной меры предосторожности рассмотрите возможность реализации программы ответственного раскрытия информации на случай, если вы что-то упустили. Этот совет актуален не только для данной уязвимости, но и в целом для любой программы безопасности.

**СОВЕТ.** Примером безопасной для памяти альтернативы языкам С и С++ является Rust. Безопасными для памяти языками являются также Java, Net (VB и C#) и Ruby on Rails.

## Аутентификация (AuthN)

Аутентификацией называется проверка подлинности личности, то есть когда система подтверждает, что это та самая Алиса, а не другой человек с таким же именем или похожий на нее. Аутентификация и авторизация часто рассматриваются как одно действие, но следует отметить, что аутентификация всегда выполняется *в первую очередь*. Ее целью является подтверждение того, что мы знаем, с кем имеем дело, прежде чем предоставить доступ к чему-либо в нашей системе (AuthZ).

В главе 1 мы говорили о трех различных факторах аутентификации (то, что у вас есть, то, что является частью вас, и то, что вы знаете). Пришло время понять, как использовать их для проверки личности пользователей в приложении.

Существуют несколько вариантов.

1. Воспользоваться уже существующим интернет-сервисом от третьей стороны. Канадское налоговое агентство использует банковские идентификаторы для проверки канадцев при уплате налогов. Многие социальные сети в интернете позволяют пройти верификацию с помощью учетных данных из Twitter, LinkedIn, Facebook<sup>[47]</sup>, Google или Microsoft.

### Плюсы

- Очень быстрая реализация.
- Нет ответственности за сопровождение или тестирование кода.
- Команды, занимающиеся разработкой таких систем, скорее всего, имеют больше времени и ресурсов, чтобы обеспечить высокую безопасность.
- Некоторые пользователи могут больше доверять такому подходу или считать его более удобным.

### Минусы

- Некоторым пользователям не нравится, когда их данные передаются между компаниями, и они предпочитают входить в систему каждого сайта отдельно.
- Некоторые пользователи могут не иметь учетной записи ни на одном из перечисленных сторонних ресурсов.

- Использование такого сервиса может быть сопряжено с определенными расходами.

- Чаще всего такие сервисы собирают личную информацию и метаданные пользователей. Взлом сторонней системы повлияет на ваш сайт.

2. Приобрести или найти бесплатную библиотеку или программную систему, которая станет частью вашей системы и будет выполнять функции идентификации.

### **Плюсы**

- Реализуется быстрее, чем разработка собственной системы.
- Ответственность возлагается не за проектирование системы, а только за реализацию.
- Команды, занимающиеся разработкой таких систем, скорее всего, имеют больше времени и ресурсов, чтобы обеспечить высокую безопасность.
- Некоторые клиенты предпочитают размещать данные у вас.

### **Минусы**

- Использование такой системы может быть сопряжено с определенными расходами.
- Если в сторонней системе будет обнаружена уязвимость, это повлияет на ваш сайт. В результате возникнет необходимость обновить систему с помощью доступного патча.
- Размещение данных подразумевает ответственность за них.

3. Написать собственную систему аутентификации с нуля.

Как уже говорилось ранее, выбирать такой вариант обычно не рекомендуется. Компания должна будет разработать выполняющий эту функцию продукт. Единственный разработчик программного обеспечения в команде проекта всегда должен использовать уже существующую проверенную систему.

## **Авторизация (AuthZ)**

Под *авторизацией* подразумевается определение того, к чему пользователь может или не может получить доступ в системе, а *управление доступом* предоставляет (или запрещает) такой доступ.

Разрешено ли Алисе просматривать документы другой команды? Может ли Боб удалить другого пользователя из системы, находящейся под его управлением? На подобные вопросы отвечают авторизация и управление доступом.

Наиболее популярной методикой определения доступа является управление доступом на основе ролей (англ. Role Based Access Control, RBAC). Она подразумевает, что «чей-либо доступ определяется на основании назначенной ему в системе роли». Можно быть чьим-нибудь начальником на своем рабочем месте, но иметь обычный доступ к системе, в то время как подчиненный сотрудник может иметь права администратора, поскольку именно эту роль он выполняет в системе (то есть является администратором).

Давайте немного углубимся в эту тему на примере Алисы и Боба.

Когда Алису впервые повысили до уровня руководителя, она была очень рада получить больше привилегий и полномочий. В ее распоряжении были счет для расходов, право подписи на сумму \$50 000 и офис на верхнем этаже. Однажды Алиса обнаружила в системе управления документами компании каталог документов, доступ к которым ей был запрещен, что неприятно ее удивило. В ИТ-отделе ей объяснили, что в данном каталоге находится реестр рисков и доступ к нему имеет только служба ИТ-безопасности и начальник отдела информационной безопасности (англ. Chief Information Security Office, CISO). Доступа не было даже у генерального директора: информацию в сжатом виде она получала от CISO. Алисе пояснили также, что у нее нет *служебной необходимости* в этой информации, а в системе управления документами ее учетной записи не была назначена роль «Операции по обеспечению безопасности», поэтому ей не могут предоставить доступ. Для добавления этой роли необходимо получить разрешение CISO, так как она не входит в структуру службы безопасности. Алису устроил такой ответ. Она лишь хотела понять причину ограничения данной части системы для нее и попросила не заморачиваться с открытием доступа.

**ПРИМЕЧАНИЕ.** *Реестр рисков* – это документ, в котором указаны все известные риски для организации. Данная информация имеет высокий уровень конфиденциальности и, оказавшись в чужих руках, может нанести компании огромный

ущерб. Такой документ, в котором описаны все слабые места организации, может стать отличным оружием для любого злоумышленника.

**СОВЕТ.** *Служебная необходимость* определяет, какую информацию должен знать человек для того, чтобы выполнять свою работу, что в свою очередь является реализацией принципа наименьших привилегий в отношении информации.

В прошлом году Боб руководил проектом по созданию нового бухгалтерского программного обеспечения. В его команду входили Сара (тестировщик программного обеспечения), Эмили (разработчик программного обеспечения) и Дженифер (администратор базы данных, DBA). Все они играли разные роли в проекте и в своей компании, поэтому им требовались разные разрешения и доступы. Эмили не нужен доступ владельца базы данных (англ. Database Owner, DBO), однако ей определенно не обойтись без прав DBA. Бобу, как руководителю проекта, никогда не требовался доступ к веб-серверу или базе данных, поэтому он не запрашивал эти права в системе.

Когда нужно было определить, кому какой доступ предоставить, Боб попросил дать ему права обычного пользователя, чтобы он мог видеть работу приложения, а для своей команды он запросил следующее.

- **Для тестировщика программного обеспечения (Сары):** регулярный доступ к приложению, а также возможность запускать все инструменты автоматизации тестирования на своем компьютере. Ей также был необходим доступ на запись к программному обеспечению DevOps Pipeline, чтобы она могла добавлять тесты в конвейер.

- **Для разработчика программного обеспечения (Эмили):** доступ к приложению, доступ к среде разработки, возможность создавать и запускать хранимые процедуры в базе данных разработки, а также разрешение запускать DevOps Pipeline, но не вносить в него изменения.

- **Для администратора базы данных (Дженифер):** доступ к серверу базы данных на уровне владельца базы данных. Но она остается без доступа к веб-серверу, конвейеру и даже коду. Дженифер ничего из этого не нужно, поэтому соответствующий запрос не был сделан.

В начале проекта Эмили хотела иметь больше прав на сервере базы данных, веб-сервере и конвейере, ведь кто, будучи разработчиком программного обеспечения, не хотел бы иметь доступ ко всему? Однако ее роль в проекте стала причиной отказа.

Сотрудник службы безопасности объяснил, что предоставление доступа только к необходимым системам усилило защиту проекта, так как, если компьютер Эмили будет взломан, заражен вирусом-вымогателем или любым другим вредоносным ПО, злоумышленник будет ограничен ролью и доступом, которые она имела к различным системам. Эмили с неохотой, но согласилась с этим доводом.

В этой проектной ситуации RBAC чрезвычайно полезен. Системный администратор может легко назначить соответствующие роли каждому из членов проекта Боба и тем самым гарантировать, что им разрешен доступ только к тому, что необходимо для работы. Системный администратор просто зашел в свою систему каталогов, назначил каждому члену команды соответствующую роль и добавил их в ресурсы проекта.

Использование ролей вместо предоставления отдельного доступа для каждого человека вручную особенно полезно в ситуациях, когда кто-то меняет команду, работу или уходит из организации. Простое изменение роли и перенаправление группы (команды) внутри организации позволяет удалять и добавлять все необходимые доступы. Не рекомендуется управлять доступом вручную – это шаг к возникновению ошибок, которые могут привести к инцидентам безопасности.

Существуют три широко распространенные модели управления доступом<sup>[48]</sup>:

- избирательное управление доступом (англ. Discretionary Access Control, DAC);
- мандатное управление доступом (англ. Mandatory Access Control, MAC);
- управление доступом на основе разрешений (англ. Permission Based Access Control, PBAC).

*Избирательное управление доступом (DAC)* означает, что владелец информации, системы или ресурсов может предоставлять и удалять доступ по своему усмотрению другим лицам, использующим ту же

систему идентификации (например, пользователям одной сети, членам одного сайта по обмену фотографиями и т. д.).

После рождения первого внука Боб сделал сотни фотографий, которые позже загрузил в свое облачное онлайн-хранилище. В тот момент система давала ему возможность предоставить определенным пользователям системы доступ к его фотографиям. Боб предоставил доступ к фотографиям всем своим родственникам, добавляя их по одному через имена пользователей (адреса электронной почты). Родственники Боба начали использовать хранилище, чтобы делиться друг с другом семейными фотографиями. Несколько лет спустя кто-то из родственников развелся, и Боб вспомнил, что он предоставил каждому доступ на чтение и запись к своей папке. Он принял решение лишить доступа бывшего члена семьи, просто на всякий случай.

В отличие от управления доступом на основе ролей на работе Боба, когда кто-то меняет роль, системы DAC не обновляются. Они основаны на идентификации в системе, а не роли.

*Мандатное управление доступом* (MAC) предоставляет доступ к информации и системам на основе уровня чувствительности ресурса и одобрения пользователя, пытающегося получить к нему доступ.

Например, у Алисы нет сверхсекретного доступа, а у Боба есть, поэтому в такой системе Боб сможет получить доступ ко всему, вплоть до сверхсекретных систем и данных. Алисе же такой доступ не предоставляется. У нее есть только публичный доступ, то есть она может получить доступ только к публичным записям.

MAC обычно используется для систем, требующих чрезвычайно высокого уровня обеспечения безопасности. Например, для правительственные военных систем или наборов данных.

*Управление доступом на основе разрешений* (PBAC) основано на разрешениях. Пользователям предоставляются определенные разрешения на информацию и использование системы. Когда пользователь пытается получить доступ к чему-либо в системе, механизм управления доступом проверяет, было ли ему предоставлено разрешение. Типами разрешений могут выступать READ, WRITE, CREATE, UPDATE, DELETE, PRINT, REBOOT и т. д.

После развода родственников Боб решил изменить систему управления доступом к своим фотографиям. Он предоставил всем READ и WRITE, но только у него были права DELETE и UPDATE.

## **Обработка, регистрация и мониторинг ошибок**

Все ошибки должны быть отловлены и корректно обработаны. На экране не должны отображаться трассировки стека или ошибки базы данных. Это необходимо не только для того, чтобы приложение выглядело профессионально, но и для того, чтобы злоумышленники не получили дополнительную информацию, которой они могли бы воспользоваться для совершения атак.

---

### **РЕЗЕРВНЫЕ КОПИИ И ОТКАТЫ**

Боб помнит, как много лет назад ему позвонил пользователь одного из приложений, разработанных его командой. Веб-приложение перестало работать, а на экране появилась трассировка стека, и пользователь подумал, что весь его компьютер сломался. Команде Боба пришлось вручную откатывать транзакцию, что заняло целую вечность. В процессе работы выяснилось, что таких сбоев было несколько, и в программном обеспечении выявили серьезный недостаток. Эту проблему можно было заметить раньше, если бы ошибки фиксировались, а информация о них отправлялась в SIEM. В результате было принято решение добавить требования по регистрации, мониторингу и оповещению об ошибках во все новые проекты программного обеспечения.

---

При возникновении ошибок приложение ни в коем случае не должно переходить в неизвестное состояние. Оно должно откатить все выполненные операции и «закрыть» все, что было открыто. Такая ситуация всегда регистрируется, чтобы в случае необходимости специалистам по реагированию на инциденты было с чем работать при расследовании, а аудиторы могли убедиться в корректной работоспособности системы. Если ошибка необъяснима или является

потенциально вредоносной, следует также запустить предупреждение системы безопасности.

Под «переходом в неизвестное состояние» имеется в виду ситуация, когда исключение (ошибка программы) не отловлено и программе некуда двигаться дальше или она не знает, какие действия предпринять. Когда появляется ошибка, мы «ловим» ее в блоке `try/catch`, затем «обрабатываем».

Возможные варианты обработки включают выдачу сообщения об ошибке, запись ошибки в журнал, выдачу предупреждения, откат транзакции или выход пользователя из системы.

Каждое приложение должно располагать всеобщим или глобальным обработчиком ошибок, который будет отлавливать все, что не было обработано (отловлено) другим способом.

**СОВЕТ.** Когда приложение переходит в неизвестное состояние, там сразу же возникает раздолье для злоумышленников. Целью тестировщиков на проникновение и исследователей безопасности всегда являются такие ситуации, поскольку именно в них кроется множество уязвимостей. Очень важно, чтобы приложение никогда не переходило в такое состояние.

## Правила работы с ошибками

Ниже приведен список правил обработки ошибок. Следование этим правилам поможет избежать перехода приложения в неизвестное состояние.

- Все ошибки приложения должны быть отловлены и обработаны. Их никогда нельзя игнорировать.
- Крайне желательно иметь всеохватывающий механизм отлова (глобальная обработка исключений) для корректной обработки внезапно возникших ошибок.
- Внутренняя информация, трассировка стека или другая информация о сбоях никогда не должны быть открыты пользователю или потенциальным злоумышленникам.
- Сообщения об ошибках должны раскрывать как можно меньше сведений. Убедитесь, что они не «сливают» информацию, например, о версии сервера или уровнях исправлений.

- Нельзя раскрывать, является ли имя пользователя или пароль неправильным, если произошла ошибка входа, так как это позволяет перебирать имена пользователя.
- Приложение должно всегда корректно производить аварийное завершение работы с закрытием всего, что было открыто, и никогда не переходить в неизвестное состояние. При возникновении ошибки приложение должно запретить доступ и не завершать транзакцию, а выполнить откат.
- Система должна предупреждать об ошибках, связанных с безопасностью (ошибках входа в систему, ошибках управления доступом, ошибках проверки ввода на стороне сервера). Идеально, когда файлы регистрации поступают в систему предотвращения или обнаружения вторжений либо в SIEM-приложение. Этую функцию можно проверить, запустив сканер уязвимостей в приложении: будут вызваны события, которые должны быть записаны в журнал.
- При регистрации ошибки с использованием информации, поступающей из внешнего источника (из внешней системы либо ввода пользователя, а не из самого приложения), необходимо защитить журналы от атак путем [\[49\]](#):
  - кодирования содержимого из внешнего источника;
  - удаления всех непечатаемых знаков (CR, LF, TAB и т. д.);
  - ограничения максимальной длины регистрируемой информации с целью предотвращения атаки переполнения [\[50\]](#).

**СОВЕТ.** Да, именно так! Злоумышленники будут атаковать даже журналы. Для них нет ничего святого. Подобная форма атаки обычно называется *инъекцией журнала* или *подделкой журнала*.

## Регистрация

Регистрация – это запись вашим приложением подробной информации о происходящих в нем событиях в очень, очень длинный документ (известный как лог-файл или файл регистрации) [\[51\]](#). Почти все содержимое в нем будет очень скучным, и именно поэтому большинство людей не сидят и не читают эти файлы напрямую. «Зарегистрирована запись в базе данных в такое-то время для такого-

то пользователя» и «Такой-то пользователь успешно вошел в систему в такое-то время» – не самые интересные истории. Размеры файлов регистрации обычно исчисляются гигабайтами и терабайтами, так что человек физически не может просмотреть их полностью. Именно поэтому были изобретены инструменты для чтения журналов: SIEM и regex.

Такие программы открывают файлы огромных размеров и дают возможность выполнять в них поиск. Операционная система Unix позволяет использовать команду grep для осуществления поиска в файлах регистрации с помощью regex (регулярных выражений), что полезно, когда происходит инцидент и необходимо найти определенный временной интервал или конкретное событие, однако не подходит для повседневного использования.

Система управления информацией и событиями безопасности (англ. Security information and event management, SIEM) – это программное обеспечение, которое «проглатывает» все журналы, а потом старается представить их в понятной для человека форме и предупредить инженеров по безопасности из операционного центра безопасности (англ. Security Operations Center, SOC) обо всем, что, по его мнению, требует их внимания. Такая система используется в большинстве крупных компаний. Для работы программы необходимо, чтобы все файлы регистрации имели формат, доступный для чтения SIEM.

## Мониторинг

Мониторинг обозначает как само внимание к тому, что происходит в системах, так и использование ИТ-инструмента, автоматизирующего процесс проверки на наличие сбоев, замедлений, аварий и любых аномалий, которые он считает потенциально опасными или проблематичными. В этой книге под «мониторингом» подразумевается использование инструмента мониторинга.

Приложения должны проходить мониторинг так же, как и сеть. Когда приложение выходит из строя, ожидается, что сообщение об этом появится мгновенно. Никому не хочется узнать о такой серьезной проблеме из социальных сетей. Разумеется, по возможности вы постараетесь сыграть на опережение и разработать автоматические ответы на определенные предупреждения. Это можно сделать с помощью триггеров и бессерверных приложений (о последних мы поговорим позже).

**СОВЕТ.** Обработка, регистрация и мониторинг ошибок неразделимы. При отсутствии регистрации мониторинг не будет работать. Без отлова ошибок нельзя говорить о полноценных журналах регистрации. Очень важно периодически тестировать корректность работы этих систем.

### Что и когда регистрировать

- Системные файлы регистрации не должны содержать конфиденциальные данные или персонально идентифицируемую информацию (ПИ).
- Необходимо регистрировать все результаты входа в систему: как неудачи и ошибки, так и успех.
- Необходимо регистрировать попытки входа, произведенные методом грубой силы (в данной книге они определяются как 10 или более последовательных неудачных попыток входа в систему менее чем за минуту или 100 или более неудачных попыток в течение часа).
- Необходимо регистрировать все события, связанные с безопасностью: аутентификацию пользователя, блокировку

пользователя после нескольких неудачных попыток входа, неучтеннюю ошибку или ошибки проверки ввода.

- Файлы регистрации должны быть доступны для SIEM и автоматически передаваться в обработку.

**СОВЕТ.** Продвинутый злоумышленник попытается удалить информацию о своих действиях из файлов регистрации, поэтому необходимо обеспечить соответствующую защиту.

В файлах регистрации должна содержаться следующая информация.

- Событие какого типа произошло (почему это событие связано с безопасностью либо имя события).
- Когда произошло событие (временная метка).
- Где произошло событие (URL, включая субдомены).
- Источник события (IP-адрес).
- Если IP приходит из заголовка X-Forwarded-For, не забудьте тщательно его проверить. Он мог быть подделан<sup>[52]</sup>.
- Результат события.
- (По возможности) личности любых лиц, пользователей или субъектов, связанных с событием.

Конечно, можно регистрировать больше информации о событии, чем указано здесь.

**ВНИМАНИЕ.** При определении того, являются ли данные конфиденциальными или персонально идентифицируемой информацией (ПИ), убедитесь, что вы рассматриваете их в совокупности, а не просто поле за полем, так как иногда ПИ становится комбинация из нескольких значений данных, собранных вместе.

### **Использование и защита файлов регистрации**

- Файлы регистрации и журналы аудита должны быть защищены, даже их резервные копии.
- В идеале все файлы регистрации следует хранить в одном месте, в одном формате, чтобы их можно было легко обработать с помощью SIEM или других инструментов безопасности.
- Ни один посторонний человек не должен иметь доступ к файлам регистрации.

- Любой доступ к файлам регистрации (если кто-то открывает их, редактирует и т. д.) также должен регистрироваться в сетевых файлах регистрации.
- Файлы регистрации должны проходить мониторинг.
- Файлы регистрации должны храниться в зашифрованном формате.
- Файлы регистрации должны быть доступны команде реагирования на инциденты, которая также обязана проверить историю их ведения, чтобы убедиться в том, что при расследовании инцидента специалист получит корректные данные.
- Файлы регистрации должны храниться на защищенном сервере или в другом безопасном месте.
- Файлы регистрации должны быть включены в общую стратегию резервного копирования данных организации.
- Файлы регистрации и их носители должны быть удалены и утилизированы так же, как и любая конфиденциальная информация.

**ВНИМАНИЕ.** Никогда не регистрируйте РИ и другие конфиденциальные данные, такие как SIN, пароли или даты рождения. РИ – это информация, относящаяся к идентифицируемому лицу<sup>[53]</sup>.

Для получения более подробной информации по этому и многим другим вопросам, связанным с безопасностью приложений, ознакомьтесь с чрезвычайно полезным проектом «памятки OWASP».

## Упражнения

- Когда следует использовать собственную идентификацию в сети (пользовательскую учетную запись), а когда – сервисную учетную запись? Приведите два примера для каждого из вариантов и поясните.
- Объясните возможные причины, по которым языки С и С++ все еще широко используются в индустрии разработки, когда существует Rust (язык, безопасный для памяти). Постарайтесь дать два или более примера, опишите соответствующие ситуации.
- Какой ваш любимый язык программирования или платформа? Почему?

- Какой язык программирования или платформа, по вашему мнению, являются наиболее безопасными? Почему?
- Почему необходимо обеспечивать защиту сессии пользователей?
- Что может сделать злоумышленник, если ему удастся завладеть чужой пользовательской сессией во время входа в систему интернет-банкинга?
- Как бы вы объяснили коллеге, не имеющему технических знаний, разницу между аутентификацией и авторизацией?
- Должны ли руководители высшего звена иметь особые привилегии в сети и других компьютерных системах? Если да, то почему? Если нет, то почему? Какие права вы бы им предоставили?
- Должны ли администраторы сетевой системы иметь особые привилегии в сети и других компьютерных системах? Если да, то почему? Если нет, то почему? Какие права вы бы им предоставили?
- Должны ли сотрудники техподдержки иметь особые привилегии в сети и других компьютерных системах? Если да, то почему? Если нет, то почему? Какие права вы бы им предоставили?
- Ваш начальник говорит, что регистрация и мониторинг будут стоить слишком дорого. Как вы объясните их ценность и важность с точки зрения обеспечения безопасности? Изложите свои доводы в одном абзаце. Не забудьте убедиться в том, что описываете возможные риски для бизнеса понятным для начальника языком (это умный, но не слишком технически подкованный человек). Если вы сформулируете свои мысли чересчур сложно, то не убедите начальника и провалите задание.

## Глава 5

# Часто встречающиеся подводные камни

В этой главе мы рассмотрим часто встречающиеся подводные камни, о которых еще не рассказывалось в предыдущих главах. Я убеждена в том, что учиться защищать программное обеспечение путем запоминания множества типов уязвимостей или статистики утечки данных – не самое эффективное использование времени. Лучше направить силы на обучение защите от всех проблем.

В главах 1–4 мы уже разобрались в различных средствах защиты, и все же некоторым отдельным ситуациям стоит уделить особое внимание. Речь пойдет о распространенных и вредоносных уязвимостях, специфические средства защиты против которых мы еще не изучали.

## OWASP

Открытый проект обеспечения безопасности веб-приложений, более известный как OWASP (от англ. Open Web Application Security Project), – это глобальное сообщество, оказывающее помочь всем желающим в создании более безопасного программного обеспечения. Официальным сайтом сообщества является [OWASP.org](http://OWASP.org). В 300 открытых отделениях по всему миру ежемесячно проходят бесплатные мероприятия по обучению защите приложений и другим полезным темам. Ежегодно OWASP организовывает несколько международных и региональных конференций, уникальные обучающие курсы по углубленной тематике, саммиты по проектам. Благодаря активистам сообщества было разработано более 100 проектов с открытым исходным кодом, которые помогают продвигать индустрию безопасности приложений. Они же стали создателями самого используемого прокси-сервера для веб-приложений и сканера безопасности Zed Attack Proxy, а также ряда других инструментов и документов, в том числе собственных бесплатных книг. OWASP – открытое сообщество, к которому может присоединиться любой желающий. Несмотря на весь упомянутый здесь поразительный вклад

в индустрию, наибольшую известность ему принес проект OWASP Топ-10.

OWASP Топ-10 представляет собой список десяти наиболее критических уязвимостей веб-приложений, который обновляется каждые несколько лет. В нем подробно описывается каждая уязвимость, связанные с ней риски и способы ее устранения. Он не позиционируется как какой-то стандарт и окончательный перечень всех уязвимостей веб-приложений. Топ-10 был создан командой добровольцев на основе их профессионального опыта и данных, предоставленных всей индустрией, для поиска наиболее опасных и наиболее распространенных уязвимостей веб-приложений. Это очень полезный документ для начинающего тестировщика на проникновение (так как он часто будет находить эти проблемы), для программиста (чтобы научиться защищаться от этих уязвимостей). Он также может помочь повысить осведомленность об обеспечении безопасности веб-приложений на работе. OWASP Топ-10 весьма информативен. Одни уязвимости можно найти с помощью автоматизированных инструментов, а для других требуется человек. Некоторые из пунктов списка представляют собой конкретную проблему (например, XSS), в то время как другие являются скорее системной проблемой или проблемой проектирования (например, «недостаточное или неправильное протоколирование»).

**ВНИМАНИЕ.** Несмотря на то что OWASP Топ-10 – очень полезный инструмент для изучения, его никогда не следует использовать в качестве контрольного списка для обеспечения безопасности приложения. Конечно, у приложений не должно быть ни одной из этих проблем, но гораздо важнее сосредоточиться на хорошей общей стратегии защиты, а не стремиться только к защите от конкретных проблем или проверке на них. Другой проект, OWASP Proactive Controls, перечисляет средства контроля, используемые для защиты от вошедших в Топ-10 уязвимостей, и многое другое. OWASP Proactive Controls – это отличный ресурс от OWASP для того, чтобы начать изучение стратегий защиты безопасности приложений.

Поскольку OWASP Топ-10 представляет собой список наиболее опасных и наиболее распространенных уязвимостей, а цель данной книги – помочь в обучении тому, как создавать безопасное программное обеспечение, мы рассмотрим этот топ лишь вкратце. После перечисления мы остановимся на трех пунктах списка, которые еще не были рассмотрены в книге, а также затронем дополнительные уязвимости, не вошедшие в Топ-10.

**СОВЕТ.** Существует огромное количество уязвимостей, которые не входят во всемирно известный список OWASP Топ-10. Не останавливайтесь на достигнутом. Это только начало пути.

Вот список версии 2017 года с моим кратким пояснением каждого пункта: [owasp.org/www-project-top-ten/OWASP\\_Top\\_Ten\\_2017](http://owasp.org/www-project-top-ten/OWASP_Top_Ten_2017).

<b>1. Инъекционные атаки</b>	
Определение	Атака обманом заставляет систему выполнить код злоумышленника вместо своего собственного
Смягчение последствий	Проверка ввода, использование параметризованных запросов при обращении к базе данных
Действие	Требуется проверка кода (SAST) и тестирование (DAST или IAST). Применение принципа наименьших привилегий ко всем учетным записям и системам для минимизации ущерба в случае успешной атаки
<b>2. Нарушенная аутентификация</b>	
Определение	Ошибки в реализации управления сессиями или аутентификации (AuthN), позволяющие получить ненадлежащий доступ или привилегии к системам, данным или сессиям пользователей
Смягчение последствий	Следование лучшим практикам, описанным в главе 4 в разделах «Управление сессиями» и «Аутентификация (AuthN)»
Действие	Требуется проверка кода (SAST) и тестирование (DAST и/или IAST)
<b>3. Раскрытие критически важных данных</b>	
Определение	Раскрытие или утечка чувствительных или конфиденциальных данных, которые не должны были попасть к пользователям или системам. Причина — нарушение защиты, неправильная маркировка или другое неаккуратное обращение с данными
Смягчение последствий	Следование лучшим практикам, описанным в главе 3 в разделе «Защита конфиденциальных данных»
Действие	Необходима проверка кода (SAST) и тестирование (DAST или IAST). Организация может потребовать проведения аудита, который проверит действия на соответствие применимым нормам, таким как GDPR (Европейский Союз) или ISTG (Канада)



<b>4. Внешние объекты XML (XXE)</b>	
Определение	Некоторые XML-процессоры позволяют загружать объекты, определенные вне документа (то есть внешние объекты). Злоумышленник, воспользовавшись этим, может дать указание уязвимому XML-процессору через собственный созданный вредоносный XML-файл отправить запросы (например, HTTP/FTP/...) к внутренним или внешним системам для выполнения определенных действий или чтения данных. По сути это неправильная конфигурация XML-процессоров, но проблема настолько распространена и опасна, что она была выделена в отдельный пункт
Смягчение последствий	Не использовать XML (использовать JSON или что-то другое, более современное). Если это невозможно, необходимо отключить эти функции на всех XML-процессорах (внешние объекты XML и DTD-обработка)
Действие	Проведите аудит всех XML-парсеров на предмет наличия этой проблемы. Специальный набор модульных тестов безопасности позволяет непрерывно проверять правильность настройки любого инстанцированного XML-парсера. Ручное тестирование, направленное на поиск этой проблемы. Более подробная информация будет представлена в разделе «Десериализация» в этой главе и в главе 6 «Тестирование и развертывание»
<b>5. Нарушение контроля доступа</b>	
Определение	Ошибки в реализации контроля доступа или авторизации (AuthZ), позволяющие получить ненадлежащий доступ или привилегии к системам или данным
Смягчение последствий	Следование лучшим практикам, описанным в главе 4 в разделах «Идентификация» и «Авторизация»
Действие	Требуется проверка кода (SAST) и тестирование (DAST или IAST)
<b>6. Неправильная конфигурация безопасности</b>	
Определение	Любая система, которая сконфигурирована таким образом, что это приводит к уязвимостям в безопасности



Смягчение последствий	Следование руководству по усилению защиты для каждого продукта в сети. Реализация стратегии «защита в глубину» для минимизации ущерба, если что-то будет упущено. По возможности для внедрения и/или аудита наймите специалиста с большим опытом работы с каждой из систем
Действие	Необходимы аудит конфигурации и управление изменениями (желательно самодокументирование), а также тестирование (DAST или IAST)

## 7. Межсайтовый скрипting (XSS)

Определение	Обман приложения, результатом которого становится запуск JavaScript злоумышленника в браузере пользователя
Смягчение последствий	Проверка ввода, кодирование вывода, политика безопасности содержимого (CSP) + заголовки безопасности x-xss-protection (главы 2, 3 и 4)
Действие	Требуется проверка кода (SAST) и тестирование (DAST или IAST)

## 8. Небезопасная десериализация

Определение	Злоумышленник заменяет сериализованный объект собственным вредоносным сериализованным объектом. Объект десериализуется во время выполнения атаки
Смягчение последствий	По возможности необходимо избегать использования сериализации. Если это неизбежно, не принимать объекты из неизвестных источников или использовать только примитивные (неизменяемые или базовые) классы данных из белого списка
Действие	Список необходимых действий длинный. Он будет подробно рассмотрен в разделе «Десериализация» данной главы. Необходимым минимумом является проверка кода (SAST) и тестирование (DAST или IAST)

## 9. Использование компонентов с известными уязвимостями

Определение	Включение в приложение библиотеки, зависимости или компонента, содержащего известную уязвимость безопасности, которая потенциально создает ту же самую уязвимость в приложении
-------------	--

Смягчение последствий	Как будет обсуждаться в главе 6 «Анализ состава программного обеспечения (SCA)», необходимо использовать инструмент анализа состава программного обеспечения (SCA), чтобы проверить компоненты на наличие известных уязвимостей
Действие	Необходимо использовать SCA в репозиториях кода и в конвейерах либо процессах сборки
<b>10. Недостаточно подробная регистрация и слабый мониторинг</b>	
Определение	Атаки и другие события или инциденты безопасности не обнаруживаются, не предотвращаются и не документируются из-за неверной конфигурации или отсутствия систем регистрации либо мониторинга
Смягчения последствий	Следование лучшим практикам, описанным в главе 4, в разделе «Обработка, регистрация и мониторинг ошибок»
Действие	Необходимо проводить аудит и тестирование этих систем на правильное и своевременное обнаружение события

## Ранее не упомянутые средства защиты и уязвимости

Далее следуют конкретные меры защиты от распространенных уязвимостей, которые еще не были упомянуты в этой книге, а именно:

- CSRF;
- SSRF;
- десериализация;
- состояние гонки.

### Межсайтовая подделка запроса (CSRF)

Ранее в книге мы вкратце рассказали о межсайтовой подделке запроса (CSRF), ласково называемой «си сёрф», однако данная

проблема важна и, к сожалению, все еще довольно распространена, поэтому необходимо рассмотреть ее более подробно. CSRF недавно была исключена из Топ-10 после того, как непосредственно в новые версии нескольких программных фреймворков была встроены автоматические средства защиты от нее. ИТ-индустрия скорректировала некоторые фреймворки, чтобы помочь устранить эту уязвимость.

**СОВЕТ.** Встраивание средств защиты непосредственно в код фреймворка – это самый лучший способ борьбы с небезопасным программным обеспечением. Разработчикам и так приходится балансировать между довольно большим количеством ответственности и знаний для того, чтобы хорошо выполнять свою работу, поэтому чем больше мы берем на себя, тем лучше будут результаты.

В отличие от XSS, CSRF обманом заставляет пользователя перейти по ссылке, которая выполняет запрос либо действия атакующего, направленные против веб-приложения. При этом против приложения используется его собственный код, а не код злоумышленника. Данная атака называется серфингом, потому что злоумышленник использует учетные данные жертв.

Тем не менее иногда CSRF путают с межсайтовым скрипtingом (XSS), потому что в обоих названиях присутствует слово «межсайтовый». Для полного понимания обсудим различия и сходства двух атак.

При XSS-атаке злоумышленник обманом заставляет веб-приложение выполнить *свой код* в браузере пользователя (на стороне клиента). Атака проводится только в JavaScript и может быть отражена от веб-сервера, сохранена или отражена от объектной модели документа (DOM) либо сохранена в вашей базе данных.

**СОВЕТ.** Межсайтовый скрипting на основе DOM означает, что уязвимость находится в коде на стороне клиента, а не на сервере. Тестирование и организацию защиты от нее провести сложнее.

Хотя цели CSRF и XSS-атак схожи (переход приложения под контроль злоумышленника), для осуществления CSRF-атаки используются функции самого приложения. Несмотря на слово «межсайтовый» в обоих названиях, эти два вида атаки совершенно не связаны между собой. Блок-схема CSRF-атаки изображена на рис. 5.1.

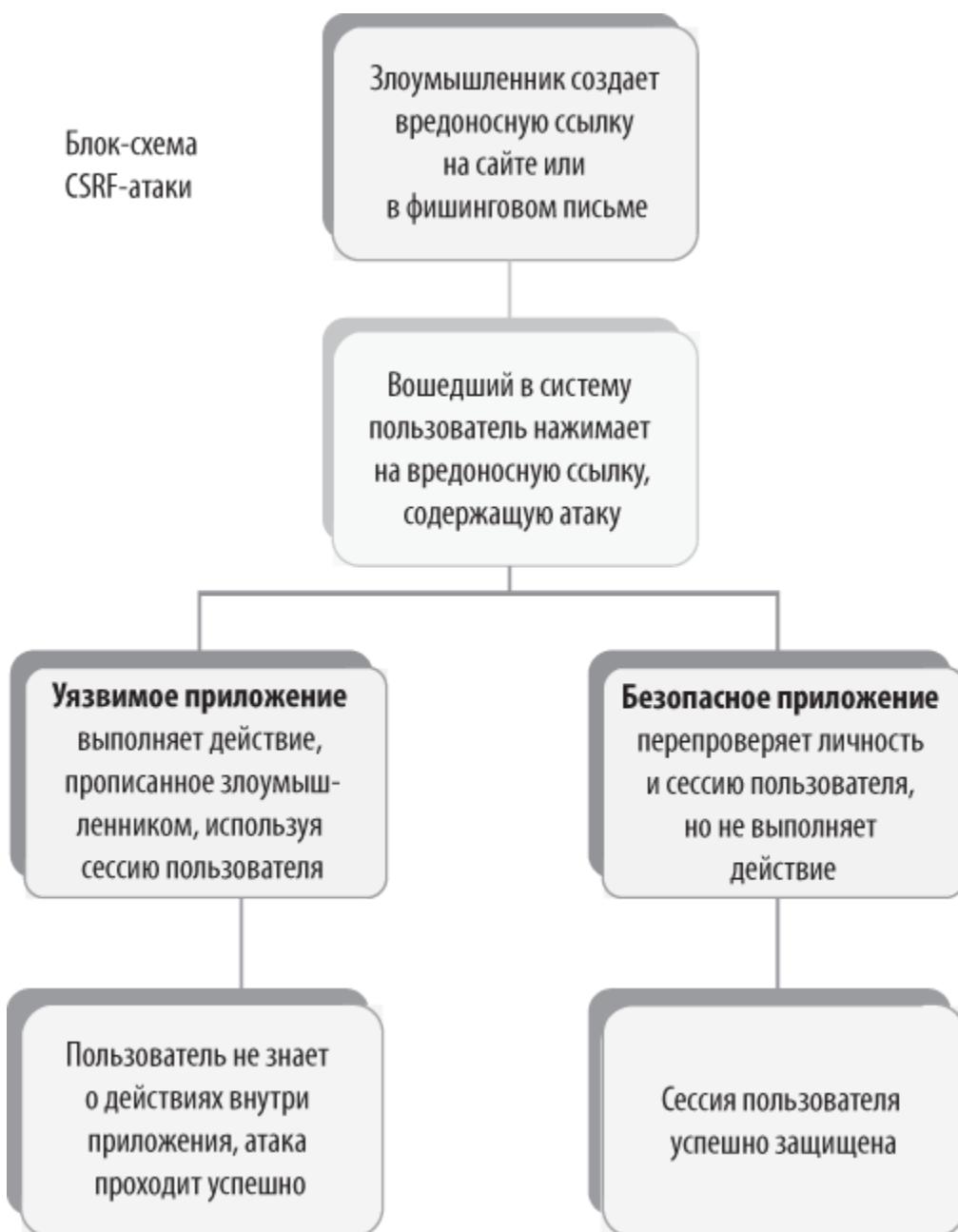


Рис. 5.1. Блок-схема CSRF-атаки

**ПРИМЕЧАНИЕ.** Если говорить в общем, то при CSRF-атаке злоумышленник захватывает активную сессию пользователя и затем выполняет вредоносное действие.

В главе 2 мы рассказали историю о том, как Алиса заходила на свой любимый сайт одежды, и обратили внимание на то, что бывает множество ситуаций, когда пользователи подолгу не выходят из своих учетных записей (в течение нескольких часов, дней или даже месяцев). Именно этим и пытается воспользоваться атакующий с помощью CSRF. В разделе «Управление сессиями» главы 4 шла речь о восстановлении сессии, однако пользователи обычно делают все, что им заблагорассудится, в то время как мы незаметно обновляем или уничтожаем сессии, чтобы обеспечить их безопасность при использовании наших приложений в интернете.

В случае CSRF-атаки злоумышленник обманом заставляет пользователя перейти по ссылке, содержащей атаку, либо атака находится внутри вредоносного сайта. В ее процессе происходят попытки выполнить действие против веб-приложения без ведома или разрешения пользователя. Если пользователь уже вошел в учетную запись на зараженном сайте (как когда Алиса покупала одежду в интернете), а этот сайт уязвим к подобным атакам, то транзакция выполняется (как правило, речь идет о краже).

Для защиты от данного типа атак многие современные платформы (Java, .Net, Ruby on Rails) осуществляют автоматическую передачу токена, в то время как другие предлагают эту функцию отдельно, и разработчику необходимо указать ее использование. Получить подробную информацию о ее реализации можно в памятке OWASP на данную тему [\[54\]](#).

Однако помимо токенов существуют и другие возможности защиты. Подходит все, что возвращает пользователя в исходное состояние и гарантирует его информированность о происходящем. Повторный вход в систему или ввод капчи являются наиболее часто используемыми вариантами.

Второй уровень защиты от данного типа атак заключается в том, чтобы перед выполнением действия убедиться, что заголовок referrer принадлежит требуемому сайту, а не другому сайту или электронной почте. Рекомендуется использовать оба способа защиты.

**ВНИМАНИЕ.** Некоторые считают, что заголовок `referrer` легко подделать. Это правда, но только если злоумышленник получил доступ к серверу, создал на нем запрос, а затем отправил поддельный запрос из приложения. Данная ситуация является уже не CSRF-атакой, а гораздо более серьезной проблемой. Заголовок `referrer` не может быть подделан с помощью одного только JavaScript (что было бы необходимо для выполнения CSRF-атаки), и поэтому проверка URL-адреса ссылки является хорошим вторым уровнем защиты.

---

## УСПЕХ В ОТРАСЛИ

OWASP Топ-10 действительно добился успеха в борьбе с CSRF, повысив осведомленность людей. В период с 2013 по 2017 год количество случаев использования этой уязвимости значительно сократилось (источник: OWASP Топ-10 2017 года), и в большинство современных платформ теперь включены соответствующие функции защиты от нее. Я признательна сообществу OWASP за все, что оно делает для нашей индустрии и всего мира.

---

## Подделка запросов со стороны сервера (SSRF)

Атака, при которой злоумышленник добавляет или изменяет параметры URL-адреса, называется подделкой запросов со стороны сервера (англ. Server-side request forgery, SSRF). В результате у злоумышленника появляется возможность инструктировать сервер либо получить информацию или доступ, которыми он не должен владеть. SSRF можно использовать для вызова внутренних API или доступа к базам данных. Приложение, уязвимое к такой атаке, – словно окно прямо в сеть с обходом периметра защиты: ужасная ситуация.

SSRF-атаку обычно сочетают с другими эксплойтами для нанесения еще большего ущерба, поэтому необходимо не только работать над защитой от нее напрямую, но и добавлять в уязвимое приложение другие средства защиты для уменьшения ущерба. На рис. 5.2 изображена блок-схема SSRF-атаки.

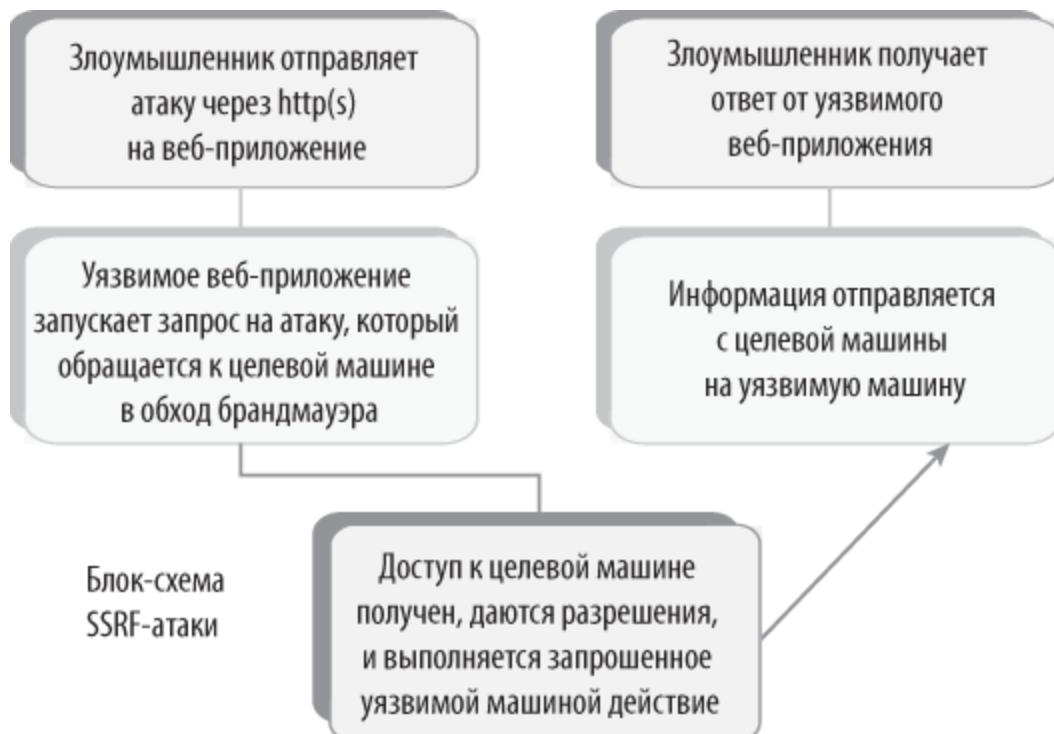


Рис. 5.2. Блок-схема SSRF-атаки

Защита (предотвращение атаки) и смягчение (уменьшение вреда, если атака все-таки произошла) состоят из нескольких уровней [\[55\]](#).

### **Защита**

1. Не использовать параметры URL для построения вызовов внутри приложения или сети, так как ими можно легко манипулировать. По возможности вообще избегать их использования.

2. Выполнять проверку вводимых данных (используя «белый список»), как указано в предыдущих главах.

3. Если приложению необходимо обращаться к другим сайтам, создать список разрешенных доменов. Все, что не входит в список, отклонять.

4. Использовать фильтрацию запросов, если она доступна в стеке. Подобно кодированию вывода, эта технология позволяет убедиться, что HTTP-ответы не содержат ничего опасного (например, SSRF-атаку) [\[56\]](#). Если она недоступна, необходимо самостоятельно проверять, не содержат ли ответы того, что было отправлено в запросе (пользовательский ввод), в неизменном виде. Данные должны быть проверены перед добавлением в ответ. Нет необходимости отражать эту информацию в неизменном и непроверенном виде.

5. Отключать неиспользуемые схемы URL, такие как file://, dict://, ftp:// и gopher://. Если они не используются, то должны быть отключены [\[57\]](#).

6. Создать специализированные модульные тесты для проверки на данную уязвимость.

7. Нанять опытного тестировщика безопасности ПО, который проверит наличие этой уязвимости во всех приложениях.

### **Смягчение**

1. Использовать конструкцию нулевого доверия в сети и приложениях, чтобы приложение было ограничено в доступе к другим областям сети и другим приложениям.

2. Применять принцип наименьших привилегий к доступу, предоставленному для веб-сервера и учетной записи (учетных записей) службы приложения, чтобы приложение, находясь под чужим контролем, имело доступ к малому количеству ресурсов.

3. Следовать руководству по усилению защиты для каждого веб-сервера, PaaS или контейнера, на котором размещены веб-приложения.

Всегда следовать всем руководствам по усилению защиты.

4. Обращаться с попытками атак так же, как и с любым другим неправильным вводом: выдавать неясное сообщение об ошибке. Никогда не следует отражать информацию об атаке на экране или в ответе.

5. Все внутренние службы должны требовать прохождения аутентификации (например, старые версии MongoDB не требовали аутентификации или авторизации для доступа к базе данных).

Существуют продукты, которые предоставляют дополнительные средства защиты от данного типа атак. Их внедрение можно обсудить со своей службой безопасности в тех случаях, если есть обеспокоенность насчет нехватки других средств защиты, если требуется очень высокий уровень безопасности или если есть вероятность стать мишенью для данного типа атак со стороны продвинутого противника.

За последние несколько лет участились случаи SSRF-атак, поскольку появилось больше информации о них. Хотя этот тип атак не входит в Топ-10, он заслуживает отдельного упоминания из-за своей опасности и распространенности.

**ПРИМЕЧАНИЕ.** При SSRF-атаке злоумышленник способен выдать инструкции веб-серверу (прочитать данные или выполнить вызов определенных функций), а затем использовать его для атаки на объекты, находящиеся за межсетевым экраном (со всем доступом и полномочиями, которыми наделен веб-сервер). В тестировании на проникновение она также известна как «поворот»: получение доступа к сетевому ресурсу, а затем использование его для получения дальнейшего доступа.

При этом данная атака также может быть использована для получения доступа к конфиденциальным данным или для вызова таких интернет-ресурсов, как API.

## Десериализация

Сериализация – это перевод данных из памяти (объекта) в файл или поток для того, чтобы их можно было передавать или хранить. Десериализация – возвращение данных в исходное состояние (память

или объект) после передачи данных или в случае их надобности. Чаще всего данный процесс осуществляется в XML, JSON и Java, однако существует несколько других ситуаций и технологий [\[58\]](#).

Если злоумышленнику удастся заменить сериализованную версию данных чем-то вредоносным, это может привести к разрушительным последствиям. Такая ситуация называется *небезопасной десериализацией*.

Самый простой способ избежать проблемы – не использовать эту функцию. Если ничего не сериализовать, то и десериализация не понадобится. Данная мысль может показаться упрощением, и так оно и есть. По возможности следует избегать любых потенциально рискованных ситуаций.

Если это невозможно, то вместо объектной сериализации данных следует использовать формат обработки данных (JSON, XML, YAML) для загрузки данных в граф объекта.

Если вам *безоговорочно* необходимо принять сериализованные объекты, они должны быть только из надежного источника и содержать лишь примитивные типы данных (типы, определенные как часть используемого языка программирования по умолчанию, например `int` и `char`).

Ниже перечислены некоторые меры предосторожности при работе с сериализованными объектами.

- Никогда не принимать сериализованный объект от ненадежного отправителя. Отправитель считается надежным, если он внесен в список одобренных. Никогда не принимать сериализованный объект от пользователя.
- Использовать защитную десериализацию с входным потоком объектов вместе со строгим утвержденным списком разрешенных классов.
- Десериализовывать только подписанные объекты и проверять контрольную сумму либо использовать HMAC с хешем SHA256 [\[59\]](#).
- Принимать только примитивные (неизменяемые) типы данных.

На сайте **OWASP.org** описано еще несколько шагов безопасной обработки сериализованных объектов из ненадежных источников. Однако, поскольку обстановка постоянно меняется, они не будут указаны здесь.

Главное, что нужно запомнить, – при любой возможности избегайте сериализации и десериализации. Если необходимо принимать сериализованные объекты, они должны быть только из надежных источников. Если следовать этому совету невозможно, то необходимо проявлять крайнюю осторожность в работе с такими объектами.

**ПРИМЕЧАНИЕ.** Если говорить в общем, то при десериализации злоумышленник заменяет сериализованную информацию своей собственной, которая содержит атаку на веб-сервер или приложение.

## Состояние гонки

Состояние гонки возникает в следующих случаях: когда действия программы зависят от времени, но потоки установлены несинхронно; когда ожидается блокировка ресурса, но она не происходит; когда программа ожидает, пока произойдет что-то еще, перед выполнением следующего шага... но программист забыл поставить соответствующую галочку.

Представьте, что Алиса хочет отредактировать файл на одном из общих рабочих дисков. Она открывает его, и программа обработки текстов накладывает на него временную блокировку. Если позже Боб захочет открыть этот же документ, то программа позволит ему увидеть его в формате «только для чтения», чтобы он не перезаписал изменения коллеги.

А представьте, что будет, если программа не заблокирует файл? Такие ситуации возникают чаще, чем вы думаете.

Предположим, что по дороге домой с работы вы захотели зайти в продуктовый магазин, чтобы купить продукты на ужин. Перед выходом вы проверили свой счет и увидели на нем \$60. Пока вы выбирали продукты, ваш супруг либо супруга произвел(а) несколько оплат, оставив на счету всего \$10. К моменту расчета на кассе счет обновился и составляет лишь \$10, из-за чего оплата по карте не проходит. Хотя данный пример, безусловно, не идеален, он описывает суть проблемы.

Теперь представьте, что кто-то намеренно пытается создать состояние гонки. Он использует карту для онлайн-покупок, пополняет ее, а в момент нажатия кнопки «Купить» одновременно пытается перевести деньги на другой счет. В идеале банковское программное обеспечение должно наложить блокировку на счет до завершения транзакции № 1, а затем попытаться провести транзакцию № 2. Порядок транзакций определяется в зависимости от того, какой запрос был получен первым.

Такая ситуация ожидаема для банков: люди пытаются украдь деньги с момента их изобретения. Однако атаки с использованием состояния гонки применимы к любому изменению в системе, зависящему от другого изменения:

- конкуренция за ресурсы (редактирование документа);
- обновление материальных ценностей (банковский баланс);
- все ситуации, где время выполнения одного действия зависит от другого условия.

Самым простым способом смягчения данной атаки является блокировка любой записи, находящейся в процессе изменения, до завершения действия. Если одна операция зависит от другой, перед началом следующей операции убедитесь, что предыдущая уже завершена. Необходимо заложить в структуру приложения программирование зависимостей. Незавершенную транзакцию следует откатить, в случае целесообразности – выдать ошибку и закрыть (а не допускать открытого или неизвестного состояния приложения).

Основная причина возникновения состояния гонки заключается в том, что при проектировании не учитывается зависимость изменений друг от друга. Моделирование угроз может помочь выявить ситуации, в которых проявляется данная уязвимость, и предотвратить их до того, как они превратятся в проблему. При проектировании приложений и других систем всегда следует указывать все зависимости.

## **Заключительные комментарии**

Не нужно заучивать OWASP Топ-10 наизусть, если только вы не создаете презентацию на данную тему или не идете на неоправданно напряженное собеседование. А вот что делать нужно, так это применять при создании программного обеспечения защитные меры, представленные в этой книге.

## **Упражнения**

- Кто-то из вашей проектной команды хочет принять сериализованные объекты из ненадежного источника. Вы знаете, что это плохая идея. Как можно эффективно объяснить риск коллеге? Запишите ответ. Объяснение должно быть убедительным и понятным.
- OWASP Топ-10 является стандартом обеспечения безопасности: да или нет?
- Назовите три пункта из OWASP Топ-10, которые были упомянуты в книге до главы 5.

- Применима ли уязвимость XXE к JSON? Применима ли она к YAML? Если да, почему? Если нет, то почему?
- Назовите пример ситуации (не обязательно связанной с компьютером), когда возникает состояние гонки.
- Почему мы откатываем незавершенные транзакции? Почему это важно? Приведите пример негативных последствий отказа от отката.

## **Часть II**

# **Как написать безупречный код**

**Глава 6.** Тестирование и развертывание

**Глава 7.** Программа безопасности приложений

**Глава 8.** Обеспечение безопасности современных систем и приложений

## Глава 6

# Тестирование и развертывание

Эта глава посвящена тестированию и развертыванию приложения и систем, от которых оно зависит: API, инфраструктуры, базы данных и т. д.

В данной книге тестированием называется проверка качества, надежности и безопасности программного обеспечения. Другими словами, это проверка того, что программное обеспечение выполняет *только* определенные клиентом функции и отвечает установленным требованиям проекта. Цель тестирования заключается также в подтверждении того, что *конфиденциальность, целостность и доступность* (CIA) системы и ее данных защищены *должным образом*.

Под развертыванием в книге понимаются «действия, предпринимаемые для публикации приложения». Это может быть нажатие правой кнопки мыши на приложение и выбор «Опубликовать», ручное копирование и вставка кода на сервер, составление формальной документации по задаче для проектной группы или создание целой автоматизированной системы непрерывной интеграции и непрерывной поставки (CI/CD).

Из этой главы вы *не узнаете*, как стать тестировщиком на проникновение. Данной теме посвящено несколько замечательных книг, но перед вами не одна из них.

## Тестирование кода

Код приложения – это письменные инструкции, передающиеся интерпретатору или компилятору. В данном разделе мы рассмотрим, как тестировать эти инструкции. Такой вид тестирования иногда называют *статическим тестированием*, поскольку в процессе проверки код не меняется и не выполняется, а представляет собой обычный текст. Его противоположностью является *динамическое тестирование*, которое обычно применяется в тех случаях, когда приложение активно на веб-сервере или другой платформе и тестировщик может взаимодействовать с ним в реальном времени. Сейчас мы рассмотрим статическое тестирование.

### Обзор кода

Первый вид тестирования, который можно провести, – обзор кода с целью обнаружения проблем, связанных с обеспечением безопасности. «Обзор программного кода безопасности» подразумевает чтение всего кода, выполняющего функцию обеспечения безопасности (то есть кода, выполняющего «контроль безопасности»), а также все точки интеграции (поскольку они *всегда* должны иметь элементы контроля безопасности). Точками интеграции называются области соприкосновения различных частей системы: например, пользовательский интерфейс веб-приложения вызывает API, API вызывает хранимую процедуру в базе данных, или приложение записывает ошибку в журнал на отдельном сервере.

При выполнении обзора кода, отвечающего за безопасность, не обязательно читать каждую строку. Часто в работу берутся только средства контроля безопасности и функции обеспечения безопасности. Каждый процесс тестирования уникален, поэтому до начала работы важно уточнить требования, соответствие которым необходимо проверить. Вместе с тем обзору *всегда* будут подвергаться средства контроля безопасности.

Какие функции обеспечения безопасности необходимо проверить?

- Аутентификацию.
- Авторизацию.

- Конфигурацию безопасности (такие особенности шифрования, как длина ключа или алгоритм, а также настройки заголовков безопасности).
- Управление идентификацией и системы идентификации.
- Управление сессиями.
- Валидацию ввода.
- Кодирование выходных данных.

Необходимо выполнить проверку того, что мы обсуждали в предыдущих главах, и убедиться в том, что все это реализовано верно.

Для чего проводится обзор? Во-первых, чтобы убедиться в наличии контроля безопасности во всех компонентах приложения, где он должен быть. Во-вторых, чтобы подтвердить, что надлежащий контроль безопасности отвечает установленным условиям и требованиям, и, в-третьих, что его реализация и настройка были выполнены корректно.

Возможно, сейчас вы задаетесь вопросом: «В каких компонентах приложения требуется контроль?» Помимо поиска ранее упомянутых элементов, необходимо просмотреть модели угроз (о чем мы поговорим позже), список требований и все пользовательские истории, связанные с безопасностью.

**СОВЕТ.** Алиса всегда поручает кому-то другому проверить свои презентации и документы, прежде чем показать их начальству, поскольку знает, что часто не замечает своих ошибок. Такой же подход верен и в отношении кода. Лучше всего, если его обзором будет заниматься не тот же человек, который его написал.

Обзор кода может проводиться в разных ситуациях на этапах кодирования, тестирования, релиза и сопровождения. Обычно проверяется каждый новый фрагмент кода до его слияния с основной веткой. Также тесты проходит самая первая версия приложения перед публикацией. Причиной проверки может стать даже утечка данных, тогда она проводится в рамках расследования произошедшего инцидента. Тем не менее обычно обзор кода безопасности делается после отправки кода на тестирование при разработке приложения по каскадной модели и перед выпуском каждой новой функции – при

работе по гибкой методологии разработки (тогда проводится проверка только нового кода). В среде DevOps выделены несколько различных временных промежутков, которые могут быть использованы для обзора кода: перед каждой загрузкой кода (внесения изменений в основную ветвь кода), перед каждым крупным релизом, для каждого отдельного внедрения кода – все зависит от потребностей команды.

Начать обзор кода безопасности можно, предложив коллеге или другу просмотреть код друг друга. Если у вас нет никого, кто мог бы вам помочь, просмотрите свой код сами. Выделите время для этой работы и напомните себе, что целью поисков являются проблемы в средствах контроля безопасности и бизнес-логике приложения. Если что-то вызвало сомнение, можно обсудить это со службой безопасности или с другом, а также поискать соответствующую информацию в интернете. Там можно найти полезные руководства, посвященные тому, как проводить обзор кода, в том числе советы по работе с конкретными языками программирования. Последовательность результатов обеспечивается с помощью контрольных списков, представленных известными руководствами или созданных собственноручно.

Ничего страшного, если вначале вы понятия не имеете, что делаете. Это именно та работа, качество которой улучшается с течением времени. С каждым обзором кода вы будете узнавать больше и повышать свой профессионализм в качестве программиста. Каждая новая обнаруженная ошибка преподаст новый урок. Возможно, обзор кода не кажется самым захватывающим из всех видов тестирования, но он позволяет с самого начала безопасно и немедленно принести пользу команде.

## Статическое тестирование безопасности приложений (SAST)

SAST расшифровывается как Static Application Security Testing (статическое тестирование безопасности приложений), но этой аббревиатурой обозначается не само статическое тестирование, а необходимые для его выполнения инструменты. Инструмент SAST может анализировать код, двоичные файлы и даже байт-код на предмет наличия потенциальных проблем, связанных с безопасностью. Каждый инструмент обладает своим набором функций, поэтому перед покупкой следует опробовать выбранный вариант. При оценке кода инструмент SAST разбирает его, как компилятор, но не разбивает его на байты для запуска на сервере, а ищет возможные риски, связанные с обеспечением безопасности: от выявления известного уязвимого класса или функции до предупреждения об отсутствии или неправильной реализации средств контроля безопасности, утечках памяти, неиспользуемых переменных или функциях, низком качестве кода и т. д.

Замечательная особенность инструментов SAST заключается в том, что они могут углубляться на несколько уровней от функции к функции, то есть работать далеко за пределами возможностей человека, проводящего проверку вручную. Данные инструменты помогают найти множество уязвимостей, которые большинство людей пропустили бы, а результаты таких проверок могут быть очень впечатляющими.

Тем не менее многие инструменты SAST выдают очень большое количество ложноположительных результатов.

**ПРИМЕЧАНИЕ.** Ложное срабатывание инструмента тестирования безопасности означает, что оно сообщило о наличии уязвимости, которая таковой не является. В случае ложноотрицательного результата инструмент пропускает уязвимость. Все инструменты выдают оба типа результатов, но в идеале ожидается как можно больше истинно положительных (инструмент сообщает об обоснованных уязвимостях) и истинно отрицательных (он ничего не пропустил).

Да, вы все правильно поняли. Инструменты SAST выдают самое большое количество ложноположительных результатов среди всех инструментов тестирования безопасности приложений, даже больше, чем при ручной проверке кода. Почему?

Первая причина заключается в том, что можно выбрать довольно строгие настройки инструмента и заставить его показывать только то, что он считает истинно положительным. В таком случае он не покажет уязвимости, в которых не будет почти полностью уверен, и пропустит множество из них, что приведет к огромному количеству ложноотрицательных результатов. Инструмент заметит потенциальную проблему, но не сообщит о ней. Вполне вероятно, что таким образом большинство проблем в приложении останутся незамеченными. Если вы хотите обеспечить безопасность ПО, такое решение будет далеко от идеала.

**ПРИМЕЧАНИЕ.** Некоторые инструменты SAST выполняют «символическое исполнение», то есть анализируют программы, чтобы определить, какие входные данные вызывают исполнение каждой ее части. Интерпретатор принимает символические значения вместо фактических входных данных, с которыми программа работала бы в нормальных условиях («Википедия») [\[60\]](#).

Для получения наиболее полного набора потенциальных проблем настройте инструмент SAST так, чтобы он сообщал обо всех подозрительных элементах, и выделите время для изучения полученных результатов.

Эти данные стоит рассматривать не как список подтвержденных уязвимостей, а скорее как набор «подсказок», где именно следует провести обзор кода. Такая интерпретация принесет гораздо больше пользы от инструмента, и, как в случае с обзором кода вручную, терпение и практика приведут к повышению качества работы.

Заменить ручной обзор кода инструментом SAST не получится, так как они подразумевают проведение разных видов анализа и получение разных результатов. Автоматический анализ кода обычно не способен выявить недостатки бизнес-логики: он просто не понимает данное условие. Таким образом, рекомендуется комбинировать два этих

подхода, чтобы увеличить охват проверки и найти максимальное количество уязвимостей.

Для обеспечения наилучших результатов конфигурация инструмента должна соответствовать программной среде, протоколам и другим используемым в разработке технологиям. Например, правильно настроенный для технологического стека инструмент SAST будет распознавать функции безопасности, исходящие от платформы (валидацию ввода или экранирование вывода), и сможет обнаружить потенциальные ошибки в их реализации.

Эффективность SAST сильно варьируется в зависимости от поддерживаемых технологий. Необходимо тщательно оценить инструмент, прежде чем решиться на его покупку. Большой список существующих бесплатных и платных инструментов доступен на сайте [owasp.org/www.community/Source\\_Code\\_Analysis\\_Tools](http://owasp.org/www.community/Source_Code_Analysis_Tools).

**ВНИМАНИЕ.** В настоящее время ни один автоматизированный инструмент (любого типа) не может обеспечить 100 %-ную точность при поиске уязвимостей, поэтому *ни в коем случае нельзя* передавать непроверенные результаты команде разработчиков или регистрировать ошибки в своем баг-трекере. Когда вы тратите время разработчиков на ложные срабатывания, их доверие к вам падает. *Всегда проверяйте результаты, полученные с помощью любого автоматизированного инструмента, прежде чем передавать их другой команде.*

## Анализ состава программного обеспечения (SCA)

Как уже говорилось в предыдущих главах, большинство современных приложений содержат множество библиотек, платформ и компонентов, имеющих в себе сторонний код (код, написанный другой командой), и вместе с ними в программное обеспечение внедряются все находящиеся в них уязвимости. Оптимальным, но непростым для реализации на практике вариантом в таком случае является ограничение стороннего кода настолько, насколько это возможно, чтобы уменьшить поверхность атаки.

Лучший способ обеспечить безопасность кода от сторонних разработчиков – это выполнение SCA.

Согласно некоторым интернет-источникам, *анализ состава программного обеспечения* (англ. Software Composition Analysis, SCA) – это процесс автоматизации контроля использования программного обеспечения с открытым исходным кодом (англ. open source software, OSS) с целью управления рисками, обеспечения безопасности и соблюдения лицензионных требований<sup>[61]</sup>. Однако данное определение является не совсем полным, так как инструменты SCA сообщают также о несвободных (с закрытым исходным кодом) библиотеках и платформах.

В большинстве крупных компаний с несколькими командами разработчиков в штате нет ни полного реестра веб-приложений (списка всего созданного программного обеспечения), ни соответствующей *спецификации материалов программного обеспечения* (англ. software bill of materials, SBoM) (перечня всех компонентов и версий) для каждого из этих активов. Инструмент SCA может создать SBoM и автоматизировать процесс его поддержки в актуальном состоянии. SBoM нужен для того, чтобы иметь правильное представление об имеющихся различных компонентах и средствах защиты. Если в платформе А версии Б обнаруживается уязвимость, необходимо как можно скорее узнать, используется ли в производстве эта платформа данной версии. Такую информацию SBoM может сообщить немедленно.

Инструменты SCA работают на основе списка «известных уязвимостей», который может быть составлен различными способами: через сбор информации из публичных списков, через покупку списков других компаний, через наем исследователей безопасности для анализа популярных платформ и компонентов, через использование автоматизированных инструментов для анализа компонентов, через подписку на рассылки по теме обеспечения безопасности от разработчиков компонентов. Некоторые компании-разработчики позволяют производителям инструментов, правительствам или крупным компаниям подписываться на их списки напрямую (обычно за определенную плату).

Инструменты SCA можно автоматизировать так, чтобы они и направляли на репозиторий кода (место хранения всего исходного

кода), и запускались при развертывании нового кода (в идеале – автоматизированно в конвейере CI/CD). Планирование регулярного сканирования репозитория кода требует доступа только для чтения и может помочь подтвердить адекватный уровень безопасности всех приложений, включая устаревшие. Даже если приложение очень давно не обновлялось, его компоненты все равно могут находиться под пристальным вниманием злоумышленников и перейти из списка «OK» в «известную уязвимость». Вот почему очень важно сканировать код инструментом SCA, даже если он не менялся в течение длительного времени. Безопасность кода должна обеспечиваться и на этапе эксплуатации.

Перед релизом код также необходимо просканировать. Можно обновить уже существующий компонент или добавить новый, но есть вероятность, что даже эта новая конкретная версия будет «заведомо уязвимой». Когда выходит новая версия, исследователи безопасности и злоумышленники часто проверяют ее на наличие проблем, связанных с обеспечением безопасности, поэтому в первые несколько дней после релиза в продуктах, патчах и платформах часто обнаруживаются новые уязвимости. Использование инструмента SCA для проверки даже новых компонентов повышает безопасность программного обеспечения.

Некоторые инструменты SCA также предлагают интеграцию с IDE (Integrated development environment – интегрированной средой разработки). Интеграция проксирует IDE и тем самым гарантирует, что разработчики могут загружать пакеты только из своего репозитория предварительно одобренных пакетов. Этой услугой необходимо воспользоваться, если она предлагается в приобретенном офисом инструменте SCA. Она позволяет предотвратить ошибки, связанные с обеспечением безопасности (что даже лучше, чем их поиск и исправление).

Онлайн-список различных инструментов SCA (платных и бесплатных) можно найти на сайте [owasp.org/www-community/Component\\_Analysis#tools-listing](http://owasp.org/www-community/Component_Analysis#tools-listing).

## Модульное тестирование

Модульное тестирование – это уровень тестирования программного обеспечения, при котором тестируются его отдельные модули или компоненты. Цель – подтвердить, что работа каждой единицы программного обеспечения соответствует ожиданиям. Модулем называется наименьшая тестируемая часть любого ПО. Обычно он имеет один или несколько входов и, как правило, только один выход [\[62\]](#).

Модульные тесты часто пишутся в IDE и хранятся вместе с кодом в репозитории. Они почти так же ценные, как и сам код. В идеале перед загрузкой кода разработчики выполняют все модульные тесты, и, если код не проходит тест, разработчик должен либо 1) исправить код, либо 2) адаптировать тест к новым изменениям кода. Не разрешается загружать сломанный код, который не прошел модульные тесты.

**СОВЕТ.** Часто при обсуждении модульных тестов люди говорят о «покрытии кода». Покрытие кода показывает процент кода, проверенного с помощью модульного тестирования. Стандартного значения не существует, и тем не менее каждая организация стремится к 100 % покрытия. В реальности же редко можно встретить очень высокое покрытие кода. Постарайтесь не быть слишком строгими к своим разработчикам, если они находятся далеко от идеальных 100 %: мало кто приближается к этому показателю.

Модульные тесты, как правило, являются *позитивными*, то есть они проверяют, чтобы код всегда выполнял заложенные в него функции. Тем не менее они могут быть и *негативными*. Их цель – проверить, что приложение *правильно отказывает* в совершении операции, если получает ненадлежащие входные данные. Негативные модульные тесты часто используются для проверки не только качества приложения, но и его безопасности.

Для проведения негативных модульных тестов поговорите с разработчиками и получите разрешение на «копание» в их коде. Затем скопируйте несколько их модульных тестов, приставьте к ним в конец

элемент, обозначающий негативную версию теста (например, `_abuse`), и добавьте соответствующую полезную нагрузку (потенциально проблемный код). Запустите негативные модульные тесты и выясните, какой код проходит их успешно, а какой – нет.

**СОВЕТ.** Если у вас на работе реализуется программа чемпионов по безопасности, попросите одного из них о помощи. Это прекрасная возможность улучшить навыки чемпиона в сфере безопасности, и со временем можно будет передать эту задачу ему. Если нет, не волнуйтесь. Мы поговорим об инициировании программы чемпионов по безопасности в последующих главах.

Например, если код будет выведен на экран, попытайтесь добавить в него тег `<script>`, затем убедитесь, что код правильно зашифровывает и не выполняет его. Это довольно экстремальный пример: если код выполнит тег, значит, у вас большие проблемы. Команда OWASP создала удобную «Шпаргалку по уклонению от XSS-фильтра»<sup>[63]</sup>, которая, без сомнений, является отличным помощником для начала работы с полезной нагрузкой (плохим кодом, добавляемым в негативные модульные тесты). Можно добавить логические тесты, ненадлежащие входные данные, повторное тестирование проблем, обнаруженных в ходе ручного тестирования безопасности, и тесты для всех разработанных моделей угроз.

Основная ценность модульных тестов, как негативных, так и позитивных, заключается в регрессионном тестировании. Регрессионное тестирование – это повторное выполнение функциональных и нефункциональных тестов с целью подтверждения того, что ранее разработанное и протестированное программное обеспечение продолжает работать после внесения изменений<sup>[64]</sup>. Негативные модульные тесты обеспечивают автоматизированное регрессионное тестирование безопасности, которое чрезвычайно полезно на ранних этапах жизненного цикла разработки ПО. Они требуют предварительной подготовки и обслуживания, но результат их выполнения весьма действенен с точки зрения безопасности.

**СОВЕТ.** Если в компании проводятся тесты на проникновение, оценка обеспечения безопасности или оценка уязвимости,

подумайте о создании модульных тестов на основе результатов этих проверок (в надлежащих случаях). Таким образом вы сможете убедиться, что допущенные ранее ошибки не повторятся. Более того, если у вас есть разрешение, примените полученные из итогового отчета знания ко всем приложениям, с которыми работаете. Создайте модульные тесты для всех возможных приложений. Разработчики обычно работают над несколькими проектами, и они могли допустить одну и ту же ошибку в нескольких местах.

## **Инфраструктура как код (IaC) и безопасность как код (SaC)**

Давным-давно, когда Боб еще занимался технической работой, а не управлял проектами, как сейчас, серверы назывались милыми именами в честь героев телешоу. Каждый сервер был отдельной физической машиной, и Боб помнит, как приходилось устанавливать их в стойки, какими тяжелыми и шумными они были. Он не забыл, как ему нужно было ходить от машины к машине с установочным компакт-диском, чтобы обновить операционную систему. Со временем появилась возможность получать обновления через интернет, и это заметно облегчило жизнь, но Боб все равно должен был ходить и нажимать кнопку Next столько раз, что уже и не упомянуть.

Со временем компания перешла на виртуальные машины (ВМ). Теперь несколько операционных систем могли находиться на одном сервере. К тому моменту уже все было подключено к сети, и обновления скачивались из интернета, а не с компакт-дисков. Стали использоваться автоматизированные системы для установки патчей, и больше не было необходимости по вечерам нажимать кнопку Next или ходить от машины к машине. Боб был в восторге от такого прогресса. Сейчас у них есть DevOps-специалисты и инженеры по надежности сайтов (SRE), которые занимаются чем-то под названием «инфраструктура как код», но их работа гораздо выше уровня понимания Боба, поэтому он занимается управлением проектами, а заботу о технических деталях оставил им.

*Инфраструктура как код (англ. *Infrastructure as Code*, IaC) – это подход для управления и описания инфраструктуры центра обработки данных через конфигурационные файлы, а не через ручное редактирование конфигураций на серверах или интерактивное взаимодействие.*

*Википедия*[\[65\]](#)

*Инфраструктура как код* (IaC), иногда называемая *конфигурацией как код*, определяет инфраструктуру с помощью написанного кода, в отличие от образа системы или установочного файла. Этот код может быть запущен для создания, замены или обновления инфраструктуры. Как и код приложения, он хранится в репозитории кода или выпускается через конвейер CI/CD.

Результаты IaC более однородны, а если код хранится в репозитории кода, то любые изменения можно легко отследить, проверить и при необходимости откатить назад. Как и к любому другому коду, к IaC можно применять тестирование безопасности.

**ПРИМЕЧАНИЕ.** IaC проверяет все виды компоненты безопасности: управление изменениями, аварийное восстановление, устранение возможности опечаток и многое другое. Хотя темой книги является создание безопасного программного обеспечения, код живет на серверах, и поэтому он имеет важное значение. Кроме того, код есть код. Давайте защитим его в полном объеме.

*Безопасность как код (англ. *Security as code*, SaC) – это практика встраивания элементов безопасности в инструменты и рабочие процессы DevOps путем составления схемы внесения изменений в код и инфраструктуру, а также поиска мест, куда можно добавить проверки, тесты и ограничения без лишних затрат и задержек.*

*О’Райли*[\[66\]](#)

Если разработчики автоматизируют все, даже инфраструктуру, мы должны автоматизировать и обеспечение безопасности. Существует

множество различных способов реализации концепции SaC, но в этом разделе мы сосредоточимся на том, как проверить IaC на наличие проблем, связанных с обеспечением безопасности. Идею автоматизации проверок безопасности мы рассмотрим в следующих главах, когда будем обсуждать DevOps и DevSecOps.

Код, написанный для IaC, можно проверить на наличие проблем, как и любой другой код, однако его необходимо протестировать до выпуска в эксплуатацию (не позже). IaC, в частности, следует не только проверить на уязвимости, но и убедиться в том, что тот, кто писал определения инфраструктуры, включил необходимые функции безопасности и конфигурации для каждого определения инфраструктуры: принудительное использование HTTPS для виртуальной машины веб-сервера, использование версии TLS, закрытие портов, размещение контейнера внутри определенной V-LAN в локальной сети и т. д.

**СОВЕТ.** Обеспечьте защиту безопасности как кода и инфраструктуры как кода так же, как и защиту кода приложения: храните их в версионном репозитории кода, отслеживайте изменения, следите за тем, кто имеет доступ, и регулярно создавайте резервные копии.

## Тестирование приложения

После компиляции кода в байт-код или его перевода в динамически работающую часть программного обеспечения можно тестировать приложение. Инструменты и тестировщики могут взаимодействовать с ПО, чтобы узнать его положительные или негативные реакции на какие-либо действия и события, тем самым проверяя, насколько оно безопасно. Обычно такие проверки называются динамическим тестированием безопасности приложений (DAST).

При тестировании приложения необходимо искать как дефекты безопасности (ошибки реализации), так и ошибки самого приложения (ошибки проектирования). Необходимо проверять приложение относительно соблюдения конфиденциальности, целостности и доступности (триады CIA), включая данные и другие системы, к которым оно подключается и от которых зависит. Приложение должно соответствовать всем применимым политикам, стандартам, юридическим требованиям и передовому опыту, а также в нем должны отсутствовать часто встречающиеся проблемы, связанные с безопасностью. Необходимо тщательно изучить каждую функцию, касающуюся обеспечения безопасности, и каждую точку интеграции на предмет возможной уязвимости, неправильной конфигурации или ошибки. И, наконец, следует убедиться, что система всегда правильно завершает свою работу, независимо от того, насколько ужасно с ней обращаются.

При проведении тестирования важно сделать все возможное, чтобы избежать «беспорядка». Будьте осторожны, ничего не ломайте, не уничтожайте данные и не выводите системы из строя. Даже если есть возможность использовать уязвимость или взять трофеи, не всегда в этом есть необходимость, ведь тестировщика нанимают для того, чтобы проверить безопасность приложения, а не для того, чтобы он продемонстрировал свои навыки. Всегда следует оставаться в рамках соглашения о тестировании и не поддаваться соблазну. По мере приобретения опыта вам будет легче контролировать свое рвение и быть более точным в действиях.

Перед началом тестирования необходимо сделать резервную копию данных и самого приложения, если отсутствует автоматическая

система развертывания. Если проверка проводится для третьего лица, необходимо иметь письменное разрешение на тестирование приложений, выданное человеком, имеющим на это полномочия. Редко бывает так, что у человека в соседнем кабинете действительно есть право «принимать риск» при тестировании безопасности, поэтому перед любой подобной проверкой следует убедиться, что вы защищены с юридической точки зрения.

**СОВЕТ.** Как правило, определение объема и утверждение тестирования документируются в рамках формального процесса во время консультирования или проведения тестирования в качестве третьей стороны. В интернете можно найти несколько книг и статей, объясняющих, как задокументировать этот процесс для юридической защиты.

По возможности следует тестировать приложение в специальной тестовой среде, а не когда оно работает в режиме реального времени. Использование выделенной тестовой среды позволит проводить тестирование без риска для эксплуатационных систем и данных.

**СОВЕТ.** Составьте официальное заявление о масштабах тестирования, в котором будет указано, какие IP- и URL-адреса будут тестироваться, а также то, будете ли вы тестировать API, сеть, WAF и т. д. Определите в нем используемые инструменты, способы проведения проверки, точки начала, откуда команды безопасности и сети могут ожидать вашего прихода (чтобы они смогли узнать вас), как долго и когда именно будет проходить тестирование, и все остальное, что можно придумать и что может потребоваться. Данный документ, который обычно называется «Правилами взаимодействия» или документом об объеме тестирования, должен быть подписан уполномоченным лицом и храниться в надежном месте.

## Ручное тестирование

Хотя в ИТ-отрасли много внимания уделяется автоматизации, ручное тестирование по сей день остается самым точным видом проверки

безопасности (если его выполняет специалист). Автоматизированные инструменты способны найти большинство проблем значительно быстрее, тем не менее существует множество типов ошибок и дефектов безопасности, которые может обнаружить только квалифицированный человек. Для достижения наилучших результатов используется сочетание ручного и автоматизированного видов тестирования.

## Браузеры

Современный веб-браузер – это лучший инструмент тестирования безопасности веб-приложения, так как через него мы общаемся с веб-приложениями. Подготовленный злоумышленник может использовать его, чтобы найти всевозможные проблемы в приложении или даже в самой сети. Большинство проблем с бизнес-логикой, которые чаще всего являются ошибками проектирования, можно найти только с помощью браузера. Автоматизированные инструменты, как правило, не способны выявить такие проблемы. Браузер является самым прямым способом общения с веб-приложениями, и он должен быть первым в арсенале инструментов для тестирования.

Браузеры обычно используются для проверки наличия проблем с бизнес-логикой. Делает ли приложение то, что не должно делать (в ущерб себе)? Можно ли заставить его выполнять действия, которые оно не должно выполнять? Может ли пользователь увидеть информацию, которую он не должен видеть? Браузер можно использовать для входа в систему от имени различных типов пользователей, чтобы убедиться, что никто из них не видит ничего выходящего за рамки его роли в системе (например, может ли обычный пользователь просматривать страницу администратора)?

Тестирование с помощью браузера дает возможность изучить URL-параметры и любые другие места для ввода пользовательских данных (например, поля ввода).

**СОВЕТ.** Необходимо иметь в своем арсенале несколько (два или три) популярных современных браузеров, потому что проблема может проявляться только в одном конкретном браузере. К тому же различные браузеры имеют различные

реализации функций безопасности, конфигурации и собственные уязвимости. Одного браузера недостаточно.

## Инструменты разработчика

При тестировании в браузере следует всегда включать «Инструменты разработчика». Если браузер имеет встроенный инструмент или функцию, предназначенные для разработчиков, можно воспользоваться ими. Полезно видеть все, что видят разработчики: в инструментах можно часто найти много интересного касаемо обеспечения безопасности.

Например, щелкнув правой кнопкой мыши на любом поле веб-страницы и выбрав «Исследовать», можно увидеть всю информацию об этом поле. Во вкладках «Хранилище» и «Сеть» можно узнать, хранится ли локально что-нибудь, о чем следует знать, и каковы сетевые настройки. Вкладка «Отладчик» также полна полезной информации, которую не следует упускать из виду.

**СОВЕТ.** Используя инструменты разработчика, всегда следует проверять, есть ли «скрытые поля» и содержатся ли в них конфиденциальные данные. Если это так, исправьте данную проблему или сообщите о ней. В скрытые поля никогда не следует помещать ценные либо конфиденциальные данные.

## Веб-прокси

Мы уже затрагивали тему веб-прокси. Они находятся между браузером и веб-приложением, отслеживая обмен данными между браузером (фронтенд) и сервером (бэкенд приложения). Однако мы не упомянули о том, что веб-прокси являются прекрасным инструментом для ручного тестирования безопасности.

Веб-прокси перехватывает HTTP-запросы и ответы между фронтендом веб-приложения (частью, работающей на стороне браузера) и бэкендом (частью, работающей на стороне веб-сервера). Веб-прокси – это программное обеспечение, запускаемое пользователем на компьютере, с которого он получает доступ к веб-приложению. Функцию прокси можно как включить, так и выключить.

Большинство веб-прокси позволяют перехватить HTTP-запрос до того, как он покинет компьютер и попадет в сеть, что дает

возможность изменять значения запроса вручную. Веб-прокси также обычно позволяют повторять и изменять запросы, применять списки слов для автоматизации нескольких запросов и часто помогают автоматизировать анализ ответов.

При использовании веб-прокси можно проверить приложение на разнообразные известные уязвимости – например, инъекционные или XSS. Поместив атаку непосредственно в веб-прокси и отправив запрос, можно узнать о ее успешности исходя из поведения приложения. Необходимо изучить все возможные входы в приложение при работе с веб-прокси.

**ВНИМАНИЕ.** В идеале при тестовой атаке на веб-приложение нужно проверить его на уязвимости, не причинив ему какого-либо вреда. Например, при тестировании на XSS-уязвимость можно вызвать всплывающее окно сообщения или украсть cookie либо сессию пользователя. При тестировании на слепую SQL-инъекцию можно послать базе данных команду перехода в сон с таймером, а затем засечь время ответа, чтобы проверить, выполнит ли она ее. Оба этих примера используют возможную уязвимость, но не наносят ущерба и не раскрывают потенциально важные данные. Некоторые клиенты захотят большего: например, получить данные из базы данных как доказательство, что их можно прочитать. В таком случае, прежде чем действовать, следует убедиться в том, что их устраивает такое глубокое погружение в приложение.

Веб-прокси также показывает все API, к которым обращается веб-приложение. Их тоже следует изучить, чтобы подтвердить их безопасность, поскольку от них зависит приложение. Используя веб-прокси, можно напрямую взаимодействовать с API, минуя фронтенд. При первом изменении значения в запросах можно попытаться подсчитать все страницы и функции веб-сайта или API. Например, у нас есть API под названием «Сотрудники» с возможностью вызова функции «Заголовок». Может быть, стоит попытаться вызвать функцию «Адрес», чтобы узнать, существует ли она? Опять же, следует изучить и протестировать все возможные входы для каждого API.

Злоумышленник, использующий веб-прокси и браузер, представляет довольно большую опасность, поэтому можно использовать эти же инструменты для глубокого тестирования своих или чужих приложений. Это сложный навык, но, как и любой другой, его можно освоить, если проявить терпение.

## Фаззинг

*Фаззинг (англ. fuzzing) – техника тестирования программного обеспечения, часто автоматическая или полуавтоматическая, заключающаяся в передаче приложению на вход неправильных, неожиданных или случайных данных. Предметом интереса являются падения и зависания, нарушения внутренней логики и проверок в коде приложения, утечки памяти, вызванные такими данными на входе.*

*Википедия*[\[67\]](#)

Простыми словами, фаззинг – это забрасывание в приложение всякой всячины до тех пор, пока что-нибудь не прилипнет.

Представьте, что в приложении есть поле ввода и вы вводите в него букву «а», затем нажимаете кнопку «Отправить». Приложение действует определенным (ожидаемым) образом. Затем вы вводите «аа», и снова получается удовлетворительный результат. Затем следуют 100, 200, 1000, 10 000 символов и т. д. После всех этих действий вы отправляете теги `<script>`, одинарные кавычки и шелл-код и смотрите, примет ли приложение эти вводные данные или станет работать по-другому. С помощью инструмента фаззинга можно добавить больше потоков и поместить его в фоновый режим, чтобы ускорить данный процесс.

Часто автоматические инструменты действительно хорошо справляются с фазз-тестированием, так как обладают довольно

большим терпением по сравнению с людьми. Как только инструмент замечает какие-то изменения в работе приложения, за дело принимается человек. Каждая аномалия тщательно изучается и проверяется на предмет того, является ли она уязвимостью. Помните, что часто уязвимости находятся тогда, когда приложение переходит в неизвестное состояние.

Фаззинг можно применять к любому входу в приложение, включая URL-параметры (API), поля ввода, файлы и папки, cookie, заголовки, обнаружение скрытых полей, а также к HTTP-методам.

## Динамическое тестирование безопасности приложений (DAST)

В этом разделе рассматриваются автоматизированные инструменты, которые каталогизируют системы (составляют список просмотренных элементов; иногда это называют «краулинг» или «спайдеринг»), а затем пытаются выполнить автоматизированный анализ безопасности, взаимодействуя с работающими системами. Вам этот процесс может быть известен также как сканирование уязвимостей, оценка уязвимости или сканирование веб-приложений. Данный раздел поделен на две категории: инфраструктура (копии) и пользовательские приложения (уникальные).

### Инфраструктура

Если два пользователя имеют копии операционной системы Windows 10 и они оба устанавливают одни и те же обновления, значит, они используют одинаковые *копии* операционной системы. Если ни один из них не устанавливал определенный патч, устраняющий конкретную проблему безопасности, значит, у них обоих одна и та же проблема безопасности.

Когда кто-то говорит «проводить оценку уязвимости», он, как правило, имеет в виду запуск автоматизированного сканирования какой-либо инфраструктуры на наличие уязвимостей. Результаты, полученные с помощью этих инструментов, как правило, точнее, чем у других инструментов: пропущенное обновление или неудачно выбранная конфигурация являются проблемой, которая может возникнуть на любой другой копии этой версии данного типа инфраструктуры. Windows 10 – это Windows 10. Компьютеры с одинаковой операционной системой могут быть настроены по-разному, но все они имеют одинаковый набор параметров, исправлений и других компонентов. В этом заключается отличие инфраструктуры от пользовательского программного обеспечения.

Сообщения об уязвимостях в инфраструктуре и платформах часто заносятся в централизованные репозитории вместе с комментариями о

том, как их найти и устраниить. Автоматизированные инструменты оценки уязвимости используют информацию из этих хранилищ, чтобы знать, что нужно искать.

## Пользовательские приложения

Пользовательские приложения отличаются друг от друга: каждое из них *的独特性* в *своем роде*. Как уже говорилось ранее, это означает, что инструменты должны искать не точные совпадения, а паттерны, которые могут быть уязвимостями, что приводит к более низкой точности как в сообщаемых результатах, так и в незарегистрированных элементах.

Если кто-то просит вас просканировать веб-приложение или использовать инструмент DAST, обычно он подразумевает программу сопоставления паттернов, ищущую уязвимости в приложениях. Такие инструменты обычно выполняют каталогизацию и фазз-тестирование, а не сопоставление паттернов и сценарные атаки.

Используемые в приложениях платформы похожи на инфраструктуру: они широко распространены, а об уязвимостях можно сообщать в централизованные репозитории. Они также версионны, и при загрузке определенной версии платформы, в которой находится известная уязвимость, автоматизированные инструменты обнаружат ее. Однако использование компонента или платформы стороннего производителя, имеющего известную уязвимость, не обязательно означает, что само приложение уязвимо, ведь эта уязвимая часть может работать только после вызова, который никогда выполняться не будет. Вместе с тем в приложении может содержаться другой код, который предотвращает использование данной уязвимости. Необходимо проверять такого рода результаты тестирования, даже если они поступают от надежного инструмента SCA.

**СОВЕТ.** Довольно часто возможности веб-прокси, фаззинга и автоматического «сканирования» DAST объединяются в одном инструменте. Для тестировщика безопасности подобное сочетание – просто находка, так как в идеале при проведении тестирования безопасности нужны все эти три возможности. Тем не менее существует несколько названий инструментов,

обладающих одной или несколькими из этих функций, что может запутать новичков. Если кто-то просит вас использовать сканер веб-приложения, DAST или сканер оценки уязвимости, скорее всего, он имеет в виду комбинированный инструмент.

## **Оценка уязвимости, оценка безопасности, пентестирование**

Зачастую, когда клиенты просят провести тест на проникновение, *на самом деле* они хотят получить оценку безопасности или оценку уязвимости (эти два термина часто используются как взаимозаменяемые. В дальнейшем до конца книги мы будем говорить «оценка безопасности»). Давайте разберемся в этом.

*Оценка безопасности* предполагает, что специалист по обеспечению безопасности пытается найти все возможные уязвимости или проблемы, связанные с безопасностью, проверяет правильную реализацию каждого элемента контроля безопасности, отсутствие в системе уязвимостей, определенных в моделях угроз, правильную работу различных ролей и настроек доступа, отсутствие возможности манипуляции бизнес-логикой и т. д. Оценка безопасности ищет все реальные и потенциальные проблемы, связанные с безопасностью, сообщает о них и предоставляет информацию для их устранения. Помимо всего прочего, отчет обычно содержит сведения о риске, который каждая проблема несет для компании и системы.

Поскольку обнаруженные уязвимости не были использованы для эксплойта, нельзя говорить об их полном *подтверждении*, поэтому при оценке безопасности возможно срабатывание ложной тревоги. Тестирование представляет меньший непосредственный риск (отсутствие эксплойта означает меньший потенциальный беспорядок), но в то же время его результаты менее точны. Тем не менее разработчики программного обеспечения часто могут сами подтвердить наличие проблем, если это необходимо.

**ПРИМЕЧАНИЕ.** Когда мы говорим об эксплойте или проникновении в систему, мы подразумеваем использование преимуществ системы не предусмотренным ее создателями способом. Это может быть сброс значений таблицы в базе данных, запуск скрипта, который не является частью системы,

повышение привилегий в системе (например, переход от обычного пользователя к суперпользователю), получение доступа к смежным нетестируемым системам и т. д. Можно также выполнить проверку концепции, чтобы доказать, что в систему можно проникнуть, например, для кражи учетных данных или отправки команд в базу данных.

*Тест на проникновение* немного отличается от оценки безопасности. Тестировщик пытается проникнуть (взломать или совершить эксплойт) в систему, чтобы доказать существование реальных, а не теоретических рисков. В некоторых тестах на проникновение от тестировщика требуется добыть трофеи (например, пароль или скриншот) или оставить после себя доказательства успешности своих действий, например файл или изображение. Значительная часть времени, отведенного на такое тестирование, уходит на разработку эксплойтов, что увеличивает время выполнения работы. Тестирование на проникновение часто называют сокращенно пентестированием или «этичным взломом».

Тесты на проникновение – это оценка безопасности *с добавлением* эксплуатации уязвимостей для доказательства их наличия.

Данный вид тестирования обладает большей известностью, чем любой другой вид деятельности по обеспечению безопасности приложений. Это объясняется как вниманием СМИ к «хакерству», так и тем, что, откровенно говоря, взлом системы может быть действительно захватывающим процессом. Когда человек часами трудился над достижением цели и результатом его работы становится утечка данных из системы, он может почувствовать себя сильным, умным и успешным. А что может быть приятнее такого ощущения на работе? Конечно, это лучше, чем заполнять формы запросов на изменения.

Тесты на проникновение иногда используются для того, чтобы «шокировать и устрашить» руководство или другие команды, которые не согласны со службой безопасности. Ничто не заставит кого-то поверить в опасность быстрее, чем демонстрация взлома одной из его систем.

Тем не менее пентестирование может усложнить отношения с коллегами. Если вы работаете в компании на постоянной основе,

лучше показать найденную уязвимость другой команде и предложить помочь в ее устраниении, а не демонстрировать руководству, насколько небезопасна система, которую вы только что взломали. Пчелы слетаются на мед, а не на уксус, и если вы не сторонний консультант, то такие действия могут оказаться и на вашей карьере, и на взаимоотношениях внутри организации.

Теперь, когда мы увидели различия, давайте рассмотрим сходство между оценкой безопасности и тестированием на проникновение. И то и другое требует наличия квалифицированного тестировщика безопасности, нескольких типов инструментов, комбинирования ручного и автоматизированного видов тестирования и достаточно большого количества времени. Цель обоих видов проверки – найти уязвимости в системах и сообщить о них клиенту. Оба подвергают тестируемую систему риску повреждения (при этом тестирование на проникновение более опасно), и оба предоставляют довольно много ценной информации для организации.

Возвращаясь к тому, с чего мы начали данный раздел: как клиенту выбрать между этими двумя видами тестирования? Именно здесь стоит показать себя экспертом, совету которого можно доверять: задайте клиенту вопросы, касающиеся того, каких целей он хочет достичь и какие риски готов принять, а затем предложите соответствующий вариант. В любом случае необходимо заранее объяснить, какие действия будут произведены при оценке или проникновении, а затем изложить эту информацию в письменном виде. Как всегда, независимо от выбранного варианта до начала тестирования следует убедиться, что были созданы резервные копии всех систем.

---

## НАНЯТЬ ХАКЕРА

Много лет назад Алиса работала в одном из тех мест, где произошел серьезный инцидент с безопасностью, приведший к утечке данных. Будучи руководителем, она не понимала технических деталей, но имела точное представление о последствиях этого инцидента для бизнеса: цена акций компании резко упала, и это было главной темой каждого заседания совета

директоров. Ей поручили «нанять хакера», чтобы выяснить, где еще может произойти утечка данных, и создать план по ее предотвращению. Алиса понятия не имела, с чего начать. Она позвонила нескольким друзьям, и один из них порекомендовал фирму, в которой работали *этичные хакеры*. Такое словосочетание она слышала впервые.

На встрече с консультантами Алису спросили о произошедшем и о целях ее обращения к ним. Она рассказала об инциденте и определила цель: ей необходимо знать, откуда еще злоумышленник может проникнуть в пять особо важных систем компании. Консультанты, в свою очередь, высказали мнение, что необходимо на всякий случай провести быструю проверку шести других общедоступных веб-приложений. Они предложили провести оценку безопасности пяти особо важных приложений, а не тестировать их на проникновение, поскольку у Алисы уже было понимание того, что к безопасности нужно относиться серьезно, и они хотели посвятить как можно больше времени поиску уязвимостей. Алиса сразу же согласилась и почувствовала огромное облегчение от того, что ей дали объяснения и предложили варианты. Теперь она могла вернуться к совету директоров, изложить ему план действий, не сомневаясь, что в данной ситуации компания предприняла все возможные меры.

---

## Гигиена безопасности

Проверка приложения на соответствие установленным политикам и стандартам в отношении шифрования и других конфигураций безопасности является очень важным и легко автоматизируемым тестированием.

Первое, что мы рассмотрим в этом разделе, – шифрование. Несколько бесплатных и платных инструментов анализируют настройки шифрования для платформ, на которых работают приложения, и передачу данных по сети. Сюда входят используемые протоколы и алгоритмы, длина ключей, сложность шифров и многое другое. При использовании публичного облака вряд ли возникнут серьезные проблемы, однако следует обратить внимание на применяемую в нем политику: чем она строже, тем выше степень защиты. Некоторые результаты сканирования системы, находящейся в традиционном центре обработки данных, скорее всего, крайне удивят проверяющего.

Второй вид анализа, представленный здесь, – проверка заголовков безопасности и настроек cookie. Как уже говорилось ранее, эти настройки подобны ремням безопасности для программного обеспечения: в экстренной ситуации они могут спасти, обеспечив еще один уровень защиты (однако чаще всего они не используются). Нужно всегда добавлять все применимые заголовки безопасности и проверять, правильно ли они установлены.

Иногда эти два вида тестов совмещаются внутри других инструментов. При выборе набора инструментов для тестирования следует ориентироваться на то, чтобы один или несколько из них проверяли шифрование, заголовки безопасности и настройки cookie.

Существует несколько способов проведения данных тестов.

- Посещение одного из бесплатных сайтов, который занимается таким тестированием, введение URL-адреса вручную, сохранение отчетов и отслеживание результатов. Такая задача будет хорошим летним проектом для студента, недавнего выпускника или младшего сотрудника и принесет отличные результаты. Можно воспользоваться такими высококачественными и бесплатными сайтами, как [securityheaders.com](http://securityheaders.com), [hardenize.io](http://hardenize.io) и [SSL labs](http://SSL Labs).

- Добавление тестов, проверяющих шифрование, заголовки безопасности и настройки cookie, в конвейеры CI/CD. Несколько бесплатных и платных продуктов могут помочь сделать это. Тесты проводятся быстро, а их результаты обладают высокой точностью.
- Использование инструмента SAST в базе кода для проверки настроек файлов cookies и заголовков безопасности.
- Проведение оценки уязвимости в инфраструктуре для проверки настроек протокола защиты транспортного уровня (англ. Transport Layer Security, TLS), обеспечивающего шифрование информации при передаче.
- Ручной обзор кода для проверки cookie и заголовков безопасности.
- Использование веб-прокси или инструмента DAST для проверки заголовков безопасности и cookie: большинство из них имеют соответствующие функции.

Заголовки безопасности, настройки безопасности cookie и использование передового опыта в шифровании – вот ваши козыри в решении вопросов, связанных с безопасностью.

---

## ОПАСНОЕ ТЕСТИРОВАНИЕ

Тестирование на отказ в обслуживании (англ. Denial-of-service, DoS) редко проводится в естественных условиях. Как правило, вы либо приобретаете защиту от DoS, либо нет. Такое тестирование может быть довольно опасным, а его результаты, как правило, оказываются негативными. Если продвинутый злоумышленник (например, целое государство) захочет провести распределенную DoS-атаку (англ. a distributed DoS, DDoS), то с большой вероятностью она окажется успешной, независимо от того, кто будет защитником. Тем не менее лишь немногие DoS-атаки осуществляются такими мощными группами, как государство, и поэтому защита против подобных атак принесет большую пользу. Ключевыми моментами здесь являются успешное сопротивление злоумышленникам среднего уровня и наличие плана на случай необходимости отражения продвинутой атаки.

---

## Стресс-тестирование и тестирование производительности

С помощью стресс-тестирования определяется, какую нагрузку (количество пользователей, трафик и т. д.) может выдержать приложение и при этом работать корректно. Оно позволяет определить, соответствуют ли верхние пределы системы *требованиям* (в соответствии с проектом и потребностями компании). Полезно знать, что система сможет выдержать ожидаемую максимальную нагрузку и даже больше.

Тестирование производительности – это измерение отзывчивости, надежности и качества работы приложения при определенной рабочей нагрузке. Например, если ожидается, что в вебинаре одновременно будут участвовать 20 000 сотрудников, необходимо протестировать устройство при нагрузке, немного превышающей это значение, и подтвердить, что в таких условиях оно работает на приемлемом уровне.

Некоторые компании предлагают тестирование на отказ в обслуживании, которое представляет собой смесь тестирования производительности и стресс-тестирования. Они гарантируют, что выведут систему из строя и покажут результаты такой атаки.

При наличии соответствующих инструментов вполне возможно провести стресс-тестирование и тестирование производительности своими силами. Однако необходимо проявлять крайнюю осторожность и использовать предварительную среду, а не эксплуатируемую, поскольку эти тесты могут привести к поломке других систем.

Если вспомнить наше предписание по защите конфиденциальности, целостности и доступности (триады CIA) систем и данных, то становится ясно, что эти тесты проверяют *доступность* систем.

Очень важно проводить стресс-тестирование или тестирование производительности любой системы, находящейся в общем доступе, в том числе и API.

## Интеграционное тестирование

Под интеграционным тестированием подразумевается объединение двух или более различных частей приложения и проверка их работы вместе. При интеграционном тестировании новый код загружается в основную часть кода и проверяется их общая работоспособность. Ко всему прочему, интеграционное тестирование означает подключение всей системы к другой системе и обеспечение их совместной работы. Все эти ситуации должны быть протестированы. При соединении двух измененных частей вместе существует возможность возникновения ошибок, в том числе тех, что связаны с обеспечением безопасности.

Точки интеграции между компонентами и системами часто имеют функции безопасности, которые можно проверить с помощью кода или любого из способов, описанных выше. Смысл интеграционного тестирования заключается в том, чтобы подтвердить, что две или более частей приложения, действуя вместе, не создают ошибок (безопасности).

---

## РАЗРАБОТКА НА ОСНОВЕ ГЛАВНОЙ ВЕТКИ

Боб до сих пор вспоминает младшего разработчика, которым он руководил. В течение примерно шести недель работы над проектом новичок был очень тихим, и Боб начал подозревать неладное. Когда он спросил разработчика, оказалось, что тот создал ветку кода проекта для себя и все эти шесть недель работал над собственными изменениями. Боб попросил его интегрировать созданный им код в общий – результаты были катастрофическими. Этот код содержал так много ошибок, что потребовалась целая неделя, чтобы объединить его с основной веткой. С тех пор Боб установил правило, согласно которому команда должна вести «разработку на основе главной ветки» (англ. Trunk-Based Development) и регулярно проводить загрузку кода, чтобы проверять его на совместимость с остальной частью приложения.

---

Интеграционное тестирование может проводиться как вручную, так и автоматизированно.

Ручное интеграционное тестирование чаще всего встречается в компаниях, работающих по каскадной (англ. Waterfall) и по гибкой (англ. Agile) моделям разработки. При каскадной модели разработки интеграция происходит на поздней стадии жизненного цикла проекта, на этапе тестирования. В гибкой модели интеграция доступна, но не обязательно реализуется с каждым добавлением новой функции или спринтом. Ручное интеграционное тестирование может быть выполнено по-разному, так как не существует установленного стандарта.

В среде DevOps для развертывания программного обеспечения используется конвейер непрерывной интеграции, непрерывной поставки и развертывания (англ. Continuous Integration / Continuous Delivery / Deployment, CI/CD). CI/CD-конвейеры интегрируют, testируют и развертывают программное обеспечение без вмешательства человека. Как правило, разработчики и команда эксплуатации создают собственные интеграционные тесты для добавления в конвейер, чтобы подтвердить, что их новый код хорошо сочетается со всем остальным приложением.

Более подробно мы обсудим CI/CD в разделе «Развертывание».

## Интерактивное тестирование безопасности приложений

Интерактивное тестирование безопасности приложений, сокращенно называемое IAST (англ. Interactive Application Security Testing), – это новый вид автоматизированного инструмента, созданного несколькими компаниями, занимающимися обеспечением безопасности приложений. IAST работает как развернутый в приложении агент, который следит за приложением и *во время его работы* сообщает о найденных уязвимостях. IAST может собирать информацию в ходе ручного и автоматизированного тестирования, а также в ходе использования приложения пользователем, находя уязвимости и сообщая о них в режиме реального времени (по мере их возникновения) [\[68\]](#).

IAST тестирует только фактически используемые части базы кода, то есть для исчерпывающего результата необходимо провести другие тесты и использовать все приложение. Идеальным решением здесь будет проведение IAST во время QA-тестирования и оценки безопасности.

IAST работает постоянно, следовательно, он может продолжать находить ошибки во время работы или обновления. Он также сочетает в себе просмотр как кода (статический подход), так и работающего приложения (динамический подход). IAST обычно используется для обозначения инструментария, а не действий, которые может выполнять человек.

Как и в случае с SAST, инструменты IAST из-за своей агенто-ориентированности сильно зависят от целевой технологии, в которой они используются. При выборе инструмента IAST можно применять те же правила, что и при выборе инструмента SAST, то есть стоит обратить свое внимание на тот, который поддерживает используемый стек технологий (язык программирования, платформу и протоколы), иначе могут возникнуть проблемы с интеграцией IAST в рабочую среду приложения, *а также* с нахождением максимального количества ошибок.

## Регрессионное тестирование

В ИТ-среде есть шутка, которая звучит примерно так.

Вопрос: Приложение имеет 10 ошибок, было исправлено 10. Сколько ошибок осталось?

Ответ: 1 или больше.

Суть шутки в том, что каждый раз, когда мы вносим изменения (включая исправления ошибок), существует вероятность возникновения новой проблемы. Как уже говорилось ранее, регрессионное тестирование – это повторное выполнение тестов с целью подтверждения, что работа программного обеспечения не была нарушена после внесения изменений.

Если в приложение вносятся серьезные изменения, необходимо провести регрессионное тестирование на функциональность, а также на безопасность. Если провести оценку безопасности, устраниТЬ найденные проблемы, а затем больше не тестировать приложение, в

которое и дальше будут вноситься изменения, то оно не будет считаться безопасным. Если приложение продолжает меняться, работа службы безопасности в нем не окончена, и именно поэтому мы проводим регрессионное тестирование.

По возможности автоматизируйте все тесты, которые будут повторяться, – обычно для этого используются модульные тесты.

## Тестирование инфраструктуры

Приложения живут в инфраструктуре: физические серверы, виртуальные машины (ВМ), контейнеры. Даже бессерверные приложения недолго работают в контейнере, прежде чем самоуничтожиться. В обязанности специалиста по безопасности приложений или разработчика обычно не входит исправление таких систем, но если приложение скомпрометировано из-за небезопасной инфраструктуры, это станет и его проблемой в том числе. В данной книге мы не станем рекомендовать вам создавать собственную инфраструктуру или проводить оценку безопасности сети, но выполнение базовых проверок может быть весьма полезным.

Все три типа инфраструктуры – физические серверы, виртуальные машины и контейнеры – работают под управлением операционных систем, которые могут быть просканированы на предмет отсутствия необходимых патчей, плохой конфигурации и других аномалий, требующих внимания разработчика (или команды эксплуатации).

При наличии автоматизированного конвейера развертывания, выпускающего инфраструктуру, стоит добавить соответствующее программное обеспечение для его сканирования. Сотрудники эксплуатационного отдела должны своевременно получать и использовать результаты проверки.

Например, если для размещения приложения применяются контейнеры, можно воспользоваться руководством по стандартам Центра интернет-безопасности (англ. Center for Internet Security, CIS) для усиления их защиты. Можно использовать несколько различных видов инструментов для сканирования контейнера на предмет соответствия его конфигурации стандарту CIS ([github.com/dev-sec/cis-docker-benchmark](https://github.com/dev-sec/cis-docker-benchmark)).

При ручном запуске программного обеспечения можно узнать у своей команды эксплуатации и службы безопасности, что им нужно для обеспечения безопасности платформы. Они могут ответить «ничего», а могут и поделиться отличными идеями. В ваших интересах сделать все от вас зависящее для повышения уровня безопасности в рамках инфраструктуры.

## Тестирование базы данных

Базы данных представляют собой инфраструктуру и управляют вашими данными. Как и любой другой сервер, они должны быть защищены, однако нужно также обеспечить безопасность самих данных.

Необходимо регулярно сканировать инфраструктуру с помощью сканера оценки уязвимостей.

Следует убедиться, что:

- каждая учетная запись имеет доступ только к тем областям и базам данных, к которым она должна иметь доступ;
- нельзя подключиться ни к одной части сервера без учетной записи;
- все порты закрыты, кроме необходимых для работы;
- сервер подключен к сети через модель нулевого доверия (все закрыто либо заблокировано, кроме приложений, которые он обслуживает);
  - на сервере нет другого программного обеспечения, кроме операционной системы сервера базы данных;
  - все данные засекречены и маркированы, вплоть до низового уровня (секретно, совершенно секретно или несекретно);
  - эксплуатационные данные не используются ни в каких других средах. Данные клиента ни в коем случае не должны использоваться для тестирования или разработки;
  - доступ контролируется и проверяется на предмет раскрытия конфиденциальных данных и ненадлежащего доступа (например, сотрудники не ищут в базах данных сведения о людях из своей личной жизни);
  - для каждого приложения используются уникальные сервисные учетные записи, и ни одна из них не наделена правами владельца базы данных или любыми другими чрезмерными разрешениями;
  - когда это возможно, используется подсервисная учетная запись, соответствующая типу операции: одна учетная запись предназначена для операций READ, а другая – для операций CREATE/UPDATE/DELETE с целью предотвращения изменения данных в случае компрометации пользователя с правами READ;
  - регулярно создаются резервные копии и практикуется откат;

- данные шифруются в пути (на пути к БД и обратно) и в состоянии покоя (вся БД шифруется как единое целое);
  - поля конфиденциальных данных зашифрованы в базе данных (второй уровень шифрования);
    - все возможные средства контроля и функции, связанные с обеспечением безопасности, включены в соответствии с руководством по усилению безопасности, если нет документально подтвержденной причины не делать этого;
    - сообщения об ошибках не содержат слишком много информации;
    - в настройках разрешения DNS установлено значение «только для внутреннего пользования»;
    - все остальное, что можно придумать. Этот список не является исчерпывающим.

Также можно обратиться к руководству от CIS, чтобы определить, какие меры по усилению безопасности можно применить к базе данных в дополнение к передовым практикам обеспечения безопасности, предоставленным поставщиком базы данных.

## Тестирование API и веб-серверов

API используются веб-приложениями для взаимодействия с веб-службами, ни одна из которых не имеет графического интерфейса пользователя (англ. graphical user interface, GUI). Они требуют такого же тестирования, как и обычные веб-приложения, за исключением тех частей, которые связаны с GUI. Тем не менее проверка API и веб-служб достаточно сложна по нескольким причинам.

Во-первых, как правило, API скрыты. Они не имеют графического интерфейса и поэтому видны только при вызове их из приложения (увидеть вызов можно лишь с помощью веб-прокси). Необходимо попытаться вручную перечислить API (найти их все), разобраться во всех функциях, и *только затем* можно приступать к тестированию – это большой объем работы. В идеале заказчик предоставляет как можно больше подробностей о системе (например, файл с определением API), однако все равно тестировщику приходится перечислять все самому, поскольку зачастую API больше, чем люди думают.

Во-вторых, на момент написания этой книги на рынке представлено мало автоматизированных инструментов тестирования безопасности, ориентированных на API, и мне еще не попадался тот, который работал бы так же хорошо, как современные сканеры веб-приложений (DAST). Использование инструмента DAST для поиска уязвимостей в веб-приложении не требует длительного обучения, API же обычно тестируется вручную посредством веб-прокси или инструмента, который может сам выполнять вызовы API. На данный момент существуют новые инструменты, которые ищут файлы определения API и тем самым облегчают тестирование, но полностью автоматизированного решения для поиска уязвимостей в этой области нет. Таким образом, тестирование API должен проводить человек с соответствующей подготовкой, что увеличивает стоимость выполнения данной задачи.

Для тестирования API следует составить запрос к нему для вызова какой-либо функции. Если вы знаете, как приложение выполняет этот вызов, просто скопируйте его. Если это API, на который вы наткнулись при выполнении перечисления, может потребоваться какое-то время, чтобы правильно оформить первый запрос. Сформировать запросы поможет файл определения. Как только у вас будет правильный запрос, начните отправлять «ненадлежащие» запросы, измененные, запросы с атаками и т. д. Пройдитесь по каждому API и всем функциям внутри него. Проверьте работоспособность функций, вызвав каждую из них с помощью различных HTTP-методов (GET, POST, PUT, DELETE и т. д.). Протестируйте пределы (насколько большим или маленьким может быть значение, тип, количество вызовов и т. д.). Попробуйте добавить «плохие» вводные данные и попытки атаки: одинарные кавычки, теги `<script>` и т. д. Сделайте все то же самое, что сделали бы с полем ввода веб-приложения, которое может вызывать API в бэкенде. Это ваш шанс обратиться непосредственно к API без вмешательства веб-приложения.

Как и любое ручное тестирование, проверка API таит в себе больше нюансов, чем мы упомянули, но со временем вы отточите этот навык. Запомните главное: API необходимо тестировать так же тщательно, как и любое веб-приложение, поскольку это не менее ценная мишень для атак.

## Тестирование интеграций

Там, где соединяются две системы, непременно должна быть функция, связанная с обеспечением безопасности, или средство контроля безопасности, поэтому необходимо проверять их наличие и работоспособность. Если приложение вызывает API, который запускает бессерверное приложение, обновляющее базу данных, необходимо протестировать каждую из этих интеграций.

Сначала следует убедиться, что между соединениями существует контроль безопасности (как минимум всегда должна быть аутентификация, *затем* авторизация), после чего проверить, правильно ли он настроен.

Во-вторых, если это возможно, необходимо подключиться ко второй системе и «пообщаться» с ней. Если ею является API, то можно вызвать его и следовать советам, данным в предыдущем разделе. Бессерверное приложение можно вызвать через логическое приложение (при его наличии) или собственноручно, а затем отправить ему различные значения и основательно поэкспериментировать с ним.

**ПРИМЕЧАНИЕ.** Логические приложения – это триггеры, вызывающие бессерверные приложения. Они срабатывают при наступлении определенных событий. Например, при превышении квоты ресурсов для API, при многократных попытках входа в систему за 24 часа либо при «слишком быстром» входе в систему (быстрее, чем это способен сделать человек).

Имеющиеся логические приложения можно запускать самостоятельно (если это возможно), чтобы убедиться в их правильной и бесперебойной работе. Необходимо проверить отсутствие ложных срабатываний.

В итоге: проверяйте пределы каждой части приложения и систем, от которых оно зависит, а также средства контроля безопасности между ними.

**СОВЕТ.** При тестировании логических приложений следует всегда спрашивать себя, кажутся ли значения триггеров подходящими. Возможно, 100 попыток входа в систему за 5 секунд – слишком большое значение настройки логического приложения? Может ли человек войти в систему 10 раз за 5 секунд? Скорее всего, нет. Следовательно, хороший тестировщик предположит, что настройку безопасности можно улучшить, снизив порог до 10 попыток входов. Любой совет по блокировке ботов будет иметь высокую ценность. Вас наняли за ваше экспертное мнение, так предоставьте же его заказчику.

## Тестирование сети

В обязанности разработчика программного обеспечения или специалиста по безопасности приложений не входит тестирование сети организации. Тем не менее можно и нужно проверять, соответствует ли система политике сетевой безопасности организации.

Если в компании используется зонирование (межсетевые экраны между различными областями сети, предполагающие доверие внутри каждой зоны), необходимо подтвердить правильную работу архитектуры приложения в том случае, если приложение находится в одной зоне, API – в другой, а база данных – в третьей. Разговор с командами эксплуатационного и сетевого отделов о потребностях приложения приведет к менее болезненному развертыванию в будущем.

Если компания применяет в своей сети модель нулевого доверия, необходимо убедиться в том, что все компоненты приложения следуют этой политике и соответствующим руководящим принципам.

Все необязательные для работы приложения порты должны быть закрыты. API, бессерверные приложения, вызовы баз данных и тому подобные компоненты должны выполняться только через сервисную учетную запись, выделенную для системы (ни в коем случае не через общую или личную). Списки доступа должны включать только необходимые сервисные и административные учетные записи. Остальные учетные записи должны быть заблокированы, а попытки подключения через них – зафиксированы с отправлением уведомления.

Система всегда должна быть спроектирована таким образом, чтобы не нарушать установленную политику сетевой безопасности. Сотрудничество с другими командами с самого начала разработки позволит избежать внесения изменений в архитектуру на поздних этапах проекта.

В целом нет необходимости самостоятельно проводить тестирование сетевой безопасности, но *следует* подтвердить, что система отвечает политике сетевой безопасности, а ее структура соответствует передовому опыту в данной области. Следование лучшим практикам и «более строгое» отношение к разработке, чем предполагает действующая политика на рабочем месте, скорее всего, будет воспринято положительно, поэтому не нужно бояться «переборщить с безопасностью». Пока приложение правильно работает и не мешает другим системам, все будет в порядке.

## Развертывание

Развертывание находится в одной главе с тестированием, потому что предоставляет большое количество возможностей для него. Эти два этапа жизненного цикла разработки системы тесно связаны между собой.

Развертывание означает «акт выпуска системы в среду» (обычно в эксплуатационную). Вместо слова «развертывание» часто используются такие термины, как «релиз», «выпуск», «публикация», «запуск», но все они подразумевают одно и то же: все элементы помещаются в нужное место, включаются – и приложение выводится в мир.

---

### ПРЕДУПРЕЖДЕНИЕ О ПРЕДВЗЯТОСТИ

Я положительно отношусь к автоматизации процесса развертывания, поскольку так можно избежать ошибок, сэкономить время и обеспечить *идеальную* повторяемость процесса. Всегда приятно чувствовать себя нужным на работе, однако еще приятнее знать, что все настроено таким образом,

чтобы в случае чрезвычайной ситуации системы можно было перезапустить нажатием одной кнопки.

---

Существует несколько различных способов развертывания приложения. Давайте рассмотрим их.

## Редактирование кода на сервере в реальном времени

В это трудно поверить, но многие системы до сих пор редактируются в режиме реального времени, непосредственно на сервере. Это очень опасно, так как любая ручная работа чревата ошибками, а резервная копия текущей версии кода отсутствует. Нельзя проверить код на наличие допущенных ошибок, так как он уже работает, а вместе с ним и все его ошибки. К тому же со временем приложение будет меняться, и, если по какой-то причине сервер перестанет работать или кто-то удалит файл либо отформатирует компьютер, оно исчезнет. Представьте, каково это – в одно мгновение потерять несколько лет работы.

Использование репозитория кода (иногда называемого библиотекой кода, контролем версий, контролем исходного кода) является передовой практикой в ИТ-отрасли. В репозитории хранится код вместе со всеми внесенными в него изменениями. В нем записывается, когда, кто, что изменил, и в любое время можно произвести откат приложения к предыдущей версии. В качестве бонусной функции некоторые репозитории кода могут также вводить код в эксплуатацию, регистрировать ошибки и многое другое. Репозиторий кода – лучший друг разработчика, и инженеру по безопасности приложений нужно быть хорошо с ним знакомым.

**ВНИМАНИЕ.** В репозитории кода не должны храниться секреты: пароли, строки подключения, хеши и все остальное, что необходимо приложению для работы и при этом должно оставаться скрытым. Если же секрет случайно попал в репозиторий, следует немедленно извлечь его оттуда и проанализировать причины произошедшего для предотвращения повторения инцидента.

Если кто-то в организации по привычке редактирует код на сервере в реальном времени, он создает угрозу безопасности. Отсутствие резервной копии кода, наличие у этого человека доступа для записи к производственным серверам, молчание системы предупреждения о нарушении безопасности в ответ на эти действия и сам факт

нарушения политики безопасности организации – все это достаточные причины, чтобы считать такое вмешательство инцидентом безопасности.

**ВНИМАНИЕ.** Ситуация, когда рабочий сервер может быть поврежден, отформатирован, затоплен или иным образом уничтожен, кажется маловероятной, но со мной такое случалось неоднократно. По мере продвижения к использованию публичного облака данная проблема появляется все реже, но лишь небольшой процент всех приложений в мире размещается в публичном облаке. Многие компании создают собственные центры обработки данных (по множеству причин). Независимо от того, где находится приложение, всегда нужно иметь самую последнюю копию в системе контроля версий.

## Публикация из среды IDE

Многие типы IDE позволяют разработчикам публиковать свои приложения в различных средах непосредственно с места написания кода. Данный процесс иногда ласково называют «ПКМ → Опубликовать», поскольку именно эти команды обычно используются для выполнения действия. Если разработчик уже выполнил свои модульные тесты, он выкладывает их в dev- или test-среду, и приложение становится частью большой системы. Данный процесс нередко является частью каскадной или гибкой модели разработки при исправлении ошибки или при переходе разработчика в среду разработки для выполнения тестирования до перемещения приложения в QA (среду обеспечения качества), UAT (среду для пользовательского приемочного тестирования), Preprod (копию эксплуатационной среды, которая часто используется для стресс-тестирования или тестирования производительности), а затем, наконец, prod. Сюда могут быть включены и другие среды.

По возможности следует убедиться, что разработчики выполняют модульные тесты перед отправкой кода куда-либо. Нет смысла в том, чтобы они отправили код в dev-отдел, провели там кучу тестов, а затем поняли, что половина модульных тестов не пройдена, и вернулись к проектированию. Если вы работаете в организации, которая

продвигает код таким образом, нужно будет очень постараться, чтобы получить согласие на добавление модульных тестов, ориентированных на установленные цели обеспечения безопасности.

Также необходимо добавлять тестирование безопасности к другим средам и этапам тестирования, с учетом наличия времени и загруженности других команд.

И, конечно, следует максимально автоматизировать тесты.

## «Самодельные» системы развертывания

Поскольку разработчики являются разработчиками, многие компании создают свои собственные системы развертывания, которые переносят изменения из среды в среду, обеспечивают получение различных разрешений и документируют все произведенные действия. Большой победой для всей команды будет добавление тестирования безопасности к любому из вышеперечисленных этапов. Например, пусть первая среда запускает сканирование состава программного обеспечения (SCA) и возвращает код разработчику при результатах «высокого» или «среднего» уровня важности. Если вам удастся заручиться поддержкой, вы сможете выполнить свою работу в рамках системы такого типа.

Однако пользовательская система развертывания обязательно должна проходить проверки безопасности. Если кто-то получит возможность манипулировать системой, он сможет опубликовать все, что пожелает, или добавить в базу кода лишнее. Необходимо рассматривать пользовательскую систему развертывания как критически важное приложение организации.

«Самодельные» системы развертывания должны быть созданы с учетом всех применяемых передовых практик в области разработки. Иногда разработчики забывают, что инструменты, которые они создают для своей работы, имеют значительную ценность для бизнеса. Как и любое другое ценное приложение в компании, системы развертывания должны иметь резервные копии, проходить тесты и отвечать требованиям безопасности жизненного цикла системы. Все действия с ними должны быть записаны в журнал, а код сохранен в репозитории.

## Ранбуки

Ранбуки – это инструкции (со скриншотами и скриптами), написанные командой разработчиков, которые передаются команде эксплуатации для развертывания приложения. Звучит здорово, но на практике все может оказаться сложнее. Часто приложение работает на компьютере разработчика, но не работает после его развертывания, в результате чего возникают проблемы. Приходится вручную следовать письменным инструкциям чрезвычайно сложного развертывания – это медленно и чревато ошибками.

Тем не менее, если разработчики и команда эксплуатации привыкли действовать по такой схеме, не нужно нарушать их налаженную работу. Убедитесь, что инструкции по всем функциям безопасности и их конфигурациям изложены в ранбуке как можно более подробно.

## Непрерывная интеграция, непрерывная поставка, непрерывное развертывание

Непрерывная интеграция и непрерывная поставка, также известная как CI/CD, фактически представляют собой три различные концепции в рамках одной системы, что можно увидеть на рис. 6.1.



Рис. 6.1. Непрерывная поставка и непрерывная интеграция (CI/CD)

*Непрерывная интеграция* – это практика регулярного слияния всех изменений разработчиков в основную ветвь кода с проведением

тестирования на общую работоспособность, которая, в идеале, проводится несколько раз в день. Непрерывную интеграцию можно выполнять вручную, но обычно для этого используется какой-нибудь инструмент.

*Непрерывная поставка* отвечает за очень быстрый выпуск изменений. Система всегда должна быть готова к автоматическому развертыванию (которое включает в себя все элементы непрерывной интеграции с добавлением тестирования). Непрерывная поставка требует использования автоматизированного инструмента, обычно называемого конвейером. Автоматизация ускоряет процесс, исключает человеческий фактор и позволяет добавлять автоматизированные тесты, что повышает качество кода. Такое программное обеспечение часто называют конвейером CI/CD или конвейером DevOps, а каждый новый выпуск – сборкой.

*Непрерывное развертывание* – это практика, позволяющая конвейеру CI/CD выпускать новые изменения в эксплуатационную среду без этапа ручного утверждения. Результаты автоматизированных тестов решают, будут ли выпущены изменения, а если какой-нибудь тест проваливается, развертывание останавливается и происходит так называемое прерывание сборки. Непрерывное развертывание не является обязательным компонентом для использования конвейера CI/CD и часто считается признаком продвинутости компании, использующей методологию DevOps.

Конвейеры CI/CD являются наиболее известной особенностью методологии DevOps, а цели их применения совпадают с нашими: снижение количества ошибок (безопасности), вызванных человеческим фактором, более быстрая обратная связь (по вопросам безопасности), возможность быстрого внедрения исправления ошибок (безопасности), автоматизация тестирования (безопасности) и снижение риска злонамеренной эксплуатации уязвимостей за счет регулярного выпуска множества мелких изменений вместо крупных, но редких патчей.

Если в организации используется конвейер CI/CD, нужно добавить в него процедуры по обеспечению безопасности. При этом необходимо уважать пожелания разработчиков и команды эксплуатации и использовать только выделенные ими время и пространство. Если они говорят, что тестирование можно проводить в течение восьми минут

на сборку, необходимо убедиться, что оно длится не дольше указанного времени.

Вот несколько идей для начала работы с конвейером CI/CD.

- Сканирование только что загруженного кода на предмет наличия в нем элементов, похожих на секреты. Данное тестирование проводится быстро, потому что в его ходе проверяется только дельта (измененный или новый код), а обнаружение секрета принесет огромную пользу организации и предотвратит катастрофу, связанную с обеспечением безопасности. В идеале при обнаружении секрета загрузка кода должна быть остановлена. Если блокировка невозможна, необходимо запустить процедуру управления инцидентами безопасности и немедленно удалить секрет.

- Сканирование посредством SCA с целью подтверждения, что в приложение не были добавлены новые компоненты с известной уязвимостью. Данная проверка также выполняется очень быстро, поскольку заключается в простом просмотре списков пакетов приложения с последующим сравнением их с черным списком. Сканирование посредством SCA имеет чрезвычайно высокую точность, что делает его идеальным инструментом для конвейерного тестирования.

Темой книги является не DevOps, но если в компании используется именно эта методология, то для достижения поставленных целей необходимо подстроить свою работу под ее принципы. Если применяются спринты, их тоже нельзя игнорировать. При использовании конвейеров необходимо включить в каждый из них по крайней мере один тест безопасности. Если вы не будете идти в ногу с отделами разработки и эксплуатации, *то останетесь позади*.

## Упражнения

1. Если бы вы могли выбрать только один вид тестирования для своего приложения, какой бы вы выбрали и почему?
2. Как вы думаете, какой вид тестирования будет самым быстрым? Почему?
3. Как вы думаете, какой вид тестирования будет самым медленным? Почему?

4. Какие типы уязвимостей вы бы искали при регрессионном тестировании? Назовите как минимум два типа и объясните свой выбор.

5. Сеть вашей компании спроектирована с учетом принципа нулевого уровня доверия? Если вы не знаете ответа, ваше домашнее задание – выяснить это.

6. Поддерживает ли ваша компания использование конвейера CI/CD? Если вы не знаете ответа, ваше домашнее задание – выяснить это.

7. Следует ли в среде CI/CD применять инструмент статического тестирования безопасности приложений (SAST) и проводить полное сканирование всего кода при каждой новой сборке? Почему?

8. Почему очень важно помещать все новые изменения в репозиторий кода?

9. Зачем мы тестируем точки интеграции между различными системами? Это лучше, чем полное тестирование каждой системы?

10. Почему мы тестируем базы данных, даже если они находятся в закрытом доступе?

11. Почему мы тестируем API, даже если они находятся в закрытом доступе? (Возможно, вопрос с подвохом.)

12. Когда имеет смысл проводить тестирование на проникновение, а не оценку безопасности системы? Поясните свой ответ.

## Глава 7

# Программы безопасности приложений

В этой главе мы обсудим программы безопасности приложений (англ. Application Security, AppSec), которые представляют собой формализацию деятельности по обеспечению безопасности как части жизненного цикла разработки системы. Программа безопасности приложения не является компонентом ПО: это набор взаимосвязанных действий, имеющих конкретную долгосрочную цель.

В данном случае наша цель – обеспечить безопасность создаваемого и поддерживаемого нами программного обеспечения. Мы стремимся к снижению рисков для организации и для тех, кого обслуживаем (клиентов, сотрудников, граждан и т. д.). Мы защищаем конфиденциальность, целостность и доступность систем и данных, находящихся в нашем ведении.

Мы создаем официальные программы безопасности приложения, чтобы усовершенствовать политику безопасности, обеспечить защиту *всех* наших приложений (без исключения) и доказать другим, что мы предприняли все возможные действия для защиты организации. Без официальной программы нельзя с уверенностью заявлять, что мы производим надежное ПО, которое можно безопасно разместить в интернете, а если произойдет крупный взлом системы, мы практически не сможем себя защитить.

Под программой подразумевается выполняемая нами деятельность: моделирование угроз для каждого нового проекта, рассмотрение всех запросов, касающихся проблем обеспечения безопасности, добавление проверок безопасности в конвейер и т. д. Идея программы заключается в формализации этих действий, то есть они официально становятся частью подхода организации к созданию ПО. Невозможно «пропустить» этап обеспечения безопасности в жизненном цикле разработки системы, потому что он зафиксирован в программе и тем самым является обязательным.

# ТРЕБОВАНИЯ ПРОГРАММЫ БЕЗОПАСНОСТИ ПРИЛОЖЕНИЙ

- В идеале программу осуществляет специально собранная команда или по крайней мере отдельный специалист.
  - Деятельность по обеспечению безопасности добавляется в жизненный цикл разработки системы.
  - Команда по безопасности приложений снабжает и поддерживает разработчиков всеми необходимыми ресурсами, инструментами, обучением, политикой, руководством.
  - Команда по безопасности приложений проверяет соблюдение безопасности жизненного цикла разработки системы и безопасность производимого программного обеспечения (тестирование и инструментарий).

## Задачи программы по защите приложения

В настоящее время не существует общепринятого стандарта того, что должно входить в программу безопасности приложений. Почти всегда она выполняется наугад. Существует множество общих мероприятий, однако нельзя утверждать, что они являются лучшими решениями для каждого конкретного случая. Для определения оптимального варианта действий одни организации используют модели OWASP SAMM или BSIMM, в то время как другие принимают решение самостоятельно. В естественных условиях программы безопасности приложений сильно отличаются друг от друга: некоторые из них очень подробны и включают в себя множество мероприятий, а другие представляют собой беглые проверки, осуществляемые исключительно для того, чтобы соответствовать нормам либо законам. Есть и такие редкие жемчужины, для которых активная культура обеспечения безопасности является приоритетом.

**СОВЕТ.** Проект OWASP Self Assessment Maturity Model (SAMM) чрезвычайно полезен при запуске программы безопасности приложений с нуля. Технически эта методология «не одобрена» и не признана стандартом, однако группа

экспертов поддерживает ее в качестве платформы для создания собственного безопасного жизненного цикла разработки системы. (Источник: [owasp.org/owasp\\_samm/model](http://owasp.org/owasp_samm/model).)

Методология Building Security in Maturity Model (BSIMM) предоставляет исследования, основанные на текущих тенденциях в ИТ-отрасли, тем самым помогая компаниям в поисках тех видов деятельности, которые позволят им получить наилучшую окупаемость инвестиций. (Источник: [www.bsimm.com/framework.html](http://www.bsimm.com/framework.html).)

Важно, чтобы создание собственной программы безопасности приложений проходило в условиях максимальной эффективности и результативности, поскольку у вас никогда не будет достаточно ресурсов, бюджета или времени, чтобы осуществить все задуманное. Если только вы не работаете в условиях чрезвычайно высокого риска или в другой особой ситуации, бизнес-требования всегда будут стоять выше обеспечения безопасности, что ограничивает работу команды безопасности.

Выбирая мероприятия для программы безопасности приложений, необходимо составить список целей, а затем на его основе определить действия и инструменты, которые будут наиболее эффективны в рамках существующих в организации систем. Цели ставятся с учетом приоритетов компании, чтобы обеспечить максимальную защиту наиболее важных для бизнеса элементов.

Исходя из этого, давайте рассмотрим некоторые высокоуровневые цели программы, которые мы могли бы поставить перед собой и работать над их достижением.

## **Создание и поддержка реестра приложения**

Нельзя защитить что-то, о наличии чего мы не знаем. Именно по этой причине проводится инвентаризация всех приложений и других активов, включая API и базы данных. Большинство крупных компаний имеют множество приложений, о которых они не знают. Эти неучтенные приложения не получают должного внимания со стороны службы безопасности и, следовательно, представляют собой брешь в защите всей компании.

Возможно, вы удивитесь, узнав, что создание и ведение реестра приложений является одной из самых сложных задач в сфере ИТ. Разработчики нередко выпускают обновления или API без документации, старые приложения теряют актуальность, бизнес-подразделения нанимают теневых ИТ-специалистов, которые не соблюдают никаких правил безопасности, продукты SaaS работают в сети без чьего-либо ведома. Многие организации, которые, как вы, вероятно, полагаете, очень надежны, в настоящее время отслеживают свои приложения вручную в таблице Microsoft Excel, пока вы читаете эти строки.

Самая важная причина, по которой вы как специалист в области безопасности приложений должны провести инвентаризацию своих программных активов, заключается в необходимости обеспечить внимание к безопасности каждого приложения. Если все приложения должны пройти проверку безопасности кода перед запуском в эксплуатацию, то вам необходимо знать *о каждом из них*.

**СОВЕТ.** Использование базы данных управления конфигурацией (англ. Configuration management database, CMDB) для хранения реестра является отличным способом наладить взаимодействие с остальными ИТ-отделами. Нужно, чтобы все команды могли найти информацию о находящихся под вашей ответственностью приложениях, и наоборот. Использование общего инструмента создаст почву для укрепления доверия и обмена сведениями между командами.

Злоумышленники всегда будут искать самое слабое звено компании. Приложения, которым не уделялось никакого внимания с точки зрения безопасности, скорее всего, окажутся таким слабым звеном.

*В больших организациях теневые информационные технологии относятся к системам информационных технологий (IT), развернутым отделами, отличными от центрального IT-отдела (обычно) для обхода недостатков центральных информационных систем или IT-отдела.*

## **Техническая возможность обнаружения уязвимостей в написанном, выполняемом и стороннем коде**

После внимательного изучения видов инструментов проверки безопасности можно сделать вывод, что они делятся на DAST или SAST.

DAST означает *динамическое* (*dynamic*) тестирование безопасности приложений – взаимодействие с приложением в процессе его работы.

SAST означает *статическое* (*static*) тестирование безопасности приложений – изучение кода в том виде, в котором он написан или скомпилирован.

Как правило, инженеры по безопасности и программисты обнаруживают уязвимости в стороннем коде с помощью инструмента анализа состава программного обеспечения (SCA), который исследует библиотеки, платформы и зависимости на предмет того, входят ли они в заранее составленный список «известного уязвимого» ПО. Такой анализ считается *статическим*, поскольку инструмент SCA исследует библиотеки, вызываемые из написанного кода (в частности, из файла дескриптора проекта).

Инструменты SAST находят уязвимости в пользовательском коде, то есть в коде, написанном командой компании. SCA обнаруживает известные уязвимости во всем коде приложения, который был написан другими людьми (библиотеки, пакеты, платформы и т. д.). Такие части кода, как правило, называются «сторонние библиотеки». Найти уязвимости во всем приложении можно, только выполнив обе проверки.

Мы рассматриваем SCA или обнаружение уязвимостей в стороннем коде с точки зрения потребителя – программиста или инженера по безопасности, – поэтому нередко будем условно объединять его с остальной частью SAST – *статическим* тестированием безопасности приложений.

Инструменты DAST тестируют все приложение: и написанные командой компании, и сторонние части, поскольку они работают как единое целое.

Когда дело доходит до поиска ошибок, необходимо также провести ручной анализ. По части обнаружения уязвимостей опытный

тестировщик или обзорщик кода превосходит любой инструмент. Часто при автоматизированном тестировании пропускаются недостатки бизнес-логики, которые могут причинить компании довольно много вреда. Следовательно, необходимо выделить в графике проекта достаточно времени и нанять квалифицированных специалистов по безопасности, чтобы обеспечить обнаружение максимального количества уязвимостей.

Вернемся к цели: техническая возможность находить уязвимости в написанном, выполняемом и стороннем коде. Если выявлять недостатки тремя различными способами, а не только одним, то получится более полная картина состояния безопасности приложений. Нередко компании полагаются только на один из этих трех столпов, что может быть следствием бюджетных ограничений, недостаточного количества сотрудников или отсутствия поддержки со стороны руководства. В таком случае следует выбрать тот метод, который обеспечивает наилучшую окупаемость инвестиций и вписывается в текущие процессы.

## **Знания и ресурсы, необходимые для исправления уязвимостей**

Если у вас когда-нибудь будет возможность пообщаться с опытным тестировщиком на проникновение, он в конце концов признается вам, что самое неприятное в его работе – это вернуться в приложение через год и найти точно такие же уязвимости в тех же системах. Год за годом ничего не исправляется. Компания нанимает тестировщика, чтобы просто поставить галочку о выполнении требования регламента (обычно PCI), совершенно не заботясь о реальном повышении безопасности приложения.

Нет смысла усердно работать над поиском уязвимостей, если их в дальнейшем не устранит. Необходимо позаботиться о наличии в компании людей, обладающих достаточным количеством времени, навыков и знаний для исправления обнаруженных проблем. Если у человека много времени, но нет опыта программирования, ему потребуется немало методической помощи и ресурсов. У высококвалифицированных и обученных людей, которые перегружены

работой и уже близки к выгоранию, также не будет времени на устранение уязвимостей.

**ПРИМЕЧАНИЕ.** Компании добавляют дополнительные проверки безопасности в жизненный цикл разработки системы, исходя из самых лучших намерений, однако они не осознают, насколько тем самым могут увеличить объем работы. Очень важно, чтобы в графике проекта было выделено время на устранение проблем, *а также* чтобы сотрудники имели соответствующую подготовку для выполнения этой задачи.

## Образование и справочные материалы

Когда речь заходит о безопасности, необходимо иметь в виду, что разработчики программного обеспечения нуждаются в поддержке, так как обучение, получаемое на курсах программирования, в колледжах и университетах, часто не полностью соответствует требуемым знаниям и умениям. Необходимо обучить сотрудников имеющимся в компании стандартам, политикам или рекомендациям, чтобы они смогли создавать код, который отвечает всем требованиям.

Обучение может проводиться в виде семинара, обмена мнениями в обеденное время, онлайн- и офлайн-курсов, пятиминутного выступления на общем собрании персонала, занятий с приглашенным инструктором и т. д. Суть в том, что, если вам нужно, чтобы разработчики знали об определенном элементе управления безопасностью, способе защиты или методе кодирования, нужно помочь им эти знания приобрести. В противном случае одни сотрудники будут обладать всей необходимой информацией, а другие – нет, что обернется увеличением количества уязвимостей, связанных с обеспечением безопасности разрабатываемых продуктов.

Например, обеспечением справочными материалами может быть покупка книги по безопасности Java, если разработчики в основном используют этот язык в своей работе. Также разработчикам могут быть полезны подписки на видео, блоги, образцы безопасного кода и все то, что поможет им узнать, как сделать свою работу более безопасной.

**СОВЕТ.** Я создала академию и сообщество We Hack Purple ([WeHackPurple.com](http://WeHackPurple.com)) – организацию, которая занимается обучением безопасности приложений, безопасному кодированию, безопасности облачных вычислений и многому другому. Если вам нравится эта книга, возможно, вы захотите заглянуть к нам?

## Предоставление разработчикам инструментов по обеспечению безопасности приложений

Как мы помним из первой главы, чем раньше в жизненном цикле разработки системы мы обнаружим ошибку или недостаток в системе безопасности, тем дешевле, проще и быстрее будет ее исправить. Начало обеспечения безопасности на более ранних стадиях называется сдвигом влево. Если разработчики пользуются инструментами по обеспечению безопасности, они найдут и устранит некоторые проблемы еще на этапе кодирования, то есть до этапа тестирования (когда нас, скорее всего, пригласят для выполнения действий по обеспечению безопасности). Это большая победа!

Еще одна причина, по которой следует предоставить разработчикам инструменты безопасности, – это цифры. На каждого специалиста по обеспечению безопасности приходится (в среднем) 100 или более разработчиков. То есть у разработчиков больше времени, но, как правило, меньше навыков (связанных с обеспечением безопасности). Обучение всех разработчиков в команде пользованию даже одним инструментом может дать потрясающие результаты при минимальной загрузке людей.

Поддерживая интерес разработчиков к безопасности, мы можем превратить их в чемпионов безопасности, которые возьмут на себя еще больше ответственности в данной области. Дополнительное обучение чемпионов безопасности, чтобы они могли выполнять обзор безопасности кода, писать негативные модульные тесты, выполнять более продвинутое тестирование и многое другое, – вот пример эффективного использования времени и ресурсов.

Разработчики программного обеспечения – это строители, и большинство из них гордятся своей работой: они *хотят* создавать безопасные приложения. Предоставление им инструментов, обучение тому, как ими пользоваться, и безопасная среда могут поспособствовать созданию высококачественного программного обеспечения, к которому они стремятся.

**СОВЕТ.** Поиск уязвимостей можно превратить в увлекательное занятие, создав систему поощрений и баллов для

разработчиков, которые найдут и устроят больше недостатков, чем другие. Одобрение и награда – хороший стимул. Работа, которая не является очередной задачей в списке, может быть приятной, познавательной и продуктивной.

## **Проведение одного или нескольких мероприятий по обеспечению безопасности на каждом этапе жизненного цикла разработки системы**

Необходимо формализовать (предписать или создать политику для поддержки) деятельность по обеспечению безопасности, которую вы хотите видеть частью жизненного цикла разработки системы. Если необходимо, чтобы каждый новый проект включал установленные требования безопасности, напишите политику для поддержки этого условия работы. При этом в расписании проектов также должно быть выделено время для этих действий, связанных с обеспечением безопасности, иначе их не удастся выполнить.

Ниже приведены примеры действий, которые можно осуществить на каждом из этапов жизненного цикла разработки системы (независимо от используемой методологии). Выбор любого из них на каждом этапе цикла повысит безопасность конечного продукта.

- **Требования:** требования безопасности в соответствии со второй главой, пользовательские истории безопасности, оценка безопасности и утверждение предлагаемого стека технологий.

- **Проектирование:** моделирование угроз, создание досок с планом поиска недостатков в архитектуре, оценка безопасности структуры приложения (в соответствии с третьей главой).

- **Кодирование:** линтеры, написание модульных тестов безопасности, сканирование инструментами SAST или SCA, написание безопасного кода (согласно четвертой главе), использование плагинов IDE или вебхуков для исправления небезопасных частей написанного кода, использование известных безопасных паттернов и/или библиотек.

- **Тестирование:** сканирование инструментами DAST, SAST и SCA, проведение оценки безопасности или теста на проникновение, проверка безопасности кода (также инструментом SAST), сканирование на наличие секретов плюс что-нибудь из шестой главы.

- **Развертывание, релиз и поддержка:** DAST, RASP или WAF, мониторинг, ведение журналов, оповещения безопасности, обученная и готовая к работе команда реагирования на инциденты и установленный процесс развертывания.

Этот список не является исчерпывающим, его можно расширять практически бесконечно. Основной момент здесь заключается в том, что мероприятия выбираются самостоятельно, и одним из главных условий является их регулярное проведение.

**СОВЕТ.** По возможности поддерживайте вашу политику паттернами, примерами кода, документацией и всем, что только можно придумать, чтобы помочь другим командам соблюдать установленные требования.

## **Внедрение полезных и эффективных инструментов**

Выбор, приобретение и внедрение инструментов обеспечения безопасности требуют времени, энергии и опыта. Не нужно взваливать это на разработчика. Не все инструменты одинаково полезны, ведь каждый из них имеет свои плюсы и минусы. В этой книге не будут предлагаться конкретные продукты или бренды. Лучший способ сделать выбор – провести проверку концепции (англ. Proof of concept, РОС) того или иного инструмента и протестировать его самостоятельно. Желательно подключить к данному процессу разработчиков, чтобы они сами посмотрели, подходит ли им предлагаемый вариант. В конце концов, именно для них это делается.

По возможности автоматизируйте рабочий процесс выбранных вами инструментов (часто в этом может помочь CI/CD). Ручная работа предполагает ошибки, бесполезную потерю времени и скуку. Если инструмент будет использоваться десятки, сотни или тысячи раз, не следует запускать его вручную.

**СОВЕТ.** Создайте базу знаний по теме обеспечения безопасности, чтобы разработчики могли централизованно перенимать чужой опыт и делиться своим. Данная деятельность приобретает все большую ценность, и через несколько лет нельзя будет представить жизни без нее!

## **Команда реагирования на инциденты, которая знает, когда вам звонить**

Инциденты безопасности являются самым дорогим, разрушительным и унизительным способом обнаружить уязвимость в приложении. Инцидент безопасности означает, что приложение подверглось атаке, которая могла повлечь за собой утечку данных, отказ либо недоступность систем, нарушение целостности данных, несанкционированный доступ или любые другие негативные последствия. Мы как ИТ-специалисты должны сделать все возможное для предотвращения всех видов инцидентов, связанных с ИТ-безопасностью. Необходимо также быть готовыми к реагированию на инциденты, если к нам обратятся за помощью. Ответные меры в связи с инцидентом безопасности называются *реагированием на инцидент* (англ. *incident response*, IR) и включают в себя управление ситуацией и коммуникацией между отделами, расследование произошедшего, снижение ущерба и затрат, устранение проблемы и восстановление систем и данных. Также сюда входят вскрытие инцидента и документирование произошедшего в соответствующем отчете.

В большинстве крупных компаний есть обученные группы реагирования на инциденты. Стартапу с пятью сотрудниками такая команда не нужна, а вот корпорации без нее не обойтись.

---

### **ЗНАНИЕ ЛУЧШЕ НЕЗНАНИЯ**

Алиса вспоминает, как она впервые наняла в отдел информационных технологий студента, который был помешан на безопасности. Он был сыном друга, и его взяли на работу в качестве одолжения. В итоге в первый же месяц он сообщил о шести инцидентах безопасности, что расстроило многих сотрудников ИТ-отдела. Алиса отвела молодого человека в сторону и сказала ему: «Пока ТЫ не пришел, у нас никогда не было никаких инцидентов, связанных с безопасностью!» В ответ она услышала следующее: «Они происходили месяцами, а вы попросту о них не знали. Теперь их можно предотвратить. Мы

будем в большей безопасности, потому что знаем, что происходит». Алиса была поражена. В начале разговора она собиралась приказать ему прекратить поиски, но поняла свою неправоту. Ведь и правда гораздо лучше знать о чем-то, чем не знать.

---

Возможно, это вас удивит, но на момент написания этой книги среднее время обнаружения утечки данных составляет 197 дней<sup>[70]</sup>. Прямо сейчас у вас может происходить одна или несколько утечек данных или какой-нибудь другой инцидент, связанный с безопасностью, о котором вы еще не знаете. Без активных поисков проблем вероятность их обнаружения снижается.

Если в команде реагирования на инциденты никто не разбирается в безопасности приложений или программном обеспечении в целом, реагирование и расследование таких инцидентов неизбежно станут вашей обязанностью. В идеале вы должны представиться команде, рассказать им о своих навыках работы по обеспечению безопасности и предложить им помочь и обучение.

Если вы сможете научить команду реагирования на инциденты выявлять инциденты, связанные с приложением, и объяснить, в каких именно случаях следует к вам обращаться, ваша помощь будет намного полезнее.

В идеальном мире приложения должны регистрировать происходящие события, связанные с безопасностью (предполагаемые инциденты безопасности), и своевременно оповещать о них. По крайней мере один человек должен регулярно просматривать эти сообщения. При наличии операционного центра безопасности (англ. Security Operations Center, SOC) и системы управления информацией о безопасности и событиями безопасности (англ. Security information and event management, SIEM) или другого мониторингового инструмента было бы полезным, чтобы журналы приложений поступали в эту систему. *Необходимо объяснить сотрудникам центра, какие оповещения вас интересуют.* Нет никакой пользы от оповещений, если их никто не просматривает. Принимать уведомления лучше даже

при работе с простым бессерверным приложением, которое запускает электронное письмо, получаемое на следующее утро.

## Постоянное совершенствование программы на основе показателей, экспериментов и обратной связи

Последняя цель программы может показаться очевидной и ненужной, однако часто про нее забывают. Например, нам выставили жесткие сроки, а потом потребовалась новая функция, а потом... И вот с командой разработчиков не проводили встречи уже шесть месяцев, и они решили, что про них забыли, или мы целый год не смотрели на метрики, и оказалось, что рабочая энергия направлена не в то русло. Все эти потенциальные проблемы могут ускользнуть от нас, если мы не предполагаем, что они произойдут.

Рассмотрим немного подробнее данную цель.

### Метрики

**Метрики** – это метод измерения чего-либо или полученные результаты измерений<sup>[71]</sup>.

Необходимо измерять эффективность нашей деятельности. Если не хватает времени на выполнение всех действий или задач для создания успешной программы безопасности, следует выбрать самые результативные из них.

Измерения можно проводить различными способами: использовать систему управления уязвимостями, базу данных, SaaS для бизнес-аналитики или даже обычную таблицу MS Excel.

Вот что следует учесть при сборе метрик.

- Измерения должны проводиться регулярно. Нельзя просто делать это «время от времени»: данные будут неполными.

- Нужно применять метод измерения последовательно. При использовании инструментов DAST и SAST следует использовать оба во всех приложениях. Нельзя применить DAST в одной половине приложений, а SAST – в другой, если хотите иметь возможность сравнить результаты всех приложений и команды. Можно проводить частичные измерения (например, когда измеряется степень покрытия с

помощью определенного инструмента, то есть 20 % с помощью SAST и 80 % с помощью DAST), но только если вы учитываете это условие при анализе данных. В этом примере имеется в виду сравнение всех результатов SAST отдельно от результатов DAST или использование статистики охвата для корректировки их значений в наборе данных.

- Шкала измерений также должна быть последовательной. Нельзя сопоставлять яблоки и апельсины. Если один инструмент оценивает уязвимости от 1 до 10, а другой использует значения «низкий», «средний», «высокий» и «критический уровень», то необходимо определить, как перевести одно из измерений в форму другого измерения. Например, низкий уровень – это 2, средний – 4, высокий – 8, критический – 10.

- Необходимо измерять показатели, которые *важны*, а не метрики тщеславия.

*Метрики тщеславия* (англ. vanity metrics) – это, как правило, те, которые легко измерить, но которые не представляют особой ценности. Примером *важной метрики* может быть измерение прогресса. Предположим, вы используете два разных вида деятельности или инструмента для достижения цели. Если вы измерите оба показателя и увидите, какой из них более эффективен, то, возможно, решите отказаться от наименее полезного и сосредоточить усилия на более результативном. Но без метрик вы никогда об этом не узнаете.

Вот примеры показателей, которые могут быть важны для вас.

- Повторяются ли инциденты (по одной и той же причине)?
- Сколько времени требуется для устранения уязвимостей (откат, исправление или устранение)?
- Сколько времени требуется для выявления инцидентов?
- Сколько произошло случаев несоблюдения политики?
- Есть ли ошибки в обработке инцидентов?
- Насколько вы близки к достижению цели по обеспечению безопасности?
  - Выполняет ли команда безопасности свои обязанности, определенные соглашением об уровне обслуживания, или другие команды постоянно находятся в ожидании?
  - Если проводилось обучение по конкретным уязвимостям, уменьшилось ли их количество?

- Всегда ли соблюдается процедура реагирования на инциденты?
- Стало ли больше сообщений об уязвимостях? Причина в том, что у вас появился новый инструмент (и это хорошо), или в том, что у вас новые сотрудники, которые не прошли обучение (и это плохо)?

Список того, что можно измерить, очень велик. Следует выбирать те метрики, которые помогут достичь целей программы, а не те, которые пускают пыль в глаза.

---

## НАПРАВЬТЕ СВОИ УСИЛИЯ В НУЖНОЕ РУСЛО

Много лет назад, когда Алиса занималась разработкой контента, она пришла в компанию, где измеряли все: каждый клик, каждый пост в социальных сетях, эффективность работы каждого сотрудника. Алиса отвечала за написание статей, которые помогали людям лучше использовать их продукты, и ей нравилась ее работа. Она создавала в два раза больше контента, чем все остальные. Она знала это, поскольку у них были панели мониторинга, где она могла видеть результаты работы всех сотрудников, что одновременно и ободряло, и держало в напряжении. Алиса всегда хотела быть «лучшей» во всем, что делает. Она заметила, что ее коллега получал около 5000 кликов по каждой ссылке, которую он размещал в социальных сетях, а она – в среднем лишь 300–400. Алиса не понимала, что делает не так, – ведь она старалась. Она решила провести еще несколько собственных измерений: в том числе сколько времени люди оставались на странице. Оказалось, что читатели ее коллеги в среднем тратили одну-две секунды на клик, то есть никто не читал ни одной его статьи. Читатели Алисы задерживались в среднем на полторы минуты! Это означало, что почти всегда они читали статью целиком. Воодушевленная Алиса сообщила о результатах своему начальнику, вследствие чего способ публикации контента и место его размещения были изменены, чтобы ее коллега смог привлечь людей, которые будут читать статьи целиком, а у Алисы прибавилось читателей. Беспрогрышный вариант, и все благодаря *важным метрикам!*

---

## Экспериментирование

Можно не знать, как достичь цели, поставленной впервые. Именно здесь на помощь приходит экспериментирование. Сюда может входить проведение проверки концепции для трех различных продуктов Runtime Application Security Protection (RASP), чтобы понять, какой из них лучше всего подходит для организации; проведение различных видов обучения для небольших групп разработчиков и последующее предоставление «лучшего» варианта обучения на основе оценки каждого вида; либо добавление собственного сценария в один конвейер, измерение результатов в течение нескольких недель, регулярное улучшение сценария, а затем распространение его на остальные конвейеры в организации. Нет ничего страшного в том, чтобы попробовать что-то и решить, что оно не подходит, – вы только что чему-то научились, и это имеет ценность. Кроме того, вы, скорее всего, сэкономили много времени и денег.

Сфера DevOps и стартапов славятся тягой к экспериментам, обучению и постоянному самосовершенствованию, чему способствуют структурированные методы и повторяющиеся процессы. Я настоятельно рекомендую следующие две книги по данной теме:

«Руководство по DevOps: Как добиться гибкости, надежности и безопасности мирового уровня в технологических компаниях» (Джин Ким, Джез Хамбл, Джон Уиллис и Патрик Дебуа, опубликовано IT Revolution Press, LLC), [itrevolution.com/book/the-devops-handbook](http://itrevolution.com/book/the-devops-handbook)<sup>[72]</sup>;

«Бережливый стартап: как современные предприниматели используют непрерывные инновации для создания чрезвычайно успешных предприятий» (Эрик Рис, (опубликовано издательством Crown Business), [theleanstartup.com/book](http://theleanstartup.com/book)<sup>[73]</sup>.

## Обратная связь от всех и каждой из заинтересованных сторон

*Любые отзывы важны.* Разработчики, владельцы продуктов, менеджеры проектов, другие команды безопасности и остальные сотрудники ИТ-отдела – все они ваши клиенты, так как используют

программу и сервисы по обеспечению безопасности приложений. Их мнение *имеет значение*.

Крайне важно запрашивать обратную связь, под которой подразумевается вовсе не автоматический опрос раз в год. Регулярно проводите встречи с чемпионами по безопасности, спрашивайте на проектных совещаниях об их потребностях, просите людей приходить к вам и рассказывать, что им нравится и что не нравится, а также всегда сообщать, если вы что-то сломали.

Перед выпуском какого-либо стандарта или политики нужно провести консультацию. Спросите разработчиков, какая поддержка им нужна для следования новому стандарту. Уточните, все ли им понятно в нем, согласны ли они с ним полностью либо частично. Пришедшие на обсуждение люди (даже если их будет совсем немного) – это, как правило, те, кому есть что сказать. Можно провести даже несколько собраний. Цель не в том, чтобы убедиться, что абсолютно все поддерживают нововведение, а в том, чтобы удостовериться, что наиболее активные разработчики согласны с ним и что вы не допустили грубых ошибок.

Получая обратную связь, старайтесь не принимать ее близко к сердцу. Не каждый человек умеет изящно и умело изложить свою точку зрения. Если вы плохо воспримете обратную связь, люди вряд ли доверятся вам снова, и тогда придется планировать свою программу вслепую. Постарайтесь сделать все возможное, чтобы оставаться открытым.

## Особое замечание о DevOps и Agile

При определении цели программы безопасности не имеет значения, какой методологии разработки придерживаются программисты. У людей из среды DevOps в любом случае есть требования, а код, написанный Agile-командой, в любом случае нуждается в тестировании. Потребуется изменить именно тип действий и инструментарий, а не конечные цели.

Если разработчики основывают свою работу на спринтах, необходимо подстроиться под спринты. Если команды DevOps используют конвейеры CI/CD для выпуска своего кода, то лучше выяснить, какие инструменты лучше всего работают в их конвейерах.

Очень важно скорректировать наши собственные методы, поскольку наша обязанность заключается именно в том, чтобы дать разработчикам возможность создавать безопасное программное обеспечение, а не мешать им в этом.

## Деятельность по обеспечению безопасности приложений

Список мероприятий по обеспечению безопасности приложений очень и очень длинный. По сути в него входит все, что касается защиты создаваемого ПО. Здесь можно легко запутаться. На рис. 7.1 видно, какими мерами безопасности можно дополнить жизненный цикл разработки системы.



Рис. 7.1. Деятельность по обеспечению безопасности в жизненном цикле разработки системы

Вот список хорошо известных и широко распространенных практик.

*Запуск сканирования уязвимостей (оценки уязвимости)*, вероятно, является наиболее распространенным видом деятельности инженеров

безопасности. Запуск автоматизированного инструмента для выполнения DAST (динамического тестирования приложений) – быстрой и достаточно точной проверки, которая позволяет выявить порядочное количество уязвимостей, но тем не менее многое может упустить. DAST можно автоматизировать для запуска в конвейере, запускать по расписанию или вручную.

*Оценка безопасности и оценка уязвимости* часто используются в высокоценных или других значимых приложениях, чтобы найти как можно больше уязвимостей.

*Моделирование угроз* может варьироваться от «мозгового штурма зла» до более формальных действий – например, разработки деревьев атак или следования моделям STRIDE или PASTA. Определение всех возможных угроз для системы с последующим смягчением – ручной процесс.

*Обзор программного кода безопасности* – это проверка элементов управления безопасностью в коде на наличие ошибок. Может выполняться не только вручную, но и посредством инструмента SAST (статического тестирования безопасности приложений).

*Анализ состава программного обеспечения* (SCA) – это проверка уровня безопасности сторонних компонентов (кода, написанного сотрудниками вне компании), и его проще всего выполнить с помощью соответствующего инструмента или заранее утвержденного списка. Выполнить такую задачу вручную слишком сложно, если только разработчики не ограничиваются крайне малым комплектом компонентов.

*Тестирование на проникновение* (пентестирование) часто проводится сторонними специалистами с целью интенсивного тестирования безопасности или привлечения внимания руководства к проблеме. Поиск и последующее использование уязвимостей – это отличный способ добиться поддержки со стороны команд, которые еще не начали работать с программой безопасности.

*Ответственное и координированное раскрытие информации* – это процесс, позволяющий исследователям безопасности и другим лицам безопасно сообщать вам о проблемах с защитой продуктов. В настоящее время Google и несколько других крупных компаний – разработчиков программного обеспечения настаивают на отраслевом стандарте, согласно которому информация становится достоянием

общественности в течение 90 дней, предполагая, что три месяца – достаточный срок для устранения проблемы и выпуска исправления. К сожалению, фактические сроки этого процесса сильно варьируются от организации к организации. Первоначально данный процесс был известен как «ответственное» раскрытие информации, но по мере своего развития он неуклонно переходит к «координированному» раскрытию информации.

*Программы вознаграждения за найденные ошибки (Bug Bounty)* – заранее спланированные, управляемые взаимодействия между исследователями безопасности и компаниями, позволяющие проводить тестирование безопасности продуктов либо любым исследователем, либо одобренным/приглашенным исследователем. Идея Bug Bounty заключается в создании конкуренции посредством вознаграждения только тех исследователей, которые находят уникальную и достоверную ошибку.

*Программы обучения и защиты разработчиков* предоставляют разработчикам возможность создавать безопасное программное обеспечение посредством обучения и другой поддержки. Некоторые компании имеют очень продуманные программы, в то время как другие могут, к примеру, просто подкинуть вам эту книгу.

*Политики, руководства и стандарты* – это инструменты, которые создаются, чтобы указывать разработчикам более безопасный путь, направлять и вести по нему. Будьте внимательны при их написании: они должны быть понятными, разумными и точными. Прекрасным решением будет их передача самым лучшим разработчикам.

*Предоставление разработчикам инструментов безопасности*, как уже говорилось ранее, является отличным способом улучшить качество работы команды. Необходимо научить сотрудников пользоваться этими инструментами и предоставить им подходящую среду для их применения (например, виртуализированную среду с тестовыми данными).

*Атака «красной команды»* – это разрешение команде обученных тестировщиков на проникновение атаковать производственные системы (осторожно), чтобы найти уязвимости и проверить, насколько правильно системы реагируют на атаки.

*Обзор* всех новых инструментов, компонентов, платформ и других технологий, которые разработчики хотят одобрить, и рекомендации по

их безопасному использованию. Вероятно, займет много времени, но результаты будут того стоить.

*Предоставление инструментов IDE для обеспечения и усиления безопасности кодирования и управления пакетами.*

*Добавление средств обеспечения безопасности в конвейеры CI/CD разработчиков.*

## **Инструменты по обеспечению безопасности приложений**

Как и список мероприятий по безопасности, представленный ниже список всегда можно дополнить. Люди создают собственные инструменты, делятся ими и открывают исходные коды. В настоящее время существует бесчисленное множество стартапов, изобретающих новые крутые технологии, а также немало занимающихся безопасностью компаний со стажем и стабильным положением, которые переосмыливают себя и свои продукты. Инструментов будет становиться все больше. Тем не менее вот список, который вам, возможно, будет интересно изучить.

### **Стандартные**

- DAST (динамическое тестирование безопасности приложений).
- SAST (статический анализ безопасности приложений).
- Анализ состава программного обеспечения (SCA) – проверка ваших сторонних зависимостей на известные уязвимости.
- Веб-прокси для ручного тестирования приложений и API.
- Фаззеры для поиска дыр в валидации ввода.
- Брандмауэры веб-приложений (WAF) – щит из регулярных выражений, используемых для проверки пользовательских вводных данных. Они блокируют трафик HTTP(S) к веб-приложению и от него на основе списков одобрения или блокировки и регулярных выражений. Некоторые WAF также используют искусственный интеллект или машинное обучение (англ. *artificial intelligence / machine learning, AI/ML*).

### **Новые**

- RASP (англ. *runtime application security protection*, также известен как обратный прокси) – заглушка (двоичный файл), размещаемая внутри приложения, которая действует как щит против атак на приложение. Обычно для обнаружения угроз используется искусственный интеллект и машинное обучение, а не только регулярные выражения и белые либо черные списки.
- Инструменты управления уязвимостями для сбора показателей и измерения прогресса.
- Инструменты IDE и вебхуки, помогающие разработчикам в исправлении небезопасных решений по кодированию.

- Интерактивное тестирование безопасности приложений (IAST) – инструмент, представляющий собой двоичный файл, размещенный внутри приложения, который выполняет комбинацию SAST и DAST. Работает в реальном времени, в ходе взаимодействия пользователей с проверяемым приложением.
- Сервисная сетка – это инфраструктурный уровень, который шифрует коммуникации API и управляет ими.
  - Инструменты объединения для конвейеров, которые позволяют добавить только один инструмент безопасности в конвейер, подключить несколько инструментов безопасности к инструменту объединения, а затем получить результаты работы всех инструментов на одной панели. Он запускает несколько инструментов одновременно.
  - API-шлюзы, которые обеспечивают аутентификацию и авторизацию API, защищая их от несанкционированных вызовов и других злоупотреблений.
  - Средства автоматизации наименьших привилегий, которые проверяют, какие разрешения действительно необходимы, и удаляют лишние, тем самым сводя к минимуму поверхность атаки. Это также называют «автоматизацией политики».
  - Стандартные инструменты безопасности, оптимизированные для конвейеров. Будьте осторожны с ними – многие поставщики заявляют, что они готовы к работе в конвейерах, а на самом деле это все те же старые инструменты. Всегда проводите проверку концепции, чтобы убедиться, что работа инструмента отвечает вашим потребностям.
- Прокси-серверы управления пакетами, позволяющие разработчикам загружать только безопасные пакеты.
- Инструменты инвентаризации приложений и активов, помогающие создать реестр общедоступных веб-активов.
- Создание собственных пользовательских инструментов, если нужно что-то специфическое и до сих пор не существующее.

## **Ваша программа безопасности приложения**

Если у вас появилась возможность принимать решения о направлении формирования программы обеспечения безопасности приложений, сперва следует определить ее цели, а затем выбрать мероприятия и инструменты, которые помогут в их достижении.

Постоянно измеряйте свой прогресс на пути к целям и не переставайте совершенствоваться. В результате у вас будет не только более безопасное ПО, но и высокоэффективная программа безопасности.

## Упражнения

1. Назовите одну причину, по которой поддержание реестра приложений в актуальном состоянии важно для любой организации.
2. Приведите один пример предупреждения, которое может выдать ваше приложение. Какой тип поведения может вызвать такое предупреждение и почему его нужно расценивать как проблему?
3. Является ли тестирование самой важной частью программы безопасности? Если да, то почему? Если нет, то почему?
4. Что выгоднее: купить инструмент RASP либо WAF (щит для приложения) или потратить эти деньги на обеспечение безопасности кода? Объясните свой выбор.
5. Опишите тип инцидента безопасности, при котором понадобится помочь разработчику. Что требуется от разработчика в описанной ситуации?
6. Какой один инструмент вы дали бы в помощь своим разработчикам? Объясните свой выбор.
7. Какой учебный ресурс вы предоставили бы в помощь своим разработчикам? Объясните свой выбор.
8. В чем разница между SAST и SCA?
9. В чем разница между SAST и DAST?
10. Определите цель для возможной программы безопасности приложений в вашем офисе, учебном проекте или в выдуманной компании, где хотели бы работать. Какую цель вы поставили? Почему вы выбрали именно ее? Как вы будете измерять прогресс в ее достижении?
11. Есть ли препятствия, которые мешают начать первую программу безопасности на вашем текущем месте работы? Если да, то какие? А еще лучше – попробуйте подумать, как их можно преодолеть.

# Глава 8

## Обеспечение безопасности современных систем и приложений

По мере того как разработчики и эксплуатационный отдел продолжают внедрять новые технологии в ИТ-среду, необходимо модернизировать стратегии и тактики, чтобы идти в ногу со временем. В этой главе даются подробные объяснения тактики обеспечения безопасности для:

- API и микросервисов;
- онлайн-хранилищ;
- контейнеров и оркестровки;
- бессерверных приложений;
- инфраструктуры как кода;
- безопасности как кода;
- платформы как безопасности;
- инфраструктуры как безопасности;
- конвейера CI/CD;
- DevSecOps;
- облака;
- облачного потока.

В этой главе рассматриваются новейшие инструменты обеспечения безопасности приложений, а также современные тактики выполнения программы безопасности приложений.

По какой-то причине с появлением новой технологии вопросы безопасности уходят на задний план. Мы спешим выпустить что-то новое и блестящее и почему-то забываем все выученные ранее уроки. Важно, чтобы внедряемые в эксплуатационную среду новые и интересные технологии были защищены должным образом. В песочнице (среде, отделенной от эксплуатации и интернета) можно свободно запускать новые инструменты и эксперименты без необходимости обеспечивать их безопасность. Однако перед внедрением в эксплуатацию технологии должны быть проверены на наличие слабых мест в системе безопасности, а их защита усилена.

*Перед каждым внедрением.*

**СОВЕТ.** Если вы являетесь специалистом по безопасности приложений, всегда старайтесь участвовать в процессе оценки новых технологий. Это отнимает много времени, но вы сможете помочь направить своих коллег в более безопасное русло и избавить себя от серьезной головной боли в будущем.

## API и микросервисы

API означает *программный интерфейс приложения*. Однако название не очень точно отражает его суть: из него можно сделать вывод, что API – это программное обеспечение (интерфейс), которое используется для написания кода приложения (программы), но на самом деле это определение интегрированной среды разработки (IDE), а не API.

API – это код, который обеспечивает общение между двумя частями ПО. Он также определяет протокол (*способ общения*). API не может иметь графического интерфейса, так как представляет собой обычный код, связывающий два приложения и позволяющий им отправлять и получать запросы и данные друг от друга.

Веб-API часто возвращают данные и могут функционировать как веб-приложения. Они обслуживают данные, не имеют графического интерфейса, состоят из кода, выполняемого на сервере, и не видны конечному пользователю в браузере.

Мобильные приложения и веб-приложения часто обращаются к API, чтобы запросить данные или выполнить определенные действия. В операционной системе Windows существуют Windows API, которые обмениваются данными и выполняют услуги в фоновом режиме. Существует несколько видов API, но в этом разделе мы сконцентрируем свое внимание на его *веб*-версии.

Под «микросервисами» часто имеется в виду «микросервисная архитектура», то есть приложение, построенное из множества небольших сервисов, вызываемых из одного интерфейса. Она показана на рис. 8.1 в упрощенном виде. Наличие набора из нескольких сервисов, работающих независимо друг от друга, означает, что если один из них выйдет из строя, то остальная часть приложения все равно (в основном) будет работать. Сервисы, как и API, не имеют графического интерфейса.

API и микросервисная архитектура обычно идут бок о бок, поэтому мы обсудим их защиту вместе. Оба, как правило, вызываются из веб-приложений, мобильных приложений, а также из других API или бессерверных приложений. Оба требуют защиты.

API должны использовать API-шлюз, который защищает от злоупотреблений в стиле DoS (чрезмерно запрашиваемые, чрезмерно потребляемые или неуместно вызываемые ресурсы, результатом которых является большое количество ошибок) и выполняет функции безопасности авторизации, аутентификации и даже проверки ввода. API-шлюзы также предлагают функции дросселирования и квотирования ресурсов, которые весьма полезны для защиты API. Обе функции должны быть всегда включены в тех случаях, когда API предназначен для публичного использования и часто доступен для внутреннего пользования. Пример проблемной микросервисной архитектуры приведен на рис. 8.2. Один вызов не проходит ни через шлюз, ни через бэкенд приложения – этот вызов должен быть принудительно проведен через шлюз.

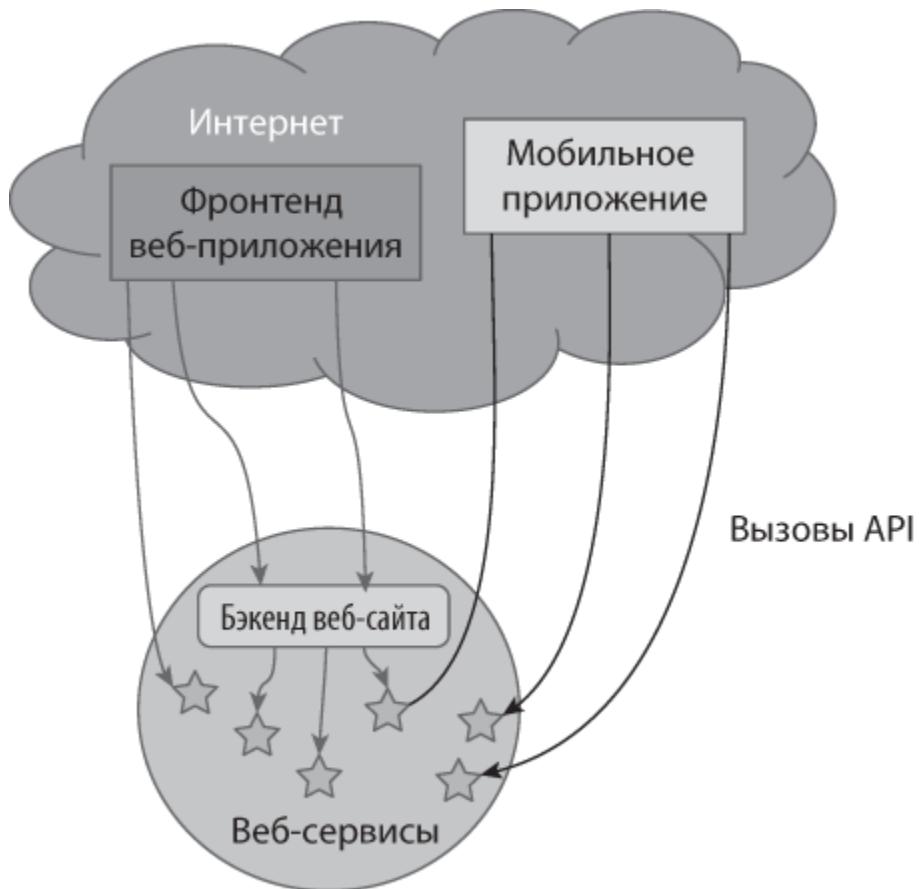


Рис. 8.1. Упрощенная архитектура микросервиса

Как уже говорилось ранее, даже если мы вводим в пользование новую технологию, важно применять лучшие практики обеспечения безопасности приложений. В случае с API и микросервисами очень важно продолжать кодировать проверку ввода, использовать параметризованные запросы и применять остальные компоненты передового опыта по написанию безопасного кода.

Отличным инструментом для защиты микросервисов является *сервисная сетка*. Сервисная сетка (англ. service mesh) – это слой инфраструктуры, который помогает микросервисам быстро и надежно взаимодействовать друг с другом. Сервисная сетка нужна только в облаке и микросервисных архитектурах. Представьте себе тысячу различных сервисов для приложения, а затем сотни экземпляров каждого сервиса, работающих одновременно. Такая сложная структура требует организованности, скорости и надежности. Именно для этого и

нужна сервисная сетка.

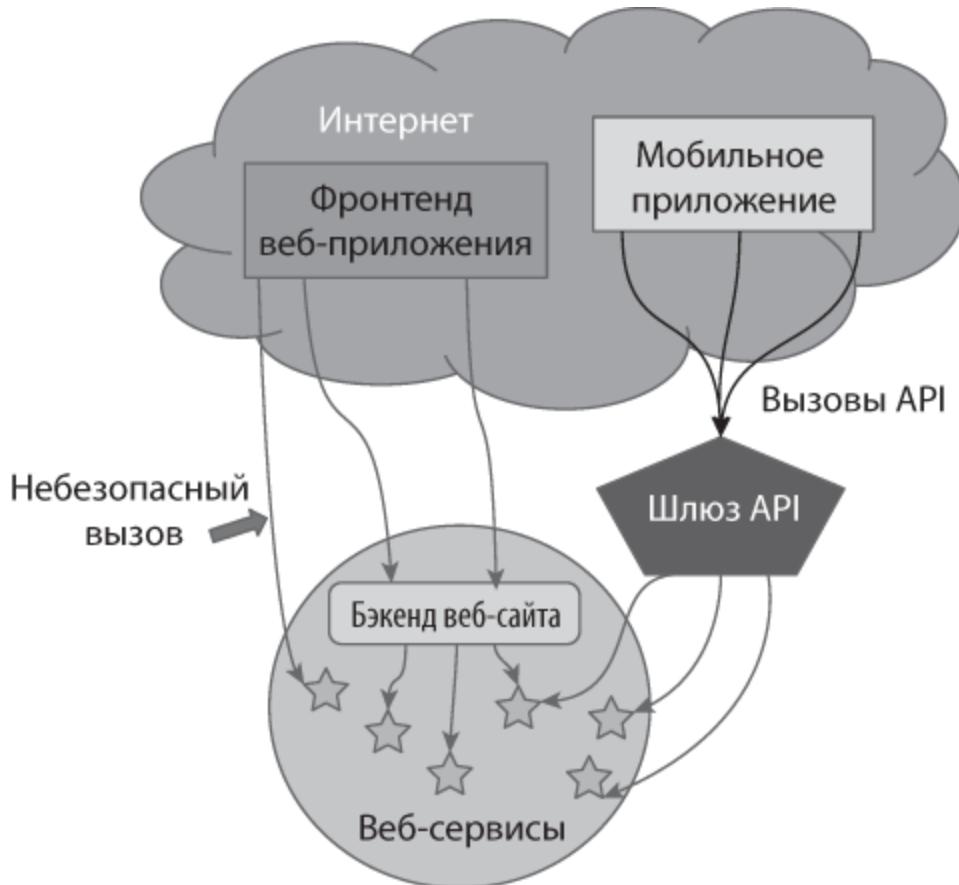


Рис. 8.2. Архитектура микросервиса с API-шлюзом

Использование сервисной сетки не требует изменений кода, так как она представляет собой инфраструктурный слой. Все провайдеры публичных облаков имеют собственные сервисные сетки, но на рынке также доступны платформенно-независимые варианты.

При создании API или сервисов любого рода необходимо, чтобы организация имела стандарты их написания – не нужно заново изобретать колесо. Стандарты также упрощают тестирование. По возможности следует предоставлять командам, недавно работающим с API и сервисами, паттерны, чтобы они знали, как следовать установленным стандартам.

Еще один метод защиты API – обеспечить максимально строгое следование протоколу и определению API (проверка через тест). Таким образом можно избежать двусмыслиности вызовов API и его переходов в неизвестное состояние из-за способа вызова.

При создании API (и всех приложений) необходимо установить обязательное условие: сначала аутентификация пользователей, а затем их авторизация. Причина этого заключается в том, что после аутентификации пользователя будет понятно, кто он в сети (раскроется его личность). Без надлежащей идентификации любой пользователь опасен. Если система сначала проводит авторизацию пользователя (предоставляет доступ), а затем аутентификацию (выясняет его личность), она подвергается опасности с точки зрения доступа.

Очень важным условием является ограничение информации, содержащейся в сообщениях об ошибках для API. С API общаются компьютеры, а не люди, и они не делают опечаток. Плохо сформированный запрос чаще всего означает попытку атаки. Если в ответ на атаку злоумышленнику придет подробная информация об ошибке, которая может помочь ему создать правильно сформированный запрос к API, опасности подвергнется вы, ваше приложение и ваша организация. В сообщениях об ошибках не должна содержаться информация, которая может раскрыть злоумышленникам, как вызвать API (например, соответствующие инструкции).

Наконец, еще один пункт, который применим ко всем приложениям, – следует разрешить вызывать только используемые приложением HTTP-методы, а остальные заблокировать.

## Онлайн-хранилище

Данные являются самым ценным техническим активом, которым владеет организация. Необходимо создать безопасные настройки по умолчанию для онлайн-хранилища, чтобы гарантировать, что каждый использующий его человек соблюдает меры предосторожности. Никаких исключений.

Крайне важно определять уровень чувствительности данных, помещаемых в онлайн-хранилище. Все данные без исключений должны быть классифицированы и маркированы. Когда специалисты по реагированию на инциденты сталкиваются с утечкой данных, они используют совершенно разные протоколы для «несекретной» и «секретной» информации. Когда данные не обозначены или не классифицированы по уровню секретности, специалист по реагированию на инциденты не знает, с чего начать, и, к сожалению, вынужден предполагать худшее. По сути незнание уровня секретности данных при ликвидации последствий инцидента приводит к кошмарным последствиям.

К дополнительным уровням безопасности для онлайн-хранилищ относятся: мониторинг доступа к онлайн-хранилищу, мониторинг целостности файлов, мониторинг на предмет прослушивания портов, добавление к мониторингу предупреждений, оповещающих операционный центр безопасности (SOC), а также настраиваемые автоматические ответы, проходящие через облачные потоки.

**СОВЕТ.** Не всем повезло работать там, где есть операционный центр безопасности. Если у вас его нет, оповещения следует отправлять команде безопасности, службе поддержки и/или владельцам компании. Важно, чтобы сообщения не уходили «в никуда» или на редко проверяемый почтовый ящик.

Необходимо, чтобы доступ к контейнерам онлайн-хранилища осуществлялся ИТ-персоналом только после проведения авторизации несколькими сторонами. Доступ обычно должен предоставляться только сервисным учетным записям – сетевым учетным записям, которые создаются для того, чтобы компьютеры могли общаться с

другими компьютерами. Таким образом, если поступит сообщение о том, что Алиса среди ночи входит в онлайн-хранилище, мы будем знать, что, скорее всего, ее учетные данные были скомпрометированы и необходимо немедленно начать изучение данного инцидента.

**СОВЕТ.** Сервисные учетные записи используются только службами операционной системы, сетей или баз данных для осуществления своей деятельности и никогда – ИТ-персоналом. Сервисные учетные записи разделяют идентификационные данные и действия человека и машины, что позволяет распознать потенциально вредоносные либо опасные паттерны и поведение и автоматизировать защитные меры против них.

Помимо этого, для защиты онлайн-хранилища можно использовать облачные средства контроля безопасности, включая инструменты категоризации трафика, управления доступом и идентификацией (англ. Access and Identity Management, AIM), а также средства автоматизации с наименьшими привилегиями для обеспечения ограниченного количества разрешений в хранилище [\[74\]](#).

Наконец, крайне важно защитить панель управления облаком. Любые облачные учетные записи с возможностью изменять разрешения в онлайн-хранилище должны иметь включенную многофакторную аутентификацию (MFA), а также длинные уникальные пароли, хранящиеся в менеджере паролей.

## Контейнеры и оркестровка

Когда центры обработки данных только зарождались, существовали *серверы*, которые занимали много места и потребляли много энергии. Они были очень шумными и требовали постоянного кондиционирования воздуха. Каждый сервер мог содержать только одно приложение.

Со временем была создана виртуализация, которая позволила посредством *виртуальных машин* (ВМ) вместить несколько операционных систем в одну физическую машину. Таким образом удалось разместить несколько приложений на одной машине.

Потом появились *контейнеры*. Контейнеры значительно меньше виртуальных машин: вместо целой операционной системы они работают только с той ее частью, которая задействована в работе приложения. Благодаря этому размер контейнеров в несколько раз меньше, чем размер виртуальной машины, что позволяет размещать их сотнями всего на одном сервере. Это огромный рост эффективности, поэтому контейнеры так быстро получили широкое распространение.

С контейнерами часто связано то, что называется *оркестровкой*. Оркестровка – это управление контейнерами. Система оркестровки может запускать, останавливать, добавлять или удалять память и многое другое в любом контейнере, находящемся под ее управлением. Идея оркестровки заключается в том, что управление сотнями или тысячами контейнеров вручную требует слишком много усилий, поэтому для выполнения всей работы используется специальная система.

Тем не менее новые технологии влекут за собой новые риски. Очень важно следовать тому же передовому опыту в отношении сетевой безопасности и облачной безопасности, что и в отношении виртуальных машин или платформы без операционной системы, включая их проектирование с учетом концепций «предположения взлома» и «нулевого доверия».

Очень важно ограничить ресурсы контейнеров и настроить оповещения о достижении лимита, чтобы мы знали, подвергаются ли контейнеры атаке<sup>[75]</sup>.

Стоит обратить особое внимание на запуск контейнеров от имени «root». Пользователь «root» может делать буквально все, и, если кто-то запустит контейнер от этого имени, он не будет иметь ограничений в действиях с находящимися там ресурсами. Хотя разработчику программного обеспечения или любому ИТ-специалисту полезно иметь суперспособности для выполнения своих обязанностей, на самом деле это почти никогда не требуется, поэтому запрет на использование root для контейнеров будет реализацией концепции наименьших привилегий, применяемой к контейнерам<sup>[76]</sup>.

Образы контейнеров часто хранятся в реестрах, которые необходимо защитить. Очень важно тщательно следить за разрешениями реестра компании. Если злоумышленник загрузит небезопасный образ,

сотрудники могут нечаянно поставить его в эксплуатацию и тем самым создать уязвимость.

Важно, чтобы разработчики программного обеспечения могли загружать образы только из доверенных источников. Для этого следует создать политику, запрещающую загрузку образов из ненадежных источников, если только речь не идет об изолированной среде, отделенной от остальной сети и ресурсов организации.

**ВНИМАНИЕ.** Безопасность цепочки поставок также относится к образам, а не только к веб-фреймворкам и библиотекам. Хорошим первым шагом будет сравнение загружаемых файлов с контрольной суммой или хешем. Если планируется запуск собственного реестра образов, создайте проект по обеспечению его защиты от манипуляций со стороны субъектов угроз.

Контейнеры и оркестровка требуют изучения новых правил конфигурации, новых инструментов сканирования, а также новых видов уязвимостей и связанных с ними рисков. Кроме того, необходимо защищать учетные записи тех, кто может создавать или редактировать контейнеры и системы оркестровки, поскольку это самые ценные учетные записи для злоумышленников. Все они должны иметь длинные, уникальные, случайно сгенерированные пароли, хранящиеся в менеджере паролей, и включенную многофакторную аутентификацию. Как и в случае с виртуальными машинами, следует сканировать все, что можно, с помощью линтера проверять соответствие конфигураций руководствам, предоставленным поставщиком, регулярно исправлять или заменять контейнеры на обновленные, более безопасные версии и применять принцип нулевого доверия ко всем проектным решениям.

**СОВЕТ.** Стандарты по защите практически любой операционной системы можно найти в стандартах CIS. Центр интернет-безопасности (CIS) предоставляет образы с предварительно усиленной защитой и свободно делится своими знаниями с другими.

## Бессерверные приложения

Бессерверное программное обеспечение – это программное обеспечение, которое размещается или работает на сервере *не* на постоянной основе. Оно доступно только при непосредственной ссылке на него. Вызываемое бессерверное приложение открывает контейнер, выполняет свой код, затем самоуничтожается, отдавая все ресурсы обратно экосистеме для использования другим процессом. Эфемерность (недолговечность) означает, что бессерверное программное обеспечение создает меньшую поверхность атаки, поскольку оно редко находится в запущенном состоянии (что часто требуется для атаки), следовательно, сервер для атаки также отсутствует. Главное преимущество бессерверных приложений заключается в том, что плата за них производится только по факту использования, то есть они стоят намного меньше, чем обычные приложения. Оплата только девяти минут времени работы в месяц может значительно сократить выставленные за облако счета. Однако в отношении безопасности здесь действуют те же принципы, что и для любых других приложений. Кроме того, нужно обратить внимание на еще несколько пунктов. Взглянем на них поближе.

Как и любые другие, бессерверные приложения необходимо отслеживать и регистрировать. В идеале журналы должны поступать в систему управления информацией о безопасности и событиями безопасности (SIEM), чтобы центр безопасности видел предупреждения и в случае необходимости принимал соответствующие меры.

**СОВЕТ.** Не страшно, если на вашем рабочем месте нет SIEM или центра безопасности. Можно настроить свои системы так, чтобы они отправляли сообщения по электронной почте вам, вашей команде, заинтересованным лицам или в службу поддержки. Кто-то должен быть оповещен, поэтому следует выяснить, кто лучше всего подходит для этого, и автоматизировать данный процесс.

Можно догадаться, что требования к написанию безопасного кода и следованию подходящему передовому опыту применимы и в данной ситуации. Как и в любом другом случае, необходимо проверять вводимые данные, проводить аутентификацию и авторизацию пользователей. Данные также должны шифроваться при передаче и в состоянии покоя. Эти требования нельзя игнорировать только потому, что приложение работает недолго.

Одним из отличий обеспечения безопасности бессерверных приложений является возможность использования API-шлюза в качестве буфера безопасности. API-шлюз проводит аутентификацию и авторизацию и подтверждает, что никто не сможет вызвать бессерверные приложения или получить доступ к ним без прохождения этих процедур. Его также можно использовать для принудительного дросселирования (замедления запросов) и квотирования ресурсов (отсечения пользователей после определенного количества запросов), что значительно повышает защиту бессерверных приложений от злоупотреблений в стиле DoS.

Бессерверные приложения также должны быть развернуты с минимальной гранулярностью, то есть они должны выполнять только одну функцию и делать это правильно. Не нужно вкладывать в бессерверное приложение функционал полноразмерного приложения, потому что в таком случае отсутствие сервера не имеет смысла. Если бессерверное приложение почти всегда находится в работе, его следует заменить на обычное приложение, размещенное на сервере или PaaS.

При создании бессерверных приложений очень важно поддерживать изолированные параметры. Имеется в виду, что каждое бессерверное приложение должно пройти аутентификацию и авторизацию во всех других бессерверных приложениях, с которыми оно взаимодействует. Не существует подразумеваемого доверия. Часто при использовании микросервисной архитектуры для создания приложений разработчики не применяют все те же средства защиты, что и для обычного приложения, предполагая доверие между своими сервисами, бессерверными и другими приложениями. Будьте осторожны, потому что злоумышленник сможет воспользоваться данной брешью и беспрепятственно вызвать бессерверное приложение, ведь оно не требует никакой аутентификации, которая остановила бы его<sup>[77]</sup>.

Хотя это касается каждого приложения, крайне важно, чтобы секреты из бессерверных приложений хранились в тайнике, а не в библиотеке кода или базе данных.

Поскольку бессерверные приложения также являются инфраструктурой, необходимо убедиться в том, что при их настройке и размещении в сети применяется принцип наименьших привилегий [\[78\]](#).

При использовании конвейера CI/CD следует подумать о добавлении сканирования безопасности для проверки конфигурации бессерверных приложений [\[79\]](#).

Наконец (это также относится ко всем приложениям), необходимо убедиться, что все компоненты, от которых зависит бессерверное приложение, проверены на отсутствие угроз безопасности, а все найденные в нем уязвимости устраниены [\[80\]](#).

## Инфраструктура как код (IaC)

Инфраструктура как код (IaC), иногда называемая «конфигурация как код», – это создание сценариев или файлов конфигурации в виде кода для автоматизации развертывания. Использование IaC является частью DevOps, которая обеспечивает возможность выпуска инфраструктуры через конвейер CI/CD. Можно также добавить в инфраструктуру средства усиления безопасности в закодированном формате и тем самым повысить ее защиту с момента создания. Не лишним будет сканирование безопасности для проверки инфраструктуры как кода по мере ее продвижения по релизному конвейеру. IaC позволяет быстрее не только выпускать исправления и изменения инфраструктуры, но и внедрять исправления безопасности и изменения в укреплении инфраструктуры в эксплуатационные системы. Рабочий процесс показан на рис. 8.3.

До изобретения IaC инженерам по эксплуатации приходилось вручную устанавливать приложения и операционные системы. Они сидели за несколькими машинами, нажимая кнопку Next, устанавливали ПО с компакт-диска или использовали программу, которая помогала автоматизировать часть этого процесса, например системный центр (англ. system center). Этот вид работы не очень хорошо масштабировался и часто приводил к ошибкам. Такая работа также не доставляла особого удовольствия. Ее, лишенную

непреходящей ценности, линейно автоматизируемую, часто называют *рутиной*<sup>[81]</sup>.

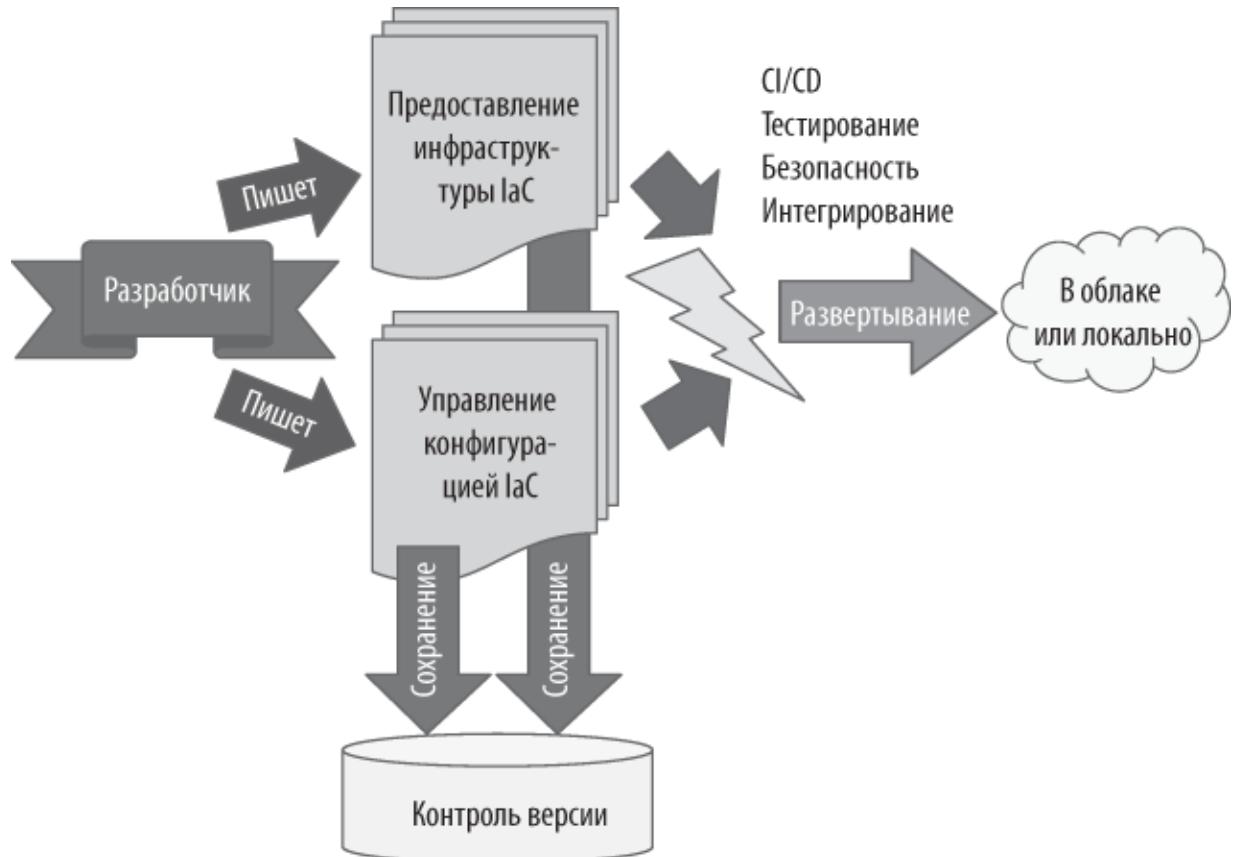


Рис. 8.3. Рабочий процесс инфраструктуры как кода

Одним из самых больших преимуществ IaC является повторяемость. Выполнение задачи вручную может привести к ошибкам, а если мы выполняем задачу довольно часто, статистически мы будем допускать ошибки. С помощью инфраструктуры как кода можно обеспечить создание идеальных копий, а наличие процессов и политики, поддерживающих обеспечение безопасности инфраструктуры как кода, принесет отличные результаты как для службы безопасности приложений, так и для организации в целом.

Еще одним большим преимуществом инфраструктуры как кода является то, что она автоматически документирует себя в репозитории кода при проверке новых версий. Как правило, в большинстве организаций изменения настроек конфигурации не очень хорошо

регистрируются или вообще не регистрируются. Если вы сначала проверяете всю инфраструктуру как код в репозитории кода во время каждого изменения, затем проводите тестирование безопасности и другие виды тестирования в конвейере CI/CD и только после этого выполняете перемещение в эксплуатацию, тем самым вы автоматизируете не только документирование, но и процесс управления изменениями. Это прекрасное решение.

---

## **ОСТОРОЖНО ПРЕДОСТАВЛЯЙТЕ РАЗРЕШЕНИЯ**

Боб помнит, как на его предыдущем месте работы впервые внедрили инфраструктуру как код. Все были очень рады использовать новый способ развертывания инфраструктуры, и каждый хотел стать частью этого процесса. К сожалению, они были очень неосмотрительны в отношении того, кто имел возможность создавать инфраструктуру как код, и один начинающий сотрудник случайно развернул 1000 контейнеров хранения вместо 100, что в итоге обошлось в *очень* большую сумму. Боб решил, что они все должны пройти обучение передовому опыту использования IaC, чтобы исключить возможность повторения подобного несчастного случая. В общем, передавайте такую ответственность и доступ только тем, кто прошел соответствующую подготовку.

---

При работе с инфраструктурой как кодом необходимо убедиться, что все изменения проводятся по соответствующей процедуре. Нельзя войти в рабочую машину и что-то поменять в конфигурации вручную, потому что тогда нарушится весь процесс. Вероятность случайного внесения ошибок, уязвимостей или слабых мест в инфраструктуру в таком случае значительно повышается, поскольку не проводятся все тесты безопасности и другие необходимые проверки, являющиеся частью CI/CD-конвейера<sup>[82]</sup>.

**СОВЕТ.** Не нужно вручную составлять документацию параллельно с инфраструктурой как кодом. Код должен говорить сам за себя, а подобная документация всегда будет отставать от того, что в данный момент проверяется в системе контроля исходных кодов. Как правило, письменная документация очень быстро устаревает, а на поддержание ее актуального состояния требуется много времени и усилий, что того не стоит. Каждый файл исходного кода должен начинаться с комментариев, объясняющих, для чего именно он предназначен, а у каждого исправления ошибок должно быть примечание, где указывается, какая ошибка была исправлена, а также номер тикета<sup>[83]</sup>.

Еще один пункт, который необходимо обсудить, когда речь идет об инфраструктуре как коде, – это *неизменяемость инфраструктуры*. Неизменяемость означает, что в созданную инфраструктуру не вносятся исправления и изменения, то есть она заменяется чем-то новым, уже имеющим новую конфигурацию, обновление безопасности или любое другое изменение. Преимуществом неизменяемости является невозможность случайной порчи образов инфраструктуры во время ее эксплуатации. Инфраструктура, которая ставится на место старой, уже должна быть проверена с помощью тестирования CI/CD. Не все патчи правильно применяются при развертывании, но этого можно избежать посредством неизменяемой инфраструктуры. В общем, она помогает снизить риски<sup>[84]</sup>.

## Безопасность как код (SaC)

Безопасность как код (англ. Security as Code, SaC) – это вплетение безопасности в DevOps, в частности с использованием кода. Хотя некоторые утверждают, что SaC – это то же самое, что и DevSecOps, большинство сходятся во мнении, что это разные вещи: DevSecOps воплощает все действия по обеспечению безопасности, которые выполняются в рамках жизненного цикла разработки системы по методологии DevOps, а безопасность как код – это только закодированные части.

Примеры SaC: добавление кода в инфраструктуру как код для улучшения политик безопасности, написание кода для реализации или автоматизации требований безопасности, создание негативных модульных тестов (модульные тесты, которые обеспечивают правильное экстренное завершение работы, также известные как регрессионное тестирование безопасности), а также добавление сценариев тестирования безопасности в конвейеры<sup>[85]</sup>.

Еще один пункт, который следует отметить в данном разделе, – это «пользовательские истории безопасности». Если в вашей компании создаются пользовательские истории, в их число должны входить также истории, касающиеся инцидентов безопасности, таких как атака методом грубой силы, XSS или инъекционная атака. Создание пользовательских историй безопасности гарантирует, что все другие усилия по тестированию, основанные на пользовательских историях, будут включать в себя обеспечение безопасности.

Пользовательские истории безопасности должны быть основаны на обеспечении целостности триады CIA, предотвращении моделей угроз, применении любых стандартов или политик, связанных с обеспечением безопасности, и реализации требований безопасности проекта.

Вот примеры пользовательских историй.

Пользовательские истории	Критерии приемки
Поскольку пользователи продукта АБВ вводят в систему частную информацию, эта информация защищена от других пользователей и общественности. (Конфиденциальность)	<ul style="list-style-type: none"> <li>Продукт протестирован с целью подтверждения того, что пользователи не могут видеть данные друг друга.</li> <li>Продукт реализует политику надежных паролей и обеспечивает MFA.</li> <li><i>Вставьте сюда все остальные применимые требования.</i></li> </ul>
В соответствии с требованиями SLA пользователи продукта АБВ могут использовать продукт 99,9% времени. (Доступность)	<ul style="list-style-type: none"> <li>Продукт прошел стресс-тесты и тесты на производительность.</li> <li>Продукт размещен в облаке с гарантированным временем работы 99,99%.</li> <li><i>Вставьте сюда все остальные применимые требования.</i></li> </ul>
Пользователи продукта АБВ вводят в систему данные, которые не могут быть изменены администраторами или любыми другими пользователями. (Целостность)	<ul style="list-style-type: none"> <li>Делаются записи обо всех изменениях.</li> <li>Перед каждым внесением изменений должна быть пройдена авторизация.</li> <li><i>Вставьте сюда все остальные применимые требования.</i></li> </ul>

## Платформа как услуга (PaaS)

Платформа как услуга (англ. Platform as a Service) – это инфраструктура, создаваемая и предоставляемая поставщиками публичных облаков для размещения приложений и API. Поставщик облака берет на себя ответственность за обеспечение безопасности платформы, включая установку патчей и настройку большинства параметров конфигурации. Клиент приобретает платформу как услугу, выбирает ее желаемый размер, операционную систему (ОС) и т. д., после чего размещает на ней свое приложение. На платформу как услугу можно поместить контейнер или код, и она его просто запустит. Она хороша тем, что обслуживание и большая часть обеспечения

безопасности не являются проблемой пользователя. Тем не менее некоторые аспекты обеспечения безопасности все же входят в его обязанности. Необходимо узнать, на какие конфигурации безопасности PaaS нужно обращать внимание, и затем поработать над ними соответствующим образом.

---

Хотя это и не относится к PaaS, включение мониторинга и протоколирования при использовании облачных сервисов – целесообразное, хотя и не дешевое решение. Основные затраты, связанные с мониторингом, – место, занимаемое журналами: размер журналов огромен. Тем не менее без ведения журналов и мониторинга невозможно быть в курсе атак, совершенных на PaaS и другие веб-ресурсы.

---

Все публичные веб-активы должны быть доступны только по зашифрованным каналам, что означает необходимость использования TLS/HTTPS. Следует обеспечить наличие заголовков безопасности, включить и правильно настроить другие конфигурации безопасности, которые находятся под вашим контролем. Наконец, вы как потребитель PaaS несете полную ответственность за создание безопасного программного обеспечения, размещаемого на платформе. Не имеет значения, что PaaS и публичное облако используют модель *разделяемой ответственности*. Поставщик облака не отвечает ни за какой ущерб или потери, понесенные из-за размещения в облаке небезопасного приложения, – это сфера ответственности клиента.

**СОВЕТ.** Независимо от того, какого поставщика услуг публичного облака вы решите использовать, прежде всего необходимо ознакомиться и согласиться с политикой *разделяемой ответственности*, которая определяет, какие конфигурации и задачи по обеспечению безопасности входят в сферу ответственности клиента и поставщика. В обязанности

поставщика входят исправления, ротация сертификатов TLS, управление ключами шифрования и многое другое. Обязанностями клиента являются: обеспечение безопасности приложений, размещенных на PaaS; решение о принудительном использовании HTTPS; решение о принятии старых версий TLS (все версии SSL будут блокироваться) и многое другое.

## **Инфраструктура как услуга (IaaS)**

Инфраструктура как услуга (англ. Infrastructure as a Service) является еще одной услугой, предлагаемой поставщиками публичных облаков. По сути, клиент выбирает операционную систему, аппаратное обеспечение и любые другие спецификации, и на основе его запросов в облаке автоматически создается такая инфраструктура. Например, чтобы создать виртуальную машину Linux с 4 Гб места на жестком диске и 8 Гб памяти, нужно заполнить онлайн-форму, нажать кнопку покупки – и в мгновение ока появится виртуальная машина, соответствующая этому описанию.

Как и платформа как услуга (PaaS), инфраструктура как услуга (IaaS) следует модели разделяемой ответственности. Но, в отличие от PaaS, IaaS требует, чтобы конечный пользователь (то есть вы) устанавливал патчи и выполнял прочее обслуживание предоставленной инфраструктуры. Облачный провайдер будет защищать физический центр обработки данных и обеспечивать надлежащий уход за физическими машинами, но не будет осуществлять поддержку предоставленной виртуальной машины – это ответственность клиента. Очень важно понимать разницу между PaaS и IaaS.

Лучшие практики защиты IaaS аналогичны тем, что применяются для всех ресурсов инфраструктуры: следование принципу наименьших привилегий; использование надежной идентификации, аутентификации и авторизации; подключение их к своей сети с использованием схем нулевого доверия и предположения взлома; регулярный выпуск исправлений, следование руководству по усилению безопасности и остальные действия по защите инфраструктуры.

## **Непрерывные интеграция, поставка и развертывание**

*Непрерывная интеграция* – это разработка на основе главной ветки. Новый код постоянно (регулярно) перепроверяется в репозитории вместе с уже написанным кодом, а затем интегрируется в конечный проект или продукт. То есть происходит постоянная проверка того, чтобы новый код не нарушил работу остальной части приложения. Непрерывная интеграция устраняет риск возникновения ошибок за счет более частого внесения небольших изменений<sup>[86]</sup>.

*Непрерывная поставка* – это использование автоматизированной системы для выполнения непрерывной интеграции вместо ручного вмешательства (которое отнимает много времени и чревато ошибками). Непрерывная поставка обычно рассматривается как использование конвейерного программного обеспечения для интеграции и развертывания кода на протяжении всего жизненного цикла разработки системы. Конвейерное программное обеспечение, также известное как CI/CD (от англ. Continuous Integration / Continuous Delivery), интегрирует новый код в ствол кодовой базы, проверяет его компиляцию, публикует его в различных тестовых средах, выполняет разнообразные автоматизированные тесты и предоставляет обратную связь на каждом этапе этого процесса. Системы непрерывной поставки имеют фантастические эффективность, повторяемость и качество. Невозможно запускать сотни тестов вручную, если код нужно проверять 10 раз в день. Конвейерное программное обеспечение облегчает данную задачу<sup>[87]</sup>.

*Непрерывное развертывание* позволяет CI/CD-конвейеру выпускать код в эксплуатацию без ручного вмешательства или одобрения человека. Непрерывное развертывание возможно в том случае, когда огромное количество проводимых тестов на качество, безопасность, надежность и т. д. позволяет доверять их результатам, обходясь без человеческого вмешательства<sup>[88]</sup>. Хотя многие компании, занимающиеся разработкой программного обеспечения, стремятся к непрерывному развертыванию, на практике оно не часто встречается в чистом виде. Это продвинутый вид деятельности, не являющейся необходимым условием для выполнения программы DevOps или использования конвейера CI/CD.

**СОВЕТ.** Использование конвейера CI/CD – не единственное условие выполнения программы DevOps. Как вы узнаете далее в этой главе, DevOps представляет собой намного больше, чем просто ПО для конвейера.

## Dev(Sec)Ops

Чтобы говорить о DevSecOps, сначала необходимо обсудить DevOps. Существует множество определений DevOps, однако я придерживаюсь того, которое приводится в DevOps Handbook и The Phoenix Project (упоминавшихся в главе 7)<sup>[89]</sup>.

DevOps – это и методология создания программного обеспечения, и особая культура, которая может существовать в отделе разработки ПО. Она требует сочетания процессов и продуктов, используемых людьми для создания надежного программного обеспечения. DevOps – это когда одна команда берет на себя обязанности по разработке и эксплуатации, разрушая существовавшее ранее разделение. Данная методология является куда более эффективным и единственным способом создания надежного и высококачественного программного обеспечения. Много лет назад в школах мы преподавали каскадную модель жизненного цикла разработки системы, затем гибкую, а теперь переходим к преподаванию DevOps, потому что она показывает лучшие результаты, а проекты, основанные на DevOps, чаще оказываются успешными.

Внедрить методологию DevOps в работу можно тремя путями, о которых говорится в The Phoenix Project<sup>[90]</sup>.

*Первый путь* внедрения DevOps заключается в обеспечении эффективности всей системы. Необходимо концентрироваться не только на своей части системы, а рассматривать весь жизненный цикл разработки в целом и вносить в него улучшения, тем самым обеспечивая быстрые и надежные результаты. Здесь подразумевается использование автоматизации (включая программное обеспечение CI/CD) и других процессов и инструментов, делающих возможным ускоренный релиз кода.

*Второй путь* внедрения DevOps – обеспечение быстрой обратной связи. Недостатком каскадного метода было отсутствие обратной связи в течение нескольких месяцев, что часто вело проектные команды по

неправильному пути: они создавали не тот проект, который на самом деле был нужен клиенту или отвечал его потребностям. Это также приводило к тому, что тестирование и другая обратная связь касательно безопасности откладывались до самого позднего этапа жизненного цикла разработки системы, когда вносить исправления было уже очень дорого и сложно, а времени на это почти не оставалось. Быстрая обратная связь означает автоматизацию различных видов проверок, включая тестирование безопасности. Здесь также очень полезен конвейер CI/CD. Однако любая автоматизация процесса тестирования безопасности или обратной связи будет соответствовать требованиям второго пути даже без использования конвейерного программного обеспечения.

**СОВЕТ.** Одним из способов обеспечить очень быструю обратную связь с разработчиками является добавление тестов безопасности в модульное тестирование. Похожего эффекта можно добиться с помощью оболочек библиотек, меняя имена небезопасных функций на такие, как «небезопасныйMD5Hash-нетрогать» или «вотпочемуунасвсеплохо»<sup>[91]</sup>.

*Третий путь* DevOps – постоянное обучение, эксперименты и принятие риска. Третий путь подчеркивает необходимость постоянно работать над повышением качества и эффективности повседневной работы сотрудников. Кроме того, он включает в себя метрики и другие измерения, показывающие направление движения рабочей деятельности. О тенденциях развития технологий в сфере безопасности можно узнавать, например, в процессе обмена мнениями в обеденное время и на занятиях, курсах, посредством книг и других материалов. Проверка концепции (PoC) различных инструментов или технологий с целью выяснения, какой из них лучше всего отвечает потребностям организации, – один из способов убедиться в актуальности и полезности выбранного для работы инструмента. Очень важно, чтобы каждую неделю или каждый месяц выделялось определенное количество времени на улучшение ежедневной работы.

Существуют издания, посвященные исключительно объяснению программы DevOps. В отличие от них, эта книга дает очень краткое определение, и, разумеется, многие идеи, процессы и концепции здесь

представлены не были. Больше о DevOps можно узнать из книг, предложенных в седьмой главе.

## DevSecOps

*Это то, чем мы всегда занимались как специалисты по безопасности приложений. DevSecOps – это просто безопасность приложений, адаптированная к среде DevOps.*

*Имран А. Мохаммед*[\[92\]](#)

При работе в среде DevOps специалисты по безопасности должны корректировать свою деятельность таким образом, чтобы она вписывалась в процессы, которым следуют инженеры DevOps. Если команда DevOps основывает свою работу на спринтах, то мы (служба безопасности) должны вписать свою работу и проекты в спринты. Если она использует конвейеры CI/CD для автоматизации тестирования, то необходимо, чтобы часть нашего тестирования безопасности была включена в эти конвейеры. Каким бы способом команда DevOps ни выполняла свои функции жизненного цикла разработки системы, нам нужно вплести себя и наши цели безопасности в этот процесс посредством нескольких существующих стратегий. Опять же, этой теме посвящены целые книги, поэтому мы не можем охватить всю информацию, но давайте обсудим хотя бы часть.

Первый и самый важный пункт, на котором должны сосредоточиться команды безопасности при попытке работать в среде DevOps, – это убедиться в том, что команды разработчиков и эксплуатации работают без оглядки на службу безопасности. Мы (служба безопасности) не можем ограничивать их работу. Другими словами, мы можем прервать сборку только в случае возникновения действительно чрезвычайной ситуации или очень серьезного риска. Служба безопасности больше не является пропускным пунктом, она стала помощником.

Еще один очень важный пункт для всех специалистов по безопасности приложений заключается в том, что необходимо настроить инструменты таким образом, чтобы максимально сократить

появление ложных тревог. Довольно часто ответственность за проверку результатов, полученных с помощью инструментов безопасности, возлагается на команду разработчиков, поскольку их больше. Однако они не обладают достаточными опытом, знаниями или полномочиями для этого, и вряд ли такой ход будет сколько-нибудь эффективен.

Последний пункт, который мы здесь упомянем, заключается в том, что при работе в среде DevOps следует обеспечивать быструю и своевременную обратную связь с командами разработки и эксплуатации. Необходимо также получать от них отзывы о своей работе, а затем включать эти замечания в свои процессы, проекты и политики для достижения наилучших результатов.

## Облако

Многие люди дают разные определения облачной безопасности, но в этой книге мы будем говорить об *облачно-ориентированной* безопасности. При обсуждении данной темы некоторые люди могут подумать, что она подразумевает почти то же самое, что и безопасность центра обработки данных или сетевая безопасность, но это не так. Говоря об облачной безопасности, мы имеем в виду, в частности, использование всех новых поразительных функций, услуг, инструментов и возможностей облака, которые недоступны в обычном традиционном центре обработки данных, а также безопасность, связанную с этими новыми технологическими предложениями.

## Облачные вычисления

*Облачные вычисления – это доступ по запросу к ресурсам компьютерной системы, особенно хранилищам данных и вычислительных мощностей, без прямого активного управления со стороны пользователя.*

*Википедия*[\[93\]](#)

Давайте разберем это определение. «Доступ по запросу» означает, что пользователь в любой момент может запросить все, что предлагает поставщик в качестве услуги, включая инфраструктуру, платформу как услугу, инструменты облачно-ориентированной безопасности и т. д. Сюда можно добавить больше сервисов, что одновременно и прекрасно, и потенциально небезопасно.

Представим, что разработчик собирается масштабировать инфраструктуру, на которой находится его веб-приложение. Допустим, он считает, что объем трафика будет небольшой, поэтому выбирает самое маленькое предложение, которое есть у провайдера облака. Однако он понимает, что, возможно, со временем сайт вырастет, бизнес станет более успешным и количество клиентов увеличится, поэтому он устанавливает флагок, разрешающий автомасштабирование. Спустя три месяца приложение подвергается атаке типа «отказ в обслуживании» (DoS), результатом которой становится гигантский счет, выставленный за пользование облаком. В этом и заключается потенциальный риск такой настройки.

Тем не менее, если заранее принять во внимание данный риск, можно выбрать автоматическое масштабирование, но при этом добавить ограничение на максимальное значение (установить квоту ресурсов) и включить соответствующее оповещение. Другим способом препятствования этому риску может быть добавление защиты от DoS-атаки, которую обычно предлагает большинство провайдеров публичного облака.

Техническому специалисту чрезвычайно удобно иметь возможность запрашивать ресурсы в случае необходимости. Возможность автоматического развертывания сотен машин одновременно или обработки гигантских объемов данных с помощью удивительной вычислительной мощности облака позволяет ИТ-специалистам выполнять гораздо больше действий, чем когда-либо прежде. Именно это мы подразумеваем под «доступом по запросу» к ресурсам в облаке.

Вернемся к нашему определению облачных вычислений и посмотрим на его концовку: «без прямого активного управления со стороны пользователя».

Несколько страниц назад мы говорили, что провайдер платформы как услуги (PaaS) берет на себя управление системами пользователей.

Доверяя своему облачному провайдеру исправление и укрепление защиты PaaS, клиент перепоручает ему выполнение этих функций. Так как пользователю облака не нужно активно управлять PaaS, у него остается больше времени на другую работу.

Облачный центр обработки данных позволяет клиенту забыть о дополнительных серверах, о кондиционировании воздуха или об установке программного обеспечения с компакт-диска. Все эти обязанности снимаются с клиента, что освобождает его от прямого управления облаком. Это и есть облачные вычисления.

## Ориентированность на облако

*Ориентированными на облако называются приложения и сервисы, автоматизирующие и интегрирующие концепции непрерывной поставки, интеграции и развертывания, DevOps, микросервисов, бессерверных технологий и контейнеров*<sup>[94], [95], [96]</sup>.

Это определение было создано путем объединения лучших частей многих различных определений. Проанализируем его вместе?

Начнем с «приложений и сервисов». Под этим подразумеваются любые услуги или ПО (включая API), предлагаемые провайдером облака, а также любые сторонние услуги, предоставляемые поставщиками, чьи продукты были приобретены для использования в облаке. По сути, вся инфраструктура, услуги и программное обеспечение, находящиеся в облаке, доступны пользователям. Сюда входит и программное обеспечение как услуга (англ. Software as a Service, SaaS), которое не устанавливается в облако, но доступно в нем, в то время как рабочая нагрузка в основном идет на другое место.

Следующая часть нашего определения – «автоматизирующие и интегрирующие». Облако сконцентрировано на автоматизации, то есть на сокращении объема утомительной ручной работы и на обеспечении точности. Оно не только автоматизирует все возможные действия, но и гарантирует бесперебойную совместную работу всех движущихся частей, которую также называют «интеграцией». Необходимо обеспечить условия для того, чтобы различные сервисы, программное обеспечение, инфраструктура и сети могли правильно работать вместе и интегрироваться без проблем, иначе в облаке наступит беспорядок, устранение которого потребует много усилий и денежных затрат.

Итак, теперь, когда мы понимаем важность правильной работы программного обеспечения и всех сервисов вместе, а также их полной автоматизации, добавим следующие компоненты: CI/CD, DevOps,

микросервисы, бессерверные приложения и контейнеры (определение которых мы уже давали в этой главе).

Все эти функции вместе создают облачно-ориентированную среду. Доступ ко всем этим инструментам и сервисам, предоставляемый поставщиком облака, инфраструктурная среда, способная поддерживать все эти концепции, а также добавление необходимых для работы инструментов сторонних производителей – все это создает потрясающие возможности для службы безопасности, если эта среда правильно спроектирована и имеет хорошую поддержку.

Однако если такая среда спланирована и разработана без учета требований безопасности (в соответствии с концепциями, которые мы определили ранее в этой книге), она может стать небезопасной онлайн-мишенью для злоумышленников.

## **Безопасность облачно-ориентированной среды**

Как и многие другие темы этой главы, обеспечение безопасности облака рассматривается в нескольких книгах. Здесь представлен всего лишь краткий обзор того, что необходимо учитывать при работе в облаке.

При проектировании или разработке архитектуры для облака следует учитывать принципы «нулевого доверия» и «предположения взлома». Очень важно обеспечить конечному пользователю безопасные настройки по умолчанию. При этом «самый безопасный способ» сделать что-то всегда должен являться и «самым простым». Это можно обеспечить с помощью предоставления безопасных паттернов (конфигураций, инфраструктуры, кода и т. д.).

Следует постоянно применять функции безопасности, которые включены в приобретенный пакет. Например, если в наличии имеется панель с функциями безопасности, необходимо пользоваться ею во всем объеме. Эти услуги уже оплачены вместе с другими приобретаемыми компонентами облачных сервисов, поэтому можно задействовать их в полной мере.

Как и в случае со всеми сетями и приложениями, следует контролировать все, что можно контролировать, сканировать все, что можно сканировать, и устанавливать оповещения обо всех важных событиях. Полезной будет активация ведения журналов и их защита от

злоумышленников с помощью мониторинга, контроля доступа и оповещений.

Многие говорят, что «идентификация в облаке – это новый защитный периметр». Можно соглашаться с этим утверждением или нет, но идентификация (разумеется, с контролем доступа на основе ролей) является замечательным способом защиты ресурсов и служб, находящихся в облаке.

Ключевым моментом при работе с облаком является заблаговременное планирование, обеспечение четкого руководства для пользователей, а также автоматизация безопасных настроек по умолчанию, применения политик и других функций безопасности при любом возможном случае. Если командам разработки и эксплуатации разрешить начать работу в облаке с не настроенными по умолчанию параметрами безопасности, это приведет к проблемам. Поэтому следует проявить инициативу и предоставить как можно больше рекомендаций и поддержки, относящихся к облачной безопасности.

## Облачные потоки

Облачный поток также известен как «пошаговые функции», «логические приложения» и «облачные потоки данных» в зависимости от поставщика публичного облака. Облачные потоки запускаются событиями в облаке, а затем выполняют заранее определенный рабочий процесс (набор действий). Логические приложения являются частью общедоступного облака, и у каждого поставщика облака они называются по-разному, хотя выполняют одну и ту же функцию – запускают рабочие процессы на основе происходящих в облаке событий. Облачные потоки могут быть запущены, когда кто-то пытается войти в приложение слишком большое количество раз; когда есть подозрение, что кто-то сканирует порты; когда кто-то с учетной записью обычного сотрудника пытается получить доступ к тому, что, согласно политике, не должно быть ему доступно, и т. д. Пользователь облака устанавливает, какое событие (или события) запускает логическое приложение, и оно выполняет соответствующий рабочий процесс.

Облачные потоки необходимо тестировать, чтобы убедиться, что они срабатывают в нужный момент и выполняют только

запрограммированные в них действия. Возможно, будет сложно проверить их на безопасность, поскольку они функционируют иначе, чем другие приложения, но, как уже можно догадаться, все равно такая проверка не будет лишней.

**СОВЕТ.** Облачные потоки отлично подходят для защиты систем. При каждом возникновении нового вида инцидента безопасности или обнаружении аномалии следует создать облачный поток, который обработает его или по крайней мере предупредит о нем.

## **Современные инструменты**

Инструментарий для обеспечения безопасности приложений менялся с течением времени, при этом многие концепции были заново изобретены или переосмыслены благодаря новым технологиям. В этом разделе мы рассмотрим новейшие (на момент написания книги) виды инструментов для обеспечения безопасности. Данный список не является исчерпывающим. Он предназначен для того, чтобы дать общее представление о множестве различных видов инструментов, доступных специалисту по безопасности приложений.

### **Интерактивное тестирование безопасности приложений IAST**

IAST – это сочетание статического и динамического тестирования безопасности приложений, которое выполняется внутри приложения во время его использования. Данные инструменты устанавливаются с помощью двоичного файла внутри приложения, работающего в реальном времени, и выдают результаты практически в реальном времени. IAST работают только в том случае, если приложение активно, поэтому, даже если выбранный для тестирования инструмент автоматизирован, следует обеспечить ему контакт с приложением. Такие инструменты тестируют только те части кода или приложения, с которыми взаимодействуют пользователи, отсюда и их название (интерактивные). Они не проверяют весь код, даже если охватывают все части приложения, которые используются конечными пользователями.

Идеальными ситуациями для использования IAST являются: тестирование качества, тест на проникновение или оценка безопасности, использование автоматизированного инструмента сканирования DAST, а также случаи, когда низкий пинг, который они создают, допустим во время эксплуатации. Уровень пинга варьируется от продукта к продукту и от приложения к приложению.

### **Запуск защиты приложений**

Инструменты запуска защиты приложений (англ. Runtime Application Security Protection, RASP) также являются устанавливаемыми в приложение двоичными файлами. Однако обычно они устанавливаются только в эксплуатационных средах. Они действуют как щит, анализируя все входные данные приложения на предмет потенциальной вредоносной активности. Многие из них могут похвастаться использованием искусственного интеллекта (AI) и машинного обучения (ML) для анализа входных данных приложения на наличие неправильных или потенциально вредоносных запросов, в результате которого найденные ненадлежащие входные данные блокируются до того, как попадут в приложение.

RASP также часто называют «обратным прокси» или «WAF нового поколения». WAF означает брандмауэр веб-приложений (англ. web application firewall) – еще один вид инструмента для защиты приложений.

Пинг варьируется от продукта к продукту и от приложения к приложению. Следует помнить, что все инструменты, обеспечивающие защиту приложения, в той или иной степени понижают пинг.

## **Контроль целостности файлов**

Контроль целостности файлов (рис. 8.4) гарантирует, что системные файлы не будут изменены без одобрения администратора. Идея контроля целостности файлов заключается в гарантии того, что вирусы и другие вредоносные программы не обойдут средства контроля приложений с помощью изменения имени на такое же, как у системного файла.

## **Инструменты контроля приложений (список одобренного программного обеспечения)**

Средства контроля приложений используются для обеспечения работы на сервере или хост-компьютере только одобренного программного обеспечения. С помощью этих инструментов компания составляет белый список, приложения из которого могут работать на

всех хост-компьютерах или только на серверных машинах. Все, что не входит в этот список, блокируется. Эти средства могут быть неудобны организациям, не утверждающим программное обеспечение для использования на серверах или хост-машинах.

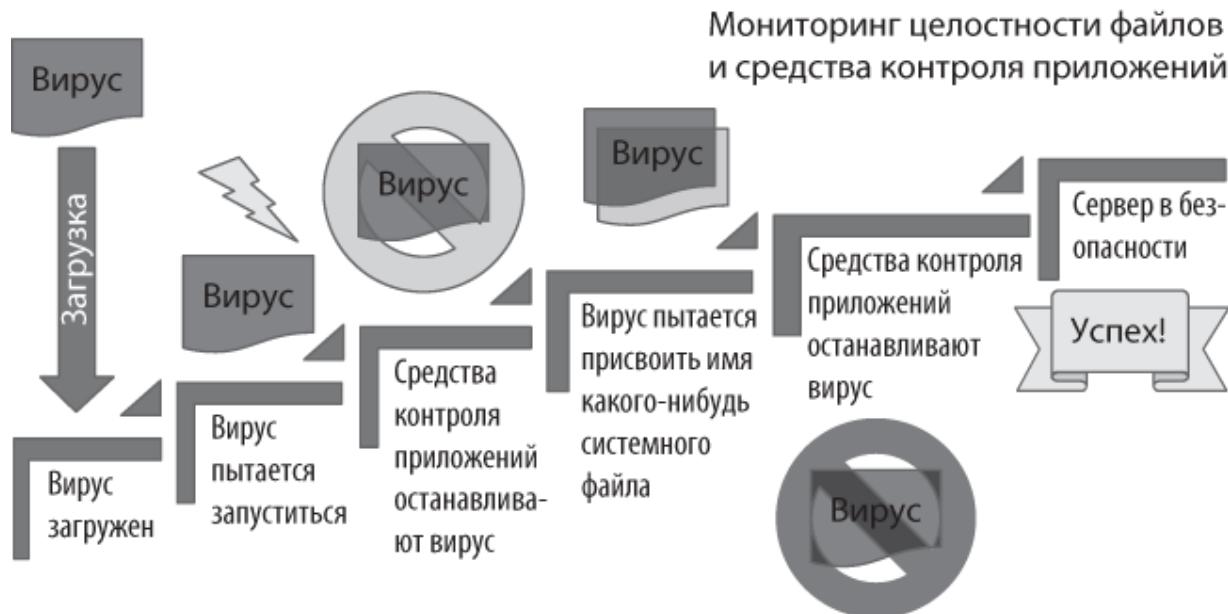


Рис. 8.4. Работа мониторинга целостности файлов и средств контроля приложения

Когда вредоносная программа загружается на компьютер, первое, что она делает, это пытается запустить себя. Если установлен инструмент контроля приложений, он сразу же увидит, что вредоносная программа не входит в список одобренных, и заблокирует ее запуск. Следующее, что делает вирус, – пытается подменить свое имя названием какого-нибудь системного файла, чтобы обмануть средства контроля приложений. Остановить эту атаку можно с помощью мониторинга целостности файлов (описанного в предыдущем разделе). Совместное использование мониторинга целостности файлов и средств контроля приложений обеспечит защиту систем от всех типов вредоносного ПО, кроме разве что самых усовершенствованных.

**ПРИМЕЧАНИЕ.** Для правильной установки и обслуживания этих инструментов требуется довольно много административных

ресурсов, но они значительно снижают риск атаки вымогателя и других типов вирусов.

## **Инструменты безопасности, созданные для конвейеров DevOps**

В настоящее время разрабатывается или выходит на рынок несколько новых видов инструментов обеспечения безопасности, предназначенных специально и исключительно для работы в конвейерах CI/CD. Некоторые из них являются повторением оригинальных идей SAST и DAST, но некоторые – абсолютно новые и довольно изобретательные инструменты.

Совершенно новые инструменты объединяют несколько инструментов вместе и запускают их одновременно, тем самым сокращают время, затрачиваемое на выполнение задач по обеспечению безопасности. Они также объединяют полученные результаты, чтобы на одной панели была видна полная информация по безопасности всех приложений, интегрируемые в конвейеры хранилища секретов и инструменты, анализирующие результаты тестирования всех приложений с помощью AI/ML, и на их основе создают прогнозы рисков и списки приоритетов.

## **Инструменты инвентаризации приложений**

Создание и поддержка в актуальном состоянии полного списка всех приложений, API и SaaS-программ – сложная задача для любого специалиста по безопасности приложений.

На момент написания книги появились новые продукты, помогающие найти приложения в локальной сети или в интернете. Невозможно собрать полный и актуальный список вручную. Новые инструменты инвентаризации приложений будут очень интересны тем, кто пытается создать полную картину всего программного обеспечения в локальной сети, доступного специалистам по обеспечению безопасности (и специалистам по реагированию на инциденты).

## **Автоматизация политики наименьших привилегий и других**

Возможность автоматизировать внедрение, аудит и проверку политик в облачной среде делает работу по внедрению политик более точной, масштабируемой и даже (возможно) приятной. Автоматизация происходит путем установки в среду агентов, которые следят за системами и проверяют их соответствие установленным политикам. Обычно автоматизация применяется в облачных средах, редко – в локальных (традиционных) центрах обработки данных.

Инструменты, автоматизирующие следование политике наименьших привилегий, проверяют привилегии в облаке и удаляют те, которые не использовались в течение длительного времени, тем самым реализуя данную концепцию без каких-либо нарушений. Другие виды автоматизации политик включают в себя применение безопасных протоколов связи (например, только HTTPS), применение заголовков безопасности, включение протоколирования, мониторинга, аудита и других защитных инструментов, оповещение о попытках определенных видов учетных записей повысить привилегии или получить доступ к тем областям сети, которые согласно политике не должны быть доступны, обеспечение многофакторной аутентификации

для всех учетных записей владельцев и т. д. Автоматизация политик чрезвычайно удобна при повышении уровня безопасности облака.

Ранее аудиторам приходилось выяснять на основе анкет и опросов, следуют ли проверяемые ими системы установленным политикам, но такие методы неэффективны и не дают точных результатов. Попытки вручную проверить и обеспечить соблюдение политик, как правило, отнимают много времени и чреваты ошибками (повышают трудоемкость). Автоматизация применения политик – качественно новый и крайне эффективный способ удостовериться, что люди действительно следуют установленным политикам безопасности (комплаенсу).

## Современные тактические приемы

Несколько корпораций, правительств и некоммерческих организаций поделились своими успешными стратегиями и тактиками модернизации деятельности в области безопасности на конференциях, в книгах, технических документах и проектах с открытым исходным кодом. В этот длинный список входят Netflix, Microsoft, Intuit и Канадское учреждение безопасности связи (CSEC).

К сожалению, гораздо чаще хорошими идеями делиться *не* принято, поскольку некоторые считают, что раскрытие их успешных программ безопасности не идет им на пользу. В итоге многие технологии вынуждены полагаться на метод проб и ошибок, а не учиться на чужих успехах и неудачах, что вредит всей IT-отрасли. В этом разделе мы обсудим несколько идей, озвученных в социальных сетях, на конференциях и в неформальных беседах.

Добавление инструментов безопасности в конвейер DevOps – хорошо известная в IT-среде идея. Тем не менее в некоторых случаях создаются асинхронные конвейеры, где длительные тесты запускаются без прерывания процесса сборки. Таким образом, в конвейер выпуска помещают критически важные тесты безопасности, а все эти долгие, медленные, тщательные сканирования, результаты которых будут изучаться еще долгое время после запуска кода в эксплуатацию, можно выполнять в асинхронном конвейере.

Некоторые технологии взяли результаты тестирования на проникновение или оценки безопасности и превратили их в модульные

тесты, чтобы предотвратить появление тех же самых ошибок в кодировании после завершения тестирования. Для получения дополнительной информации можно поискать те же результаты тестирования в остальных приложениях, ведь есть вероятность, что разработчики, которые допустили ошибку в приложении «А», допустили ее и в приложении «Б» или «В». С помощью этих результатов и оценки по ним остальных приложений можно получить больше пользы от тестирования, скорее всего, обошедшегося в круглую сумму.

Еще один вид деятельности, о котором не так часто говорят, но который может серьезно повысить эффективность, – это автоматическое сканирование не только выпускаемого кода в конвейере, но и всего репозитория кода. Можно создать «ловушки перед фиксацией» для запуска различных тестов безопасности, которые затем при необходимости могут заблокировать релиз конвейера. Однако неплохим решением *также* будет запуск тестов, которые не блокируют релизы конвейера, например еженедельное сканирование SCA или SAST всего репозитория. В проверке репозитория можно задействовать находящийся в нем код, так как он просто хранится там.

Создание для разработчиков библиотеки безопасного кода с паттернами и кодом хорошего качества – еще один способ повысить эффективность написания безопасного кода. Эта идея, возможно, кажется довольно простой, но из-за большого количества усилий, необходимых для ее поддержки, она реализуется довольно редко. Если в компании есть разработчик, который также сторонник обеспечения безопасности, его можно назначить своего рода библиотекарем. Результаты этой деятельности видны не сразу, но со временем они производят ошеломительный эффект.

По мере того как все больше организаций становятся смелее и делятся своим опытом, появляется все больше и больше удивительных идей, которые можно оценить, чтобы решить, стоит ли внедрять их в работу. Посещайте конференции, смотрите записи выступлений, следите за различными членами сообщества информационной безопасности, которые делятся идеями в интернете, и используйте другие способы своевременного ознакомления с новыми креативными стратегиями и тактиками безопасности приложений в ИТ-отрасли.

## В итоге

В данной главе мы охватили очень много информации, однако это лишь верхушка айсберга. Главный посыл изложенного здесь заключается в следующем: при использовании новых технологий нельзя забывать принципы безопасности, которые были приведены ранее в этой книге. Любой программный ввод требует проверки, будь то в автомобиле или веб-приложении. Все технологии нуждаются в тестировании безопасности, потому что наша обязанность – защита пользователей, данных и систем. Новые технологии могут иметь новые уязвимости, но на них наверняка будут распространяться некоторые из уже описанных здесь концепций.

## Упражнения

1. Что означает термин «разделяемая ответственность»?
2. В чем разница между инфраструктурой как услугой (IaaS) и платформой как услугой (PaaS)? Какую из них вам придется самостоятельно обновлять и поддерживать?
3. Почему онлайн-хранилища подвергаются (или не подвергаются) большему риску? Что из триады CIA может быть применимо к онлайн-хранилищу?
4. Назовите один новый риск, которому подвержено облако и которому не подвержены традиционный или локальный центры обработки данных.
5. В чем разница между контейнером, виртуальной машиной и физическим сервером?
6. Назовите одно преимущество инфраструктуры как кода.
7. Назовите одно преимущество DevOps перед каскадной моделью жизненного цикла разработки системы.
8. Какой из современных вариантов инструментария показался вам наиболее интересным? Почему?
9. Какая из современных тактических мер по обеспечению безопасности показалась вам наиболее интересной? Почему?
10. Прочитав эту главу, скажите: чего, на ваш взгляд, не хватает вашей организации? Что и как можно улучшить?

# **Часть III**

## **Полезная информация о том, как**

### **постоянно писать код очень высокого**

### **качества**

**Глава 9.** Полезные привычки

**Глава 10.** Непрерывное обучение

**Глава 11.** Заключение

## Глава 9

# Полезные привычки

Разработчики программного обеспечения, системные администраторы, сотрудники службы поддержки и другие ИТ-специалисты, имеющие особые привилегии в технологических системах, являются основными мишенями злоумышленников. Суперсилы, которыми они обладают в своих сетях и системах, очень нужны тем, кто хочет навредить их компании, вот почему этим людям должна быть обеспечена наилучшая защита. В данной главе мы поговорим о полезных привычках, способствующих обеспечению безопасности данных, кода и систем.

Самая первая привычка, которую нужно сформировать, может показаться кликбейтом, однако она таковой не является. Качество создаваемого кода возрастет сразу же, как только разработчики начнут искать в интернете информацию не просто о том, как написать код, а как сделать это *самым безопасным образом*. Как правило, люди посещают одни и те же сайты и останавливаются на наиболее популярных постах. Они копируют текст из них прямо в свое приложение, пытаются скомпилировать его, и если все проходит хорошо, то продолжают работу, не задумываясь о том, не добавили ли они только что уязвимость.

К сожалению, самые первые ссылки обычно рассказывают о самых небезопасных способах достичь нужных результатов<sup>[97]</sup>, поскольку удаление из кода средств защиты (также известное как *ослабление*) часто заставляет все «работать» из-за настройки «открытости». Чрезвычайно опасно вставлять такой код без предварительной оценки последствий этого действия для безопасности разрабатываемого продукта.

Поиск в интернете должен быть сосредоточен на том, как сделать что-либо *наиболее безопасным способом*. Приоритетное внимание к безопасности на каждом этапе разработки способствует улучшению безопасности программного обеспечения.

### Управление паролями

Специалисты в области информационных технологий оперируют невообразимым количеством паролей. Даже обычный пользователь имеет столько паролей, сколько запомнить он попросту не может. Кроме того, индустрия кибербезопасности так долго проповедовала сложность паролей (которые должны были состоять из специальных символов, цифр, заглавных и строчных букв), что у среднестатистического человека пароль получается мудреный, но при этом достаточно короткий, чтобы его можно было запомнить и/или беспроблемно ввести при входе в систему. В результате люди придумывают различные стратегии запоминания паролей: они их записывают, сохраняют в каком-нибудь файле либо в браузере или используют один пароль в разных учетных записях.

## Отмена правил сложности пароля

Короткие пароли очень легко и быстро взламываются с помощью специальных программ. Такие программы угадывают его на основе словарных слов, а затем применяют модели деривации (то есть пробуют различные вариации), пока не найдут совпадение. Чем короче пароль, тем быстрее он подбирается. Поскольку обычному человеку трудно запомнить чрезвычайно сложные пароли, появились пароли меньшей длины, которые, по понятным причинам, являются менее безопасными. Введение правил сложности пароля в конечном итоге приводит к снижению безопасности большинства паролей пользователей.

Длинный пароль, созданный с применением энтропии, крайне сложно взломать автоматическим инструментом [\[98\]](#). Чем больше длина пароля, тем больше времени потребуется на его взлом, поэтому использование максимального количества символов заметно повышает сложность пароля. *Здесь стоит обратить особое внимание на энтропию пароля.* Нельзя говорить о надежности 64-символьного пароля, который состоит из повторения одного и того же символа.

Энтропия подразумевает под собой произвольность. В реальности довольно сложно организовать создание случайных значений с помощью компьютеризированной системы. Ученым-информатикам понадобились десятилетия для решения этой задачи. Однако их усилия

были не напрасны, ведь компьютеры справляются с придумыванием паролей гораздо лучше людей.

Маловероятно, что значения, которые использует человек при создании пароля, имеют случайный характер. Как правило, такой пароль содержит или основан на словах, которые автоматизированная система легко может угадать. Именно поэтому произвольно сгенерированные значения в менеджере паролей представляют собой *намного более безопасные* пароли.

Каждая учетная запись должна иметь произвольный пароль максимальной длины, сгенерированный менеджером паролей.

---

## ПОВТОРНОЕ ИСПОЛЬЗОВАНИЕ ПАРОЛЕЙ

Боб использовал один и тот же пароль на всех сайтах, на которых регистрировался, пока однажды его учетные данные не попали в чужие руки в результате утечки информации. Боб подписался на сервис HaveIBeenPwned, который прислал ему электронное письмо с предложением сбросить пароль, поскольку его учетные данные были скомпрометированы. Сервис также посоветовал ему приобрести менеджер паролей. В этот момент Боб понял, что почти все его аккаунты находятся под угрозой взлома, поскольку на многих сайтах он использовал одни и те же учетные данные (имя пользователя и пароль). Следующие пару дней он потратил на сброс паролей и их замену на случайные значения, сгенерированные менеджером паролей, – не самый приятный опыт в его жизни.

---

## Использование менеджера паролей

Менеджер паролей – это программное обеспечение, которое управляет паролями, шифрует и защищает их без вмешательства пользователя. Он интегрируется в большинство современных браузеров и может работать на телефоне, компьютере, планшете и большинстве других устройств. Менеджер паролей не сохраняет их в браузере, так как представляет собой отдельное программное обеспечение, предназначеннное исключительно для защиты паролей.

Менеджеры паролей не только отслеживают учетные данные (комбинации имени пользователя и пароля) всех аккаунтов в интернете, но в большинстве своем также способны решать проблемы многофакторной аутентификации, автоматически генерируя коды. Они могут сохранять коды для резервных копий, конфиденциальную информацию, например номер социального страхования, номера кредитных карт и все, что следует держать в тайне. Менеджеры паролей также очень полезны при случайном нажатии на фишинговую ссылку, ведущую на сайт-подделку, предназначенный для кражи учетных данных. Когда пользователь переходит на сайт, для которого нет сохраненного пароля, менеджер паролей показывает пустое место, тем самым предупреждая и спасая от атаки.

Менеджеры паролей также создают случайно сгенерированные, очень длинные и сложные пароли. Взлом уникальных для каждого сайта паролей, состоящих из 64 символов, занимает очень много времени. Ко всему прочему, если произойдет утечка данных на одном из сайтов, на котором у пользователя есть учетная запись (и учетные данные станут частью утечки), будет скомпрометирована только она, поскольку злоумышленник получит только один набор учетных данных. Все остальные учетные записи пользователя останутся в безопасности.

Практически невозможно запомнить большое количество паролей, состоящих из 64 произвольных символов, но для программы это довольно простая задача. Человеку нужно помнить только один пароль – тот, который используется для входа в менеджер паролей.

Далее поговорим об этом единственном пароле – парольной фразе.

## Парольные фразы

Пароли для компьютерных систем были впервые созданы в Массачусетском технологическом институте и Bell Laboratories в 1960-х годах с целью получения доступа к системам Unix<sup>[99]</sup>. Тогда они состояли из одного слова – в то время способы защиты изобретали на ходу.

Парольная фраза – это длинное предложение, которое легко запомнить, но сложно угадать. Как правило, доступ к менеджеру паролей осуществляется через кодовую фразу, которую необходимо запомнить, не записывая.

Примерами эффективных парольных фраз являются:

- слова из песни;
- строка из шутки или стихотворения;
- выражающая что-то особенное фраза:
  - У меня две дочери, и я очень люблю их обеих;
  - Мою собаку зовут Спотти, и она просто прелесть!
  - Майор Том, на связи Центр. Вы успешно справились с заданием;
  - Почему у меня ТАК МНОГО ПАРОЛЕЙ? Слишком много!

Большинство менеджеров паролей не ограничивают парольную фразу 64 символами. По этой причине лучше делать ее максимально длинной.

## Отказ от повторного использования паролей

Многие пользователи используют один и тот же пароль, так как это действительно *облегчает* управление большим количеством учетных записей. Однако злоумышленник может обратить повторное использование паролей против своих жертв посредством проведения *атаки с подстановкой учетных данных*.

Киберпреступники за копейки покупают взломанные учетные данные, а затем с помощью скриптов проверяют их на различных популярных онлайн-платформах, то есть осуществляют *атаку с подстановкой учетных данных*. Когда какой-нибудь набор учетных данных срабатывает, злоумышленники начинают обкрадывать человека. Например, они могут купить товары для себя либо заказать что-нибудь, а потом вернуть деньги за заказ, но уже на свои счета. Они

могут украсть информацию или фото, чтобы затем вернуть их владельцу за деньги. Злоумышленник, обладающий учетными данными пользователя, которые подходят для входа на многие различные сайты (то есть в ситуации повторного использования пароля), имеет большую власть над жертвой и ее личностью в интернете.

## **Отказ от ротации паролей**

Многие организации по-прежнему применяют политику ротации паролей, заставляя пользователей менять свои пароли каждые 90 дней. Из-за этого многие пользователи добавляют в конец своих паролей цифры, чтобы их было легче запомнить, но при этом соблюсти требования политики (например, «пароль1», «пароль2», «пароль3»).

Вероятно, в данный момент самым распространенным на планете является пароль, состоящий из названия текущего сезона и года (например, «лето2022»).

---

## **РОТАЦИЯ ПАРОЛЕЙ**

Алиса работала в месте, где действовала раздражающая политика смены паролей каждые 90 дней. По прошествии каждого такого периода она тратила два часа на смену всех своих паролей, половину из которых забывала уже на следующий день, в результате чего еще два часа уходило на телефонные разговоры со службой поддержки. Чтобы сэкономить время, она начала записывать эти пароли на стикеры, которые затем убирала в стол, хотя знала, что так делать не стоит.

Не мучайте своих пользователей – откажитесь от ротации паролей.

---

К сожалению обычных пользователей, организованная преступность становится все более продвинутой в применении технологий для совершения краж в интернете и других противоправных действий.

Злоумышленники поняли, как можно обернуть политику ротации в свою пользу. *Атаки методом подстановки учетных данных с использованием радужных таблиц* предполагают запуск скриптов не только для проверки учетных данных пользователя на популярных платформах, но и для увеличения значений в начале или в конце пароля и попыток войти в учетные записи с помощью полученных вариаций.

Например, если пользователь ставит такие пароли, как «осень2020» или «мойпарольб», то вполне вероятно, что следующим вариантом будет «зима2021» или «мойпароль7». Атаки методом подстановки учетных данных с использованием радужных таблиц учитывают эту закономерность и позволяют преступникам использовать ее в своих интересах.

## Многофакторная аутентификация

Первое, что должен сделать пользователь для защиты своих учетных данных в интернете, – включить многофакторную аутентификацию (MFA). Как уже говорилось в главе 1, факторы аутентификации – это то, что у пользователя есть, то, что является частью пользователя, и то, что пользователь знает. Использование двух факторов означает, что злоумышленнику придется скомпрометировать оба, чтобы получить доступ к учетным записям жертвы. А значит, для атаки потребуется *больше усилий*.

Регистрирующийся на онлайн-платформе пользователь не может повлиять на то, как хранится его имя пользователя и пароль. В идеале все пароли должны храниться с использованием функции хеширования (англ. password-hashing function, PHF), которая усложняет пароли с помощью добавления к ним соли и создания хеша из полученных результатов. К сожалению, в реальности все не всегда так. Не каждая компания следует передовому опыту при создании пользовательского программного обеспечения. Поверхностный поиск в интернете показывает, что любые учетные данные можно приобрести в даркнете за бесценок, а в бесчисленных статьях на эту тему подробно описывается, через какое время пользователи получают уведомление о взломе. Некоторые люди так и не узнают об этом.

Если сайт не защищает данные должным образом, то в случае их утечки злоумышленники получают в открытом виде имена пользователей и пароли, используемые для доступа к этому сайту. Однако преступник не сможет попасть в учетную запись жертвы без прохождения второго фактора аутентификации, и маловероятно, что кто-то будет тратить время на это, если речь идет о взломе обычного пользователя. Преступления, связанные с использованием учетных данных, двухфакторная аутентификация обычно пресекает на корню.

MFA должна быть активирована для каждой учетной записи, представляющей ценность для вас или вашего работодателя.

## Реагирование на инциденты

Даже если вы не входите в группу реагирования, очень важно знать, какие действия от вас потребуются при возникновении инцидента. Никто не хочет узнать об этом, когда чрезвычайная ситуация уже *произошла*, поэтому лучше подготовиться к ней заранее. Представьтесь сотрудникам службы безопасности и попросите их рассказать, что вам нужно знать на случай возникновения инцидента. Скорее всего, они будут очень удивлены и рады вас услышать. Довольно часто службы безопасности забывают сообщить, чего они ожидают от всех остальных сотрудников организации в критической ситуации.

Еще лучше, если вы сами работаете в службе безопасности. В таком случае вы можете провести короткий тренинг для всех остальных сотрудников компании. Хватит пяти минут собрания на объяснение того, что такое инцидент безопасности и какие действия от них ожидаются в случае его возникновения.

Расскажите им:

- что такое «служебная необходимость» и что нельзя распространять информацию о произошедших инцидентах без разрешения;
- кто должен быть доступен в рабочее время, а кого могут вызвать в срочном порядке;
- что у вас есть разрешение привлечь их к действиям по реагированию на инциденты, вне зависимости от выполняемых ими задач или дедлайнов;
- что инциденты безопасности – это чрезвычайные ситуации и службе безопасности требуется содействие, чтобы хорошо выполнить свою работу и защитить организацию.

В конце поблагодарите всех за внимание.

Разговор не должен быть чрезвычайно сложным, но по его окончании сотрудники должны понимать, что нужно, а что не нужно делать. Намного полезнее поговорить с людьми напрямую, чем надеяться, что они обо всем догадаются сами. Широкое информирование не только способствует соблюдению принципа «безопасность – это дело каждого», но и поможет людям быть спокойнее в чрезвычайных ситуациях, поскольку они знают, как действовать и к кому обращаться.

## **Пожарные учения**

Проведение симуляций инцидентов безопасности для различных жизненно важных видов бизнес-деятельности – очень полезное использование времени. Подобно пожарным учениям, подготавливающим к действиям при пожаре, симуляция инцидентов безопасности готовит к чрезвычайным ситуациям, связанным с ИТ-безопасностью. Данный вид мероприятий позволяет еще до возникновения реального происшествия выявить, в каких процессах могут возникнуть проблемы. Лучше всего узнать и устраниить недостатки процесса реагирования на инциденты до того, как экстренная ситуация действительно возникнет. В «пожарные учения» могут входить: атаки красной команды; тестирование систем защиты и оповещения, которое позволяет убедиться в их активности и правильной работе; проверка доступности различных систем, которые в нормальных условиях используются редко; тестирование плана аварийного восстановления; проверка эффективности плана обеспечения непрерывности бизнеса и т. д. Необходимо протестировать все средства, которые могут потребоваться для урегулирования инцидента, и убедиться в их доступности в случае необходимости.

---

## **АВАРИЙНОЕ ВОССТАНОВЛЕНИЕ И ПЛАНИРОВАНИЕ НЕПРЕРЫВНОСТИ БИЗНЕСА**

Несколько лет назад во время выборов Боб вел огромный проект по предоставлению технологических услуг. Заказчики хотели, чтобы у него был не только план аварийного восстановления (англ. a Disaster Recovery, DR), но и план обеспечения непрерывности бизнеса (англ. Business Continuity Plan, BCP). Они очень серьезно относились к соблюдению принципов демократии и осуществлению права голоса. Абсолютно ничто не должно было помешать в день голосования предоставить общественности результаты выборов. Боб составил планы А, Б, В, Г и Д как часть BCP. Использование технологии, разработанной в рамках основного проекта, было планом А. План Б подразумевал переход на имеющиеся резервные реальные сервера с совершенно другим интернет-соединением и интернет-

провайдером в случае выхода из строя основной технологии. Согласно плану В, результаты выборов отправлялись вручную с помощью мобильной связи. План Г – использование спутниковой связи. План Д заключался в том, чтобы отвезти машину с результатами в ближайший город, где есть связь, и отправить их оттуда. Все было очень серьезно.

В рамках плана аварийного восстановления (DR):

- были автоматизированы все развертывания приложения с помощью системы CI/CD;
- был установлен «горячий» центр обработки данных с двумя отдельными центрами обработки данных, в каждом из которых работали точно такие же системы с постоянной синхронизацией;
- были отработаны откаты всех баз данных и серверов приложения;
- был разработан отдельный сайт на случай наводнения или возникновения любого другого физического препятствия, блокирующего доступ к основному операционному центру безопасности (SOC) и сетевому операционному центру (NOC). Это решение было дорогим, но целесообразным. Ничто не могло встать на пути демократии.

Было много решений и действий, но Бобу дали понять главное: несмотря ни на что выборы должны состояться, и от него требовался обеспечивающий это план (BCP). Заказчики также попросили его составить план аварийного восстановления, чтобы в случае настоящей аварии можно было в кратчайшие сроки восстановить работоспособность всех критически важных систем.

---

## Непрерывное сканирование

В описании многих продуктов есть обещание непрерывного сканирования, но лишь некоторые из них способны его выполнить. В этом разделе будет рассказано не о необходимости приобретения того или иного инструмента, а о необходимости самого *непрерывного сканирования* всех доступных компонентов разрабатываемого продукта.

Замечательно, если есть возможность настроить автоматическое сканирование инфраструктуры, сети, баз данных и приложений. В противном случае следует гарантировать использование всех возможностей проведения сканирования вручную и *на регулярной основе*.

При невозможности автоматизации сканирования разработчики должны найти как можно больше ошибок с помощью инструмента DAST, а затем исправить их, прежде чем отправлять код в отдел обеспечения качества или следующей команде (то есть дальше по жизненному циклу разработки).

Инструменты сканирования не идеальны, тем не менее все обнаруженные ими ошибки обычно легко находят и злоумышленник, а значит, обязательно нужно потратить время на их исправление.

Необходимо *непрерывно* выполнять сканирование. Нельзя останавливать этот процесс.

## Технический долг

Технический долг – это решение. Решение отодвинуть модернизацию, исправление и улучшение на задний план. Зачастую его принимает руководство, однако даже простой сотрудник может повлиять на ситуацию. Обновление платформ, обеспечение того, чтобы все среды были идеальными копиями либо зеркалами, рефакторинг старого кода, кажущегося опасным или ненадежным, маскировка эксплуатируемых данных, используемых в других средах, – все это можно предложить руководству добавить в предстоящие проекты, в график работ или вставить между другими задачами.

Как уже говорилось в данной книге, *технический долг – это долг безопасности*. Даже если компания не хочет заниматься повсеместным предотвращением возникновения технического долга, можно попытаться сделать это в рамках собственных проектов, отдав данной деятельности личный приоритет.

## Инвентаризация

К важнейшим мероприятиям по обеспечению безопасности можно отнести создание списка всех приложений компании, которое иногда проводится в рамках «управления портфелем приложений». Без полного и актуального реестра невозможно быстро и эффективно реагировать на инциденты или другие проблемы, так как непонятно, все ли активы компании охватывает мониторинг инструментами SAST, DAST либо IAST, а также другие мероприятия по обеспечению безопасности. По определению, неполная инвентаризация приводит к неполному покрытию средствами безопасности.

Проведение инвентаризации нельзя назвать интересной работой. В ней напрочь отсутствуют какие-либо веселые и захватывающие моменты. Тем не менее это чрезвычайно *сложная* задача. Обеспечить постоянный контроль и автоматизировать процесс ведения актуального списка всех веб-активов компании – труднодостижимая цель, и добиться ее в полной мере пока не удалось никому в ИТ-отрасли.

Реестр должен включать следующую информацию:

- название приложения или API;
- тип приложения, API, веб-приложения, SaaS-продукта и т. д.;
- URL-адрес для эксплуатации;
- URL-адрес для разработки;
- ссылку на местоположение кода;
- контактную информацию команды, ответственной за данное приложение;
- уровень чувствительности данных, используемых в приложении;
- специфику технологического стека (язык, платформа);
- местоположение приложения (облако/локально/ВМ или имя контейнера);
- все остальное, что, по вашему мнению, может быть полезной информацией.

---

## ИНВЕНТАРИЗАЦИЯ

Боб узнал, насколько сложно выполнить инвентаризацию, во время ведения проекта по управлению портфелем приложений. Ему действительно пришлось помучиться с этим проектом, хотя ему в помощь предоставили дорогостоящего консультанта, нанятого компанией специально для этой задачи. Было проведено большое количество интервью и опросов, заполнено множество чек-листов, однако в конце Боб не мог отделаться от ощущения, что этот процесс нужно было автоматизировать. Компания заплатила консультанту целое состояние за проведение опроса всех разработчиков, результатом которого стала лишь вычурная таблица Excel. Боб думал, что в актив компании входило более 200 приложений, однако консультант объяснил ему, что нашлись доказательства существования только 72 из них.

Для Боба так и осталось загадкой, сколько же их было *на самом деле...*

---

## Другие полезные привычки

Ниже перечислены еще несколько полезных привычек типичного ИТ-специалиста, заботящегося о безопасности.

### Политики

Вероятно, это само собой разумеющийся факт, однако нельзя нарушать политику безопасности на рабочем месте. Если возникла ситуация, когда выполнение должностных обязанностей требует нарушения политики, значит, существует проблема, связанная с установленными правилами либо с поставленной задачей. Довольно часто при создании политики ее авторам не удается продумать все ситуации, к которым она может быть применима, поэтому в ней могут быть допущены ошибки. Если политика создает проблемы,

необходимо сообщить об этом своему руководителю, чтобы ее решили пересмотреть.

## Загрузки и устройства

Не стоит скачивать случайные программы, коды, музыку или другие файлы из интернета и запускать их на рабочем компьютере. Нельзя подключать собственные устройства к рабочей сети без зафиксированного в политике разрешения на это. В большинстве случаев в рабочей сети могут располагаться только одобренные устройства, программное обеспечение и другие компоненты.

---

### НЕИЗВЕСТНЫЙ КОД

Боб помнит, как один его коллега скопировал в разрабатываемое им приложение непроверенный код. Этот код обратился к вредоносной библиотеке (к сайту командно-контрольной службы в интернете) и тем самым вызвал инцидент безопасности, который, к счастью, заметила и заблокировала служба безопасности. У коллеги Боба тогда были серьезные проблемы.

---

Все мы уверены в безопасности наших личных вещей, которые приносим на рабочее место. Однако все мы ошибаемся.

### Блокировка рабочей техники

Всегда блокируйте компьютер, когда отлучаетесь от него, если только речь не идет о работе из дома. Нельзя оставлять рабочие системы открытыми на время своего отсутствия. Доверие к коллегам не отменяет опасность инсайдерских угроз, к которым следует относиться с максимальной серьезностью.

---

## **БОЛЬШЕ, ЧЕМ ПРОСТО ШУТКИ**

Каждый раз, когда Алиса видит, что ее подчиненный отлучился, не заблокировав свой компьютер, она отправляет от его имени электронное письмо всему коллективу с темой «Бесплатное пиво!». Несмотря на то что этот поступок немного детский и несерьезный, ей нравится слышать смех команды. Все знают, что произошло.

---

## **Приватность**

С появлением социальных сетей многие начали утверждать, что частная жизнь умерла. Однако это не обязательно так.

Важно хранить конфиденциальную информацию о себе подальше от интернета, поскольку все, что выкладывается в сеть, может быть использовано против человека в атаках социальной инженерии. Например, если разместить в интернете свою дату рождения, девичью фамилию матери и домашний адрес, кто-то может легко воспользоваться этой информацией и получить доступ к интернет-банкингу, сменить номер телефона, к которому привязаны аккаунты, и т. д.

Общественность (люди, не работающие в сфере информационной безопасности) часто недооценивает ценность личной информации. Ниже приведен список того, чем не следует делиться в интернете:

- девичья фамилия матери;
- дата рождения;
- фотография удостоверения личности либо рабочего пропуска;
- домашний адрес;
- номер домашнего телефона;
- фотография водительских прав, свидетельства о рождении или любого другого удостоверения личности;
- номер социального страхования;
- любая информация, связанная с банковским счетом или номерами кредитных карт;
- любая информация, используемая для создания пароля или ответа на контрольный вопрос для доступа к аккаунту;

- фотография номерного знака машины;
- фотография фасада дома (по которому можно угадать домашний адрес);
- фотографии офиса, особенно те, на которых изображена физическая или интеллектуальная собственность.

Довольно часто мы не задумываемся о том, что загружаем в социальные сети. Прежде чем нажать кнопку «Твитнуть», стоит потратить секунду и подумать, может ли выкладываемая информация быть использована против вас или вашего работодателя. Эта секунда значительно снизит риск атаки злоумышленника.

В случае неуверенности по поводу какого-либо аспекта, связанного с безопасностью, следует проконсультироваться с сотрудниками службы безопасности. Они готовы дать вам совет, ведь это буквально их работа.

## Итоги

Очень важно обращать особое внимание на управление своими и чужими паролями (до тех пор, пока необходимость в паролях полностью не исчезнет). Лучшими практиками являются использование менеджера паролей (который требует только одной парольной фразы для доступа ко всем хранящимся там паролям), отмена правил сложности и требования ротации паролей, а также избегание повторного использования паролей. Единственный пароль, который *ни в коем случае нельзя* забывать, – парольная фраза, используемая для входа в менеджер паролей.

В каждой важной учетной записи должна быть включена многофакторная аутентификация. Второй уровень защиты абсолютно оправдывает потраченные дополнительные усилия.

Следует поговорить с членами группы реагирования на инциденты и узнать, не нужно ли им что-нибудь от вас. Проводите учебные тренировки по всем важным вопросам касательно действий в критической ситуации: например, какие действия необходимо предпринять, если расплавится сервер, или сколько времени потребуется на восстановление резервных копий базы данных.

Еще одной важной привычкой является сканирование всего, что только можно, с последующим исправлением найденных ошибок.

Необходимо также сделать все, что в ваших силах, чтобы команда не накапливалася слишком большой технический долг, даже если придется перенести его в будущий проект. Также следует постоянно обновлять реестр приложений, чтобы знать об объектах защиты.

Не следует забывать блокировать компьютер при необходимости отлучиться от него. Рабочие устройства требуют такой же заботы, как и личные. Кроме того, важно внимательно относиться к приватности в интернете, поскольку любая фотография может быть использована против ее владельца. По этой причине стоит подумать дважды перед публикацией какой бы то ни было информации.

Формирование полезных привычек не требует особых усилий, но дает отличные результаты. Станьте защитником безопасности, внеся эти небольшие изменения в свои рабочие будни.

## **Упражнения**

1. Какие риски влечет за собой технический долг?
2. Стоит ли публиковать частную информацию в социальных сетях, если она доступна только друзьям?
3. Почему пользователи редко включают многофакторную аутентификацию? С помощью каких трех способов можно повысить частоту ее применения?
4. Каким менеджером паролей вы пользуетесь? (Ответ «никаким» не принимается.)
5. Какая политика безопасности в вашей компании затрудняет выполнение рабочих обязанностей? Обсуждали ли вы со службой безопасности ее обновление? Можно ли найти компромисс по этому вопросу?
6. Назовите вид деятельности, который, по вашему мнению, мог бы быть полезен в качестве «пожарных учений».

# Глава 10

## Непрерывное обучение

---

Непрерывное обучение – это концепция постоянного развития навыков и знаний в ответ на изменения на работе[\[100\]](#).

---

Осведомленность об изменениях и тенденциях в сфере безопасности повышает эффективность использования проектного времени, а также дает возможность добавить очень привлекательные пункты в свое резюме, произвести положительное впечатление на начальника и коллег и стабильно создавать более качественные и безопасные продукты. Обучение чему-то новому – интересное и перспективное занятие, которое делает профессиональную жизнь более увлекательной. Непрерывное обучение дает только положительные результаты.

Руководитель, составляющий программу обучения для своих сотрудников, знает, что регулярное обучение приводит к более эффективному удержанию сотрудников в компании и высокой степени их удовлетворенности. Они будут работать не только лучше, но и дольше![\[101\]](#)

---

Непрерывное обучение – это концепция постоянного расширения своих знаний для приобретения новых навыков и опыта. С точки зрения бизнеса непрерывное обучение предполагает стимулирование сотрудников к постоянному обучению путем предоставления им соответствующих инструментов[\[102\]](#).

---

## Что изучать

Так как вы читаете книгу о безопасности приложений, рискну предположить, что вам хочется получить больше знаний по этой теме. При обучении очень важно сосредоточить усилия на чем-то одном, поскольку существует так много различных ресурсов, что использовать их все попросту невозможно.

Сперва необходимо решить, какая область безопасности приложений представляет для вас наибольший интерес. Моделирование угроз? Тестирование безопасности? Обучение разработчиков? Все это будет определять, на что именно стоит сделать упор в учебном плане.

### Нападение = защита

Довольно часто ИТ-отрасль фокусируется на наступательной безопасности. Можно найти бесчисленное количество курсов о том, как «взломать это» или как «сломать то». Очень странно, ведь в сфере защиты гораздо больше рабочих мест. Имейте это в виду при выборе предмета для изучения. Если вы не специализируетесь на тестировании безопасности, исследованиях в области безопасности или поиске ошибок, вам незачем зубрить все виды уязвимостей или способы создания эксплойтов. Если есть желание выучить эти вещи – отлично! Однако для выполнения большинства задач в области безопасности приложений вам не нужно будет запоминать, как использовать 100 различных видов уязвимостей.

Если вам интереснее наступательная техника, то можно сосредоточить свою работу и обучение именно на ней, а на изучение защитной техники тратить меньше сил, и наоборот. Нет ничего плохого в том, чтобы отдать предпочтение тому, что нравится больше. Но помните, что *отличная* программа по безопасности полностью охватывает оба этих аспекта.

### Не забывайте о «гибких навыках»

В области безопасности приложений никак не обойтись без умения открыто общаться с людьми и видеть разные точки зрения. Безразличие к тому, что у разработчиков есть свои потребности, сроки и требования, которым они должны следовать, из союзника превращает вас в противника. То же верно и в обратную сторону: понимание специфики работы и задач службы безопасности улучшает взаимодействие с ней.

Кроме того, следует научиться читать и писать отчеты, так как придется интерпретировать результаты работы множества инструментов, обобщать сложную информацию, а затем доносить ее до других. Умение четко описывать и понимать результаты, полученные с помощью различных инструментов и из отчетов, является обязательным условием для успешной карьеры в сфере ИТ-безопасности.

---

## **ЦЕННОСТЬ ГИБКИХ НАВЫКОВ**

Проработавший с разработчиками и другими ИТ-специалистами более 25 лет, Боб повидал немало людей, которым необходимо было поработать над своими гибкими навыками. Его всегда раздражали те, кто считал, что отличные технические навыки отодвигают на задний план вежливость и профессиональное поведение на рабочем месте. Всякий раз, когда Бобу приходилось иметь дело с человеком, который просто не мог найти общий язык с другими, он отправлял его на тренинг по развитию гибких навыков и коммуникации. Обычно сотрудник поначалу негативно реагировал на такое решение, однако по возвращении с курсов начинал делать небольшие успехи. Тогда Боб понимал, что поступил правильно. Некоторые сотрудники даже признавались ему спустя годы, что эти курсы заметно улучшили их жизнь и межличностные отношения вне работы. Иногда руководство проектами тяжело давалось Бобу, однако именно это решение всегда оказывалось верным.

---

Проведите самоанализ: возможно, вам недостает каких-то социальных или «гибких» навыков? Если да, то необходимо включить их в свой учебный план. Их освоение улучшит и другие сферы вашей жизни – вы не пожалеете!

Для своего учебного плана составьте список всех предметов и тем, которые вы хотели бы изучить. Запишите полученные пункты.

## Лидерство!= менеджмент

Обучение лидерским качествам может пригодиться во всех аспектах жизни. Всегда полезно знать, как брать на себя ответственность в нужный момент, а *не при любой возможности*. Также не помешает умение эффективно убеждать людей следовать за вами и помогать в достижении ваших целей. Обучение лидерству – стоящее дело практически для любого человека.

Лидерство нельзя приравнивать к менеджменту. Менеджмент – это о том, как руководить сотрудниками в профессиональной среде; другими словами, как быть начальником.

Менеджер отслеживает больничные, разбирается с нерадивыми подчиненными и распределяет задачи. Получить работу менеджера может только тот, кто обладает этими цennыми навыками.

## Варианты обучения

Существует множество способов обучения и соответствующих ресурсов, поэтому список вариантов будет очень длинным [\[103\]](#).

Однако здесь важно быть избирательным и не тратить свое время впустую. Подумайте о том, какие виды обучения подходят вам больше всего, и на этой основе сделайте выбор. Не пытайтесь ухватиться за все сразу – так вы себя измотаете.

---

## ОБЕДЕННЫЙ СЕМИНАР

Алиса помнит, как работала специалистом по анализу политики: ее повысили до руководителя группы, и она сразу же разочаровалась в своей команде, которая отставала от трендов в отрасли. Алиса хотела, чтобы члены ее команды научились чему-то новому, однако не располагала бюджетом на их обучение, поэтому решила организовать серию «обеденных семинаров». Она пригласила специалистов из других компаний (своих друзей) выступить с презентациями в обеденный перерыв, чтобы ее

подчиненные ели и учились. В этот промежуток времени большинство залов заседаний были пустыми, поэтому найти свободное место было несложно. Семинары получились очень веселыми, и уже через несколько месяцев команда показала очевидный прогресс. Алиса получила премию, *а еще* это достижение было добавлено в оценку эффективности работы отдельным пунктом, в котором говорилось, что она «проявила инициативу, выходящую за рамки ее служебных обязанностей». Алиса и ее команда были очень довольны результатами.

---

### **Действия, которые можно выполнять самостоятельно**

- Читать книги, подобные *этой*.
- Следить за блогами и авторами по интересующим темам.
- С помощью предпочтаемой поисковой системы окунуться с головой в информацию по заинтересовавшей теме.
  - Присоединяться к проектам с открытым исходным кодом, чтобы изучить используемые в них технологии.
  - Рассмотреть возможность получения сертификата и прохождения соответствующей программы обучения.
  - Стать частью сообщества InfoSec: ходить на встречи, заводить друзей, общаться.
  - Создавать разные компоненты, атаковать их, а затем улучшать их защиту.
  - Участвовать в геймификациях: можно записаться на соревнования типа «захват флага» или соревнования cyber range.
  - Слушать подкасты или аудиокниги во время поездки на работу или занятий спортом.
  - Развивать собственную сеть связей с другими людьми для обучения и обмена информацией.
  - Бесплатно смотреть онлайн-выступления на конференциях.
  - Присоединиться к каналам по интересующим темам в Discord или Slack.
  - Следить за деятельностью известных лидеров в области информационной безопасности, подписаться на них в соцсетях

(например, на аккаунт автора этой книги – [@shehackspurple](#)).

- Заново пройти технологические тренинги, которые показали свою результативность. Есть большая вероятность, что при повторении откроется информация, которая была забыта или пропущена.
- Пройти курсы на таких платформах обучения, как Coursera, Udemy и [WeHackPurple.com](#).
- Создавать собственный контент (блоги, видео, семинары): лучший способ закрепить свое понимание темы – объяснить ее другим людям.
- С помощью [#CyberMentoringMonday](#) найти профессионального наставника.
- Стать волонтером и помочь в проведении конференций, тем самым получив бесплатный вход на них.
- Подписаться на бесплатные информационные рассылки по интересующим темам.
- Подписаться на платный контент (например, [Safari Books Online](#), [Pluralsight](#), [lynda.com](#) и [wehackpurple.com](#)).
- Искать ответы в сообществе с помощью хештега [#AskInfoSec](#).
- Не забывать о гибких навыках!

---

## ОБУЧЕНИЕ ДРУГИХ ЧЛЕНОВ КОМАНДЫ

Однажды Боб присутствовал на совещании команды разработчиков программного обеспечения. Ведущий разработчик обнаружила, что один младший разработчик больше года правил свой код сразу на сервере, не проверяя изменения в системе контроля исходных кодов. Она была очень зла, так как в ответ на объяснение *необходимости* использования репозитория младший сказал, что считает это пустой тратой времени. Дискуссия становилась все жарче. Тогда Боб велел всем прекратить кричать и попросил ведущего разработчика вернуться на следующей неделе с подробным объяснением, почему все должны использовать контроль исходных кодов и какую пользу он приносит. «Нельзя убедить команду одними криками о передовом опыте», – сказал ей Боб. Ведущий разработчик очень сердилась, однако на следующей неделе она вернулась с полной

презентацией о преимуществах контроля исходных кодов, рисках, связанных с его игнорированием, и убедила всех членов команды в том, что его использование – в их интересах. С тех пор никто больше не редактировал код сразу на сервере.

---

## **Действия, которые можно выполнять на работе (или о чем можно попросить начальника)**

- Больше узнавать о специфике работы, проходя формальное обучение на рабочем месте.
- Создать на работе базу знаний со всеми советами и рекомендациями для других сотрудников, а также попросить всех внести свой вклад в нее.
- Попросить начальника оплатить конференции по полезным темам.
- Учиться, инструктируя всех новых членов команды.
- Выступить перед своей командой и рассказать о своих планах по обучению. Тогда вам *придется* заняться этим. Закреплению полученных знаний способствует передача их другим людям. Назначение срока будет давить, а некоторым людям такое давление действительно необходимо.
- Попросить выделить из бюджета деньги на нужные учебные материалы для офиса, например на книги, подписки и онлайн-курсы.
- Составить расписание регулярных самостоятельных занятий – возможно, по два часа в неделю, – а затем постараться выделить это время на обучение, не забыв получить разрешение начальника.
- Составить собственный учебный план, предоставить его начальнику и попросить взять вас под контроль.
- Стать частью программы «чемпионов безопасности» при ее наличии в компании.
- Выполнять различные задачи, тем самым получая опыт работы с интересующими технологиями и видами деятельности.
- Спросить, есть ли возможность стажироваться у других сотрудников, имеющих больший опыт или занимающих интересующую должность.
- Начать программу «обеденных семинаров» и приглашать на них специалистов, разбирающихся в интересующих вопросах.
- Присоединиться к программе наставничества или начать ее самостоятельно (многому можно будет научиться в любой из ролей).
- Всегда задавать вопросы. Всегда.

## **Действия, которые можно выполнять в отношении своих сотрудников**

- Проводить с командой открытые обсуждения или дебаты о различных технологиях или технологических решениях.
- Ввести премии за обучение и включать его результаты в оценку эффективности работы.
- Попросить о проведении внебирючих тимбилдингов для совместного обучения.
- Делиться найденным решением или новой для себя информацией с остальными членами команды.
- В некоторых компаниях есть программы, которые позволяют сотрудникам в течение короткого периода времени поработать в другом месте, а затем вернуться с новыми знаниями. Если есть возможность, следует записаться на такие стажировки.
- Открыто поощрять стремление своих сотрудников и коллег к новым знаниям, поздравлять и хвалить их за это.
- Покупать необходимые сотрудникам учебные ресурсы, если вы руководитель и имеете такую возможность.
- Вместе со своими сотрудниками составить учебный план, поддерживать их мотивацию и ежеквартально проверять достигнутые результаты.
- Создать групповой чат только для сотрудников или команды для обсуждения новой информации и взаимоподдержки.
- Внедрить собственную систему управления обучением (англ. learning management system, LMS) в случае ее отсутствия.
- Сформировать учебную группу и создать программу обучения на основе политики, стандартов, руководящих принципов и бизнес-требований компании.
- Проводить тренинги по адаптации для новых сотрудников или членов команды.

## **Подотчетность**

Когда дело доходит до обучения, серьезной проблемой для многих людей становится мотивация. Только вы точно знаете, что может вас вдохновить и почему вы решили учиться, – но принятие на себя

определенных обязательств наверняка поможет не свернуть с намеченного пути к важным для вас целям.

Чтобы обеспечить подотчетность, можно попробовать следующее.

- Найти профессионального наставника (по возможности нескольких), который поможет поднять вашу карьеру на новый уровень.

- Если друг согласится быть партнером по подотчетности, попросите его не давать вам расслабиться и постоянно спрашивать о прогрессе.

- Сделать подотчетность частью официального учебного плана, чтобы знать, что делать.

- Попросить начальника проверять вас и требовать отчеты.

- Награждать себя за достижение важных результатов: например, «Я съем мороженое сегодня вечером, только если закончу читать десятую главу».

- Установить штрафные санкции за недостижение намеченных целей: например, «Если я не закончу десятую главу к концу недели, то пожертвую 50 долларов на благотворительность».

- Добавить автоматические напоминания: уведомление в календаре эффективно подстегивает к выполнению работы.

Не нужно ограничивать себя в идеях. Годится все что угодно, лишь бы это работало для вас.

---

## **ПОДОТЧЕТНОСТЬ**

Когда Алиса ставит перед собой какую-нибудь цель, она объявляет о ней в социальных сетях и всем друзьям и родителям.

Она так делает, потому что знает, что в таком случае ей будет очень стыдно не достичь ее. Всякий раз, когда Алиса чувствует усталость или у нее нет желания что-либо делать, она представляет, как ей будет неловко сказать маме, что она сдалась, и как люди в социальных сетях узнают о ее провале. Эти мысли помогают ей продолжать заниматься. Конечно, она понимает, что никто не расстроится из-за ее неудачи, но, рассказав о ней, она взяла на себя определенную ответственность. Данный трюк творит чудеса с Алисой, и она всегда достигает поставленных

перед собой целей. Это та мотивация, которая заставляет ее идти вперед.

---

## Составление плана

Самым важным шагом при составлении учебного плана является выделение времени для регулярных занятий. Если вы впихнете в первую неделю все, что только можно, то, скорее всего, ко второй неделе у вас не останется ни сил, ни мотивации, а к третьей неделе, возможно, вы и вовсе утратите интерес к обучению. Планирование регулярных учебных занятий с постановкой реальных сроков и требований обеспечит успех в обучении. План можно внести в свое расписание дня, если оно у вас есть.

*Как лучше всего учиться?* Существует несколько различных стилей обучения, но наиболее очевидными являются чтение, слушание, просмотр, письмо и действие. Какой из них кажется вам самым «правильным»? А может быть, таких несколько? Планируйте свою учебную базу с учетом того, что больше всего подходит именно вам.

Определите, что вы уже знаете, а что следует подучить, и тем самым задайте направление своей учебной деятельности.

В конце этой главы приведен шаблон, использование которого неограниченно. На его основе можно сформировать и составить собственный план.

---

### НАМ ВСЕМ НУЖЕН ОРИЕНТИР

В какой-то момент своей жизни Боб решил, что хочет стать этичным хакером. Он понятия не имел, с чего начать, поэтому прочитал все книги и блоги, какие смог найти, посмотрел все видео, которые попались ему на глаза, – у него не было ориентира. Он потратил бесчисленное количество часов, но даже не дошел до уровня скрипт-кидди (в хакерской культуре так называют тех, кто пользуется скриптами или программами, разработанными другими, для атаки компьютерных систем и сетей, не понимая механизма их действия). В итоге Боб сдался. Став старше, он многое узнал об обучении, в частности – о важности составления плана перед началом работы. Такой план он использовал, чтобы стать менеджером проектов: он знал,

какова его цель, определился с сертификатом, который ему нужно было получить, собрал список тем для изучения и на этой основе составил план подготовки. Он выполнил все пункты, сдал экзамен и теперь работает менеджером проектов. Если бы все было так же просто, когда он хотел стать тестировщиком на проникновение...

---

## Действуйте

Теперь, когда вы многое узнали об обучении, пришло время *действовать*. Начните с выполнения упражнений, представленных ниже, затем *заполните* шаблон в конце главы. Составленный план покажите тому, кто будет проверять ваши результаты.

Вы никогда не пожалеете, что овладели новым навыком. Никогда не почувствуете, что потратили время впустую, если достигнете своей цели. Если вы взяли в руки данную книгу и дочитали до этого момента, то, очевидно, вы серьезно настроены на обучение. Составьте план и *начните следовать ему прямо сейчас*.

## Упражнения

1. Приготовьтесь составить для себя план обучения на следующий год.

- Какие книги вы планируете прочитать?
  - Планируете ли участвовать в конференциях? В каких?
  - Планируете ли стать членом профессионального объединения или другого профессионального сообщества? Какого?
  - Планируете ли слушать подкасты? Какие?
  - Планируете ли посещать митапы? Какие?
  - Планируете ли создавать собственный проект для обучения либо проводить соревнования типа «захват флага» или другие практические занятия? Какие именно?
  - Планируете ли проходить формальное обучение?
2. Где вы найдете время на обучение? Предоставит ли его вам ваш начальник? Выделите ли вы на это свое личное время? Какой

приоритет имеет обучение в вашей жизни?

3. Перечислите три новых навыка, которым вы хотели бы овладеть в этом году. Почему вы выбрали именно их?

4. Назовите три навыка, обучение которым помогло бы вам улучшить качество выполняемой вами работы.

5. Какой стиль обучения вам подходит больше всего? Чтение? Слушание? Действие? Просмотр? Какой-нибудь другой стиль? Расскажите о нем как можно подробнее.

6. Что мотивирует вас учиться? Вознаграждение? Признание? Дополнительный досуг? Сладкие десерты? Подумайте о вашей личной мотивации и как ее можно использовать, чтобы продолжать учиться круглый год.

7. Какой формат лучше всего подходит вам для усвоения информации? Занятия? Учебная стажировка? Наставничество? Соревнования? Назовите свои любимые форматы.

8. Где вы будете учиться? Дома? На работе? В кафе рядом с домом по субботам? Используете ли вы рабочий или домашний компьютер? Какое оборудование вам необходимо? Где вы планируете выделить место для учебы? Необходимо понимать, когда и где вы будете заниматься.

9. Как вы получите доступ к ресурсам, необходимым для обучения? Бесплатны ли они (подкасты)? Сможете ли вы убедить начальника приобрести для вас платные ресурсы? При каких условиях вы сможете платить за них самостоятельно? Составьте бюджет в конце учебного плана.

## Учебный план

Имя: \_\_\_\_\_

Предмет / темы изучения / посещаемое мероприятие: \_\_\_\_\_

Необходимые ресурсы: \_\_\_\_\_

Время/расписание обучения: \_\_\_\_\_

Формат(ы) обучения: \_\_\_\_\_

Необходимый бюджет и его источник: \_\_\_\_\_

План подотчетности: \_\_\_\_\_

Я, \_\_\_\_\_ (имя), обязуюсь изучить этот предмет/тему к \_\_\_\_\_ (дата достижения цели).

\_\_\_\_\_ (*подпись*)

# Глава 11

## Заключение

*Безопасность – дело каждого.*

**Таня Янка**

Каждый человек несет ответственность за выполнение своей работы наиболее безопасным способом, в противном случае он подрывает доверие своего работодателя. Если в компании существует политика безопасности, следует ее *соблюдать* наравне с любой другой политикой. Нанимая сотрудника, работодатели ожидают от него добросовестного выполнения обязанностей, под которым, помимо всего прочего, подразумевается обеспечение безопасности.

Если вы работаете в службе безопасности, то лучше других знаете, что невозможно обеспечивать безопасность от имени всех сотрудников организации или самолично контролировать соблюдение всех необходимых мер. Например, невозможно быть рядом каждый раз, когда сотрудник принимает решение о том, пустить ли подозрительного человека в здание, или когда он придумывает очередной пароль. Невозможно постоянно наблюдать за работой программиста и следить за тем, чтобы его код отвечал рекомендациям безопасности. Как бы мы ни старались, невозможно проверить каждое предпринятое сотрудником действие, связанное с обеспечением безопасности. Необходимо предоставить людям соответствующую подготовку, а затем надеяться на то, что они имеют честные намерения и делают все возможное, чтобы выполнять свою работу самым безопасным и надежным способом. Мы рассчитываем на их поддержку в деле обеспечения безопасности нашей организации, работающих в ней людей, систем, данных и клиентов.

Некоторые полагают, что «если безопасность – дело каждого, значит, никого», но это далеко от истины. Специалист службы безопасности изготавливает служебные пропуска, внедряет инструмент SCA или пишет политику безопасности. Рядовой сотрудник выполняет

предписанные ему действия по обеспечению безопасности: для входа в здание использует пропуск, исправляет код по результатам сканирования SCA, соблюдает политику безопасности. Служба безопасности осуществляет руководство, направляет и предоставляет необходимые инструменты. Благодаря этому сотрудники стараются соблюдать все меры безопасности при выполнении своей работы. Помимо всего прочего, службе безопасности также иногда приходится выполнять дополнительные обязанности (помогать завершить работу по обеспечению безопасности), реагировать на отзывы (корректировать политику или процессы) и делать все возможное, чтобы обеспечить поддержку всей компании в соблюдении ее политики и руководящих принципов. Это командная работа.

При написании программного обеспечения рамки действия этого правила становятся шире. Каждый разработчик обязан написать самый лучший код, на который он только способен; участвовать во всех мероприятиях по обеспечению безопасности; действовать в соответствии с результатами проверок, выполненных различными инструментами, и предупреждать службу безопасности об обнаруженных проблемах. Если в коде отсутствуют компоненты защиты, то, скорее всего, он является представляющим угрозу слабым местом.

Работа по обеспечению безопасности компании заключается в повседневных решениях и действиях всех сотрудников, а не только тех, кто работает в службе безопасности. Мы все вместе отвечаем за защиту нашей организации, а служба безопасности лишь предоставляет инструменты, ресурсы и руководство для этого.

Невозможно достичь цели, если к ней не будут стремиться все работники или хотя бы их большинство. *Необходимо* работать сообща. Осознание того, что безопасность – сфера ответственности и часть работы каждого человека, – это залог ее обеспечения.

## **Вопросы, оставшиеся без ответа**

После прочтения этой книги у вас могут остаться некоторые наболевшие вопросы:

- Когда можно говорить о достаточности предпринятых мер по обеспечению безопасности?
- Как привлечь руководство к обеспечению безопасности?
- Как привлечь разработчиков к обеспечению безопасности?
- С чего начать?
- Откуда брать помощь?

Рассмотрим их по очереди, прежде чем отправиться на борьбу с угрозами безопасности приложений.

### **Когда можно говорить о достаточности предпринятых мер по обеспечению безопасности?**

Если в организации отсутствует программа обеспечения безопасности или имеются очень слабые средства безопасности, любые действия в этом направлении приведут к положительным результатам. Возможностей для прогресса в данной ситуации бесчисленное множество, поэтому выглядеть хорошо в глазах начальства легче легкого.

Что делать, если в организации очень продвинутая и устоявшаяся программа безопасности, с которой все согласны, и обеспечена правильная работа всех процессов? Как понять, когда нужно остановиться?

Наименее удовлетворительный, но наиболее правильный ответ – все зависит от обстоятельств.

**СОВЕТ.** Приемлемый для организации уровень риска – это показатель того, какой риск она готова принять. Например, небольшой стартап может быть готов пойти на большой риск, в то время как очень крупная или бюрократическая организация, скорее всего, будет избегать рискованных решений.

Все зависит от стоимости защищаемых активов: вероятно, нет смысла тратить миллион долларов на защиту чего бы то ни было стоимостью в сто тысяч.

Все зависит от уровня риска, на который может пойти компания. Вносит ли она большие изменения в технологию одной из первых или ждет основной волны внедрения? Насколько быстро принимаются решения в компании? Они представляют собой сложные процессы с несколькими уровнями утверждения или действует культура доверия? Ответы на эти и другие вопросы помогут понять, насколько компания готова к риску, и определить необходимое количество уровней защиты.

Все зависит также от уровня чувствительности защищаемых данных и систем. Например, содержит ли система ценную интеллектуальную собственность или государственные секреты? Поддерживает ли она жизнь человека (как кардиостимулятор Боба)?

Все зависит от бюджета. Любая компания может написать на своем веб-сайте: «Мы серьезно относимся к вашей безопасности», но крошечный бюджет, выделяемый на ее обеспечение, и отсутствие квалифицированных сотрудников позволяют усомниться в данном заявлении.

Все зависит от заинтересованности высшего руководства и ведущих специалистов других отделов. Если высшее руководство хочет совместимость с PCI (англ. Payment Card Industry – стандарт безопасности индустрии платежных карт), значит, именно она станет вашей главной целью. Если оно хочет увидеть план и бюджет для запуска новой программы безопасности, стоит перечитать седьмую главу. Если руководство хочет проверить безопасность нового разработанного компанией телевизора, но практически не выделяет время на тестирование, это также говорит само за себя. О достаточности предпринятых вами мер можно с уверенностью судить только после беседы с высшим руководством о соответствии результатов вашей работы его ожиданиям.

---

## РИСК НЕГАТИВНЫХ ПОСЛЕДСТВИЙ

Когда-то Алиса работала в банке. Банк боялся вносить какие-либо изменения до такой степени, что этот страх фактически

приводил к риску возникновения негативных событий. В нем совершенно не торопились обновлять систему и средства контроля безопасности или пробовать внедрять новые технологии, и в результате иногда оказывалось, что вносить какие-либо изменения уже слишком поздно. Для наиболее важных систем все еще имелись мейнфреймы, которые сдерживали модернизацию банковских систем. Все многочисленные аргументы Алисы в пользу изменений игнорировались. В банке слишком боялись перемен.

---

## СПЕЦИАЛЬНОЕ КОНСУЛЬТАНТОВ

У консультантов, проводящих аудит, оценку или тест на проникновение в системе безопасности, скорее всего, время для завершения работ будет ограничено по контракту. Многие из них говорят, что «сделали все возможное в рамках контракта» – за что клиент заплатил, то он и получил.

В качестве альтернативы в данной ситуации можно составить список всех пунктов, которые необходимо выполнить для обеспечения безопасности систем заказчика, а также определить, какие из них не покрываются условиями контракта. Например, можно порекомендовать провести не вошедшее в контракт тестирование безопасности API, сделать обзор кода для определенных функций безопасности или выполнить стресс-тестирование, поскольку нагрузки представляют собой огромный риск для бизнес-модели заказчика. Возможно, клиенту следует подумать о создании команды безопасности и запуске соответствующей программы. В такой список следует включить даже ту работу, которую невозможно выполнить. Хороший консультант предоставляет клиенту право решать, игнорировать его рекомендации или нет. Необязательно выполнять весь список задач, однако следует хотя бы объяснить, как, по вашему мнению, должен выглядеть «результат выполненных работ». Такое действие часто называют «оценкой упущений» (англ. gap assessment).

## ПРИМЕЧАНИЕ

## ДЛЯ

При решении того, что необходимо добавить в оценку упущений, обычно используются такие контрольные списки, как ASVS (Стандарт проверки безопасности приложений OWASP), MASVS (мобильная версия ASVS от OWASP), WSTG (Руководство по тестированию веб-безопасности OWASP), MSTG (мобильная версия WSTG). С их помощью можно подтвердить, что были охвачены все пункты, представляющие важность для клиента. Заранее созданный шаблон сделает это занятие не требующим больших усилий и принесет большую пользу клиентам.

Результатом такой работы может стать также увеличение количества контрактов.

---

## Как привлечь руководство к обеспечению безопасности?

Работа в компании, где никто не относится к безопасности серьезно, может стать крайне неприятным испытанием. Человеку очень сложно быть удовлетворенным своей работой, если 50 % времени он тратит на переговоры и просьбы, от которых зависит выполнение его задач.

Есть несколько способов привлечь руководство и другие команды к обучению безопасности приложений, если отсутствие взаимопонимания представляет серьезную проблему.

Сначала стоит попробовать привлечь пчел медом. Проводите обеденные семинары, выступайте с презентациями и делитесь информацией о проблемах безопасности, с которыми сталкивается компания. Просвещение – отличный способ привлечь людей к делу. Даже если человек пришел «просто посидеть» на таком собрании, возможно, он поймет, что в случае необходимости всегда сможет обратиться за помощью. Сотрудники часто вносят свой вклад в обсуждения, независимо от того, хотят они этого или нет.

Далее следует объяснить риск так, чтобы мог понять каждый. Не надо восклицать: «Это входит в CVE (CVE – Распространенные уязвимости и риски (англ. Common Vulnerabilities and Exposures)) наивысшей степени риска!» Лучше сказать: «Данная уязвимость в платформе Java Struts позволит злоумышленникам взять контроль над зараженными веб-серверами с помощью всего одной строки кода. Таким образом они окажутся внутри нашей сети, минуя межсетевой экран, и смогут атаковать наши ресурсы напрямую. Мы уязвимы, я проверил(-а). Мне нужно разрешение на создание виртуального патча с использованием инструмента RASP для блокировки подобных атак, а также ваша помощь в его тестировании, чтобы я был(-а) уверен(-а), что все делаю правильно. Вы со мной?» Кто сможет отказать в такой аргументированной просьбе? Когда кажется, что мир рушится, но поддержки нет, часто причиной ее отсутствия является именно проблема коммуникации.

Другой вариант получения поддержки – провести проверку концепции (РоС) беспокоящей проблемы. «Не используете заголовки безопасности? Позвольте мне показать наш веб-сайт во фрейме, из которого злоумышленник легко может украсть учетные данные

пользователей». Проверка концепции эксплойта для уязвимости, которую никто не хочет исправлять, – отличный способ вернуть эту проблему в топ списка в системе отслеживания ошибок. Тем не менее данный подход довольно трудоемок, поэтому, прежде чем приступать к его применению, следует убедиться в том, что он будет принят положительно. Проверку концепции надо проводить только в особых случаях.

Последней идеей, которую можно представить по этой теме, является использование документа «Подписанный список рисков», который я придумала в ответ на игнорирование моих отчетов по пентестированию. В нем подробно описываются все оставшиеся уязвимости, выявленные в ходе тестирования безопасности, а также связанные с ними бизнес-риски. Документ подается на подпись высшему руководителю службы безопасности организации (обычно это главный специалист по информационной безопасности (CISO) или главный специалист по безопасности (CSO)), который тем самым «принимает изложенные в нем риски». Цель такого документа – указать на то, что руководство было проинформировано об имеющихся проблемах и дальнейшее действие будет означать принятие соответствующих рисков. Если произойдет взлом системы через указанную в списке уязвимость, ответственность за него будет нести подписавший его человек. Возможно и такое, что, ознакомившись с этим документом, руководство даст его составителю полномочия на устранение проблем силами других команд.

Насколько я знаю, никто никогда не подписывал такой документ. Всем моим знакомым, кто использовал эту тактику, всегда предоставлялись полномочия для проведения необходимых изменений.

**СОВЕТ.** Используйте последний метод только в случае крайней необходимости. Данный трюк работает лишь один раз.

## **Как привлечь разработчиков к обеспечению безопасности?**

Разработчики программного обеспечения – это строители, которые ежедневно буквально из ничего создают что-то. Большинство из них гордится результатами своей работы и хочет, чтобы их приложения были безопасными. Разработчики пишут небезопасный код не

специально: либо они не знают, как улучшить защиту, либо им не хватает времени и ресурсов.

Поскольку программисты стремятся к созданию качественного ПО, а это невозможно без обеспечения высокого уровня безопасности, то привлечь их к делу – не такая уж сложная задача. Есть три области, на которые необходимо обратить внимание: знания, понимание и ресурсы.

**Знания:** как уже говорилось ранее, большинство курсов и тренингов по программированию не учат тому, как обеспечить безопасность создаваемого ПО. Вот здесь-то можно вступить в игру и научить их! Дайте разработчикам знания, необходимые для выполнения их работы. Отправляйте их на тренинги. Покупайте им книги (например, эту... да, купите им эту). Сделайте все возможное, чтобы они получили информацию, необходимую для создания максимально безопасного программного обеспечения.

**Понимание:** всем ИТ-специалистам нужно знать приоритеты, потребности и мандаты в области обеспечения безопасности. Если разработчик понимает важность вашей просьбы, он с гораздо большей вероятностью включит ее в свой список приоритетов. Вместо того чтобы указывать ему, что конкретно нужно делать, лучше показать значимость поставленной задачи и способ ее выполнения.

**Ресурсы:** ошибки не исправляются в вакууме. Разработчикам требуется время, отведенное в проектных графиках, чтобы выполнить действия по обеспечению безопасности, в противном случае они будут проигнорированы. Также необходимы инструменты, справочные материалы, а иногда и дополнительная пара рук. Ваша задача – предоставить разработчикам возможность создавать безопасное программное обеспечение, и иногда под этим подразумевается, что нужно самому засучить рукава и потратить несколько дней на исправление ошибок, связанных с безопасностью, чтобы помочь им закончить работу в срок. Обеспечьте разработчиков всеми необходимыми ресурсами, затем сядьте и наблюдайте за их трансформацией.

## С чего начать?

Вы только что прочитали книгу о безопасности приложений, так что уже движетесь к созданию безопасного программного обеспечения.

Разработчику можно посоветовать начать вносить изменения в свою работу, внедряя в нее полученные здесь знания. В каждом последующем проекте применяйте руководящие принципы касательно безопасного кодирования, безопасного проектирования и требований безопасности. Познакомьтесь со службой безопасности и спросите, не нужен ли им новый боец. Всегда выясняйте, как написать код наиболее безопасным способом, а не копируйте первые результаты поиска.

Специалист в области безопасности приложений может использовать полученные знания для повторной оценки своей текущей программы безопасности. Каковы ее цели? Они достигаются? Какие новые виды деятельности или инструменты можно добавить в программу? Измерения проводятся регулярно? Если нет, то уже сейчас следует вносить соответствующие изменения. Кроме того, чемпионы по безопасности компаний, вероятно, оценят те знания, которыми вы с ними поделитесь.

Если вы надеетесь работать в сфере безопасности приложений, данная книга – первый шаг на этом пути. Чтобы помочь людям понять, какая позиция в нашей сфере интересна им больше всего, и рассказать о соответствующих ей требованиях, я создала постоянный цикл прямых трансляций, проходящих каждый четверг в 18:00 по тихоокеанскому времени на YouTube, под названием «We Hack Purple: Поиск своего места в области информационной безопасности». Все ролики сохраняются на YouTube-канале We Hack Purple. Каждый из них посвящен отдельной позиции. В видео различные специалисты в IT-области рассказывают о специфике той или иной работы, требуемом образовании и первых шагах.

Не будет лишним также заглянуть в различные сообщества, посвященные теме обеспечения безопасности, завести в них друзей и присоединиться к веселью. Там всегда можно узнать что-то новое, а сообщество InfoSec постоянно нуждается в новых членах.

## Откуда брать помощь?

В области информационной безопасности есть несколько вариантов действий на тот случай, если вы зашли в тупик.

Для начала попытайтесь разобраться самостоятельно. Прежде чем задавать вопросы другим людям, поищите информацию в интернете. Если проблема связана с работой, попросите помощи у коллег. Однако лучше сначала попытаться справиться своими силами и проверить собственные идеи.

Второй шаг – расширить круг поиска. Можно попросить помощи у своего наставника или у других знакомых, пользующихся вашим доверием. Если вы регулярно участвуете в митапах, напишите кому-нибудь из группы или задайте ему интересующий вопрос во время следующей встречи. Затем попробуйте перефразировать вопрос, поскольку к этому моменту вы, скорее всего, будете обладать большей информацией.

Третий шаг – еще больше расширить круг поиска, обратившись к сообществу. Опубликуйте вопрос в Twitter с хештегом #askinfosec или разместите сообщение на LinkedIn, отметив в нем близких друзей. Обратитесь к OWASP, We Hack Purple, DevOpsDays и другим сообществам, которые проводят онлайн-собрания и другие мероприятия по обучению и общению.

Вопрос без ответа может стать предметом личного исследования, и, когда вы в конце концов найдете решение, обязательно поделитесь им с другими пользователями интернета. Ваше открытие поможет всем.

## Заключение

Спасибо, что прочитали эту книгу. Теперь вы готовы к борьбе за правое дело. Вся представленная здесь информация наверняка поспособствует заметному улучшению безопасности создаваемого вами программного обеспечения. Время, которое было потрачено на получение знаний, изложенных в этой книге, показывает ваше стремление к обучению, а также высокую степень важности для вас данной темы. Пожалуйста, используйте эту информацию в помощь другим, чтобы мы все могли жить в более безопасном мире.

*Мы, сотрудники службы безопасности, зависим от всех и каждого из вас. Не подведите нас.*

# Приложение

## Ответы

Разобрать все вопросы из этой книги можно, приняв живое участие в открытых онлайн-обсуждениях или просмотрев их запись. Мы не хотим оставлять вас в подвешенном состоянии.

Подписывайтесь на рассылку новостей SheHacksPurple, чтобы получать приглашения на обсуждения, которые будут проходить в прямом эфире на сайте [newsletter.shehackspurple.ca](http://newsletter.shehackspurple.ca), а также на YouTube-канал SheHacksPurple, чтобы получить доступ к их записям: [youtube.com/shehackspurple](http://youtube.com/shehackspurple).

### Глава 1. Основы безопасности

1. Боб установил в настройках Wi-Fi на своем кардиостимуляторе запрет на передачу имени своего Wi-Fi. Как называется эта стратегия обеспечения безопасности?

Безопасность через неясность.

2. Назовите пример значения, которое может быть жестко закодировано, и объясните почему. (Почему программист сделал бы это?)

В свое время, будучи разработчиком, автор книги использовала жестко закодированные строки подключения для dev-, QA- и prod- сред, которые могла переключать при тестировании. Она тогда и понятия не имела, какие проблемы с безопасностью вызвала этим.

3. Является ли капча удобной мерой безопасности? Почему?

Нет. Капча очень сложна для людей с нарушениями зрения и вообще для людей с ограниченными возможностями. Пользователи терпеть не могут ее, она раздражает и надоедает.

4. Приведите один пример хорошей реализации удобства и безопасности.

Менеджер паролей 1Password можно использовать в качестве многофакторного аутентификатора. Он генерирует код, который нужно

ввести для входа в учетную запись сайта в качестве второго фактора. Как только пользователь вводит свое имя пользователя и пароль, менеджер паролей автоматически копирует сгенерированный код в буфер обмена, который сразу можно вставить в нужное поле. После успешного входа код в буфере обмена будет заменен тем, что было там до него, на случай, если вам это понадобится. Вот пример одновременного обеспечения удобства и безопасности. Использование менеджера паролей делает многофакторную аутентификацию менее трудоемкой, более приятной и менее подверженной ошибкам (и, по мнению автора, справляется с этим довольно успешно). Данное решение обеспечивает высокий уровень безопасности, но в то же время оно очень простое для реализации пользователями.

5. Необходимо ли подтверждать данные, полученные при использовании параметров URL? Почему?

Заменить данные в параметрах URL со стороны пользователя легче легкого, поэтому им нельзя доверять. Если пользователь заходит на сайт без HTTPS, используя сеть кафе, злоумышленнику не составит труда изменить параметры посредством MITM-атаки (Man-In-The-Middle или Manipulator-In-The-Middle). Существует множество ситуаций, способных привести к злонамеренным или даже случайным изменениям входных данных, которые могут представлять опасность для приложения. Необходимо всегда проверять *все* вводимые пользователем данные.

6. Если сотрудник узнает коммерческую тайну, а затем продаст ее конкуренту, какую часть (или какие части) триады CIA он нарушит?

Конфиденциальность, так как сотрудник наверняка нарушил условия найма, изложенные в подписанных им документах о неразглашении, а также, скорее всего, закон.

7. Вы купили смарт-холодильник и подсоединили его к домашней сети. К нему подключился злоумышленник и в настройках повысил температуру, в результате чего ваше молоко испортилось. Какую часть (или какие части) триады CIA он нарушил?

Целостность, поскольку данные все еще доступны, а их конфиденциальность, вероятно, не нарушена (как правило, данные настроек не считаются секретом). Изменение же настроек, результатом которого стало повреждение продукта в холодильнике, нарушает основу принципа целостности.

8. Если кто-то взломает ваш умный термостат и отключит отопление, какая часть или какие части триады CIA будут нарушены?

Доступность, потому что в результате взлома вы лишились доступа к теплу. Надеюсь, вы живете не в Оттаве, а если и живете там, то у вас сейчас не февраль!

9. Считается ли инсайдерской угрозой добавленная программистом «пасхалка» (дополнительный код, выполняющий незарегистрированные функции, в качестве «сюрприза» для пользователей, о котором неизвестно руководству и команде безопасности)? Почему?

Да, это инсайдерская угроза. Обычные розыгрыши, происходящие внутри компаний и затрагивающие только ее работников, могут быть веселыми и смешными. Неодобренные функции или «неожиданные» параметры функциональности – худший кошмар службы безопасности.

10. Какие меры предосторожности можно предпринять при подключении к общественному Wi-Fi, чтобы обеспечить «глубокую защиту»?

Можно обеспечить посещение веб-сайтов только через HTTPS для шифрования трафика. Можно использовать VPN (виртуальную частную сеть), которая создает «туннель» либо к безопасной сети, либо к рабочей сети (как правило). Можно установить на свой компьютер антивирус или антивирусное ПО. Можно просто использовать сотовую связь, если есть подозрение, что сеть, к которой будет проведено подключение, особенно опасна (например, если пользователь находится в одном из кафе Лас-Вегаса во время проведения Def Con).

11. Если вы живете в квартире с несколькими соседями и у каждого из вас есть ключ от двери, считается ли ключ «фактором аутентификации»?

Нет. Ключ идентифицирует каждого из хозяев ключей как того, кто живет в квартире, но не различает их. Аутентификация должна идентифицировать личность, а не члена группы.

## Глава 2. Требования безопасности

1. Назовите два дополнительных возможных требования безопасности для веб-приложения (не упомянутых в списке).

1. Пользователь не может сбросить пароль более одного раза в 24 часа.

2. Каждый API должен подключаться через специально выделенную сервисную учетную запись, которая не предназначена для повсеместного использования.

2. Назовите два дополнительных возможных требования безопасности для операционной системы автомобиля.

1. При нарушении работы тормозов, подушки безопасности или любых функций двигателя система автомобиля немедленно отправляет сигнал тревоги пользователю и в указанный им автосервис. Движение автомобиля невозможно без разрешения автомеханика.

2. Каждый владелец автомобиля имеет собственный ключ для идентификации. Завести автомобиль можно, только пройдя еще один фактор аутентификации (например, отсканировав отпечаток большого пальца). Всем не прошедшем аутентификацию водителям должно быть отказано в доступе.

3. Назовите два дополнительных возможных требования безопасности для «умного тостера».

1. Температура тостера не может превышать X градусов. Если температура вышла за пределы разрешенного значения, должно быть выдано предупреждение и приостановлена работа тостера.

2. Чтобы установить в режиме управления тостером такие температурные параметры, которые превышают рекомендованные производителем, требуется пройти второй фактор аутентификации.

4. Назовите два дополнительных возможных требования безопасности для приложения, работающего с кредитными картами.

1. Приложение соответствует стандарту PCI (стандарту безопасности индустрии платежных карт).

2. База данных, содержащая информацию о кредитных картах, помечена как «для служебного пользования». Доступ к ней имеет только одна служебная учетная запись (приложения), попытки несанкционированного доступа фиксируются с выдачей соответствующего предупреждения. Раз в месяц проводится ручной аудит доступа к этой базе данных.

5. Какое требование безопасности является наиболее важным для вас или вашей организации? Почему?

Я бы выбрала «**Не доверяйте никому: проверяйте (а в особых обстоятельствах санируйте) все данные, даже из вашей собственной базы данных**». Я твердо убеждена, что, если бы каждое приложение выполняло строгую проверку ввода, это сразу заметно повысило бы уровень безопасности интернета.

6. Если бы вам нужно было убрать одно из упомянутых в этой главе требований из проекта веб-приложения, какое бы это было требование? Почему?

Я бы убрала HTTPS, так как знаю, что заказчик проекта все равно вернет его обратно. Браузеры не будут пускать пользователей на сайт без HTTPS, что может нанести вред бизнесу. Вот почему я бы выбрала именно этот вариант.

## Глава 3. Безопасность при проектировании ПО

1. Когда следует шифровать данные? Выберите все подходящие варианты.

- А. Когда один API отправляет данные другому API.
  - Б. Если данные расположены на выключенной виртуальной машине.
  - С. Если данные хранятся в базе данных.
  - Д. При отправке данных с сервера в браузер.
- Во всех перечисленных случаях.

2. Какими способами можно обеспечить безопасность используемых компонентов от сторонних производителей? Как минимизировать риск в этой области?

Каждый сторонний компонент можно проверить либо посредством ручного обзора, либо с помощью инструмента анализа состава программного обеспечения.

3. Где необходимо хранить секреты приложения? Каким образом приложение должно получать доступ к ним?

Лучшим способом обеспечить сохранность секретов (учетных данных, паролей, хешей, строк подключения и т. д.) и безопасный доступ к ним является использование хранилища секретов.

4. Назовите три типа «секретов».

Учетные данные (имя пользователя и пароль), строки подключения, пароли, хеши.

5. С какими угрозами может столкнуться мобильное приложение банка? Назовите три угрозы и оцените вероятность их возникновения и степень опасности (низкая, средняя или высокая).

1. Люди пытаются войти в систему под чужим именем путем перебора учетных данных.

Вероятность: высокая.

Степень опасности: высокая.

Меры по снижению серьезности риска: блокировка доступа после 10 попыток входа.

2. Атаки с подстановкой учетных данных (украденных или взломанных).

Вероятность: высокая.

Степень опасности: критическая.

Смягчение последствий: активация MFA для пользователей. Подписка на сервис, который будет предупреждать компанию о взломе учетных данных пользователя на других сайтах, чтобы можно было произвести принудительный сброс его имени и пароля. Разрешение использования менеджеров паролей и функции копирования в полях для пароля. Отказ от ротации паролей и контрольных вопросов.

3. Пользователи пытаются создать состояние гонки в целях своего денежного обогащения.

Вероятность: низкая.

Степень опасности: средняя.

Смягчение последствий: тщательное тестирование на предмет наличия состояния гонки и реализация блокировки остатка на счете при переводе денежных средств.

6. Назовите три возможные угрозы для «умного» автомобиля. Оцените вероятность их возникновения (низкая, средняя или высокая) и уровень потенциального ущерба.

1. Сбой операционной системы автомобиля, в результате чего владелец не может им воспользоваться. Угроза доступности.

Вероятность: низкая.

Потенциальный ущерб: рассерженные клиенты, которые не могут воспользоваться своим автомобилем по назначению. В результате наносится ущерб репутации компании.

2. Операционная система автомобиля заражена вирусом или вредоносным ПО.

Вероятность: низкая.

Потенциальный ущерб: рассерженные клиенты, которые не могут воспользоваться своим автомобилем по назначению. Вероятна попытка требования выкупа взамен на возвращение возможности управления. В результате наносится ущерб репутации компании.

3. Кража GPS и других данных автомобиля. Угроза конфиденциальности.

Вероятность: низкая.

Потенциальный ущерб: нанесение урона репутации компании. Возможно нанесение огромного ущерба пользователю. Если владелец автомобиля посещает секретные военные базы или за ним ведется слежка, его местонахождение является очень секретной информацией.

7. Назовите пять различных типов функциональности для обеспечения безопасности, которые может предоставить современная платформа.

1. Авторизация.
2. Аутентификация.
3. Контроль доступа.
4. Передача анти-CSRF-токенов.
5. Функции валидации ввода.

## Глава 4. Безопасность кода ПО

1. Когда следует использовать собственную идентификацию в сети (пользовательскую учетную запись), а когда – служебную учетную запись? Приведите два примера для каждого из вариантов и объясните их.

Пользовательская учетная запись предназначена для:

- 1) чтения личной электронной почты;
- 2) доступа к файлам компании и входа в системы в качестве сотрудника.

Служебная учетная запись предназначена для:

- 1) доступа к базе данных через веб-приложение;
- 2) доступа к онлайн-хранилищу через API.

2. Объясните возможные причины, по которым языки С и С++ все еще широко используются в индустрии разработки, когда существует Rust (язык, безопасный для памяти). Постарайтесь дать два или более примера, опишите соответствующие ситуации.

1. Компании не могут себе позволить перевести все свои активы на совершенно новый язык только из-за желания улучшить их безопасность. Бюджеты существуют не просто так.

2. Найти программистов, пишущих на Rust, может быть сложнее, чем тех, кто пишет на С/С++.

3. Не все слышали о Rust и его преимуществах.

4. «Потому что мы всегда так делали» – в любом случае неправильный ответ, независимо от вопроса.

3. Какой ваш любимый язык программирования или платформа? Почему?

Я предпочитаю Net, так как у меня больше всего опыта работы с ним, он имеет хорошее сопровождение, по нему есть масса документации и это, в общем, очень безопасная платформа.

4. Какой язык программирования или платформа, по вашему мнению, являются наиболее безопасными? Почему?

По моему мнению, это Net, потому что 1) я работала в Microsoft и знаю из первых рук, насколько серьезно эта компания относится к обеспечению безопасности; 2) его сопровождением занимается целая компания (а не отдельные люди на добровольных началах), следовательно, ему уделяется больше времени и внимания и он никогда не будет «заброшен», а также 3) он проприетарный, то есть никто, кроме сотрудников Microsoft, не может работать над ним, в отличие от платформ с открытым исходным кодом. К тому же каждый сотрудник проверяется на благонадежность, а также регулярно проходит тренинги по этике и безопасности, что гарантирует наибольшую надежность. Люди, работающие над проектами с открытым исходным кодом, редко подвергаются такой тщательной проверке, прежде чем получают доступ к коду.

Тем не менее существует множество хороших продуктов с открытым исходным кодом, которые можно использовать.

5. Почему необходимо обеспечивать защиту сессии пользователей?

Незащищенная пользовательская сессия может быть «украдена», то есть злоумышленник может завладеть ею и использовать систему от

имени взломанного пользователя. Ему не составит труда опустошить банковский счет жертвы, заказать ненужные уродливые туфли или через ее Twitter-аккаунт попросить людей отправить биткоины (и, конечно, указать при этом свой крипто кошелек). В такой ситуации злоумышленник имеет безграничные возможности.

6. Что может сделать злоумышленник, если ему удастся завладеть чужой пользовательской сессией во время входа в систему интернет-банкинга?

Злоумышленник может отправить деньги взломанного пользователя куда угодно и кому угодно: например, он может проспонсировать какую-нибудь террористическую группировку от имени своей жертвы и обвинить ее в этом преступлении. С помощью чужой пользовательской сессии изобретательный преступник может нанести достаточно большой ущерб за короткий промежуток времени.

7. Как бы вы объяснили коллеге, не имеющему технических знаний, разницу между аутентификацией и авторизацией?

Аутентификация – это проверка компьютером подлинности личности человека (то есть он именно тот, кого за себя выдает). Авторизация – это решение компьютера о том, что этому человеку разрешено и запрещено делать в системе.

8. Должны ли руководители высшего звена иметь особые привилегии в сети и других компьютерных системах? Если да, то почему? Если нет, то почему? Какие права вы бы им предоставили?

Руководители высшего звена часто утверждают, что они «заслуживают» особого доступа к системам и сети, однако часто для этого нет никаких оснований с технической точки зрения. Поскольку они часто становятся объектом хакерских атак, при предоставлении им какого-либо доступа крайне важным будет следование принципу наименьших привилегий. Тем не менее у руководителей есть право указать вам на дверь, поэтому сделайте все возможное, чтобы и защитить свою организацию, и избежать увольнения. В такой ситуации лучшим решением будет объяснение высшему руководству рисков, связанных с предоставлением необязательного доступа кому бы то ни было.

9. Должны ли администраторы сетевой системы иметь особые привилегии в сети и других компьютерных системах? Если да, то почему? Если нет, то почему? Какие права вы бы им предоставили?

Да, для выполнения своих должностных обязанностей сетевые администраторы должны иметь особые привилегии. Лучше всего, если они будут входить в свою электронную почту и пользоваться интернетом через обычный набор учетных данных (без полномочий администратора), а для выполнения работы станут использовать отдельную учетную запись (или учетные записи) со специальными правами: либо путем повторного входа в систему, либо с помощью функции «запуск от имени администратора». Сетевым администраторам как минимум нужен доступ ко всем сетевым настройкам и системам.

10. Должны ли сотрудники техподдержки иметь особые привилегии в сети и других компьютерных системах? Если да, то почему? Если нет, то почему? Какие права вы бы им предоставили?

Да, сотрудникам техподдержки нужны особые привилегии. Лучше всего, если они будут входить в свою электронную почту и пользоваться интернетом через обычный набор учетных данных (без полномочий администратора), а для выполнения работы станут использовать отдельную учетную запись (или учетные записи) со специальными правами: либо путем повторного входа в систему, либо с помощью функции «запуск от имени администратора». Им понадобится доступ кбросу паролей пользователей, управлению доступом и многому другому, чтобы выполнять свои рабочие функции.

11. Ваш начальник говорит, что регистрация и мониторинг будут стоить слишком дорого. Как вы объясните их ценность и важность с точки зрения обеспечения безопасности? Изложите свои доводы в одном абзаце. Не забудьте убедиться в том, что описываете возможные риски для бизнеса понятным для начальника языком (это умный, но не слишком технически подкованный человек). Если вы сформулируете свои мысли чересчур сложно, то не убедите начальника и провалите задание.

Уважаемый \_\_\_\_\_ (начальник)!

Когда в прошлом месяце у нас произошел инцидент и вы попросили меня провести расследование, я не смог этого сделать. Не было журналов, которые можно было бы просмотреть и на их основе выяснить, как наши данные попали в даркнет. Мы знали только сам факт того, что они там были. Мне пришло предупреждение от

приложения, что что-то пошло не так, но без журналов я не смог понять, что именно. Мне было очень неприятно, что я не сумел объяснить, что произошло. Я хочу защитить нашу организацию как можно лучше, и для этого мне нужно знать, что происходит во время инцидента. Так можно будет предотвратить его повторение. Поэтому прошу вас посодействовать и выделить бюджет для ведения журналов.

С уважением,  
специалист по безопасности.

## **Глава 5. Часто встречающиеся подводные камни**

1. Кто-то из вашей проектной команды хочет принимать сериализованные объекты из ненадежного источника. Вы знаете, что это плохая идея. Как можно эффективно объяснить риск коллеге? Запишите ответ. Объяснение должно быть убедительным и понятным.

Уважаемый \_\_\_\_\_ (сотрудник)!

Я просматривал ваш проектный документ и заметил, что вы собираетесь принимать сериализованные объекты из публичного источника. Это вызвало у меня беспокойство с точки зрения безопасности. Обычно, принимая данные или что-либо еще из публичного источника, мы проверяем и сканируем их, то есть относимся к ним как к особо опасным, пока не доказано обратное. Перед проверкой сериализованных объектов нужно провести их десериализацию, и есть большая вероятность, что какой-нибудь из них содержит в себе атаку, которая может попасть прямо в нашу сеть. Небезопасная десериализация настолько страшна, что включена в список OWASP Топ-10, поэтому относиться к ней нужно крайне серьезно. Мы можем встретиться, чтобы вместе подобрать менее рискованный способ получить нужные вам данные? Я уверен, что мы сумеем найти компромисс, который устроит нас обоих.

С уважением,  
специалист по безопасности.

2. OWASP Топ-10 является стандартом обеспечения безопасности: да или нет?

Нет. Он является отличным информационным документом, с которого можно начать знакомство с проблемами, связанными с безопасностью веб-приложений, и прекрасной основой для преподавания, но не стандартом.

3. Назовите три пункта из OWASP Топ-10, которые были упомянуты в книге до главы 5.

Межсайтовый скрипting, инъекционные атаки и регистрация. (Их больше трех.)

4. Применима ли уязвимость XXE к JSON? Применима ли она к YAML? Если да, почему? Если нет, то почему?

Нет, она применима только к XML. JSON, YAML и даже мой любимый. Net подвержены такой уязвимости, как десериализация.

5. Назовите пример ситуации (не обязательно связанный с компьютером), когда возникает состояние гонки.

В компании Starbucks в рамках программы Bug Bounty выявили классический случай «состояния гонки». Исследователь безопасности Егор Хомаков мог получить неограниченное количество кофе высшего сорта, практически одновременно выполнив несколько переводов с одной подарочной карты на другую. Остаток на счете не «заморозился», поэтому при параллельной проверке баланса оказалось, что деньги с первой карты еще не были сняты. Этот дефект позволил осуществить множество переводов между подарочными картами, несмотря на нулевой баланс на первой карте.

6. Почему мы откатываем незавершенные транзакции? Почему это важно? Приведите пример негативных последствий отказа от отката.

Представим, что Алиса пытается купить кофе в мобильном приложении, но в середине этого процесса у нее пропадает мобильная связь. Покупка происходит следующим образом: клиент нажимает кнопку «купить», приложение снимает деньги с его счета, отправляет заказ в местную кофейню, затем отправляет клиенту подтверждение. Если Алиса потеряла сигнал после снятия денег, но до момента отправления заказа, получается, что она заплатила за кофе, но 1) не получила заказ и 2) не знает о снятии денег со счета. Она знает только то, что осталась без кофе. Откат всей операции подразумевает, что

приложение может попросить Алису повторить заказ и деньги с ее счета снимут только один раз.

## Глава 6. Тестирование и развертывание

1. Если бы вы могли выбрать только один вид тестирования для своего приложения, какой бы вы выбрали и почему?

Пользовательское приемочное тестирование. Хотя темой данной книги является обеспечение безопасности ПО, невозможно игнорировать тот факт, что приложение должно *работать*. Оценка безопасности или тест на проникновение идут уже после приемочных тестов. Тем не менее лучше всего провести все виды тестирования и подтвердить, что приложение находится в хорошем состоянии.

2. Как вы думаете, какой вид тестирования будет самым быстрым? Почему?

Модульное тестирование, так как обычно оно автоматизировано и выполняется быстро.

3. Как вы думаете, какой вид тестирования будет самым медленным? Почему?

Тестирование на проникновение, потому что оно подразумевает тщательность проверки.

4. Какие типы уязвимостей вы бы искали при регрессионном тестировании? Назовите как минимум два типа и объясните свой выбор.

Инъекционные атаки, потому что они наиболее опасны, и XSS-атака – из-за ее распространенности.

5. Сеть вашей компании спроектирована с учетом принципа нулевого уровня доверия? Если вы не знаете ответа, ваше домашнее задание – выяснить это.

Только вы можете ответить на этот вопрос.

6. Поддерживает ли ваша компания использование конвейера CI/CD? Если вы не знаете ответа, ваше домашнее задание – выяснить это.

Только вы можете ответить на этот вопрос.

7. Следует ли в среде CI/CD применять инструмент статического тестирования безопасности приложений (SAST) и проводить полное

сканирование всего кода при каждой новой сборке? Почему?

Нет! SAST работает очень медленно и дает много ложноположительных результатов. Его можно использовать одним из следующих способов: 1) запустить на совершенно новом коде или 2) запустить вне конвейера, самолично проанализировать результаты его работы, а затем поместить их в баг-трекер. Вместо SAST можно попробовать включить в свой конвейер сканирование на наличие секретов, SCA или пассивную проверку инструментом DAST.

8. Почему очень важно помещать все новые изменения в репозиторий кода?

Если вы не проверяете изменения своего кода в системе контроля версий, то в конечном итоге кто-то другой загрузит свои изменения, выложит их в prod-среду и тем самым перезапишет сделанные вами изменения. Вся ваша тяжелая работа исчезнет в одно мгновение – этот день станет одним из худших в вашей жизни.

9. Зачем мы тестируем точки интеграции между различными системами? Это лучше, чем полное тестирование каждой системы?

Я считаю, что интеграционное тестирование не менее важно, чем тестирование каждой отдельной системы. Если системы не работают вместе, то конечному пользователю будет казаться, что они неисправны. Как правило, пользователя не волнует причина, по которой он не может использовать систему, а только сам факт ее неисправности.

10. Почему мы тестируем базы данных, даже если они находятся в закрытом доступе?

Данные представляют большую ценность, поэтому им, конечно же, необходимо обеспечить надежную защиту. Кроме того, приложение в процессе работы затрагивает базу данных, и, даже если оно окажется уязвимым, безопасная БД сможет сама отразить атаку.

11. Почему мы тестируем API, даже если они находятся в закрытом доступе? (Возможно, вопрос с подвохом.)

Необходимо обеспечить безопасность всех технологий, даже если они находятся в закрытом доступе, так как существуют еще инсайдерские угрозы, возможность проникновения в сеть кого-либо извне или внедрения потенциально опасной системы внутрь сети с последующей установкой контакта между ней и основной системой приложения.

12. Когда имеет смысл проводить тестирование на проникновение, а не оценку безопасности системы? Поясните свой ответ.

При ограничении времени или хрупкости системы имеет смысл провести оценку безопасности. Если нужно получить полное подтверждение результатов или привлечь внимание руководства, то лучше выполнить тест на проникновение.

## Глава 7. Программа безопасности

1. Назовите одну причину, по которой поддержание реестра приложений в актуальном состоянии важно для любой организации.

Если подвергшееся атаке приложение зафиксировано в реестре (в базе данных управления конфигурацией), вам доступна вся информация, с помощью которой можно своевременно провести расследование инцидента.

2. Приведите один пример предупреждения, которое может выдать ваше приложение. Какой тип поведения может вызвать такое предупреждение и почему его нужно расценивать как проблему?

Предупреждение о вызове API, содержащем тег `<script>`, должно расцениваться как проблема, связанная с обеспечением безопасности, поскольку оно говорит о попытках внедрения постороннего кода в систему.

3. Является ли тестирование самой важной частью программы безопасности? Если да, то почему? Если нет, то почему?

Тестирование – это проверка (не)эффективности приложенных усилий по обеспечению безопасности приложения. По моему мнению, оно не является самой важной частью в обеспечении безопасности ПО, но *подтверждает* эффективность затраченных усилий. Тестирование также позволяет выявить все моменты, упущенные из виду на предыдущих этапах разработки.

4. Что выгоднее: купить инструмент RASP либо WAF (щит для приложения) или потратить эти деньги на обеспечение безопасности кода? Объясните свой выбор.

Обеспечение безопасности кода – это без исключений задача номер один, а WAF или RASP – только *дополнительный* уровень защиты.

5. Опишите тип инцидента безопасности, при котором понадобится помочь разработчику. Что требуется от разработчика в описанной ситуации?

Разработчик нужен в том случае, если приложение подверглось атаке SQL-инъекции. Он поможет исправить дефект, получить доступ к необходимым журналам и понять, что произошло.

6. Какой один инструмент вы дали бы в помощь своим разработчикам? Объясните свой выбор.

DAST, потому что он прост в использовании и находит все легкие для злоумышленника мишени. В частности, я бы выбрала OWASP Zap, потому что он бесплатный и является отличным инструментом.

7. Какой учебный ресурс вы предоставили бы в помощь своим разработчикам? Объясните свой выбор.

Я бы подарила им эту книгу!;-D

8. В чем разница между SAST и SCA?

SAST анализирует код, написанный командой компании, на предмет проблем безопасности. SCA рассматривает только сторонние компоненты (код, который был написан другими людьми).

9. В чем разница между SAST и DAST?

SAST – это статический анализ кода, а DAST – динамический анализ функционирующего приложения.

10. Определите цель для возможной программы безопасности приложений в вашем офисе, учебном проекте или в выдуманной компании, где хотели бы работать. Какую цель вы поставили? Почему вы выбрали именно ее? Как вы будете измерять прогресс ее достижения?

Только вы можете ответить на этот вопрос.

11. Есть ли препятствия, которые мешают начать первую программу безопасности на вашем текущем месте работы? Если да, то какие? А еще лучше попробуйте придумать, как их можно преодолеть.

Только вы можете ответить на этот вопрос.

## Глава 8. Обеспечение безопасности современных систем и приложений

1. Что означает термин «разделяемая ответственность»?

Он означает, что часть ответственности за обеспечение работы облака лежит на клиенте, а часть – на облачном провайдере. Другими словами, речь о том, кто и что должен делать, чтобы предотвратить возникновение каких бы то ни было проблем.

2. В чем разница между инфраструктурой как услугой (IaaS) и платформой как услугой (PaaS)? Какую из них вам придется самостоятельно обновлять и поддерживать?

IaaS – это предоставляемая облачным провайдером виртуальная машина, поддержку работы и исправление ошибок которой осуществляет ее пользователь. PaaS – это предоставляемая облачным провайдером платформа, на которой размещается приложение пользователя и исправлением ошибок которой занимается сам провайдер.

3. Почему онлайн-хранилища подвергаются (или не подвергаются) большему риску? Что из триады CIA может быть применимо к онлайн-хранилищу?

Чаще всего онлайн-хранилище всегда «доступно» (CIA), так как размещено в интернете. Однако в случае неправильной настройки онлайн-хранилище проще обнаружить с целью взлома.

4. Назовите один новый риск, которому подвержено облако и которому не подвержены традиционный или локальный центры обработки данных.

Риск состоит в том, что внешние ресурсы (например, сотрудники облачного провайдера) будут иметь доступ к находящимся в облаке системам и контроль над ними.

5. В чем разница между контейнером, виртуальной машиной и физическим сервером?

Контейнер – это виртуализированная операционная система с минимальным набором функций, предназначенная для запуска одного приложения. Виртуальная машина – это полная виртуализированная операционная система для запуска одного или нескольких приложений.

Физический сервер – это машина, которая может содержать одну операционную систему, множество операционных систем (посредством виртуализации) или множество контейнеров.

6. Назовите одно преимущество инфраструктуры как кода.

При механической поломке сервера возвращение серверной ОС в эксплуатацию занимает считанные секунды. Кроме того, можно использовать контроль версий для управления изменениями. Преимуществ очень много, попробуйте сами придумать еще несколько.

7. Назовите одно преимущество DevOps перед каскадной моделью жизненного цикла разработки системы.

В большинстве случаев выпуск небольших изменений позволяет очень быстро исправить дефект безопасности. Это победа!

8. Какой из современных вариантов инструментария показался вам наиболее интересным? Почему?

Только вы можете ответить на этот вопрос.

9. Какая из современных тактических мер по обеспечению безопасности показалась вам наиболее интересной? Почему?

Только вы можете ответить на этот вопрос.

10. Прочитав эту главу, скажите, чего, на ваш взгляд, не хватает вашей организации. Что и как можно улучшить?

Только вы можете ответить на этот вопрос.

## Глава 9. Полезные привычки

1. Какие риски влечет за собой технический долг?

1. Приложение может настолько отстать от актуальной версии используемой платформы, что для обновления придется переписывать его целиком.

2. Все системы могут настолько отстать от своих актуальных версий, что выпустить исправления ошибок для них будет практически невозможно.

3. Если для используемого языка программирования выпущено 11 различных версий одной и той же платформы, появляется «нулевой день», и очень сложно понять, какая из этих версий имеет уязвимость.

2. Стоит ли публиковать частную информацию в социальных сетях, если она доступна только друзьям?

Не следует размещать в интернете ничего из того, что может вас опозорить, навредить вам или вашим близким, привести к увольнению или как-нибудь еще быть использовано против вас. Данные могут быть

украдены или взломаны, могут произойти ошибки, а друзья – оказаться ненадежными. *Кроме того, интернет помнит все.*

3. Почему пользователи редко включают многофакторную аутентификацию? С помощью каких трех способов можно повысить частоту ее применения?

Низкие показатели внедрения MFA объясняются тем, что она требует дополнительных действий, и человеческой ленью. Большинство людей не понимают масштабы защиты, предоставляемой MFA, и на какой риск они идут, выкладывая что-либо в интернете.

Уровень внедрения MFA можно повысить:

- 1) упростив ее реализацию;
- 2) сделав ее обязательной для использования на рабочем месте;
- 3) повысив осведомленность людей о ценности MFA и о том, как ее использовать.

4. Каким менеджером паролей вы пользуетесь? (Ответ «никаким» не принимается.)

Я использую 1Password, но лучше использовать любой менеджер паролей, чем никакой вовсе.

5. Какая политика безопасности в вашей компании затрудняет выполнение рабочих обязанностей? Обсуждали ли вы со службой безопасности ее обновление? Можно ли найти компромисс по этому вопросу?

На этот вопрос вы должны ответить сами. Однако в качестве примера можно привести требование ротации паролей. Обычному пользователю очень трудно запомнить свой пароль, поэтому он записывает его или добавляет в его конец числа, что снижает уровень безопасности в целом.

6. Назовите вид деятельности, который, по вашему мнению, мог бы быть полезен в качестве «пожарных учений».

Откат базы данных или приложения, тестирование планов обеспечения непрерывности бизнеса (BCP) или аварийного восстановления (DR).

## Глава 10. Непрерывное обучение

На все вопросы из главы 10 необходимо ответить *самостоятельно*.

**Удачи!**

# ЛУЧШИЕ КНИГИ О БИЗНЕСЕ С ЛОГОТИПОМ ВАШЕЙ КОМПАНИИ? ЛЕГКО!

Удивить своих клиентов, бизнес-партнеров, сделать памятный подарок сотрудникам и рассказать о своей компании читателям бизнес-литературы? Приглашаем стать партнерами выпуска актуальных и популярных книг. О вашей компании узнает наиболее активная аудитория.

## ПАРТНЕРСКИЕ ОПЦИИ:

- Специальный тираж уже существующих книг с логотипом вашей компании.
  - Размещение логотипа на супер-обложке для малых тиражей (от 30 штук).
  - Поддержка выхода новинки, которая ранее не была доступна читателям (50 книг в подарок).

## ПАРТНЕРСКИЕ ВОЗМОЖНОСТИ:

- Рекламная полоса о вашей компании внутри книги.
  - Вступительное слово в книге от первых лиц компании-партнера.
  - Обращение первых лиц на суперобложке.
  - Отзыв на обороте обложки вложение информационных материалов о вашей компании (закладки, листовки, мини-буллеты).



У вас есть возможность обсудить свои пожелания с менеджерами корпоративных продаж. Как?

Звоните:  
+7 495 411 68 59, доб. 2261

Заходите на сайт:  
[eksmo.ru/b2b](http://eksmo.ru/b2b)





ТАНЯ ЯНКА

ИСЧЕРПЫВАЮЩИЙ ГИД  
ДЛЯ НАЧИНАЮЩИХ  
РАЗРАБОТЧИКОВ

# БЕЗОПАСНОСТЬ WEB-ПРИЛОЖЕНИЙ

ПРОЕКТИРОВАНИЕ  
БЕЗОПАСНОЙ  
АРХИТЕКТУРЫ

МЕХАНИЗМЫ  
ЗАЩИТЫ  
ДАННЫХ

ТЕСТИРОВАНИЕ НА  
ПРОНИКНОВЕНИЕ

БОМБОРА  
ИЗДАТЕЛЬСТВО



# Примечания

1

Отчет о нарушениях данных за 2016, 2017, 2018 годы.

[Вернуться](#)

2

[\*\*time.com/3404330/home-depot-hack.\*\*](http://time.com/3404330/home-depot-hack)

[Вернуться](#)

3

[\*\*www.infoworld.com/article/2626167/third-party-code-putting-companies-at-risk.html.\*\*](http://www.infoworld.com/article/2626167/third-party-code-putting-companies-at-risk.html)

[Вернуться](#)

4

Пример атаки на цепь поставок: [\*\*www.trendmicro.com/vinfo/hk-en/security/news/cybercrime-and-digital-threats/hacker-infects-node-js-package-to-steal-from-bitcoin-wallets.\*\*](http://www.trendmicro.com/vinfo/hk-en/security/news/cybercrime-and-digital-threats/hacker-infects-node-js-package-to-steal-from-bitcoin-wallets)

[Вернуться](#)

5

«Инъекционная» уязвимость признана многими специалистами по безопасности угрозой № 1 для безопасности программного обеспечения:

[\*\*www.owasp.org/index.php/Top\\_10-2017\\_A1-Injection;\*\*](http://www.owasp.org/index.php/Top_10-2017_A1-Injection;)  
[\*\*www.owasp.org/index.php/Top\\_10\\_2013-A1-Injection;\*\*](http://www.owasp.org/index.php/Top_10_2013-A1-Injection;)

[www.owasp.org/index.php/Top\\_10\\_2010-A1-Injection](http://www.owasp.org/index.php/Top_10_2010-A1-Injection).

[Вернуться](#)

**6**

Кодирование выходных данных и предотвращение XSS-атак:  
[portswigger.net/web-security/cross-site-scripting/preventing](http://portswigger.net/web-security/cross-site-scripting/preventing).

[Вернуться](#)

**7**

Проверка границ: [en.wikipedia.org/wiki/Bounds\\_checking](http://en.wikipedia.org/wiki/Bounds_checking).

[Вернуться](#)

**8**

Целочисленное переполнение:  
[en.wikipedia.org/wiki/Integer\\_overflow](http://en.wikipedia.org/wiki/Integer_overflow).

[Вернуться](#)

**9**

Переполнение буфера: [en.wikipedia.org/wiki/Buffer\\_overflow](http://en.wikipedia.org/wiki/Buffer_overflow).

[Вернуться](#)

**10**

Шпаргалка OWASP по профилактике XSS-уязвимости:  
[owasp.org/www-project-cheat-sheets/cheatsheets/Cross\\_Site\\_Scripting\\_Prevention\\_Cheat\\_Sheet.html](http://owasp.org/www-project-cheat-sheets/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html).

[Вернуться](#)

**11**

Rust – безопасный для памяти язык программирования:  
[acks.mozilla.org/2019/02/rewriting-a-browser-component-in-rust](https://acks.mozilla.org/2019/02/rewriting-a-browser-component-in-rust).

[Вернуться](#)

**12**

OWASP Топ-10 за 2017 год: 77 % приложений содержит XSS-уязвимости. [www.ptsecurity.com/ww-en/Analytics/Web-Application-Vulnerabilities-Statistics-2019](http://www.ptsecurity.com/ww-en/Analytics/Web-Application-Vulnerabilities-Statistics-2019).

[Вернуться](#)

**13**

База данных CVE Mitre: [cve.mitre.org](http://cve.mitre.org).

[Вернуться](#)

**14**

Несмотря на высокую сложность эксплойта, POST-запросы по-прежнему уязвимы для данного типа атак: [developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-XSS-Protection](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-XSS-Protection).

[Вернуться](#)

**15**

Кликджекинг: [portswigger.net/web-security/clickjacking](http://portswigger.net/web-security/clickjacking).

[Вернуться](#)

**16**

Предзагрузка HSTS-суффикса: [developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Strict-Transport-Security#Preloading\\_Strict\\_Transport\\_Security](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Strict-Transport-Security#Preloading_Strict_Transport_Security).

[Вернуться](#)

## 17

Предзагрузка HSTS-суффикса: [developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Strict-Transport-Security#Preloading\\_Strict\\_Transport\\_Security](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Strict-Transport-Security#Preloading_Strict_Transport_Security).

[Вернуться](#)

## 18

Список разрешенных Feature Policy источников: [developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Feature-Policy](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Feature-Policy).

[Вернуться](#)

## 19

Пакет [www.nuget.org/packages/NWebsec.AspNetCore.Middleware](https://www.nuget.org/packages/NWebsec.AspNetCore.Middleware).

Nuget:

[Вернуться](#)

## 20

Значение Same-Site ‘Lax’ по умолчанию:

- а) [web.dev/samesite-cookies-explained/#changes-to-the-default-behavior-without-samesite](https://web.dev/samesite-cookies-explained/#changes-to-the-default-behavior-without-samesite);
- б) [tools.ietf.org/html/draft-west-cookie-incrementalism-00](https://tools.ietf.org/html/draft-west-cookie-incrementalism-00).

[Вернуться](#)

## 21

Рабочий фактор и «засыпка перцем»: [cheatsheetseries.owasp.org/cheatsheets/Password\\_Storage\\_Cheat\\_Sheet.html#work-factors](https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html#work-factors).

[Вернуться](#)

## 22

Рабочий фактор и «засыпка перцем»:  
[cheatsheetseries.owasp.org/cheatsheets/Password\\_Storage\\_Cheat\\_Sheet.html#work-factors](https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html#work-factors).

[Вернуться](#)

## 23

Технология Deep Packet Inspection:  
[en.wikipedia.org/wiki/Deep\\_packet\\_inspection](https://en.wikipedia.org/wiki/Deep_packet_inspection).

[Вернуться](#)

## 24

Технический долг: [en.wikipedia.org/wiki/Technical\\_debt](https://en.wikipedia.org/wiki/Technical_debt).

[Вернуться](#)

## 25

«Безопасность через проектирование»: [Wikipedia.org](https://en.wikipedia.org).

[Вернуться](#)

## 26

Из статьи онлайн-журнала Slashdot: ошибка, обнаруженная на этапе требований, стоит \$1, на этапе проектирования – \$10, на этапе кодирования – \$100, на этапе тестирования – \$1000:  
[developers.slashdot.org/story/03/10/21/0141215/software-defects-delay-bugs-really-cost-more](https://developers.slashdot.org/story/03/10/21/0141215/software-defects-delay-bugs-really-cost-more).

[Вернуться](#)

## 27

Canadian Center for Cyber Security: **cyber.gc.ca**.  
[Вернуться](#)

**28**

Люксембург: **www.circl.lu**.  
[Вернуться](#)

**29**

Япония: **www.jpcert.or.jp/english/at/2020.html**.  
[Вернуться](#)

**30**

Великобритания (Отчет об угрозах): **www.ncsc.gov.uk**.  
[Вернуться](#)

**31**

Соединенные Штаты Америки: **www.us-cert.gov/ncas/alerts**.  
[Вернуться](#)

**32**

Новая Зеландия (Оповещения): **www.cert.govt.nz/it-specialists**.  
[Вернуться](#)

**33**

ITSG-33: **www.cse-cst.gc.ca/sites/default/files/itsg33-ann4a-eng.xls**.  
[Вернуться](#)

## 34

Специальная публикация NIST 800-53 S-3: [nvd.nist.gov/800-53/Rev4/control/SC-3](http://nvd.nist.gov/800-53/Rev4/control/SC-3).

[Вернуться](#)

## 35

ITSG-33: [www.cse-cst.gc.ca/sites/default/files/itsg33-ann4a-eng.xls](http://www.cse-cst.gc.ca/sites/default/files/itsg33-ann4a-eng.xls).

[Вернуться](#)

## 36

Специальная публикация NIST 800-53 S-2: [nvd.nist.gov/800-53/Rev4/control/SC-2](http://nvd.nist.gov/800-53/Rev4/control/SC-2).

[Вернуться](#)

## 37

Подделка межсайтовых запросов (CSRF): [www.owasp.org/index.php/Cross-Site\\_Request\\_Forgery\\_\(CSRF\)](http://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)).

[Вернуться](#)

## 38

Сравнение открытого исходного кода и закрытого исходного кода: [www.researchgate.net/publication/220891308\\_Security\\_of\\_Open\\_Source\\_and\\_Closed\\_Source\\_Software\\_An\\_Empirical\\_Comparison\\_of\\_Published\\_Vulnerabilities](http://www.researchgate.net/publication/220891308_Security_of_Open_Source_and_Closed_Source_Software_An_Empirical_Comparison_of_Published_Vulnerabilities).

[Вернуться](#)

## 39

Нехватка специалистов в области информационной безопасности:  
[www.infosecurity-magazine.com/news/cybersecurity-skills-shortage-tops](http://www.infosecurity-magazine.com/news/cybersecurity-skills-shortage-tops).

[Вернуться](#)

**40**

Высокая стоимость услуг специалистов информационной безопасности: [www.mondo.com/blog-highest-paid-cybersecurity-jobs](http://www.mondo.com/blog-highest-paid-cybersecurity-jobs).

[Вернуться](#)

**41**

Деревья атак: [en.wikipedia.org/wiki/Attack\\_tree](http://en.wikipedia.org/wiki/Attack_tree).

[Вернуться](#)

**42**

STRIDE: [en.wikipedia.org/wiki/STRIDE\\_%28security%29](http://en.wikipedia.org/wiki/STRIDE_%28security%29).

[Вернуться](#)

**43**

PASTA: [www.owasp.org/images/a/aa/AppSecEU2012\\_PASTA.pdf](http://www.owasp.org/images/a/aa/AppSecEU2012_PASTA.pdf).

[Вернуться](#)

**44**

Обнаружение уязвимостей в Struts в 2016 году:  
[www.securitymetrics.com/blog/apache-struts-vulnerability-what-you-should-do](http://www.securitymetrics.com/blog/apache-struts-vulnerability-what-you-should-do).

[Вернуться](#)

## 45

Шпаргалка OWASP по управлению сессиями: [owasp.org/www-project-cheat-sheets/cheatsheets/Session\\_Management\\_Cheat\\_Sheet](http://owasp.org/www-project-cheat-sheets/cheatsheets/Session_Management_Cheat_Sheet).

[Вернуться](#)

## 46

«Разбиение стека ради удовольствия и прибыли»: [www-inst.eecs.berkeley.edu/~cs161/fa08/papers/stack\\_smashing.pdf](http://www-inst.eecs.berkeley.edu/~cs161/fa08/papers/stack_smashing.pdf).

[Вернуться](#)

## 47

Деятельность социальной сети Facebook запрещена на территории РФ по основаниям осуществления экстремистской деятельности (согласно ст. 4 закона РФ «О средствах массовой информации»).

[Вернуться](#)

## 48

Модели управления доступом (из памятки OWASP): [cheatsheetseries.owasp.org/cheatsheets/Access\\_Control\\_Cheat\\_Sheet.html](http://cheatsheetseries.owasp.org/cheatsheets/Access_Control_Cheat_Sheet.html).

[Вернуться](#)

## 49

Атаки на журналы (три источника):

- а) [owasp.org/www-community/attacks/Log\\_Injection](http://owasp.org/www-community/attacks/Log_Injection);
- б) [www.sans.org/blog/what-works-in-appsec-log-forging](http://www.sans.org/blog/what-works-in-appsec-log-forging);
- в) [github.com/votd/vulnerability-of-the-day/wiki/Log-Overflow](http://github.com/votd/vulnerability-of-the-day/wiki/Log-Overflow).

[Вернуться](#)

# 50

Шпаргалка OWASP по обработке ошибок:  
[cheatsheetseries.owasp.org/cheatsheets/Error\\_Handling\\_Cheat\\_Sheet.html](https://cheatsheetseries.owasp.org/cheatsheets/Error_Handling_Cheat_Sheet.html).

[Вернуться](#)

# 51

Шпаргалка OWASP по регистрации:  
[cheatsheetseries.owasp.org/cheatsheets/Logging\\_Cheat\\_Sheet.html](https://cheatsheetseries.owasp.org/cheatsheets/Logging_Cheat_Sheet.html).

[Вернуться](#)

# 52

Заголовок X-Forwarded-For: Доминик Ригетто предоставил доказательства.

[Вернуться](#)

# 53

PII: [en.wikipedia.org/wiki/Personal\\_data](https://en.wikipedia.org/wiki/Personal_data).

[Вернуться](#)

# 54

Шпаргалка OWASP по CSRF:  
[cheatsheetseries.owasp.org/cheatsheets/Cross-Site\\_Request\\_Forgery\\_Prevention\\_Cheat\\_Sheet.html](https://cheatsheetseries.owasp.org/cheatsheets/Cross-Site_Request_Forgery_Prevention_Cheat_Sheet.html).

[Вернуться](#)

# 55

SSRF:

a)

[owasp.org/www-](http://owasp.org/www-community/attacks/Server_Side_Request_Forgery)

[community/attacks/Server\\_Side\\_Request\\_Forgery](http://owasp.org/www-community/attacks/Server_Side_Request_Forgery);

б) [portswigger.net/web-security/ssrf](http://portswigger.net/web-security/ssrf).

[Вернуться](#)

**56**

SSRF: [www.darkreading.com/edge/theedge/ssrf-101-how-server-side-request-forgery-sneaks-past-your-web-apps-/b/d-id/1337121](http://www.darkreading.com/edge/theedge/ssrf-101-how-server-side-request-forgery-sneaks-past-your-web-apps-/b/d-id/1337121).

[Вернуться](#)

**57**

SSRF: [www.acunetix.com/blog/articles/server-side-request-forgery-vulnerability](http://www.acunetix.com/blog/articles/server-side-request-forgery-vulnerability).

[Вернуться](#)

**58**

Десериализация:  
[cheatsheetseries.owasp.org/cheatsheets/Deserialization\\_Cheat\\_Sheet.html](http://cheatsheetseries.owasp.org/cheatsheets/Deserialization_Cheat_Sheet.html).

[Вернуться](#)

**59**

HMAC SHA256: [cryptobook.nakov.com/mac-and-key-derivation](http://cryptobook.nakov.com/mac-and-key-derivation).

[Вернуться](#)

**60**

Символическое  
[en.wikipedia.org/wiki/Symbolic\\_execution](http://en.wikipedia.org/wiki/Symbolic_execution).

исполнение:

[Вернуться](#)

**61**

Определение SCA: [www.flexerasoftware.com/blog/what-is-software-composition-analysis](http://www.flexerasoftware.com/blog/what-is-software-composition-analysis).

[Вернуться](#)

**62**

Модульное тестирование: [softwaretestingfundamentals.com/unit-testing](http://softwaretestingfundamentals.com/unit-testing).

[Вернуться](#)

**63**

Шпаргалка по уклонению от XSS-фильтра: [owasp.org/www-community/xss-filter-evasion-cheatsheet](http://owasp.org/www-community/xss-filter-evasion-cheatsheet).

[Вернуться](#)

**64**

Регрессионное  
[en.wikipedia.org/wiki/Regression\\_testing](http://en.wikipedia.org/wiki/Regression_testing).

тестирование:

[Вернуться](#)

**65**

IaC: [en.wikipedia.org/wiki/Infrastructure\\_as\\_code](http://en.wikipedia.org/wiki/Infrastructure_as_code).

[Вернуться](#)

**66**

SaC:

[www.oreilly.com/library/view/devopssec/9781491971413/ch04.html](http://www.oreilly.com/library/view/devopssec/9781491971413/ch04.html).

[Вернуться](#)

**67**

Фаззинг: [en.wikipedia.org/wiki/Fuzzing](http://en.wikipedia.org/wiki/Fuzzing).

[Вернуться](#)

**68**

IAST:

- a) [www.veracode.com/security/interactive-application-security-testing-iast](http://www.veracode.com/security/interactive-application-security-testing-iast);
- б) [www.contrastsecurity.com/knowledge-hub/glossary/interactive-application-security-testing](http://www.contrastsecurity.com/knowledge-hub/glossary/interactive-application-security-testing).

[Вернуться](#)

**69**

Теневые IT: [en.wikipedia.org/wiki/Shadow\\_IT](http://en.wikipedia.org/wiki/Shadow_IT).

[Вернуться](#)

**70**

Время обнаружения утечки данных: [www.varonis.com/blog/data-breach-response-times](http://www.varonis.com/blog/data-breach-response-times).

[Вернуться](#)

**71**

Определение метрик: [www.lexico.com/en/definition/metrics](http://www.lexico.com/en/definition/metrics).

[Вернуться](#)

## 72

Приобрести «Руководство по DevOps: как создать гибкость, надежность и безопасность мирового уровня в научно-технических организациях» напрямую у авторов: [itrevolution.com/book/the-devops-handbook](http://itrevolution.com/book/the-devops-handbook).

[Вернуться](#)

## 73

«Бережливый стартап...»: [theleanstartup.com/book](http://theleanstartup.com/book).

[Вернуться](#)

## 74

Инструменты категоризации трафика:  
[www.esecurityplanet.com/cloud/data-storage-security.html](http://www.esecurityplanet.com/cloud/data-storage-security.html).

[Вернуться](#)

## 75

Ограничение ресурсов контейнеров и настройка оповещения о достижении лимита: [kubernetes.io/docs/concepts/policy/resource-quotas](http://kubernetes.io/docs/concepts/policy/resource-quotas).

[Вернуться](#)

## 76

Не запускать контейнеры от имени root:  
[resources.whitesourcesoftware.com/blog-whitesource/docker-container-security-challenges-and-best-practices](http://resources.whitesourcesoftware.com/blog-whitesource/docker-container-security-challenges-and-best-practices).

[Вернуться](#)

**77**

Изоляция функциональных объектов: ссылка на Snyk:  
[snyk.io/blog/10-serverless-security-best-practices](https://snyk.io/blog/10-serverless-security-best-practices).

[Вернуться](#)

**78**

Применение принципа наименьших привилегий при настройке бессерверных приложений: [www.serverless.com/blog/serverless-deployment-best-practices](https://www.serverless.com/blog/serverless-deployment-best-practices).

[Вернуться](#)

**79**

Сканеры для CI/CD: [blog.checkpoint.com/2020/04/29/9-serverless-security-best-practices-you-must-read](https://blog.checkpoint.com/2020/04/29/9-serverless-security-best-practices-you-must-read).

[Вернуться](#)

**80**

Изоляция функциональных объектов: ссылка на Snyk:  
[snyk.io/blog/10-serverless-security-best-practices](https://snyk.io/blog/10-serverless-security-best-practices).

[Вернуться](#)

**81**

Определение труду в своей речи «Инфраструктура как код» дал Стивен Муравски на технической конференции Microsoft Ignite the Tour 2019.

[Вернуться](#)

**82**

Код – единственный источник истины: [stackify.com/what-is-infrastructure-as-code-how-it-works-best-practices-tutorials](https://stackify.com/what-is-infrastructure-as-code-how-it-works-best-practices-tutorials).

[Вернуться](#)

**83**

Будьте осторожны с предоставлением права на разработку IaC: [techbeacon.com/enterprise-it/infrastructure-code-engine-heart-devops](https://techbeacon.com/enterprise-it/infrastructure-code-engine-heart-devops).

[Вернуться](#)

**84**

Неизменяемость: [www.thorntech.com/2018/02/infrastructure-as-code-best-practices](https://www.thorntech.com/2018/02/infrastructure-as-code-best-practices).

[Вернуться](#)

**85**

Безопасность как код: [www.bmc.com/blogs/security-as-code](https://www.bmc.com/blogs/security-as-code).

[Вернуться](#)

**86**

Определение CI/CD дал Стивен Муравски на технической конференции Microsoft Ignite the Tour 2019.

[Вернуться](#)

**87**

Определение CI/CD дал Стивен Муравски на технической конференции Microsoft Ignite the Tour 2019.

[Вернуться](#)

## 88

Определение CI/CD дал Стивен Муравски на технической конференции Microsoft Ignite the Tour 2019.

[Вернуться](#)

## 89

Три пути внедрения DevOps: [itrevolution.com/wp-content/uploads/files/PhoenixProjectExcerpt.pdf](http://itrevolution.com/wp-content/uploads/files/PhoenixProjectExcerpt.pdf).

[Вернуться](#)

## 90

Три пути внедрения DevOps: [itrevolution.com/wp-content/uploads/files/PhoenixProjectExcerpt.pdf](http://itrevolution.com/wp-content/uploads/files/PhoenixProjectExcerpt.pdf).

[Вернуться](#)

## 91

Оболочки для библиотек: Джастин Осборн, Зейн Лэки, Клинт Гиблер и Даг Де Перри на групповом обсуждении на конференции по вопросам информационной безопасности 2020 года. «Вотпочемуунасвсеплохо» – шутка Джастина.

[Вернуться](#)

## 92

Определение DevSecOps: Имран А. Мохаммед – основатель и глава Practical DevSecOps.

[Вернуться](#)

## 93

Облачные вычисления: [en.wikipedia.org/wiki/Cloud\\_computing](https://en.wikipedia.org/wiki/Cloud_computing).  
[Вернуться](#)

**94**

Часть определения ориентированности на облако:  
[www.infoworld.com/article/3281046/what-is-cloud-native-the-modern-way-to-develop-software.html](http://www.infoworld.com/article/3281046/what-is-cloud-native-the-modern-way-to-develop-software.html).

[Вернуться](#)

**95**

Часть определения ориентированности на облако: [thenewstack.io/10-key-attributes-of-cloud-native-applications](https://thenewstack.io/10-key-attributes-of-cloud-native-applications).

[Вернуться](#)

**96**

Часть определения ориентированности на облако: Дэвид Блэнк-Эдельман и Дженифер Дэвис на Microsoft Ignite the Tour 2019.

[Вернуться](#)

**97**

Самые популярные решения являются самыми небезопасными – пример: [stackoverflow.com/a/14968272/451455](https://stackoverflow.com/a/14968272/451455).

[Вернуться](#)

**98**

Длина и энтропия пароля:  
[en.wikibooks.org/wiki/Cryptography/Key\\_Lengths](https://en.wikibooks.org/wiki/Cryptography/Key_Lengths).  
[Вернуться](#)

## 99

Происхождение паролей: [theconversation.com/the-long-history-and-short-future-of-the-password-76690](https://theconversation.com/the-long-history-and-short-future-of-the-password-76690).

[Вернуться](#)

## 100

Определение непрерывного обучения 1:  
[www.ispringsolutions.com/blog/continuous-learning](http://www.ispringsolutions.com/blog/continuous-learning).

[Вернуться](#)

## 101

Определение непрерывного обучения 1:  
[www.ispringsolutions.com/blog/continuous-learning](http://www.ispringsolutions.com/blog/continuous-learning).

[Вернуться](#)

## 102

Определение непрерывного обучения 2:  
[www.learnupon.com/blog/continuous-learning](http://www.learnupon.com/blog/continuous-learning).

[Вернуться](#)

## 103

Список ресурсов для обучения частично представлен в твиттере @shehackspurple.

[Вернуться](#)