
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. Исчерпывающий гид для начинающих разработчиков

Серия «КиберБез. Лучшие книги о безопасности в сети»

indd предоставлен правообладателем
http://www.litres.ru/pages/biblio_book/?art=69351946
ISBN 978-5-04-188679-0

Аннотация

У вас в руках идеальное руководство для тех, кто только начинает свой путь в веб-разработке и хочет научиться создавать безопасные веб-приложения. Автор подробно описывает основные уязвимости веб-приложений и предлагает практические советы по их предотвращению. Книга содержит множество примеров кода и наглядных иллюстраций, которые помогут вам лучше понять, как работают уязвимости и как их можно избежать.

В формате a4.pdf сохранен издательский макет.

Содержание

Таня Янка
Безопасность веб-приложений
Исчерпывающий гид для начинающих разработчиков

Alice and Bob Learn Application Security

* * *

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!
***Дэфидд Штуттард**, соавтор бестселлера The Web Application Hacker's Handbook, создатель приложения Burp 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 идее. Его домашняя страница – **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, хорошей передышки в горах и стакана вкусного виски (односолодового, конечно). Вы можете найти его в Twitter (**@7hunderSon**) и на GitHub (**thunderson**). С ним можно связаться также по электронной почте **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)[1]. 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

[1] Отчет о нарушениях данных за 2016, 2017, 2018 годы.

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 многого 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, архитекто-

ром 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. B.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 уви-

деть 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, обфускация 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** 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** 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

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 IT-безопасности заключается в защите *конфиденциальности, целостности* и *доступности* систем и данных компании, 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 устрой-

ства 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 отрас-

ли. 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 пра-

вительстве 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»[2]. 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 то-

[2] **time.com/3404330/home-depot-hack**.

го, 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 устранению 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 устранить 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.

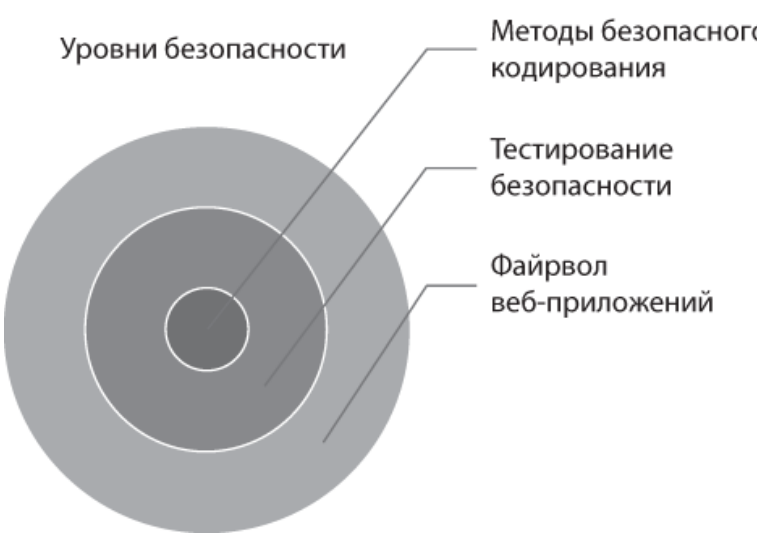

Рис. 1.5. Три уровня безопасности приложения; пример глубокой защиты

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 ин-

струментов 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 уни-

чтожить 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 изготовляться 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[3] (того, что написали вы и ваши коллеги), в остальном – из сторонних компонентов, часто называемых зависимостями. При подключении зависимостей к

[3] **www.infoworld.com/article/2626167/third-party-code-putting-companies-at-risk.html**.

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[4].

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.

---

[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**.

Безопасность через неясность

Концепция *безопасности через неясность* подразумевает, что если какой-то фрагмент системы скрыт, то он находится в «большей 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 обфускации своего кода, изменяя его таким образом, чтобы его было гораздо сложнее или вообще невозможно понять тому, кто попытается провести обратную разработку продукта.

> **ПРИМЕЧАНИЕ.** *Обфускация* – 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 обфускацию.

Другим примером «безопасности через неясность» явля-

ется 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 усилена. 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-устройства-

ми», 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 ничему 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 скриптинг* (*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 утруждают себя аутентификацией (подтвержде-

нием 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).

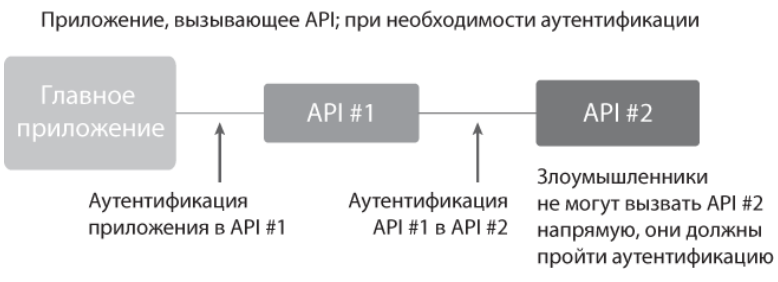

Рис. 1.7. Пример вызова API приложением и при необходимости аутентификации

Вот несколько примеров.

• Веб-сайт заражен хранимым межсайтовым скриптингом, 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 дан-

ным, 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-то, *что у вас есть*, что-то, *что*

является частью вас, и что-то, *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 «хорошей» 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?)

Конец ознакомительного фрагмента.