УДК 004.04
ББК 32.372
В29

Венц К.
В29 Безопасность ASP.NET Core / пер. с англ. Д. А. Беликова. – М.: ДМК Пресс, 2023. – 386 с.: ил.

ISBN 978-5-97060-176-4

В книге рассматриваются методы защиты веб-приложений ASP.NET Core: безопасное взаимодействие с браузером, распознавание и предотвращение распространенных угроз, развертывание уникальных API безопасности этого фреймворка. Приводятся способы написания безопасного кода и примеры с аннотациями, а также полное описание встроенных инструментов безопасности ASP.NET Core. Обсуждаются реальные нарушения в системе безопасности, включая мошеннические расширения Firefox и кражу паролей в Adobe. В универсальных рекомендациях по обеспечению безопасности учтены уникальные потребности приложений ASP.NET Core.

Книга адресована опытным разработчикам ASP.NET Core.

УДК 004.04
ББК 32.372

Copyright © DMK Press 2023. Authorized translation of the English edition © 2022 Manning Publications. This translation is published and sold by permission of Manning Publications, the owner of all rights to publish and sell the same.

Все права защищены. Любая часть этой книги не может быть воспроизведена в какой бы то ни было форме и какими бы то ни было средствами без письменного разрешения владельцев авторских прав.

ISBN 978-1-6334-3998-6 (англ.)
ISBN 978-5-97060-176-4 (рус.)

© Manning Publications, 2022
© Перевод, оформление, издание, ДМК Пресс, 2023

Оглавление

Часть I ПЕРВЫЕ ШАГИ ... 20
1 ▪ О безопасности веб-приложений ... 21

Часть II НЕЙТРАЛИЗАЦИЯ РАСПРОСТРАНЕННЫХ АТАК ... 35
2 ▪ Межсайтовый скриптинг ... 36
3 ▪ Атака на управление сессиями ... 74
4 ▪ Межсайтовая подделка запросов ... 93
5 ▪ Данные, не прошедшие валидацию ... 121
6 ▪ Внедрение SQL-кода (и другие виды внедрений) ... 142

Часть III БЕЗОПАСНОЕ ХРАНЕНИЕ ДАННЫХ ... 156
7 ▪ Хранение секретных данных ... 157
8 ▪ Работа с паролями ... 185

Часть IV КОНФИГУРИРОВАНИЕ ... 208
9 ▪ HTTP-заголовки ... 209
10 ▪ Обработка ошибок ... 224
11 ▪ Журналирование и проверка работоспособности ... 236

Часть V АУТЕНТИФИКАЦИЯ И АВТОРИЗАЦИЯ ... 256
12 ▪ Защита веб-приложений с помощью ASP.NET Core Identity ... 257
13 ▪ Защита API и одностраничных приложений ... 291

Часть VI БЕЗОПАСНОСТЬ КАК ПРОЦЕСС ... 338
14 ▪ Безопасные зависимости ... 339
15 ▪ Инструменты аудита ... 350
16 ▪ OWASP Top 10 ... 369

Содержание

Вступительное слово от сообщества ... 11
Введение ... 12
Об этой книге ... 14
Об авторе ... 18
Об иллюстрации на обложке ... 19

Часть I ПЕРВЫЕ ШАГИ ... 20

1 ***О безопасности веб-приложений*** ... 21
1.1 ASP.NET Core: история и фреймворки ... 22
1.1.1 История версий ASP.NET Core ... 23
1.1.2 MVC ... 23
1.1.3 Razor Pages ... 26
1.1.4 Веб-API ... 27
1.1.5 Blazor ... 29
1.2 Выявление и нейтрализация угроз ... 30
1.2.1 Компоненты веб-приложений ... 30
1.2.2 Эшелонированная защита ... 32
1.3 API, предназначенные для выполнения функций безопасности ... 33
1.4 Безопасность важна ... 33
Резюме ... 34

Часть II НЕЙТРАЛИЗАЦИЯ РАСПРОСТРАНЕННЫХ АТАК ... 35

2 ***Межсайтовый скриптинг*** ... 36
2.1 Анатомия атаки с использованием межсайтового скриптинга ... 37

2.2 Предотвращение межсайтового скриптинга ... 44
2.2.1 Правило ограничения домена ... 44
2.2.2 Экранирование HTML ... 46
2.2.3 Экранирование в другом контексте ... 50
2.3 Политика безопасности контента ... 53
2.3.1 Пример приложения ... 54
2.3.2 Как работает CSP ... 55
2.3.3 Рефакторинг приложений для CSP ... 57
2.3.4 Рекомендации по CSP ... 65
2.3.5 Функциональные возможности CSP третьего уровня ... 68
2.4 Дополнительные меры безопасности, предоставляемые браузерами ... 70
Резюме ... 73

3 ***Атака на управление сессиями*** ... 74
3.1 Анатомия атаки ... 75
3.1.1 Кража сеансовых файлов cookie ... 77
3.1.2 Файлы cookie и управление сессиями ... 78
3.2 Файлы cookie и параметры сессий ASP.NET Core ... 81
3.3 Поддержка протокола HTTPS ... 85
3.4 Обнаружение перехвата сессий ... 90
Резюме ... 91

4 ***Межсайтовая подделка запросов*** ... 93
4.1 Анатомия межсайтовой подделки запросов ... 94
4.2 Меры противодействия для защиты от межсайтовой подделки запросов ... 102
4.2.1 Делаем HTTP-запрос непредсказуемым ... 102
4.2.2 Защита сеансового файла cookie ... 106
4.3 Кликджекинг ... 108
4.4 Совместное использование ресурсов между разными источниками ... 112
Резюме ... 119

5 ***Данные, не прошедшие валидацию*** ... 121
5.1 Взгляд на HTTP ... 122
5.2 Валидация в ASP.NET Core ... 126
5.3 Массовое переназначение параметров ... 130
5.4 Безопасная десериализация ... 137
Резюме ... 141

6 ***Внедрение SQL-кода (и другие виды внедрений)*** ... 142
6.1 Анатомия атаки с использованием внедрения SQL-кода ... 143
6.2 Подготовленные инструкции ... 146
6.3 Entity Framework Core ... 149

6.4 Внешние сущности XML 151
6.5 Другие виды внедрений 153
Резюме 154

Часть III БЕЗОПАСНОЕ ХРАНЕНИЕ ДАННЫХ 156

7 *Хранение секретных данных* 157
7.1 О шифровании 158
7.2 Secret Manager 161
7.3 Файл appsettings.json 164
7.4 Хранение секретов в облаке 167
7.4.1 Хранение секретов в Azure 168
7.4.2 Хранение секретов в AWS 172
7.4.3 Хранение секретов в Google Cloud 174
7.5 Использование API, обеспечивающего защиту данных 177
7.6 Локальное хранение секретов с помощью Blazor 181
Резюме 184

8 *Работа с паролями* 185
8.1 От утечки данных до кражи паролей 186
8.2 Реализация хеширования паролей 190
8.2.1 MD5 (и почему не стоит его использовать) 192
8.2.2 PBKDF2 195
8.2.3 Argon2 199
8.2.4 scrypt 201
8.2.5 bcrypt 202
8.3 Анализ шаблонов ASP.NET Core 204
Резюме 207

Часть IV КОНФИГУРИРОВАНИЕ 208

9 *HTTP-заголовки* 209
9.1 Сокрытие информации о сервере 211
9.2 Заголовки безопасности браузера 213
9.2.1 Заголовок Referrer-Policy 214
9.2.2 Заголовки Feature-Policy и Permissions-Policy 217
9.2.3 Предотвращение сканирования содержимого файла 218
9.2.4 Политики CORS 220
9.2.5 Дополнительные заголовки 221
Резюме 223

10 *Обработка ошибок* 224
10.1 Страницы ошибок для веб-приложений 225
10.1.1 Собственные страницы ошибок 227

10.1.2 Страницы ошибок для заданных кодов состояния 230
10.2 Обработка ошибок в API 232
Резюме 235

11 *Журналирование и проверка работоспособности* 236
11.1 Проверка работоспособности 237
11.1.1 Настройка проверки работоспособности 237
11.1.2 Расширенная проверка работоспособности 240
11.1.3 Форматирование вывода 242
11.1.4 Пользовательский интерфейс проверки работоспособности 243
11.2 Журналирование 247
11.2.1 Создание записей журнала 248
11.2.2 Уровни сообщения журнала 250
11.2.3 Области журналирования 252
Резюме 255

Часть V АУТЕНТИФИКАЦИЯ И АВТОРИЗАЦИЯ 256

12 *Защита веб-приложений с помощью ASP.NET Core Identity* 257
12.1 Настройка ASP.NET Core Identity 258
12.2 Основы ASP.NET Core Identity 262
12.3 Расширенные возможности ASP.NET Core Identity 271
12.3.1 Параметры пароля 272
12.3.2 Параметры файлов cookie 274
12.3.3 Блокировка пользователей 275
12.3.4 Работа с утверждениями 276
12.3.5 Двухфакторная аутентификация 279
12.3.6 Аутентификация с внешними поставщиками 284
Резюме 289

13 *Защита API и одностраничных приложений* 291
13.1 Защита API с помощью токенов 292
13.2 OAuth и OpenID Connect 303
13.2.1 OAuth и OpenID Connect 304
13.2.2 Потоки OAuth 305
13.3 Защита приложений 308
13.3.1 Сторонние инструменты 308
13.3.2 Учетные данные клиента 310
13.3.3 Код авторизации + PKCE 317
13.3.4 Одностраничные приложения и паттерн Backend-for-Frontend 325
Резюме 337

Часть VI БЕЗОПАСНОСТЬ КАК ПРОЦЕСС 338

14 ***Безопасные зависимости*** 339
14.1 Использование команды npm audit 340
14.2 Поддержание зависимостей NuGet в актуальном состоянии 344
Резюме 349

15 ***Инструменты аудита*** 350
15.1 Поиск уязвимостей 351
15.2 OWASP ZAP 353
15.3 Security Code Scan 359
15.4 GitHub Advanced Security 363
Резюме 367

16 ***OWASP Top 10*** 369
16.1 OWASP Top 10 370
16.1.1 Процесс создания списка OWASP Top 10 370
16.1.2 № 1: Нарушение контроля доступа 372
16.1.3 № 2: Криптографические ошибки 373
16.1.4 № 3: Внедрение 373
16.1.5 № 4: Небезопасный дизайн 374
16.1.6 № 5: Ошибки настроек безопасности 374
16.1.7 № 6: Уязвимые и устаревшие компоненты 375
16.1.8 № 7: Ошибки идентификации и аутентификации 375
16.1.9 № 8: Ошибки программного обеспечения и целостности данных 375
16.1.10 № 9: Ошибки журналирования и мониторинга 376
16.1.11 № 10: Подделка запросов со стороны сервера 376
16.2 OWASP API Top 10 378
16.3 Другие списки 379
Резюме 381

Предметный указатель 383

Вступительное слово от сообщества

ASP.NET Core – это профессиональный и гибкий инструмент для создания современных веб-приложений, и безопасность – это ключевой аспект, особенно когда речь идет о работе с конфиденциальной информацией. Сам по себе фреймворк ASP.NET Core безопасен благодаря разработкам и стандартам безопасности Microsoft. Однако, несмотря на старания Microsoft, разработчику требуется понимать основы кибербезопасности и уметь выстраивать механизмы защиты.

Создание безопасных приложений – это сложный и ответственный процесс, требующий глубоких знаний и опыта в области информационной безопасности. Автор рассказывает о лучших практиках и технологиях работы с секретными данными, векторах атак, защите веб-приложений как с точки зрения создания безопасного кода, так и с точки зрения защиты от попыток получения контроля извне.

Рекомендуем не только прочитать книгу, но и попробовать все практические примеры. Это эффективный способ узнать о том, как именно эксплуатируются наиболее популярные ошибки и уязвимости. Обязательно попробуйте разными способами «взломать» ненадежный магазин соков от OWASP, это увлекательный процесс.

Мы надеемся, что эта книга станет вашим надежным помощником в создании безопасных приложений на платформе .NET и поможет вам защитить свои приложения от различных угроз. Приятного чтения!

Над переводом работали представители сообщества DotNet.Ru:

- Игорь Лабутин;
- Дмитрий Жабин;
- Илья Лазарев;
- Рустам Сафин;
- Радмир Тагиров;
- Сергей Бензенко;
- Андрей Беленцов;
- Андрей Брижак;
- Алексей Ростов;
- Ринат Сафиуллин;
- Анатолий Кулаков.

Введение

Я до сих пор помню, как впервые столкнулся с темой безопасности веб-приложений, хотя в то время и не осознавал ее важности. Где-то в 1997 г., когда я занимался созданием веб-приложений (точнее, сайтов), услуги хостинга были очень дорогими. В случае с одним из моих проектов единственным вариантом, который я мог себе позволить, была возможность создать только одну страницу (!), и для этого мне приходилось использовать инструменты хостинг-провайдера – никакого собственного HTML или CSS. У меня было много свободного места на бесплатном хостинге, но я не мог использовать там собственный домен: у меня было что-то вроде http://home.someprovider.com/mysite.

Одной из очень немногих доступных мне функциональных возможностей была настройка ключевых слов страницы (раньше поисковые системы анализировали эту информацию). Если бы я использовал, например, словосочетание *web application security, hacking*, оно было бы преобразовано в следующую HTML-разметку:

```
<meta name="keywords" content="web application security, hacking">
```

Поэкспериментировав, я обнаружил, что могу попробовать такое «ключевое слово»:

```
"><meta http-equiv="refresh" content="0;
     url=http://home.someprovider.com/mysite"><"
```

Оказалось, что провайдер помещал эти данные в тег `<meta>` как есть, что привело к следующему результату (для удобочитаемости код отформатирован, а ввод выделен жирным шрифтом):

```
<meta name="keywords" content="">
<meta http-equiv="refresh" content="0;
     url=http://home.someprovider.com/mysite">
<"">
```

Таким образом я внедрил еще один тег `<meta>`, который перенаправлял посетителя на мой настоящий сайт, размещенный на бесплатном хостинге в другом месте.

Потребовалось время, прежде чем я понял значение того, что обнаружил, – на эту страницу можно было внедрить произвольный контент. Моя «атака» была безобидной, но можно было бы добавить и другую, более вредоносную разметку. Это пробудило во мне интерес к изучению безопасности веб-приложений, и с тех пор я не оглядывался назад. Я проверил бесчисленное количество приложений, работал с клиентами до и после аудита, учил разработчиков писать безопасные веб-приложения, выступал на трех континентах на конференциях, посвященных безопасности

веб-приложений, и изо всех сил старался сделать приложения, за которые я отвечал, максимально безопасными. В 2004 г. я впервые получил звание Microsoft MVP по ASP.NET. Я очень внимательно следил за API безопасности, подводными камнями и проблемами в этом фреймворке на протяжении многих лет.

Я думал написать книгу об опыте и знаниях, которые приобрел за последние 25 лет, но не было подходящего времени. В середине 2021 г. оно, наконец, настало, и я отправился в многомесячное путешествие, чтобы изложить все, что я знаю и считаю важным, в книге, которую вы собираетесь прочитать.

По моему опыту, недостаточно просто знать контрмеры, применяемые против определенных угроз. Разработчики должны понимать, как работают атаки, поскольку легче защититься от вещей, с которыми вы уже сталкивались. Вот почему во многих главах сначала будет продемонстрирована сама атака, а затем я объясню, как ее предотвратить. Помимо того что это облегчает восприятие материала, это еще и весело: мы видим, как что-то можно сломать, и называем это работой!

Как следует из названия, центральная тема книги – ASP.NET Core, куда входят Razor Pages и ASP.NET Core MVC. Также там, где это возможно, в ней рассматривается Blazor – еще один фреймворк для создания веб-приложений от компании Microsoft. Во всех примерах используется язык C#, а в качестве основы для них была выбрана платформа .NET 6 (в надежде, что они по-прежнему будут работать и со многими последующими версиями).

Благодарности

Многие, кто принимал участие в подготовке этой книги, упомянуты на обороте титульного листа (и это правильно!), и кроме них есть еще большое количество людей, которые помогли мне и внесли свой вклад.

Я в долгу как перед рецензентами, которые предоставляли полезные комментарии на различных этапах подготовки книги, так и перед читателями издания, которое было доступно по программе Manning Early Access Program (MEAP): Аль Пезевски, Билли Мигель Ванегас, Даниэль Васкес, Даррен Гиллис, Дэвид Пакку, Деннис Хейс Джордже, Дмитрий Сергеевич Дорогой, Дойл Тернер, Эммануил Шардалас, Гай Лэнгстон, Гарри Полдер, Джедиджа Буржуа, Джо Куэвас, Хосе Луис Перес , Марчин Сенк, Марек Петак, Маркус Вольф, Мэттью Харвелл, Майкл Холмс, Милош Тодорович, Ник МакГиннесс, Ник Римингтон, Онофри Джордж, Пол Браун, Ричард Вон, Рон Лиз, Сэмюэл Бош, Стэнли Анози, Сумит К. Сингх, Том Гет, Вьорель-Мариан Муазе и Уэйн Мэтер – спасибо за ваш вклад и помощь в улучшении книги.

Некоторые из моих близких друзей и коллег также оставили неоценимые отзывы и сделали книгу намного лучше. Спасибо всем вам за ваши идеи и поддержку!

Особая благодарность редактору по развитию Дагу Раддеру, который не только занимался поддержкой этого проекта, но и ловил меня каждый раз, когда я пытался «схалтурить».

Об этой книге

Название книги говорит само за себя: в ней рассказывается о безопасности приложений, созданных с использованием ASP.NET Core, поэтому здесь подробно описываются различные угрозы и риски для веб-приложений, основанных на технологии .NET. Я верю в принцип «показывай, а не рассказывай», поэтому вы увидите не только API и меры противодействия, но и то, как происходит атака. Основой для многих глав служат реальные инциденты.

Кому адресована эта книга?

Вы должны понимать основы .NET и владеть хотя бы одним из фреймворков ASP.NET Core (Razor Pages или MVC/Web API). Еще лучше, если вы хорошо разбираетесь в HTML и CSS («я понимаю это, когда вижу»). По крайней мере, в некоторых главах будет полезен поверхностный опыт работы с JavaScript. В качестве предпочтительного языка здесь используется C#, поэтому это еще одно необходимое условие, чтобы получить максимальную отдачу от прочтения книги.

Структура книги

Книга разбита на пять частей, которые содержат 16 глав. В первой части закладывается основа для последующих тем:

- в главе 1 обсуждается, почему важна безопасность веб-приложений и какие существуют параметры ASP.NET Core, а также то, как они могут быть затронуты. Вы получите краткую информацию о настройках проекта, которые предоставляет ASP.NET Core.

Во второй части показаны наиболее распространенные атаки на веб-приложения и способы защиты от них:

- в главе 2 рассматривается межсайтовый скриптинг (XSS) – очень распространенная атака, обычно основанная на внедрении вредоносного кода JavaScript. Пример с внедрением HTML, описанный во введении, также попадает в эту категорию;
- в главе 3 обсуждаются способы атаки на управление сессиями и как сделать сессии более безопасными. Сюда входят функциональные возможности, предоставляемые современными веб-браузерами;
- в главе 4 рассматривается межсайтовая подделка запросов (CSRF) – очень опасная атака, которую можно нейтрализовать

с помощью встроенных функций ASP.NET Core и механизмов защиты, которые есть в последних версиях браузеров;
- в главе 5 описываются потенциальные последствия использования данных, не прошедших валидацию, и как в этом может помочь ASP.NET Core. Сюда входит удобное и мощное средство: валидация модели;
- глава 6 посвящена внедрению SQL-кода, очень старой атаке, редко встречающейся в мире ASP.NET Core благодаря простым в использовании мерам противодействия и появлению инструментов объектно-реляционного отображения, в частности Entity Framework Core.

Третья часть посвящена безопасному хранению данных:

- в главе 7 рассказывается о том, как хранить секретные данные, в частности токены. Один из вариантов – использовать шифрование; другой – использовать предложения облачных провайдеров;
- в главе 8 обсуждается работа с паролями и способы их безопасного хранения. На самом деле пароли вообще не стоит хранить, в отличие от их хешей.

В четвертой части рассматриваются различные параметры конфигурации, связанные с безопасностью:

- в главе 9 подробно описаны HTTP-заголовки, поддерживаемые современными веб-браузерами, которые добавляют в приложение дополнительный уровень безопасности, а также рассказывается, как предотвратить отправку заголовков, отображающих скрытую информацию, клиенту;
- глава 10 знакомит вас с обработкой ошибок в приложениях ASP.NET Core с использованием передовых практик;
- в главе 11 рассматриваются две темы, которые отличаются друг от друга, но тем не менее в какой-то степени они связаны: журналирование может обеспечить сохранение диагностической информации о сайте для последующего ее извлечения, а проверка работоспособности обеспечивает механизм наблюдения за доступностью сайта и его сервисов.

Пятая часть посвящена проверке подлинности и авторизации в приложениях ASP.NET Core:

- в главе 12 вы познакомитесь с системой ASP.NET Core Identity, благодаря которой управлять пользователями и выполнять аутентификацию на сайте становится проще;
- в главе 13 описывается защита API и одностраничных приложений с помощью решения на основе токенов, а также рассматриваются протоколы OAuth и OpenID Connect с точки зрения ASP.NET Core.

Шестая часть охватывает ряд аспектов, являющихся частью процесса обеспечения безопасности:

- в главе 14 обсуждается, как обеспечить безопасность зависимостей, включая различные инструменты аудита;
- в главе 15 основное внимание уделяется инструментам аудита, которые могут помочь обнаружить уязвимости в веб-приложениях;
- в главе 16 рассматривается OWASP Top 10, регулярно обновляемый список десяти главных уязвимостей в веб-приложениях и то, как они описаны в этой книге.

Большинство глав не зависят друг от друга, но при необходимости приводятся соответствующие ссылки.

О коде

В этой книге содержится большое количество примеров исходного кода. Часть из них представляет собой пронумерованные листинги, а часть встречается в самом тексте. В обоих случаях исходный код отформатирован соответствующим шрифтом фиксированной ширины, чтобы отделить его от обычного текста. В некоторых случаях исходный код был переформатирован. Я добавил разделители строк и переделал отступы, чтобы уместить их по ширине книжных страниц. В редких случаях, когда этого оказалось недостаточно, в листинги были добавлены символы продолжения строки (➥). Также из многих листингов, описываемых в тексте, были убраны комментарии. Некоторые листинги сопровождают аннотации, выделяющие важные понятия.

Отзывы и пожелания

Мы всегда рады отзывам наших читателей. Расскажите нам, что вы думаете об этой книге, – что понравилось или, может быть, не понравилось. Отзывы важны для нас, чтобы выпускать книги, которые будут для вас максимально полезны.

Вы можете написать отзыв на нашем сайте www.dmkpress.com, зайдя на страницу книги и оставив комментарий в разделе «Отзывы и рецензии». Также можно послать письмо главному редактору по адресу dmkpress@gmail.com; при этом укажите название книги в теме письма.

Если вы являетесь экспертом в какой-либо области и заинтересованы в написании новой книги, заполните форму на нашем сайте по адресу http://dmkpress.com/authors/publish_book/ или напишите в издательство по адресу dmkpress@gmail.com.

Список опечаток

Хотя мы приняли все возможные меры для того, чтобы обеспечить высокое качество наших текстов, ошибки все равно случаются. Если

вы найдете ошибку в одной из наших книг, мы будем очень благодарны, если вы сообщите о ней главному редактору по адресу dmkpress@gmail.com. Сделав это, вы избавите других читателей от недопонимания и поможете нам улучшить последующие издания этой книги.

Нарушение авторских прав

Пиратство в интернете по-прежнему остается насущной проблемой. Издательства «ДМК Пресс» и Manning Publications очень серьезно относятся к вопросам защиты авторских прав и лицензирования. Если вы столкнетесь в интернете с незаконной публикацией какой-либо из наших книг, пожалуйста, пришлите нам ссылку на интернет-ресурс, чтобы мы могли применить санкции.

Ссылку на подозрительные материалы можно прислать по адресу электронной почты dmkpress@gmail.com.

Мы высоко ценим любую помощь по защите наших авторов, благодаря которой мы можем предоставлять вам качественные материалы.

Об авторе

Кристиан Венц – пионер Интернета, специалист по информационным технологиям и предприниматель. С 1999 г. он написал около 150 книг, посвященных веб-технологиям и смежным темам, которые были переведены на десять языков. На своей основной работе он занимается консультацией предприятий по вопросам цифровизации и Индустрии 4.0, или Четвертой промышленной революции. Будучи неизменным участником международных конференций для разработчиков, он выступал на трех континентах. С 2004 г. Кристиан носит звание MVP для ASP.NET, является ведущим автором официальной сертификации по PHP и время от времени участвует в проектах по разработке программного обеспечения с открытым исходным кодом. У него высшее образование в области компьютерных наук и бизнес-информатики, и он двукратный обладатель премии Кнута.

Часть I

Первые шаги

Не проходит и недели без каких-либо громких инцидентов в области информационной безопасности: будь то утечка данных в открытый доступ, популярные библиотеки кода, получающие обновления с вредоносным ПО, распространение новых программ-вымогателей и сайты, подверженные уязвимостям. Многие из событий, о которых вы читаете в IT-новостях, стали возможными благодаря ошибкам в коде. Поскольку центральная тема этой книги – ASP.NET Core, в первой главе будут представлены возможности веб-приложений, предоставляемые этой технологией, и анализ того, где могут произойти атаки. Мы построим «ментальную модель» для понимания остальной книги.

1 О безопасности веб-приложений

В этой главе:

- почему важна безопасность веб-приложений;
- использование ASP.NET Core для создания веб-приложений и API;
- почему определенные части приложения находятся под угрозой;
- чего ожидать от этой книги.

Девять из десяти веб-приложений имеют уязвимости. Таков довольно пугающий вывод исследования, опубликованного в 2020 г. компанией Positive Technologies (http://mng.bz/mOj2), поставщиком различных решений в области безопасности. Очевидно, что те, кто проводит такие исследования, часто могут быть предвзяты, но ряд других исследований, проведенных в предыдущие годы, дал аналогичные результаты. Вот отчет по одному из исследований 2009 г.: http://mng.bz/5Qo1.

Авторы исследования также обнаружили, что примерно четыре из пяти уязвимостей веб-приложений находятся в коде, а не, скажем, в конфигурации сервера. Основываясь на этом, можно выделить две тенденции:

- основная угроза безопасности веб-приложений заключается в коде;

- проблема носит масштабный характер, и ситуация, похоже, не улучшается.

Часто недостатки в системе безопасности проявляются не сразу, а только когда станет слишком поздно и веб-приложение будет успешно взломано. Поэтому необходимо сделать безопасность веб-приложений главным приоритетом и использовать передовые методы для ее обеспечения с самого начала проекта.

Большинство угроз связаны с работой веб-браузеров, протокола HTTP, серверов баз данных и другими «сетевыми аспектами». Следовательно, эти риски не зависят от технологий. Вот один из примеров: теоретически внедрение кода JavaScript на сайт работает независимо от используемого языка сервера или фреймворка. На практике существуют следующие отличия:

1. некоторые языки и фреймворки обладают встроенными средствами противодействия, которые помогают предотвратить распространенные атаки без каких-либо дополнительных усилий на этапе разработки;
2. в разных технологиях и фреймворках функции, методы и API, используемые для защиты от определенных атак и рисков, естественно, называются по-разному.

Таким образом, книга, посвященная безопасности веб-приложений, должна представлять и описывать распространенные атаки в более или менее общем виде, после чего необходимо будет ввести меры противодействия, привязанные к определенной технологии. Стек, который мы будем использовать в этой книге, – это .NET; а поскольку мы говорим о веб-приложениях, то в центре внимания будет фреймворк ASP.NET Core. Книга была написана с использованием .NET 6 и ASP.NET Core 6, но надеюсь, что она будет полезна и при работе с более новыми версиями.

1.1 ASP.NET Core: история и фреймворки

У ASP.NET долгая история, связанная с историей платформы .NET, которая впервые была выпущена в виде бета-версии в 2001 г. и в качестве окончательной версии 1.0 в начале 2002 г. Тогда программный пакет назывался «.NET Framework» и содержал фреймворк для разработки серверных веб-приложений под названием ASP.NET (первые три буквы были взяты из названия предыдущей веб-технологии Microsoft ASP. Это сокращение от «Active Server Pages»). Вместе с .NET Framework появился новый язык программирования C#, который будет использоваться в этой книге, хотя существуют и другие варианты (Visual Basic for .NET, или функциональный язык F#).

1.1.1 История версий ASP.NET Core

ASP.NET и .NET Framework развивались на протяжении многих лет, но мы намеренно не рассматриваем их в этой книге. Возможно, это неожиданно, особенно учитывая название книги, но в 2010-х гг. компания Microsoft работала над новой версией .NET, кульминацией которой стал выпуск .NET Core 1.0 в середине 2016 г. Это была новая версия .NET с открытым исходным кодом, более или менее независимая от платформы, и она больше не была привязана к Windows. Слово *Core* использовалось, чтобы избежать путаницы с .NET Framework, особенно что касалось номеров версий. Сработало это или нет – другой вопрос, но, чтобы еще больше все запутать, Microsoft отказалась от этого слова в названии, когда появилась версия .NET 5.0. Причина: последняя и, вероятно, окончательная версия .NET Framework и ASP.NET – 4.8, поэтому .NET Framework 5 не будет; таким образом, «.NET 5» явно означает новую версию .NET.

Однако с ASP.NET все немного сложнее. У фреймворка ASP.NET MVC есть свои номера версий. Последний выпуск пакета NuGet ASP.NET MVC для .NET Framework – 5.2.8 (http://mng.bz/2nE0), поэтому на самом деле «ASP.NET 5» может означать три вещи:

- ASP.NET MVC 5 (на основе .NET Framework);
- ASP.NET Core 5 (на основе .NET 5, ранее известного как .NET Core);
- ASP.NET как часть .NET 5. Это предыдущее название проекта, который позже стал .NET Core 1.0.

Я думаю, можно согласиться с тем, что имело смысл оставить слово *Core*, чтобы сделать название продукта явным, поэтому на данный момент это *ASP.NET Core*. Не нужно быть пророком, чтобы предсказать, что в какой-то момент в будущем от него, скорее всего, откажутся. Но пока, если в названии есть это слово, мы говорим о текущей версии веб-фреймворка Microsoft, а не об устаревшей. В этой книге используется .NET 6, где слово *Core* все еще присутствует.

1.1.2 MVC

Архитектурный паттерн «модель–представление–контроллер» (MVC) был придуман в 70-х гг. прошлого века. Он появился в приложениях с графическим интерфейсом, но стал очень популярным при разработке веб-приложений. Написание HTML и CSS, отвечающих за внешний вид страницы, – это совершенно иной навык, нежели реализация серверной части. Поэтому отделение пользовательского интерфейса от логики имеет смысл, и MVC – один из доступных вариантов. Будучи адаптированным под веб-приложения, MVC в основном работает следующим образом (рис. 1.1):

- контроллер принимает пользовательский ввод (в случае с веб-приложением это данные HTTP-запроса);
- контроллер получает модель (часто данные из базы данных) и манипулирует ею, а затем передает эту модель представлению (обычно HTML-странице);
- клиент получает представление и может использовать его для создания нового запроса.

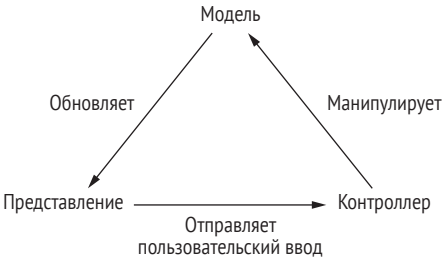

Рис. 1.1 Как работает паттерн MVC

В ASP.NET MVC эти компоненты обычно представлены следующим образом (поскольку ASP.NET MVC обладает широкими возможностями настройки, многие детали могут быть изменены, но мы опишем поведение по умолчанию):

- контроллер – это класс C#. Запросы сопоставляются с «методами действия». По сути, это общедоступные методы C#;
- модель обычно представляет собой объект или класс C#, часто заполненный содержимым базы данных (но не обязательно в соотношении 1:1). В примерах Microsoft обычно используют Entity Framework Core, инструмент объектно-реляционного отображения, что, конечно же, не обязательно. Контроллер получает доступ к модели и может манипулировать ею, а затем при необходимости передает ее представлению;
- представление – это, по сути, HTML-страница с дополнительной разметкой для привязки значений из модели или выполнения кода. Поскольку мы используем язык C#, эти страницы имеют расширение *cshtml*. Движок представлений Razor позволяет включать код C# в эти файлы с помощью специального символа @. Файлы компилируются, давая возможность выполнить C# код; браузер, разумеется, получает итоговый HTML-код.

При создании нового проекта в Visual Studio выбранный шаблон задает технологический стандарт для приложения. На рис. 1.2 показан ряд доступных шаблонов проектов. Обратите внимание, что четвертый вариант, ASP.NET Core Web App (Model-View-Controller), также включает веб-API, так как они очень похожи с точки зрения кода.

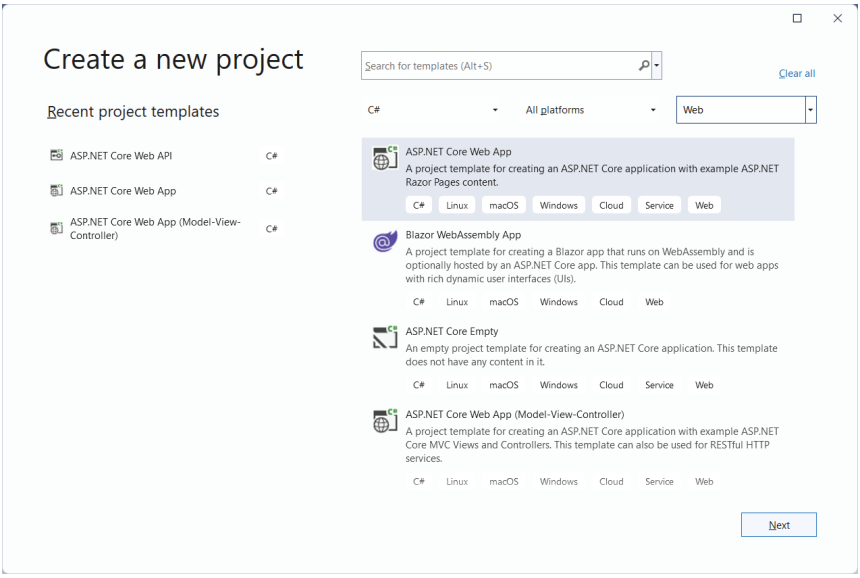

Рис. 1.2 Создание нового веб-приложения в Visual Studio

Рассмотрим основные элементы простого приложения. В следующем листинге показан контроллер.

Листинг 1.1 Контроллер простого приложения, в котором используется паттерн MVC

```
using Microsoft.AspNetCore.Mvc;

namespace AspNetCoreSecurity.MvcSamples.Controllers
{
      public class HomeController : Controller
      {
          public IActionResult Index()
          {
              var outcome = new Random().Next(1, 7);

              var roll = new DiceRoll(outcome);

              return View(roll);
          }
      }

      public record DiceRoll(int outcome);
}
```

Метод действия

Отправляет результат броска кости в представление, которое возвращается клиенту

Класс `HomeController` реализует метод действия `Index()`, который возвращает представление с результатом броска кости. Тип `DiceRoll` определен в том же файле исключительно для простоты. Представление показано в следующем листинге.

Листинг 1.2 Представление простого приложения с MVC

```
@model AspNetCoreSecurity.MvcSamples.Controllers.DiceRoll
@{
    Layout = null;
}

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Dice Roll -  MVC</title>
</head>
<body>
    <h1>Dice Roll: @Model?.outcome</h1>
</body>
</html>
```

Определяет тип модели для страницы

Выводит результат броска кости

В представлении результат броска, свойство `outcome`, отображается в элементе `<h1>`.

1.1.3 Razor Pages

Помните движок представлений Razor из предыдущего раздела? Простой, но эффективный синтаксис был улучшен, чтобы получить собственный подход к веб-разработке в рамках ASP.NET Core.

По сути, Razor Pages – это HTML-страницы с расширением *cshtml*, которые поддерживают синтаксис Razor. В отличие от фреймворка MVC, здесь нет необходимости в контроллере. Весь код, отвечающий за получение представления и обработку пользовательского ввода, теперь является частью страницы. Для приложений с простой логикой взаимодействия пользовательского интерфейса и кода для страницы такая архитектура гораздо удобнее и уменьшает сложность, присущую архитектуре MVC. В следующем листинге показана модель страницы простого приложения.

Листинг 1.3 Модель страницы простого приложения

```
using Microsoft.AspNetCore.Mvc.RazorPages;

namespace AspNetCoreSecurity.RazorSamples
{
    public class IndexModel : PageModel
    {
        public void OnGet()
        {
            var outcome = new Random().Next(1, 7);

            Roll = new DiceRoll(outcome);
        }
```

Метод вызывается при получении HTTP-запроса типа GET

```
        public DiceRoll? Roll { get; set; }

        public record DiceRoll (int outcome);
    }
}
```

Это свойство, которое будет использоваться страницей Razor

В данном случае, при отсутствии отдельного контроллера, такой класс IndexModel уже содержит код, выполняющий бросок кости при загрузке страницы. Модель, в свою очередь, привязана к представлению, которое показано в следующем листинге.

Листинг 1.4 Страница Razor простого приложения

```
@page
@model IndexModel

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8" />
    <meta name="viewport"
        ➥content="width=device-width, initial-scale=1.0" />
    <title>Dice Roll - MVC</title>
</head>
<body>
    <h1>Dice Roll: @Model?.Roll?.outcome</h1>
</body>
</html>
```

Ссылается на класс модели страницы

Выводит значение свойства из модели

Представление Razor выглядит почти так же, как представление из приложения в листинге 1.2. Результат показан в заголовке страницы.

1.1.4 Веб-API

С помощью ASP.NET Web API компания Microsoft предоставляет фреймворк для реализации REST-совместимых API. Создать свою реализацию конечных точек API довольно просто – извлеките данные из базы данных, преобразуйте их в формат JSON или XML и верните клиенту. Однако в Web API часть тяжелой работы выполняется за вас:

- в зависимости от значения заголовка HTTP-запроса `Accept` используется соответствующий формат (например, JSON или XML);
- корректно форматируются сообщения об ошибках и исключениях;
- упрощается возврат правильного кода состояния HTTP;
- управление версиями API осуществляется на основе HTTP-заголовков, компонентов пути или строки запроса.

С точки зрения разработки, веб-API работает аналогично ASP.NET MVC. Различаются только несколько базовых классов (и обычно веб-API работает быстрее, поскольку Razor Pages не используется). По сути, вы работаете с классом, который выглядит точно так же, как контроллер, и с методами, которые ведут себя – и выглядят – как методы действия MVC. В следующем листинге показан контроллер веб-API простого приложения.

Листинг 1.5 Контроллер веб-API простого приложения

```
using Microsoft.AspNetCore.Mvc;

namespace AspNetCoreSecurity.WebApiSamples.Controllers
{
    [ApiController]
    [Route("[controller]")]
    public class DiceRollController : ControllerBase
    {
        [HttpGet]
        public DiceRoll Get()
        {
            var outcome = new Random().Next(1, 7);

            var roll = new DiceRoll(outcome);

            return roll;
        }
    }

    public record DiceRoll(int outcome);
}
```

В зависимости от конфигурации метод вызывается при запросе GET к /DiceRoll

Возвращаемое значение автоматически преобразуется (например) в формат JSON

Контроллер – это класс с методом `Get()`, который будет выполняться при отправке `GET`-запроса к связанной конечной точке. Возвращаемые данные автоматически преобразуются в формат JSON.

> **ПРИМЕЧАНИЕ** В .NET 6 появились «минимальные API», в которых используется меньше кода – не требуется даже класс контроллера, однако никакой разницы с точки зрения функциональности и дополнительных (или не принятых во внимание) последствий для безопасности нет.

SignalR

Наиболее часто применяемый подход к вызову API со стороны клиента заключается в использовании JavaScript в качестве языка программирования и HTTP в качестве протокола прикладного уровня. Однако с точки зрения производительности данный протокол не оптимизирован по скорости. Все современные браузеры поддерживают технологию

WebSocket, протокол, который был признан как стандарт еще в 2011 г. (https://tools.ietf.org/html/rfc6455). Он может работать поверх HTTP (и поэтому совместим с существующими настройками и правилами брандмауэра), но поддерживает двоичные данные и работает в двунаправленном режиме.

Для использования SignalR (https://signalr.net/) Microsoft предлагает библиотеку с открытым исходным кодом, которая предоставляет API для создания конечных точек на сервере и компонент JavaScript для связи с этими конечными точками. В качестве предпочтительного протокола используется WebSocket, но если, скажем, старый браузер его не поддерживает, то будут задействованы другие средства, в частности HTTP-запросы.

1.1.5 Blazor

Помните веб-формы ASP.NET, попытку Microsoft мотивировать разработчиков, которым не нравятся веб-технологии, всё-таки пробовать создавать веб-приложения? Можно сказать, что они канули в Лету, но с выпуском Blazor Microsoft пробует другой подход, на сей раз ориентированный исключительно на разработчиков, которые не очень любят JavaScript. Все современные браузеры поддерживают стандарт WebAssembly, определяющий двоичный формат кода, выполняемого в браузере. С помощью Blazor код C# компилируется в WebAssembly, который затем может выполнить браузер. Следовательно, можно писать приложения на C# (или других языках .NET), а браузер может их запускать.

В настоящее время Blazor поддерживает два подхода (и их будет больше!), которые представлены на рис. 1.3 в виде схемы:

- *Blazor Server* – в браузер отправляется только пользовательский интерфейс, тогда как весь код C# находится на веб-сервере. Сгенерированный код JavaScript автоматически вызывает сервер (используя SignalR), который затем выполняет код C# и отправляет все изменения в DOM (объектная модель документа – по сути, содержимое) страницы обратно клиенту. Приложение загружается быстрее, но не работает офлайн;
- *Blazor WebAssembly* – все компилируется в WebAssembly, включая части .NET, используемые приложением. В этом случае загрузка приложения может занять несколько дополнительных секунд, но затем оно запускается в браузере без какого-либо взаимодействия с сервером, за исключением возможных вызовов API. Как следствие код приложения можно подвергнуть реверс-инжинирингу.

В следующем листинге показана страница Blazor как часть простого приложения.

Листинг 1.6 Страница Blazor простого приложения

```
@page "/"  <-- Указывает URL-адрес, связанный с этой страницей

<PageTitle>Dice Roll - Blazor</PageTitle>

@{
   var outcome = new Random().Next(1, 7);
}

<h1>Dice Roll: @outcome</h1>
```

На этот раз и бросок кости, и вывод находятся в одном файле. И снова мы используем синтаксис Razor с символом @.

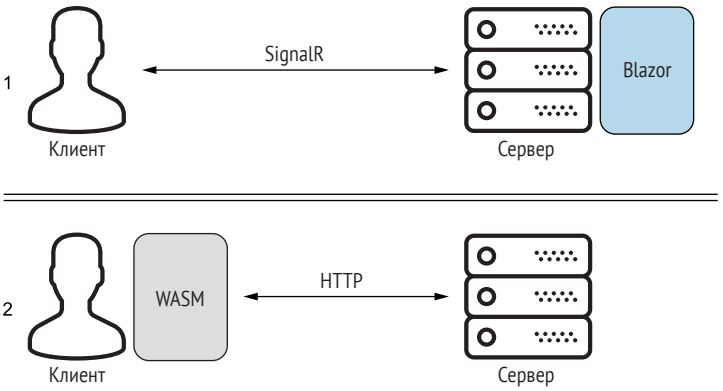

Рис. 1.3 Две архитектурные модели Blazor

1.2 Выявление и нейтрализация угроз

Как было показано в предыдущих разделах, ASP.NET Core предоставляет много возможностей для создания сайтов: различные серверные фреймворки и подходы, а также различные методы и форматы для клиентского кода. По сути, большое количество компонентов приводит к большой площади поверхности атаки, поэтому есть много мест, где что-то может пойти не так с точки зрения безопасности.

1.2.1 Компоненты веб-приложений

В современном веб-приложении есть и другие компоненты, например:

- базы данных;
- различные внешние сервисы;
- ресурсы файловой системы;
- CDN (сети доставки контента), содержащие библиотеки и другие, по большей части статические, ресурсы.

На рис. 1.4 показана типичная архитектура приложения. С точки зрения безопасности бóльшая часть компонентов может оказаться под угрозой, в том числе и связь между ними.

Вот некоторые угрозы, от которых веб-приложения должны обеспечивать защиту. О них и пойдет речь в этой книге:

- пользовательские данные, отправляемые в базу данных, могут содержать вредоносные команды, которые затем выполняются в этой базе. Такая атака называется *внедрением SQL-кода*;
- пользовательские данные, отправляемые на сервер, а затем клиенту, могут содержать нежелательный контент, например вредоносный код JavaScript. Обычно это называют *межсайтовым скриптингом*;

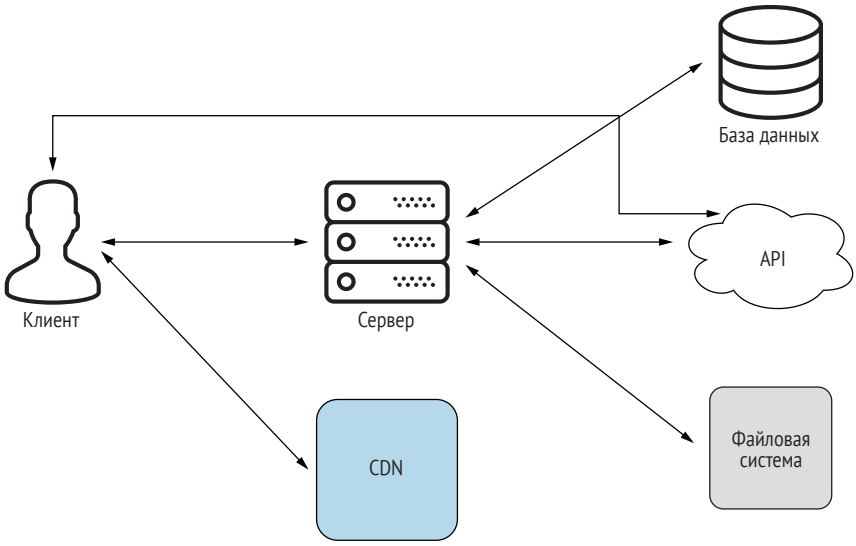

Рис. 1.4 Компоненты типичного веб-приложения

- данные, которыми обмениваются клиент и сервер, могут быть перехвачены или украдены, что подвергает риску некоторые части приложения. Здесь может помочь использование защищенного канала обмена данными;
- недостаточная авторизация может предоставить пользователям доступ к ресурсам сервера, которых они не должны видеть;
- то, как происходит аутентификация в интернете, может быть использовано для создания HTTP-запросов, обработка которых может приводить к нежелательным действиям в приложении. Межсайтовая подделка запросов – типичная атака этого типа;
- неожиданный ввод данных пользователем, например слишком большой или маленький объем данных или неверные данные, приводит к нежелательному поведению веб-приложения или провоцирует появление сообщений об ошибках;

- сообщения об ошибках могут раскрывать внутреннюю информацию о сервере, которая может быть полезна злоумышленнику. С другой стороны, внутренние ошибки необходимо правильно обрабатывать и журналировать;
- ресурсы, размещенные на стороннем сайте (например, CDN), могли быть изменены.

1.2.2 Эшелонированная защита

Если говорить о безопасности веб-приложений, то здесь существует эмпирическое правило: если что-то может пойти не так, то так и будет. Вот почему нужно предвидеть все вышеупомянутые риски и многое другое, использовать защитное программирование, при котором код приложения пишется в ожидании худшего сценария и реализуется как можно больше мер по обеспечению безопасности. Во второй части этой книги будут показаны все самые распространенные и известные на данный момент атаки, после чего мы реализуем меры противодействия.

Иногда, когда дело касается мер безопасности, избыточность может быть полезна. Лучше использовать два средства защиты, чем одно (или не использовать вообще). Кроме того, можно использовать меры безопасности на различных уровнях приложения. Когда речь заходит о производительности, избыточность часто считается чем-то плохим. Зачем делать две вещи, если достаточно одной? С другой стороны, когда дело доходит до доступности, избыточность помогает поддерживать работу приложения: если одна система выйдет из строя, есть резервная копия. Безопасность веб-приложений более тесно связана с доступностью, поэтому избыточные меры не являются проблемой; на самом деле обычно они приветствуются.

Как правило, мы говорим о механизмах эшелонированной защиты при внедрении нескольких уровней безопасности. Даже если один из уровней окажется недостаточным, мы надеемся, что есть и другие, которые предотвратят сбой приложения.

Если вы работаете в офисе, где есть охрана, то, скорее всего, дверь в офис или здание заперта. Вы можете возразить, что в этом нет необходимости, поскольку есть охранник, который заблокирует доступ нежелательных лиц в помещение, но у охранника может быть перерыв или он может отвлечься. Поэтому здесь применяется эшелонированная защита. Любая мера, направленная на обеспечение безопасности, может дать сбой, поэтому наличие еще одной такой меры может быть чрезвычайно полезным. В этой книге мы встретимся с механизмами, которые обеспечивают дополнительный уровень безопасности, но сами по себе они не способны полностью нейтрализовать угрозы. Однако в сочетании с другими средствами веб-приложение может стать оплотом против атак.

1.3 API, предназначенные для выполнения функций безопасности

Возможно, наиболее важным аспектом безопасности веб-приложений является нейтрализация рисков путем прогнозирования атак и реализации подходящих мер противодействия. Однако ASP.NET Core и связанные с ним технологии обладают рядом API, предназначенных для выполнения функций безопасности, которые не всегда можно напрямую сопоставить с конкретной атакой. Например:

- ASP.NET Core Identity предоставляет API для проверки подлинности и авторизации в веб-приложении, включая вход/выход пользователя, данные профиля, роли и многое другое;
- безопасное взаимодействие можно обеспечить различными вариантами и подходами, включая перенаправление на HTTPS, защиту файлов cookie и даже отключение HTTP;
- HTTP-заголовки, предназначенные для выполнения функций безопасности, можно настроить явно в файле web.config и контроллерах или неявно путем определения параметров в классе `Program` (в версиях .NET до 6 класс `Startup` был идеальным местом для этого);
- пароли не должны храниться в виде открытого текста. Они должны быть зашифрованы, а еще лучше, если они будут хешированы. В ASP.NET Core есть соответствующие форматы и алгоритмы для этого;
- у облачных провайдеров, в частности Microsoft Azure или AWS (Amazon Web Services), имеются собственные API для хранения строк подключения или паролей.

Крайне важно знать эти API, а также философию и механизмы безопасности ASP.NET. В третьей части книги вы узнаете, как именно они работают и когда их использовать. Однако на практике основной причиной являются фактические уязвимости в коде, и аудит информационной безопасности подтверждает это снова и снова.

1.4 Безопасность важна

Не могу не подчеркнуть: безопасность веб-приложений – чрезвычайно важная тема для всех участников процесса. Вот мотивация для лиц, принимающих участие в работе над веб-проектом:

- разработчики должны писать безопасные приложения. Нельзя просто исправить то, что в принципе не работоспособно. Им необходимо понимать существующие риски и то, как ASP.NET Core может помочь нейтрализовать их;

- менеджеры проектов и/или инженеры должны иметь представление об атаках, расставлять приоритеты касательно усилий по обеспечению безопасности и могут организовывать тестирование безопасности уже на ранних этапах разработки;
- технический директор должен знать о последствиях недостаточного обеспечения безопасности и понимать, что такие вещи не даются бесплатно; для этого необходимо прилагать усилия, которые должны быть предусмотрены в бюджете.

Кроме того, разве не весело пытаться взламывать приложения и получать за это деньги? Но поверьте, чем меньше я нахожу уязвимостей после аудита, тем счастливее я себя чувствую.

Резюме

Повторим, что нам известно на данный момент:

- подавляющее большинство веб-приложений имеют те или иные уязвимости, большая часть которых обусловлена кодом;
- большинство атак не зависят от конкретной используемой серверной технологии. Они используют общий знаменатель всех веб-приложений: HTML/CSS/JavaScript и HTTP;
- многие части веб-приложения могут быть подвержены риску: клиентская часть, сервер, база данных, внешние ресурсы и пути взаимодействия между ними;
- в этой книге будет использоваться ASP.NET Core, фреймворк для разработки веб-приложений на платформе .NET;
- в ASP.NET имеются встроенные средства защиты. Некоторые из них активированы по умолчанию, другие готовы к настройке и активации. Для остальных нужно писать собственный код.

Теперь, когда основа подготовлена, пришло время подробно изучить безопасность веб-приложений, начиная с распространенных (и опасных!) атак и того, как предвидеть их и защититься от них. Лучше быть параноиком, чем сидеть офлайн!