

Оглавление

1	■ О защите в деталях	19
Часть I	ОСНОВЫ КРИПТОГРАФИИ	32
2	■ Хеширование.....	33
3	■ Хеш-функции с ключом	48
4	■ Симметричное шифрование.....	60
5	■ Асимметричное шифрование	74
6	■ Transport Layer Security.....	87
Часть II	ПРОВЕРКА ЛИЧНОСТИ И ПРЕДОСТАВЛЕНИЕ ПРАВ.....	110
7	■ Сеанс HTTP	111
8	■ Проверка личности.....	128
9	■ Пользовательские пароли.....	147
10	■ Авторизация.....	172
11	■ OAuth 2.....	190
Часть III	ПРОТИВОСТОЯНИЕ АТАКАМ.....	214
12	■ Работа с операционной системой.....	215
13	■ Никогда не доверяйте вводу.....	227
14	■ Атаки методом межсайтового скриптинга	247
15	■ Политики защиты содержимого	268
16	■ Подделка межсайтовых запросов	285
17	■ Совместное использование ресурсов между разными источниками	299
18	■ Кликджекинг	314

Содержание

Оглавление	5
Предисловие	12
Об авторе	17
Об иллюстрации на обложке	18

1 О защите в деталях	19
1.1 Пространство для атаки	20
1.2 Глубокая оборона	22
1.2.1 Стандарты обеспечения защиты	23
1.2.2 Проверенные приемы	24
1.2.3 Основные принципы безопасности	25
1.3 Инструменты	27
1.3.1 Меньше слов, больше дела	30
Итоги	31

Часть I ОСНОВЫ КРИПТОГРАФИИ	32
---	----

2 Хеширование	33
2.1 Что такое хеш-функция?	33
2.1.1 Свойства криптографических хеш-функций	36
2.2 Архетипичные персонажи	38
2.3 Целостность данных	39
2.4 Выбор криптографической хеш-функции	40
2.4.1 Безопасные хеш-функции	40
2.4.2 небезопасные хеш-функции	41
2.5 Криптографическое хеширование в Python	43
2.6 Функции контрольного суммирования	45
Итоги	47

3 Хеш-функции с ключом	48
3.1 Подлинность данных	48
3.1.1 Генерация ключа	49
3.1.2 Хеширование с ключом	52
3.2 HMAC-функции	54
3.2.1 Проверка подлинности данных между системами	55

3.3	Атака по времени	57
	Итоги	59

4	Симметричное шифрование	60
4.1	Что такое шифрование?	60
4.1.1	Управление пакетами	62
4.2	Пакет cryptography	63
4.2.1	«Взрывчатые вещества»	63
4.2.2	«Готовые рецепты»	64
4.2.3	Смена ключа	66
4.3	Симметричное шифрование	67
4.3.1	Блочные шифры	67
4.3.2	Потоковые шифры	69
4.3.3	Режимы шифрования	70
	Итоги	73

5	Асимметричное шифрование	74
5.1	Загвоздка с передачей ключей	74
5.2	Асимметричное шифрование	75
5.2.1	RSA	76
5.3	Неопровержимость деяния	80
5.3.1	Цифровые подписи	80
5.3.2	Подписание данных криптосистемой RSA	82
5.3.3	Проверка подписи, созданной криптосистемой RSA	82
5.3.4	Подписание данных на базе эллиптических кривых	84
	Итоги	86

6	Transport Layer Security	87
6.1	SSL? TLS? HTTPS?	88
6.2	Атака «человек посередине»	88
6.3	Процедура подтверждения связи	90
6.3.1	Переговоры о наборе шифров	90
6.3.2	Обмен ключами	91
6.3.3	Проверка подлинности сервера	94
6.4	Общаемся по HTTP с Django	98
6.4.1	Параметр DEBUG	99
6.5	Общаемся по HTTPS с Unicorn	101
6.5.1	Самозаверенные сертификаты	102
6.5.2	Заголовок ответа Strict-Transport-Security	103
6.5.3	Переадресация на HTTPS	104
6.6	Пакет requests и TLS	105
6.7	Соединение с БД через TLS	106
6.8	Электронная почта через TLS	107
6.8.1	Режим «только TLS»	108
6.8.2	Проверка подлинности почтового клиента	108
6.8.3	Данные для доступа к SMTP-серверу	109
	Итоги	109

Часть II ПРОВЕРКА ЛИЧНОСТИ И ПРЕДОСТАВЛЕНИЕ ПРАВ 110

7	Сеанс HTTP	111
7.1	Что такое сеанс HTTP?	111
7.2	HTTP-куки.....	113
7.2.1	Атрибут <i>Secure</i>	114
7.2.2	Атрибут <i>Domain</i>	114
7.2.3	Атрибут <i>Max-Age</i>	115
7.2.4	Сеанс, пока запущен браузер	116
7.2.5	Установка куки в программном коде	116
7.3	Параметры сеанса	117
7.3.1	Упаковщик сеанса	117
7.3.2	Механизм на основе кеша	119
7.3.3	Механизм на основе кеша и базы данных	121
7.3.4	Механизм на основе базы данных	121
7.3.5	Механизм на основе файлов	122
7.3.6	Механизм на основе куки	122
	Итоги	127

8	Проверка личности	128
8.1	Регистрация пользователя.....	129
8.1.1	Шаблоны	133
8.1.2	Боб заводит учетную запись.....	135
8.2	Проверка личности.....	137
8.2.1	Встроенные представления	137
8.2.2	Создание приложения Django	139
8.2.3	Боб входит и выходит.....	141
8.3	Просим представиться загодя.....	143
8.4	Тестируем проверку личности и скрытое за ней	144
	Итоги	145

9	Пользовательские пароли	147
9.1	Сценарий смены пароля.....	148
9.1.1	Собственное средство проверки пароля	150
9.2	Хранение паролей.....	153
9.2.1	Засолка	156
9.2.2	Функции формирования ключа.....	158
9.3	Настройка хеширования паролей.....	162
9.3.1	Встроенные средства хеширования паролей	163
9.3.2	Собственное средство хеширования	164
9.3.3	Хеширование паролей через Argon2.....	164
9.3.4	Смена средства хеширования	165
9.4	Сценарий восстановления пароля	169
	Итоги	171

10	Авторизация	172
10.1	Авторизация на уровне приложения.....	173
10.1.1	Разрешения.....	174
10.1.2	Администрирование пользователей и групп.....	176
10.2	Принудительная авторизация.....	181
10.2.1	Сложный низкоуровневый путь.....	181
10.2.2	Простой способ высокого уровня.....	184
10.2.3	Отображение по условию.....	186
10.2.4	Тестирование авторизации.....	187
10.3	Антишаблоны и эффективные практики.....	188
	Итоги.....	189
11	OAuth 2	190
11.1	Типы авторизации.....	192
11.1.1	Процесс предоставления кода авторизации.....	192
11.2	Боб авторизует Чарли.....	196
11.2.1	Запрос авторизации.....	196
11.2.2	Предоставление авторизации.....	197
11.2.3	Обмен токенами.....	198
11.2.4	Доступ к защищенным ресурсам.....	198
11.3	Django OAuth Toolkit.....	200
11.3.1	Обязанности сервера авторизации.....	201
11.3.2	Обязанности сервера ресурсов.....	205
11.4	request-oauthlib.....	209
11.4.1	Обязанности клиента OAuth.....	210
	Итоги.....	213
Часть III ПРОТИВОСТОЯНИЕ АТАКАМ		214
12	Работа с операционной системой	215
12.1	Авторизация на уровне файловой системы.....	215
12.1.1	Определение разрешений.....	216
12.1.2	Работа с временными файлами.....	217
12.1.3	Работа с разрешениями файловой системы.....	218
12.2	Запуск внешних выполняемых файлов.....	221
12.2.1	Решение задач с помощью внутренних API.....	222
12.2.2	Использование модуля subprocess.....	224
	Итоги.....	226
13	Никогда не доверяйте вводу	227
13.1	Управление пакетами с помощью Pipenv.....	228
13.2	Удаленное выполнение кода YAML.....	230
13.3	Расширение сущностей XML.....	233
13.3.1	Атака квадратного взрыва.....	234
13.3.2	Атака миллиард насмешек.....	234

13.4	Отказ в обслуживании	236
13.5	Атаки с использованием заголовка Host	237
13.6	Атаки с непроверенной переадресацией	240
13.7	Внедрение SQL	243
13.7.1	Обычные SQL-запросы	244
13.7.2	Запросы на подключение к базе данных	245
	Итоги	246

14	Атаки методом межсайтового скриптинга	247
14.1	Что такое XSS?	248
14.1.1	Хранимый XSS	248
14.1.2	Отраженный XSS	249
14.1.3	XSS на основе DOM	250
14.2	Проверка ввода	252
14.2.1	Проверка формы Django	255
14.3	Экранирование вывода	258
14.3.1	Встроенные утилиты отображения	260
14.3.2	Заключение атрибутов HTML в кавычки	262
14.4	Заголовки HTTP-ответа	263
14.4.1	Отключение доступа к файлам cookie из JavaScript	263
14.4.2	Отключение анализа тина MIME	265
14.4.3	Заголовок X-XSS-Protection	267
	Итоги	267

15	Политики защиты содержимого	268
15.1	Конструирование политик защиты содержимого	270
15.1.1	Директивы извлечения	271
15.1.2	Директивы навигации и документов	276
15.2	Развертывание политики с помощью django-csp	276
15.3	Использование индивидуальных политик	278
15.4	Отчеты о нарушениях CSP	281
15.5	CSP Level 3	283
	Итоги	284

16	Подделка межсайтовых запросов	285
16.1	Что такое подделка запроса?	285
16.2	Управление идентификатором сеанса	287
16.3	Соглашения об управлении состоянием	290
16.3.1	Проверка метода HTTP	291
16.4	Проверка заголовка Referer	292
16.4.1	Заголовок ответа Referrer-Policy	294
16.5	Токены CSRF	295
16.5.1	POST-запросы	295
16.5.2	Другие небезопасные методы запроса	297
	Итоги	298

17	Совместное использование ресурсов между разными источниками	299
17.1	Политика одного источника.....	300
17.2	Простые запросы CORS.....	301
17.2.1	Асинхронные запросы между источниками.....	302
17.3	Реализация CORS с django-cors-headers.....	303
17.3.1	Настройка Access-Control-Allow-Origin.....	304
17.4	Предварительные запросы CORS.....	305
17.4.1	Отправка предварительного запроса.....	306
17.4.2	Отправка ответа на предварительный запрос.....	309
17.5	Отправка файлов cookie между источниками.....	311
17.6	Устойчивость к CORS и CSRF.....	312
	Итоги.....	313
18	Кликджекинг	314
18.1	Заголовок X-Frame-Options.....	317
18.1.1	Индивидуализация ответов.....	317
18.2	Заголовок Content-Security-Policy.....	318
18.2.1	X-Frame-Options и CSP.....	319
18.3	Идите в ногу с Мэллори.....	320
	Итоги.....	321
	Предметный указатель.....	322

Предисловие

Когда-то давно решил я посмотреть на Amazon книги о написании безопасных приложений на Python. Будет из чего выбрать, подумал я. К тому времени было издано множество книг о программировании на Python на разные темы: оптимизация кода, машинное обучение, веб-разработка и т. д.

К моему удивлению, такой книги не оказалось. А ведь эта тема касается задач, с которыми мы с коллегами сталкиваемся ежедневно. Как убедиться, что весь сетевой трафик зашифрован? На каком фреймворке построить безопасный веб-сайт? Каким алгоритмом стоит подписывать данные?

Спустя годы практики мы нашли проверенные приемы для решения стандартных задач и без применения несвободного ПО. За это время мы создали с нуля несколько проектов, где в безопасности хранились личные данные миллионов пользователей. Трое наших конкурентов, кстати, были взломаны.

Начало 2020-го изменило наши жизни. Отовсюду доносились известия о COVID-19, и удаленная работа нежданно стала обыденностью. Каждый провел эту пандемию по-своему. Меня же настигла невыносимая скука.

Так что я решил одним выстрелом убить двух зайцев. Написание книги, во-первых, оказалось восхитительным способом унять тоску на протяжении целого года на карантине. Осенью 2020-го в Силиконовой долине это оказалось настоящим спасением. Из-за смога от бушующих лесных пожаров большинство местных сидели дома.

Во-вторых, что важнее, оказалось очень приятно написать книгу, которую я так и не смог тогда приобрести. Многие открывают стартапы в Силиконовой долине, чтобы звать себя основателями. Так же и множество книг создаются ради того, чтобы написавший звал себя автором. Но что стартап, что книга должны решать насущные проблемы и приносить пользу.

Надеюсь, эта книга станет вам подспорьем при написании безопасного кода.

Благодарности

Написание книги – тяжкий уединенный труд. Потому легко упустить из виду тех, кто оказывал помощь. Я хотел бы поблагодарить всех нижеуказанных людей. Перечисляю в порядке нашей встречи.

Спасибо Катрин Берковиц (Kathryn Berkowitz), вы были моей лучшей учительницей английского в старших классах. Извините,

что докучал вам. Амит Ратор (Amit Rathore), дружище, спасибо, что порекомендовал меня издательству Manning. Хочу поблагодарить Джея Филдса (Jay Fields), Брайна Гётса (Brian Goetz) и Дина Уомплера (Dean Wampler) за ваши советы и поддержку, пока я искал издателя. Кэри Кемпстон (Cary Kempston), благодарю за содействие. Без вашего опыта эта книга просто бы не состоялась. Майк Стивенс (Mike Stephens), спасибо, что взглянули на мою рукопись и увидели в ней потенциал. Тони Арритола (Toni Arritola), мой редактор-консультант, спасибо вам – вы объяснили, что к чему. Ваши советы просто бесценны, благодаря вам я столько узнал о написании технических текстов. Майкл Дженсен (Michael Jensen), мой научный редактор, спасибо вам за содержательные рекомендации в короткие сроки. Благодаря вашим замечаниям и предложениям эта книга стала отменной.

И наконец, я хотел бы поблагодарить всех рецензентов издательства Manning, уделивших время на прочтение и поделившихся впечатлением. Аарон Бартон (Aaron Barton), Адриан Байерц (Adriaan Beiertz), Бобби Лин (Bobby Lin), Дайвид Морган (Daivid Morgan), Даниэль Васкес (Daniel Vasquez), Доминго Салазар (Domingo Salazar), Гжегож Мика (Grzegorz Mika), Ховард Уолл (Håvard Wall), Игорь ван Ооствин (Igor van Oostveen), Дженс Кристиан Бредал Мадсен (Jens Christian Bredahl Madsen), Камеш Ганешан (Kamesh Ganesan), Ману Сарина (Manu Sareena), Марк-Энтони Тейлор (Marc-Anthony Taylor), Марко Симон Зуппон (Marco Simone Zuppone), Мари Анн Тюгесен (Mary Anne Thygesen), Николас Актон (Nicolas Acton), Нинослав Серкез (Ninoslav Cerkez), Патрик Реган (Patrick Regan), Ричард Воан (Richard Vaughan), Тим ван Дорсен (Tim van Deurzen), Вина Гарapati (Veena Garapaty) и Уильям Джамир Силва (William Jamir Silva), ваши отклики помогли сделать книгу лучше.

Об этой книге

Python здесь является средством, чтобы обучить написанию защищенных программ. Проще говоря, эта книга больше про безопасность, чем про Python, и тому есть две причины. Во-первых, тема безопасности ПО куда обширнее, чем сам Python. Во-вторых, писать свои средства защиты – не лучшая идея. Серьезную работу стоит поручить алгоритмам, уже реализованным в Python, библиотеках кода либо сторонних утилитах.

В этой книге рассказывается о нюансах безопасного кода для начинающих программистов и профессионалов средней руки. Примеры иллюстрированы несложным кодом на Python. Для прочтения книги быть продвинутым профессионалом не требуется.

Для кого эта книга

Все примеры кода воспроизводят реальные задачи, стоящие перед разработчиками. Эта книга будет наиболее полезна тем, кто поддер-

живает уже работающие сервисы. Читателю потребуются начальные знания Python либо хорошее знакомство с другим популярным языком программирования. Эта книга будет полезна не только веб-разработчикам. Однако базовое понимание того, как работает интернет, пригодится при прочтении второй половины книги.

Возможно, вы не разработчик, а тестировщик. В таком случае вам станет ясно, что в первую очередь следует подвергать проверке. Но наша книга не рассказывает, как именно проводить тесты. Это все-таки разные навыки.

Эта книга не похожа на другие книги о безопасности. Здесь ни разу не будет показан процесс атаки со стороны злоумышленника, так что им здесь не будет особо что почерпнуть. Чтобы сгладить их разочарование, скажу, что временами злодеям будет дозволено одержать верх.

Структура книги

Первая глава рассказывает о том, что вам повстречается в книге. В ней кратко говорится о проверенных приемах написания безопасных программ. Остальные семнадцать глав делят книгу на три части.

Часть I «Основы криптографии» рассказывает о ее базовых понятиях. Описанное в ней вам еще повстречается во второй и третьей частях книги.

- Глава 2 начинает рассказ о криптографии с хеширования данных и проверки их целостности. Кроме того, мы впервые познакомимся с персонажами нашей книги.
- Глава 3 вытекает из материала второй главы. В ней говорится о проверке подлинности данных с помощью генерации ключей и последующего хеширования данных секретным ключом.
- Глава 4 затрагивает две темы, без которых не обходится ни одна книга по безопасности. Это симметричное шифрование и неразглашение содержимого.
- Глава 5, опять же, вытекает из материала предыдущей. В ней рассказывается об асимметричном шифровании, цифровых подписях и неопровержимости деяния.
- Глава 6 на основе идей из предыдущих глав рассказывает об общепринятом сетевом протоколе Transport Layer Security.

Часть II «Проверка подлинности и предоставление прав на доступ» включает в себе самую востребованную информацию. Она обязательно пригодится при поддержке коммерческих сервисов. В этой части содержатся пошаговые инструкции для воплощения типичных сценариев того, как пользователь взаимодействует с приложением.

- Глава 7 повествует о том, как установить пользовательский сеанс по протоколу HTTP, и в том числе о *куках* (cookies). Эта информация является базой для осуществления множества атак, которые мы обсудим позже.

- В главе 8 мы говорим о том, как узнавать нашего пользователя: о регистрации и проверке подлинности.
- Глава 9 рассказывает про обращение с пользовательскими паролями. В этой главе я порядочно разошелся. Для ее понимания нужно прочтение предыдущих.
- Глава 10 переходит от проверки подлинности пользователя к разграничению его прав.
- Глава 11 завершает вторую часть рассказом об OAuth. Это широко используемый протокол предоставления прав на доступ.

Настоящее противоборство разворачивается в части III «Противостояние атакам». Она проще для понимания и в целом увлекательная.

- Глава 12 погружается в недра операционной системы. Файловая система, запуск одной программой других, доступ к командной оболочке.
- Глава 13 учит вас противостоять различным атакам с целью внедрения вредоносного кода через пользовательский ввод.
- Глава 14 целиком посвящена самой печально известной атаке с целью внедрения кода – *межсайтовому скриптингу* (cross-site scripting – XSS). В самом деле, куда же без него.
- Глава 15 знакомит вас с *политиками защиты содержимого* (Content Security Policy – CSP). В каком-то смысле это бонусная глава про межсайтовый скриптинг.
- Глава 16 рассказывает о *межсайтовой подделке запросов* (cross-site request forgery – CSRF). В этой главе затрагиваются детали из нескольких предыдущих плюс обсуждается, как правильно использовать архитектурный стиль REST.
- Глава 17 описывает *политику одинакового источника* (same-origin policy) и объясняет, для чего нужно временами разрешать *использование ресурсов между разными источниками* (cross-origin resource sharing – CORS).
- Глава 18 завершает книгу рассказом о *кликджекинге* (clickjacking). Кроме того, в ней приводятся веб-ресурсы, которые стоит временами посматривать, чтобы ваши знания не устаревали.

О фрагментах кода

В этой книге приводится много примеров исходных кодов, как в виде отдельных пронумерованных листингов, так и прямо в тексте. В любом случае программный код для наглядности напечатан вот таким моноширинным шрифтом. Иногда код может быть **выделен жирным**. Таким образом выделено то, что поменялось по сравнению с предыдущим вариантом. Например, когда в существующей строке кода добавлен новый функционал.

Чаще всего исходный код был перекомпонован, чтобы вместить его на страницы книги. Были добавлены переносы строк и сокраще-

ны отступы. В редких случаях этого оказалось недостаточно. Если строка кода продолжается ниже, это обозначается символом ➔. Кроме того, если пояснение к коду дается в тексте, то из него удалены комментарии.

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

Деннис Бирн (Dennis Byrne) работает в команде IT-архитекторов сервиса 23andMe. В задачи команды входит защита медицинских данных более десятка миллионов клиентов. Ранее Деннис трудился разработчиком в LinkedIn. Он бодибилдер, а также кейв-дайвер под началом Global Underwater Explorers. Живет в Кремниевой долине, но вырос и отучился далеко от этих краев – на Аляске.

О защите в деталях



Темы этой главы:

- где находится пространство для атаки;
- что такое глубокая оборона;
- стандарты и проверенные приемы защиты;
- средства защиты в среде Python.

Никогда ранее нам еще не приходилось настолько доверять компаниям хранение вашей личной информации. Увы, некоторые из них уже сдали ее с потрохами взломщикам. Если вам трудно в это поверить, зайдите на <https://haveibeenpwned.com>. Этот сайт позволяет узнать по адресу электронной почты, есть ли ваши персональные данные среди миллиардов утекших учетных записей. Со временем эта база только растет. Если ваших учетных данных там еще нет – стоит благодарить специалистов по обеспечению безопасности сервисов, которыми вы пользуетесь.

Раз вы открыли эту книгу, вас наверняка интересует безопасность приложений не только как пользователя. Как и я, вы не только хотите быть клиентом защищенных сервисов – вы хотите их создавать. Большинство программистов признают важность написания безопасного кода, но у них не всегда есть понимание, как этого добиться. Эта книга дает крепкий базис для такого понимания.

Безопасность – это способность противостоять атакам. В этой главе подробно говорится о безопасности с ее лицевой стороны: как сервисы могут быть атакованы. Следующие главы рассказывают об обеспечении защиты изнутри с помощью инструментов, доступных в Python.

Для каждой атаки требуется место проникновения. Совокупность мест проникновения некоего сервиса называется *пространством для атаки* (attack surface). За пространством для атаки у защищенного приложения находятся уровни обороны. Этот архитектурный прием называется *глубокой обороной* (defence in depth). Уровни обороны строятся на основе стандартов и проверенных приемов, что исключает очевидные дыры.

1.1 Пространство для атаки

Обеспечение безопасности данных когда-то было просто небольшим перечнем того, чего стоит придерживаться и чего стоит избегать. Сейчас же это объемная область знания. Что же сделало ее таковой? Обеспечение безопасности стало нетривиальной наукой потому, что сами атаки стали нетривиальными. Какими они только не бывают. Стоит хорошо в них разбираться, прежде чем писать безопасный код.

Как говорилось выше, для каждой атаки требуется место проникновения. Совокупность возможных мест проникновения составляет пространство для атаки вашего приложения. Для каждого сервиса это пространство свое.

Атаки, как и пространство для них, изменчивы. Взломщики со временем осваивают новые приемы. Регулярно обнаруживаются доселе неизвестные уязвимости. Именно поэтому охрана вашего пространства – это непрекращающийся процесс. Компания должна быть озабочена этим постоянно.

Местом проникновения может быть пользователь, сам сервис или сеть связи между ними. Если говорить о пользователях, то взломщик в некоторых случаях может найти себе жертву через электронную почту либо чат. Средство подобных атак – обманом заставить пользователя активировать вредоносное содержимое, которое эксплуатирует уязвимость. Среди подобных атак можно перечислить следующее:

- непостоянный межсайтовый скриптинг (reflected cross-site scripting);
- социальная инженерия;
- межсайтовая подделка запросов;
- непроверенная переадресация (open redirect).

Но также и сам сервис может быть местом проникновения. Подобные атаки часто основываются на недостаточной проверке данных, поступаемых приложению. Классические примеры этого:

- внедрение в запрос к базе данных (SQL injection);
- удаленное исполнение кода;
- изменение HTTP-заголовка Host (Host header attack);
- отказ в обслуживании (denial of service).

Местами проникновения могут быть одновременно и пользователь, и сервис. Среди таких атак – хранимый межсайтовый скриптинг и кликджекинг.

Наконец, злоумышленник может воспользоваться сетью связи между пользователем и сервисом, в том числе промежуточными устройствами в сети, как местом проникновения. Среди таких атак – «человек посередине» (man-in-the-middle) и «попугай» (replay attack).

Эта книга учит вас обнаруживать подобные атаки и противостоять им. Некоторым из них посвящена целая глава, а межсайтовому скриптингу – даже две. Рисунок 1.1 иллюстрирует пространство для атаки типичного сервиса. Четыре злоумышленника одновременно прощупывают пространство, как показано пунктирными линиями. Пока что не погружайтесь в детали. Это всего лишь обзор того, что вас ожидает в нашей книге. После прочтения вам будет понятно, из чего состоит та или иная атака.

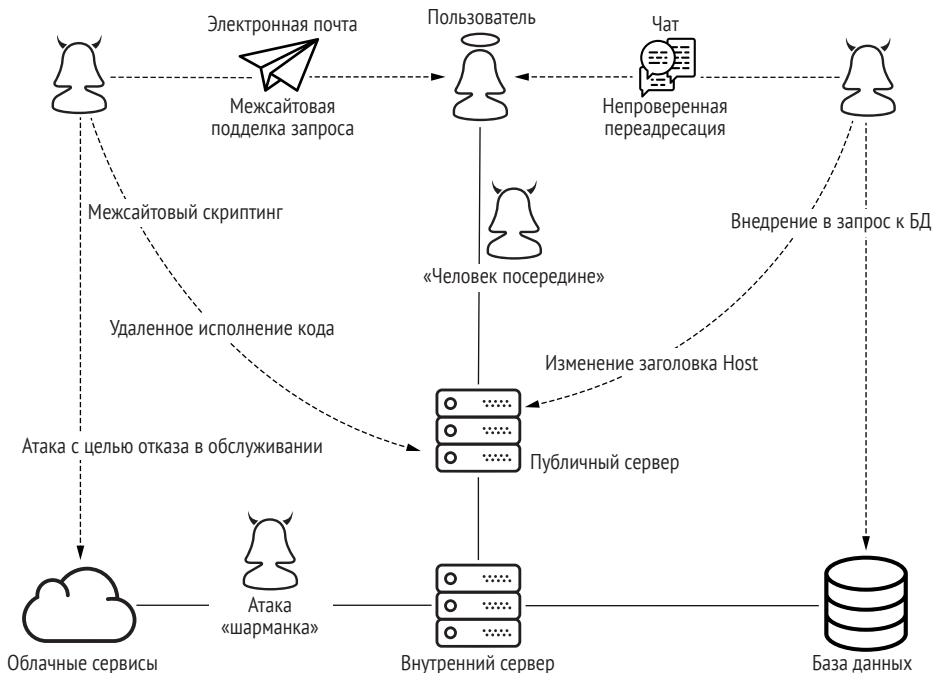


Рис. 1.1 Четыре злоумышленника используют места проникновения через пользователя, сервис и сеть связи

Под пространством для атаки у любого защищенного сервиса находятся уровни обороны. Одной защиты по периметру недостаточ-

но. Как упоминалось выше, подобный многоуровневый подход называется *глубокой обороной*.

1.2 Глубокая оборона

Этот подход зародился в Агентстве национальной безопасности США. Глубокая оборона подразумевает, что сервис должен противодействовать угрозам с помощью отдельных уровней. У каждого уровня две задачи: противостоять атакам и принимать удар на себя, если остальные уровни не справились с отражением атаки. Все потому, что не стоит класть яйца в одну корзину. Даже опытные программисты допускают ошибки, а новые уязвимости обнаруживаются постоянно.

Понятие «глубокая оборона» можно объяснить на таком примере. Представьте замок, у которого есть один уровень обороны – армия. Она непрерывно защищает замок от посягательств. Допустим, в 10 % случаев армия терпит поражение. Это весьма сильная армия, но королю не по себе и от 10%-ного риска. Что вас, что меня вряд ли бы устроил сервис, который пропускает 10 % атак. наших пользователей – тоже.

У короля есть два способа снизить риск. Первый – усилить армию. Это возможно, но экономически неоправданно. Устранение оставшихся 10 % риска в разы затратнее, чем устранение первых 10 %. Вместо того чтобы усиливать армию, король решает добавить дополнительный уровень обороны: выкопать ров вокруг замка.

Насколько наличие рва снижает риски? Теперь, чтобы завоевать замок, нужно преодолеть и армию, и ров. Чтобы посчитать риски, королю понадобится обыкновенное умножение. Допустим, ров тоже не может сдерживать 10 % атак. Итак, теперь захват замка возможен только в 10 % от 10 % случаев, что равняется 1 %. Представьте, насколько затратно может быть собрать армию, против которой устоит только 1 % врагов, по сравнению с тем, чтобы просто выкопать ров и залить его водой.

Как последнюю меру король возводит стену вокруг замка. Она тоже не выдержит натиск лишь 10 % атак. Теперь любая атака будет успешна только в 10 % от 10 % от 10 % случаев. Это 0,1 %.

В итоге подсчет выгоды от глубокой обороны сводится к умножению вероятностей. Добавление очередного уровня обороны всегда выгоднее шлифовки существующего. Концепция глубокой обороны признает, что стремиться к идеалу напрасно, и обращает это в достоинство.

Время показывает, какая реализация уровня обороны успешнее отражает атаки и пользуется большей популярностью. Способов выкопать тот же ров не так уж и много. Распространенная проблема рождает популярное решение. Специалисты замечают однообразную задачу, и экспериментальные приемы ее решения со временем

становятся стандартом. Стандарт в подробностях описывает шаблонную задачу и определяет ее решение.

1.2.1 Стандарты обеспечения защиты

Многие успешные стандарты обеспечения защиты были определены этими организациями: Национальный институт стандартов и технологий (National Institute of Standards and Technology – NIST), Инженерный совет Интернета (Internet Engineering Task Force – IETF), Консорциум Всемирной паутины (World Wide Web Consortium – W3C). Эта книга научит вас защищать сервисы, используя следующие стандарты:

- алгоритм симметричного шифрования Advanced Encryption Standard (AES);
- семейство криптографических хеш-функций Secure Hash Algorithm 2 (SHA-2);
- защищенный сетевой протокол Transport Layer Security (TLS);
- протокол предоставления прав доступа к разделенным ресурсам OAuth 2.0;
- протокол для использования ресурсов между разными источниками (CORS), применяющийся в веб-браузерах;
- стандарт политики защиты содержимого (CSP), который предотвращает выполнение некоторых атак в веб-браузере.

Зачем нужно создавать стандарты? Для того, чтобы программа, написанная в одной компании, могла взаимодействовать с программами от других разработчиков. Например, веб-сервер высылает любому браузеру один и тот же TLS-сертификат. У браузера же не возникнет проблем с обработкой TLS-сертификата от какого бы то ни было веб-сервера.

Кроме того, стандартизация позволяет переиспользовать код. Например, `oauthlib` это стандартная реализация протокола OAuth. На этой библиотеке построены как Django OAuth Toolkit, так и `flask-oauthlib`. В итоге один и тот же код пригодился и сервисам на Django, и приложениям на Flask.

Буду откровенен, стандартизация – это не таблетка от всех недугов. Иногда уязвимость обнаруживается в стандарте, который используется десятки лет. В 2017 году исследователи объявили о взломе SHA-1 (<https://shattered.io/>). Это криптографическая хеш-функция, которую повсеместно использовали более 20 лет.

Иногда разработчики запаздывают с реализацией стандартов в своих продуктах. Реализация политик защиты содержимого в популярных веб-браузерах растянулась на годы. Но практически всегда установление стандартов идет во благо, и потому нельзя ими пренебрегать.

В дополнение к стандартам безопасности возникают также и проверенные приемы. Глубокая оборона – сама по себе проверенный

прием. Как и стандарты, проверенные приемы используются в разработке защищенных сервисов. Отличие проверенных приемов от стандартов – в том, что для приемов не существует нормативной документации.

1.2.2 Проверенные приемы

Проверенные приемы берутся не из технических документов. Они, как байки, передаются из уст в уста, в том числе через подобные книги. Это то, чего обязательно стоит придерживаться, и никто, кроме вас, этого не проконтролирует. Эта книга поможет как замечать использование, так и придерживаться этих проверенных приемов, а именно:

- шифрования хранимых и передаваемых данных;
- «не изобретай свое шифрование»;
- принципа наименьших привилегий.

Данные могут либо передаваться, либо обрабатываться, либо храниться. Когда специалист говорит о шифровании хранимых и передаваемых данных, имеется в виду необходимость шифровать данные всегда, когда они передаются между компьютерами либо записываются для хранения.

Когда речь идет о том, чтобы не изобретать свое собственное шифрование, речь идет в целом о велосипедостроении в сфере безопасности. Смысл в том, чтобы использовать проверенное решение от умудренных опытом экспертов, а не пытаться сделать самому. Программисты полагаются на сторонние инструменты вовсе не из-за горящих сроков и не из желания писать меньше кода. Сторонний код испытан на прочность. Увы, большинство программистов приходят к пониманию этого лишь на своем горьком опыте. Вам же будет достаточно прочесть нашу книгу.

Принцип наименьших привилегий (principle of least privilege – PLP) подразумевает, что пользователю либо системе разрешен доступ к достаточному, но минимально возможному набору полномочий. Этот принцип встретится в книге при обсуждении различных тем: разграничения прав пользователей, протоколов OAuth и CORS, и не только.

Рисунок 1.2 показывает, как стандарты безопасности и проверенные приемы сочетаются в типичном веб-сервисе.

Ни один уровень обороны не панацея. Ни один стандарт либо прием никогда не предотвратят инцидентов сами по себе. Поэтому книга и содержит в себе, как и типичная программа на Python, множество как стандартов, так и проверенных приемов. Каждая из глав предлагает наметки для очередного уровня обороны в вашем сервисе.

Стандарты и приемы могут описываться по-разному, но по сути своей каждый из них говорит об одних и тех же азах. Эти азы и есть тот самый базис, на котором выстраивается защита.

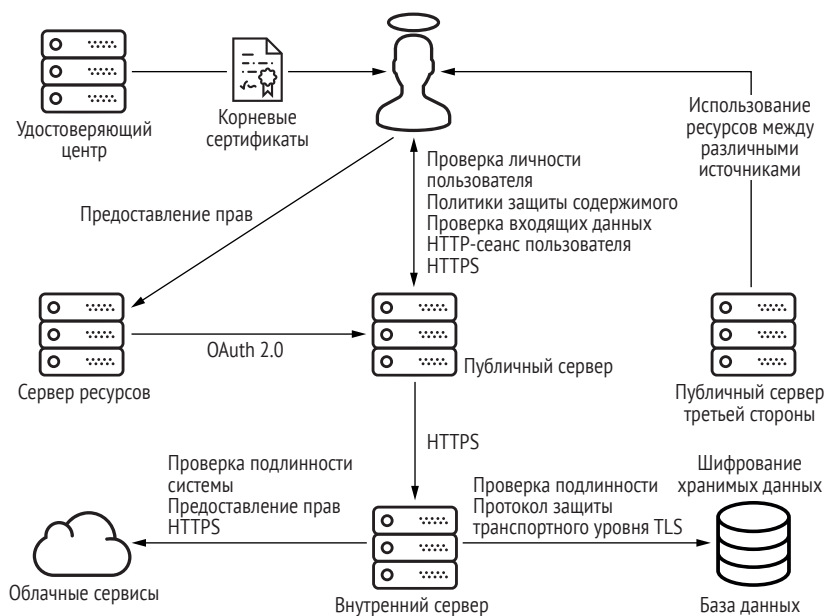


Рис. 1.2 Пример глубокой обороны, построенной на стандартах и проверенных приемах

1.2.3 Основные принципы безопасности

Ни один разговор о построении защищенных сервисов, и эта книга в том числе, не обходится без упоминания *основных принципов безопасности* (security fundamentals). Как алгебра и тригонометрия зиждутся на математике, так и стандарты и приемы вытекают из азов безопасности. Эта книга учит, как обезопасить систему на основе следующих принципов:

- целостность данных (изменены ли данные?);
- проверка подлинности (это кто?);
- проверка подлинности данных (кем созданы данные?);
- неопровержимость деяния (кто это сделал?);
- предоставление прав (что можно и что нельзя?);
- неразглашение (у кого есть доступ к данным?).

Целостность данных (data integrity, иногда message integrity) служит для проверки, не было ли содержимое случайно искажено из-за деградации данных (bit rot). Эта дисциплина служит для ответа на вопрос «изменены ли данные?». Таким образом мы можем быть уверены, что прочитали те же данные, которые были записаны. Целостность данных можно установить вне зависимости от того, кем они были произведены.

Проверка подлинности (authentication) отвечает на вопрос «кто это?». С этим процессом люди сталкиваются ежедневно. Это может быть как проверка личности, так и проверка подлинности чего бы то

ни было. Личность персоны устанавливается в момент ввода верной пары логина и пароля. Проверять можно не только людей, но и машин. Например, сервер непрерывной интеграции также проходит проверку подлинности, прежде чем забрать свежие изменения из системы контроля версий.

Проверка подлинности данных (data authentication, часто message authentication) обеспечивает возможность при чтении данных проверить личность того, кто данные записал. Здесь дается ответ на вопрос «кем созданы данные?». Как и в случае с целостностью данных, проверены могут быть любые данные, даже записанные той же стороной, которая их читает.

Неопровержимость деяния (nonrepudiation) в ответе за «кто это сделал?». В результате кто бы то ни был – ни личность, ни организация – не сможет отрицать содеянного. Неопровержимость может быть задействована при производстве любых действий, но она обязательна при совершении деловых операций и заключении юридических соглашений.

Предоставление прав (authorization, иногда access control) часто путают с проверкой подлинности. По-английски оба понятия звучат похоже, но означают они разные вещи. Как сказано выше, проверка подлинности отвечает на вопрос «кто это?». Предоставление прав же занимается вопросом «что можно и что нельзя?». Просмотр финансового отчета, отправка сообщения, отмена заказа – все это действия, которые могут быть позволены либо не позволены пользователю.

Неразглашение (confidentiality) дает ответ на вопрос «у кого есть доступ к данным?». Эта дисциплина служит для гарантирования того, что две или более стороны обмениваются данными исключительно частно. Информация, не подлежащая разглашению, при обмене ею для любого неадресата выглядит бессмыслицей.

В основе всех архитектурных решений лежат вышеперечисленные фундаментальные принципы. Таблица 1.1 перечисляет принципы и соответствующие им архитектурные решения.

Каждый основной принцип дополняет друг друга. Поодиночке от них немного толку, но вместе они сила. Приведу пример. Допустим, сервер электронной почты предоставляет проверку подлинности данных, но не проверяет их целостность. Как получатель, благодаря проверке подлинности данных вы можете быть уверены, что отправитель тот, за кого себя выдает. Но вот в том, что письмо дошло в первоизданном виде, сервис уже не гарантирует. Так себе сервис электронной почты, пожалуй. Нет никакого смысла проверять подлинность отправителя, если нельзя проверить целостность пришедшего.

Представьте новый классный сетевой протокол. Он гарантирует неразглашение, но не проверку подлинности. Любопытные глазки передаваемое сообщение увидеть не могут (неразглашение), а вот кто получает его на той стороне – уверенным быть нельзя. Так-то и получателем вполне может оказаться кто-то любопытный, а не

предназначенный адресат. Как давно вы последний раз делились сокровенным тет-а-тет неизвестно с кем? Как правило, охота знать и быть уверенным, кому отправляются секретные сведения.

Таблица 1.1. Основные принципы безопасности

Фундаментальный принцип	Архитектурное решение
Целостность данных	Защищенные сетевые протоколы Система контроля версий Диспетчер пакетов
Проверка подлинности	Проверка личности Проверка подлинности системы
Проверка подлинности данных	Регистрация пользователей Ввод пользователем учетных данных Механизм сброса и восстановления пароля Пользовательские сеансы
Неопровержимость деяния	Посылка данных об операции на сервер Цифровые подписи Доверенные третьи стороны
Предоставление прав	Выдача прав пользователям Выдача прав системам Выдача прав на доступ к файлам и папкам
Неразглашение	Алгоритмы шифрования Защищенные сетевые протоколы

И напоследок, пусть есть онлайн-банк, который умеет предоставлять права, но не умеет проверять подлинность личности. То есть авторизация ему по силам, а вот аутентичностью пользователя он не озабочен. Этот банк позволяет вам распоряжаться вашими деньгами, чужими не получится. Но вы ли это на самом деле, он и не спрашивает. Как же можно выдавать права, не убедившись сначала, кому эти права выдаются? Не стал бы я доверять этому банку деньги. А стали бы вы?

Основные принципы безопасности являются самыми базовыми кирпичиками защищенной системы. Из одного кирпича трудно что-либо построить. Но вот составляя их в разной последовательности, можно возводить уровни обороны один за одним. Для постройки каждого уровня обороны вместо голых рук следует подобрать инструмент. Какие-то из инструментов уже доступны в Python, какие-то из них доступны в отдельных пакетах.

1.3 Инструменты

Все примеры в данной книге написаны на Python – версии 3.8, если быть точным. Почему на Python? Ну, зачем вам читать книгу, которая быстро бы стала никому не нужной. Да и мне зачем писать такую. А Python – популярный язык, и становится только популярнее.

Индекс популярности языков программирования (Popularity of Programming Language Index – PYPL) составляется на основе данных

Google Trends. По состоянию на середину 2021 года Python занимает первое место с долей рынка 30 % (<http://pypl.github.io/PYPL.html>). За последние пять лет распространенность Python росла больше, чем у любого другого языка программирования.

Почему же Python настолько востребован? Есть множество ответов на этот вопрос. Многие согласны с двумя причинами. Во-первых, Python – язык программирования, подходящий для новичков. Его несложно учить, на нем легко писать и читать. Во-вторых, платформа вокруг языка переживает взрывной рост. В 2017-м Python Package Index (PyPI) насчитывал сто тысяч пакетов. Всего за два с половиной года это число удвоилось.

Мне не хотелось писать книгу исключительно о безопасности веб-приложений на Python. Поэтому некоторые главы повествуют о криптографии, генерации ключей и взаимодействии с операционной системой. Эти темы объясняются с помощью нескольких модулей Python:

- `hashlib` (<https://docs.python.org/3/library/hashlib.html>) для криптографического хеширования;
- `secrets` (<https://docs.python.org/3/library/secrets.html>) для генерации непредсказуемых случайных чисел;
- `hmac` (<https://docs.python.org/3/library/hmac.html>) для проверки подлинности сообщений по хешам;
- `os` и `subprocess` (<https://docs.python.org/3/library/os.html>, <https://docs.python.org/3/library/subprocess.html>) для доступа к возможностям операционной системы.

Некоторые инструменты заслужили свою собственную главу. О некоторых же рассказывается походя, а о каких-то и вовсе мельком. Вот они:

- `argon2-cffi` (<https://pypi.org/project/argon2-cffi/>) – функция для защиты паролей;
- `cryptography` (<https://pypi.org/project/cryptography/>) – пакет с распространенными криптографическими функциями;
- `defusedxml` (<https://pypi.org/project/defusedxml/>) – безопасный способ разбора XML;
- `Gunicorn` (<https://gunicorn.org>) – веб-сервер, написанный на Python;
- `Pipenv` (<https://pypi.org/project/pipenv/>) – пакетный менеджер с акцентом на безопасности;
- `requests` (<https://pypi.org/project/requests/>) – простая в использовании библиотека для отправки HTTP-запросов;
- `requests-oauthlib` (<https://pypi.org/project/requests-oauthlib/>) – реализация клиента протокола OAuth 2.0.

Большую роль в пространствах для атаки играют публичные веб-серверы. Поэтому в книге так много глав посвящено защите веб-приложений. Перед написанием этих глав мне пришлось задаться знакомым для многих питонистов вопросом: Flask или Django? Оба

фреймворка имеют хорошую репутацию. Главное различие между ними: первый аскетичен, второй же весьма функционален прямо «из коробки». По сравнению друг с другом, Flask предоставляет только самое необходимое, а Django подобен швейцарскому ножу.

Мне, как минималисту, нравится Flask. Увы, его аскетичность распространяется и на функционал обеспечения безопасности. Приложения, написанные на Flask, возлагают большинство уровней обороны на плечи сторонних библиотек.

Django же, наоборот, меньше полагается на сторонний код. В нем много встроенных функций защиты, они включены по умолчанию. В этой книге для демонстрации построения обороны веб-сервисов используется Django. Но и Django не панацея. Вдобавок применяются следующие библиотеки:

- `django-cors-headers` (<https://pypi.org/project/django-cors-headers/>) – серверная реализация протокола для использования ресурсов между разными источниками;
- `django-csp` (<https://pypi.org/project/django-csp/>) – серверная реализация политик защиты содержимого;
- `Django OAuth Toolkit` (<https://pypi.org/project/django-oauth-toolkit/>) – серверная реализация протокола OAuth 2.0;
- `django-registration` (<https://pypi.org/project/django-registration/>) – библиотека для реализации регистрации пользователей.

На рис. 1.3 изображено взаимодействие перечисленных инструментов. Unicorn пересылает трафик от пользователя и к нему через TLS. Пользовательский ввод проверяется встроенными в Django валидаторами форм и моделей, а также через объектно-реляционное связывание (object-relational mapping – ORM). Вывод очищается экранированием HTML. `django-cors-headers` и `django-csp` снабжают каждый ответ сервера соответствующими HTTP-заголовками протоколов CORS и CSP. На `hashlib` и `hmac` возложено хеширование. Пакет `cryptography` занимается шифрованием. `requests-oauthlib` взаимодействует с сервером OAuth. И наконец, `Pipenv` предохраняет от найденных уязвимостей в пакетах.

Это не значит, что надо использовать только упоминаемые фреймворки и библиотеки, а другие не надо. Прошу не брать на свой счет, если ваш любимый фреймворк оказался не у дел. Каждый инструмент в этой книге был выбран по двум критериям.

- *Состоявшийся ли это проект?* Чего уж точно не стоит делать, так это строить карьеру на фреймворке, который зародился буквально вчера. Я также нарочно не рассказываю об инструментах, стоящих на острие прогресса. О них можно и порезаться. Нельзя расценивать инструменты на такой жизненной стадии как безопасные. Именно поэтому все, о чем рассказывается в книге, прошло проверку временем.
- *Популярный ли это проект?* Здесь я задумывался скорее о будущем, чем о настоящем. Прошлое проекта меня не интересовало

вовсе. Если конкретно, то насколько велика вероятность, что этот же инструмент будет использоваться читателями книги в будущем? Но не столько важно, с помощью какого стороннего проекта я демонстрирую техники защиты. Самое важное – понять суть описанной техники.

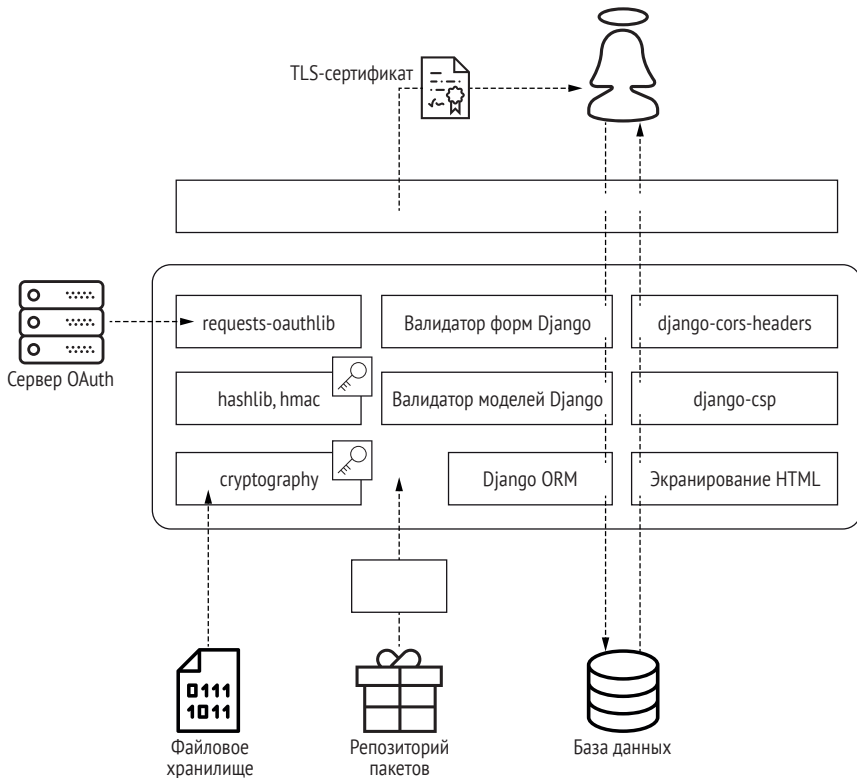


Рис. 1.3 Набор компонентов Python, создающий стойкую к атакам многоуровневую систему

1.3.1 *Меньше слов, больше дела*

Эта книга – прямое руководство, а не учебник. Она скорее для действующих программистов, нежели для еще постигающих азы. Я не хочу сказать, что академической стороной компьютерной безопасности можно пренебречь. Она очень важна. Но безопасность и Python – широкие темы. В этой книге дается выжимка всего важного и полезного.

В книге говорится о функциях для хеширования и шифрования. Я не затрагиваю вычисления, которые они производят под капотом. Вы узнаете, как работают эти функции, но не узнаете, как они реализованы. Вы увидите, когда и как их использовать, а когда не стоит.

Наша книга повысит вашу квалификацию как программиста, но не сделает экспертом по безопасности. И ни одна книга не делает. Не доверяйте рекламным проспектам. Пишите безопасные приложения на Python вместе с этой книгой! Сделайте существующий сервис безопаснее. С уверенностью разворачивайте ваш код на боевом окружении. Но не пишите в вашем LinkedIn, что вы специалист по криптографии.

Итоги

- Каждая атака начинается с точки проникновения. Совокупность точек проникновения системы является пространством для атаки.
- Нетривиальность атак зародила потребность в глубокой обороне. Это архитектурный подход, в котором оборона делится на уровни.
- Большинство уровней обороны действуют по стандартам и проверенным приемам ради совместимости, переиспользования кода и безопасности.
- Стандарты безопасности и проверенные приемы являют собой разные способы применения одних и тех же основных принципов.
- Вместо голых рук стремитесь подбирать инструменты, как то: фреймворки и библиотеки. Большинство программистов приходит к этому на своем горьком опыте.
- Эта книга повысит вашу квалификацию, но не сделает из вас эксперта по криптографии.

Часть I

Основы криптографии

Человечество каждый день полагается на хеширование, шифрование и цифровые подписи. Но вся слава обычно достается шифрованию. О нем чаще говорят на выступлениях и лекциях, на него чаще обращают внимание журналисты. Программистам тоже, как правило, любопытнее всего именно оно.

В первой части книги регулярно обращается внимание на то, почему без цифровых подписей и вычисления хешей невозможно обойтись – настолько же, насколько и без шифрования. Последующие части тоже постоянно напоминают об этом. Главы 2–6 полезно прочитать даже в отрыве от всей книги, но они будут хорошим подспорьем в понимании дальнейшего материала.

Хеширование

Темы этой главы:

- что такое хеш-функции;
- знакомство с архетипичными персонажами;
- проверка целостности данных с помощью хеширования;
- как выбрать криптографическую хеш-функцию;
- модуль `hashlib` для подсчета криптохешей.

В этой главе рассказывается о том, как применять хеш-функции для проверки целостности данных. Это необходимо для построения любой защищенной системы. Также говорится о том, как различать безопасные и небезопасные хеш-функции. Между делом мы познакомимся с Алисой, Бобом и другими архетипичными персонажами. Их обыкновенно можно увидеть в схемах атаки и защиты приложений. Эта книга не исключение. И в конце следует рассказ о модуле `hashlib`, с помощью которого и будут высчитываться хеши.

2.1 Что такое хеш-функция?

Любой хеш-функции можно подать на вход данные и получить на выходе результат. Входные данные хеш-функции называются *сообщением*. Сообщением могут быть любые данные. «Война и мир», кар-

тинка с котиком, пакет Python – все это может служить сообщением. На выходе хеш-функция выдает очень большое число. Это число носит много названий: *хеш*, *хеш-сумма*, *отпечаток*.

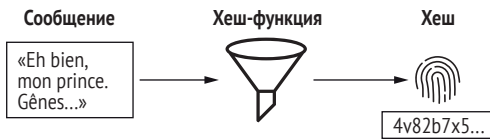


Рис. 2.1 Хеш-функция превращает входные данные, то есть сообщение, в хеш

В этой книге оно будет называться *хеш*. Хеш обычно представляет из себя строку из латинских букв и арабских цифр. Хеш-функция превращает набор любых сообщений в набор хешей. Рисунок 2.1 схематично описывает, как эти понятия взаимодействуют между собой.

В нашей книге для обозначения хеш-функций используется воронка. Что хеш-функция, что воронка принимают на вход содержимое неопределенного размера, но размеры результата предопределены. Хеш обозначается отпечатком пальца. Отпечаток уникален для человека, хеш уникален для сообщения, и оба могут быть использованы для опознания.

Хеш-функции отличаются друг от друга. Различия в общем сводятся к их свойствам, о которых рассказано чуть ниже. Чтобы познакомиться с некоторыми свойствами, нам пригодится встроенная в Python функция с говорящим названием `hash`. Она используется в Python для обработки словарей (`dictionary`) и наборов данных (`set`). Нам она пригодится в качестве примера.

Встроенная функция `hash` отлично подходит для знакомства с понятиями, так как она весьма незатейливее других хеш-функций, о которых мы поговорим дальше. Этой функции требуется один аргумент, сообщение, и на выходе получается хеш:

```
$ python
>>> message = 'message' ← Сообщениe, поданное на вход
>>> hash(message)
2010551929503284934 ← Хеш на выходе
```

Хеш-функциям присущи три основных свойства:


- детерминированное поведение;
- хеши неизменной длины;
- лавинный эффект.

ДЕТЕРМИНИРОВАННОЕ ПОВЕДЕНИЕ

Любая хеш-функция является *детерминированной*: для одного и того же сообщения на входе функция всегда выдает одинаковый результат. Иначе говоря, хеш-функциям присуща воспроизводимость

полученного результата, он не случаен. Встроенная функция `hash` всегда возвращает один и тот же хеш для любого отдельно взятого сообщения в рамках одного процесса Python. Запустите следующие две строки кода в интерактивной консоли Python. Полученные вами значения хешей будут совпадать друг с другом, но будут отличаться от полученных автором:

```
>>> hash('same message')
1116605938627321843
>>> hash('same message')
1116605938627321843
```



Один и тот же хеш

Хеш-функции, которые обсуждаются далее, абсолютно детерминированные. Они выдают один и тот же результат вне зависимости ни от чего.

ХЕШИ НЕИЗМЕННОЙ ДЛИНЫ

У сообщений произвольная длина, у хешей же для каждой хеш-функции длина строго определена. Если функции не присуще это качество, то она не может считаться хеш-функцией. Длина сообщения не должна влиять на длину хеша. Подача на вход встроенной функции `hash` различных сообщений даст на выходе различные хеши, но каждое значение всегда будет целым числом в заданных пределах.

ЛАВИННЫЙ ЭФФЕКТ

Когда незначительное изменение сообщения разительно сказывается на получаемом хеше, это означает, что хеш-функции присущ лавинный эффект. В идеальном случае каждый бит хеша зависит от каждого бита входных данных. Если два сообщения различаются хотя бы на бит, тогда в среднем только половина битов хеша должны совпасть. По каждой конкретной хеш-функции судят отдельно, насколько она близка к идеалу.

Взгляните на данный код. Значения хешей как для строки, так и для целого числа имеют заданную длину, но только на хеши от строк действует лавинный эффект.

```
>>> bin(hash('a'))
'0b100100110110010110110010001110011110011111011101010000111100010'
>>> bin(hash('b'))
'0b10111101111110110110010100110000001010000011110100010111001110'
>>>
>>> bin(hash(0))
'0b0'
>>> bin(hash(1))
'0b1'
```

Встроенная функция `hash` подходит для образовательных целей, но ее нельзя назвать криптографической хеш-функцией по трем причинам. Следующий раздел рассказывает о них.

2.1.1 Свойства криптографических хеш-функций

Криптографическая хеш-функция должна обладать тремя дополнительными признаками:

- быть вычислительно необратимой;
- иметь слабое сопротивление поиску коллизий;
- иметь сильное сопротивление поиску коллизий.

По-научному это называется *сопротивление поиску прообраза, сопротивление поиску второго прообраза, стойкость к коллизиям*. Для целей книги эти термины можно не использовать, никакого умысла задеть специалистов в этом нет.

ВЫЧИСЛИТЕЛЬНО НЕОБРАТИМЫЕ ФУНКЦИИ

Все криптографические хеш-функции без исключения обязаны быть *вычислительно необратимыми* (one-way functions). Эта такая функция, результат которой легко вычисляется, но найти аргумент по результату трудно. Иначе говоря, по выходным данным должно быть сложно определить входные. Если злоумышленнику попал в руки хеш, нам нужно, чтобы определить сообщение для этого хеша было очень непросто.

Насколько непросто? Можно сказать, что *недостижимо*. Это означает *очень сложно* – настолько сложно, что взломщику не остается другого способа, кроме метода «грубой силы».

Что подразумевается под «грубой силой»? Любому злоумышленнику, даже не особо смышленному, под силу написать несложный скрипт, чтобы создать очень много сообщений, вычислить хеш от каждого и сравнить этот хеш с имеющимся. Для этого атакующему нужно много времени и вычислительных мощностей: сила есть – ума не надо. Иногда этот метод еще называется *полным перебором*.

Сколько же потребуется времени и мощностей? Сложно сказать, это величина переменная. Если говорить о хеш-функциях, которые будут обсуждаться в дальнейшем, то потребуются миллиарды долларов и миллионы лет. Это то, что специалист по безопасности счел бы *недостижимым*. Но *недостижимым* не значит *невозможным*. Стоит признать, что идеальных хеш-функций не существует, так как все они подвержены перебору «грубой силой».

Недостижимое вполне может стать *достижимым*. То, что невозможно было подобрать полным перебором лет тридцать назад, может быть успешно подобрано сегодня или в недалеком будущем. Компьютерные комплекты продолжают дешеветь, вместе с ними дешевеет и применение атаки перебором. Увы, шифры со временем теряют былую стойкость. Это не значит, что любая система рано или поздно будет подвержена взлому. Это значит, что любой системе со временем следует переходить на более стойкие хеш-функции. В этой главе еще будет рассказано, как разумно выбирать хеш-функцию.

СОПРОТИВЛЕНИЕ ПОИСКУ КОЛЛИЗИЙ

Все без исключения используемые в криптографии хеш-функции должны обладать *сопротивлением к поиску коллизий*. Что такое коллизия? Несмотря на то что длина хешей разных сообщений одинакова, их значение всегда является различным. Почти всегда. Когда хеш двух разных сообщений оказывается одинаковым, это и называется коллизией. Коллизия – это плохо. Хеш-функции проектируются так, чтобы свести их возникновение к минимуму. Стойкость функции к появлению коллизий – это важный фактор. Некоторые функции справляются лучше, некоторые – хуже.

Если для заданного сообщения найти другое сообщение с таким же хешем недостижимо, то такая хеш-функция обладает *слабым сопротивлением к поиску коллизий*. Другими словами, если у взломщика на руках есть входные данные, найти другие данные с таким же хешем ему должно быть недостижимо.

Если недостижимо обнаружить какую бы то ни было коллизию в принципе, то такая хеш-функция обладает *сильным сопротивлением к поиску коллизий*. Разница между сильным и слабым сопротивлениями едва заметна. Слабое сопротивление касается нахождения коллизии хеша от известного, предварительно заданного сообщения. Сильное сопротивление касается нахождения коллизий хеша между любыми двумя сообщениями. На рис. 2.2 разница отображена наглядно.

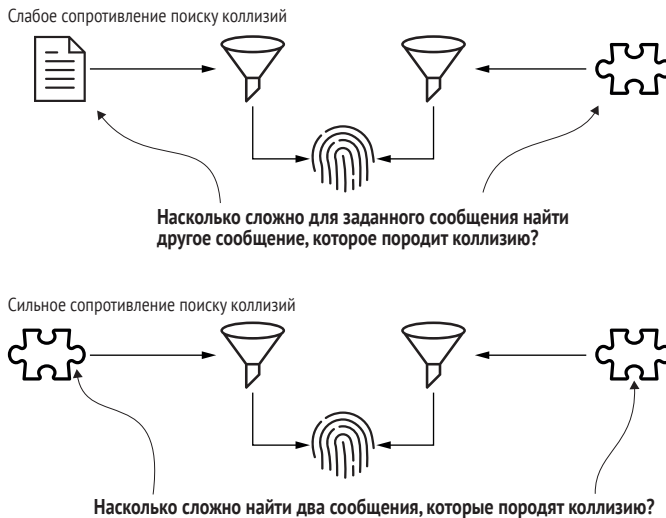


Рис. 2.2 Слабое и сильное сопротивления поиску коллизий

Если функция обладает сильным сопротивлением поиску коллизий, то обладает и слабым. Обратное же неверно. Хеш-функции со слабым сопротивлением поиску коллизий необязательно присуще

сильное сопротивление. Таким образом, реализация сильного сопротивления – весьма непростая задача. Если взломщик либо исследователь находит брешь в криптографической хеш-функции, то именно это свойство она теряет первым. Чуть позже вы увидите живые примеры таких атак.

Вся соль – в слове *недостижимо*. И хотелось бы повстречать хеш-функцию без коллизий, но такой просто-напросто не существует. Смотрите сами: сообщения могут быть любой длины, хеши могут быть только одной. Количество возможных сообщений заведомо превышает количество возможных хешей. *Принцип Дирихле* в действии.

В этом разделе было рассказано о том, что такое хеш-функция. Теперь перейдем к тому, как с помощью хеширования проверяется целостность данных. Но сперва позвольте познакомить вас с горсткой архетипичных персонажей. В течение книги они часто будут играть роль в пояснительных схемах, начиная как раз с понятия целостности данных.

2.2 Архетипичные персонажи

В схемах и иллюстрациях этой книги можно повстречать пятерых архетипичных персонажей. Вы можете увидеть их на рис. 2.3. Гарантирую вам, с их помощью вам будет куда проще понять материал, а мне – объяснить его. Задачи в этой книге построены вокруг ситуаций, в которых оказываются Алиса и Боб. Если вы читали другие книги о безопасности, то они вам уже должны быть знакомы. Алиса и Боб, как и вы, хотят безопасно создавать данные и обмениваться ими. Иногда к ним будет присоединяться их друг Чарли. Данные в целом будут пересылаться между ними. Алиса, Боб и Чарли – положительные персонажи. Можете представлять себя на их месте.

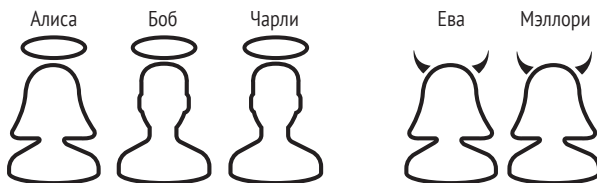


Рис 2.3 Положительные архетипичные персонажи обозначены нимбом, взломщикам пририсованы рожки

Ева и Мэллори – отрицательные действующие лица. Они атакуют Алису и Боба: пытаются похитить либо подменить данные между ними, а также пытаются выдать себя за них. Ева – пассивный обозреватель. Из всего пространства для атаки она скорее выберет обмен данными по сети. Мэллори – активная взломщица, она атакует куда

более изощренно. Чаще всего точкой проникновения она выбирает саму систему либо пользователей сервиса.

Запомните их, они вам еще встретятся. У Алисы, Боба и Чарли светятся нимбы. У Евы и Мэллори растут рожки. В следующем разделе Алиса воспользуется хешированием для проверки целостности данных.

2.3 Целостность данных

Целостность данных, иногда *целостность сообщения*, позволяет быть уверенным, что данные не были непреднамеренно изменены. Она дает ответ на вопрос «изменены ли данные?». Допустим, Алиса хранит деловые бумаги в системе документооборота. Сейчас, чтобы отслеживать целостность данных, в системе хранятся две копии каждого документа. Чтобы проверить их целостность, они сравниваются побайтово. Если копии разнятся, документ считается искаженным. Алиса недовольна тем, сколько места для хранения потребляет система. Этот сервис уже обходится в копейчку, и чем больше в нем документов, тем дороже стоит его содержать.

Алиса понимает, что стоит перед распространенной проблемой, и находит для нее распространенное решение. Она решает применить криптографическую хеш-функцию. При создании документа система высчитывает и сохраняет его хеш. Чтобы удостовериться в целостности данных, приложение заново высчитывает хеш и сравнивает его с сохраненным ранее значением. Если значения хешей не совпадают, документ считается искаженным.

Рисунок 2.4 пошагово описывает процесс. Фрагмент пазла означает процесс сравнения двух хешей.

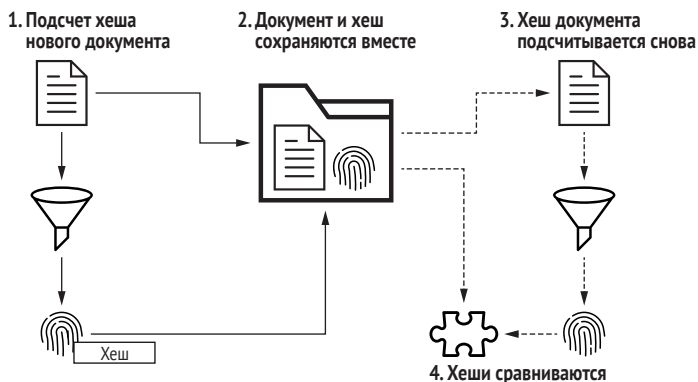


Рис. 2.4 Алиса убеждается в целостности данных, сравнивая хеши, а не документы

На этом примере ясно, почему сопротивление поиску коллизий – это важное свойство криптографической хеш-функции. Допустим,

Алиса бы использовала хеш-функцию, допускающую коллизии. Сервис бы просто не смог определить искаженный файл, если бы хеш оригинального и искаженного файла оказался одинаковым.

Этот раздел продемонстрировал важную область применения хеширования для определения целостности данных. В следующем разделе рассказывается о том, как из существующих хеш-функций выбрать подходящую для этой задачи.

2.4 Выбор криптографической хеш-функции

В Python уже встроена поддержка криптографического хеширования. Сторонние фреймворки либо библиотеки для этого не понадобятся. Встроенный модуль `hashlib` предлагает все, что может понадобиться большинству разработчиков для криптографического хеширования. В множестве `algorithms_guaranteed` хранятся все хеш-функции, которые гарантированно поддерживаются на всех платформах. Из этого множества вам и предстоит выбирать. Мало кому требуются функции за пределами данного набора:

```
>>> import hashlib
>>> sorted(hashlib.algorithms_guaranteed)
['blake2b', 'blake2s', 'md5', 'sha1', 'sha224', 'sha256', 'sha384',
'sha3_224', 'sha3_256', 'sha3_384', 'sha3_512', 'sha512', 'shake_128',
'shake_256']
```

Вряд ли вам хоть когда-то повстречаются хеш-функции не из этого списка.

Само собой, настолько широкий выбор может озадачить. И прежде чем выбирать, нужно разделить эти функции на безопасные и небезопасные.

2.4.1 Безопасные хеш-функции

Безопасные хеш-функции из списка `algorithms_guaranteed` принадлежат семействам:

- SHA-2;
- SHA-3;
- BLAKE2.

SHA-2

Семейство хеш-функций SHA-2 было представлено NSA в 2001 году. Оно состоит из функций SHA-224, SHA-256, SHA-384 и SHA-512. Основными функциями являются SHA-256 и SHA-512. Можете не запоминать их названия, пока что нас интересует только SHA-256. Она вам еще не раз встретится на протяжении книги.

Для криптографического хеширования по умолчанию стоит использовать SHA-256. Это очевидный выбор, ведь эта функция уже используется в любом сервисе. Операционные системы и сетевые протоколы, поверх которых работает приложение, уже полагаются на SHA-256. Выбирать не приходится: пришлось бы серьезно постараться никак не задействовать эту функцию. Она является безопасной, широко поддерживается и используется повсеместно.

В названиях всех функций SHA-2 уже указана длина их хешей. О хеш-функциях часто судят по длине ее хеша, и его длина нередко фигурирует в названии. SHA-256, например, выдает хеш длиной, как вы уже догадались, 256 бит. Чем длиннее хеш, тем вероятнее, что он уникален, и тем меньше вероятность коллизии. Чем длиннее, тем лучше.

SHA-3

Семейство хеш-функций SHA-3 состоит из SHA3-224, SHA3-256, SHA3-384, SHA3-512, SHAKE128 и SHAKE256. Семейство SHA-3 безопасно, и оно считается наследником SHA-2. Увы, на момент написания книги оно еще не набрало популярности. Стоит подумать об использовании функции этого семейства, например SHA3-256, если требуется повышенная безопасность. Но не забывайте, что поддержка данного семейства не настолько широкая, как у SHA-2.

BLAKE2

Алгоритм BLAKE2 не настолько популярен, насколько SHA-2 или SHA-3, но у него есть козырь в рукаве. BLAKE2 умело использует возможности современных ЦП, чтобы считать хеши на сверхвысоких скоростях. Именно поэтому BLAKE2 – ваш выбор, если вам требуется подсчитывать хеши для солидного объема данных. Есть две разновидности BLAKE2: BLAKE2b и BLAKE2s. BLAKE2b предназначен для 64-битных платформ. BLAKE2s разработан для платформ от 8 до 32 бит.

Мы познакомились с безопасными хеш-функциями и узнали, как выбирать между ними. Теперь пора узнать в лицо небезопасные, чтобы избегать их.

2.4.2 Небезопасные хеш-функции

Хеш-функции множества `algorithms_guaranteed` пользуются популярностью и отличаются кросс-платформенностью. Но это не значит, что все они безопасны для криптографических целей. Небезопасные хеш-функции оставлены в Python для обеспечения обратной совместимости. Знать о них стоит, потому что они могут повстречаться вам в устаревших системах. Небезопасные функции среди `algorithms_guaranteed` следующие:

- MD5;
- SHA-1.

MD5

MD5 – устаревшая 128-битная хеш-функция родом из начала 90-х. Это самая широко используемая хеш-функция всех времен и народов. Увы, она до сих пор в ходу, несмотря на то что исследователи продемонстрировали коллизии в ней еще в 2004-м. В наше время криптоаналитикам нужно менее часа, чтобы создать коллизию MD5-хешей на домашнем компьютере.

SHA-1

SHA-1 – устаревшая 160-битная хеш-функция, разработанная NSA в середине 90-х. Как и MD5, эта функция была некогда популярна, но она больше не считается безопасной. Google в сотрудничестве с Центром математики и информатики (Centrum Wiskunde & Informatica), научно-исследовательским институтом, расположенным в Нидерландах, сообщили о первых коллизиях в ней в 2017 году. Говоря языком терминов, они лишили эту функцию сильного сопротивления поиску коллизий. Слабое сопротивление по-прежнему в строю.

Многие разработчики знакомы с SHA-1 по системам контроля версий Git и Mercurial. Там хеши SHA-1 используются для проверки целостности коммитов и их идентификации. Линус Торвалдс, создатель Git, в 2007 году на Google Tech Talk сказал: «Применение SHA-1, во всяком случае в Git, не для безопасности вовсе. Это лишь способ наведения порядка».

ВНИМАНИЕ! MD5 либо SHA-1 ни за что не должны использоваться для целей безопасности при создании новых систем. Любой устаревший сервис, использующий эти функции, должен быть переписан с использованием безопасных альтернатив. Эти функции были некогда популярны, но сейчас популярной и безопасной является SHA-256. Устаревшие функции быстрые, но BLAKE2 еще быстрее и безопаснее.

Итак, вспомним, как стоит выбирать криптографическую хеш-функцию.

- Для большинства задач подходит SHA-256.
- Для обеспечения высокой безопасности подходит SHA3-256, но за это придется заплатить не настолько широкой поддержкой.
- Для объемных сообщений подходит BLAKE2.
- Ни за что не используйте MD5 либо SHA1 для целей безопасности.

В этом разделе вы узнали, как выбрать безопасную криптографическую хеш-функцию. Давайте применим эти знания на практике.