网络空间安全丛书

Python 全栈安全

[美] 丹尼斯 • 伯恩(Dennis Byrne) 著

赵宏伟 姚领田 冯少栋 唐进 译

清華大學出版社

北 京

北京市版权局著作权合同登记号图字：01-2021-6447
Dennis Byrne
Full Stack Python Security: Cryptography, TLS, and Attack Resistance
EISBN: 978-1-61729-882-0
Original English language edition published by Manning Publications, 178 South Hill Drive, Westampton, NJ 08060 USA. Copyright © 2021 by Manning Publications. Simplified Chinese-language edition copyright © 2022 by Tsinghua University Press. All rights reserved.

本书中文简体字版由 Manning 出版公司授权清华大学出版社独家出版。未经出版者书面许可，不得以任何方式复制或抄袭本书内容。

本书封面贴有清华大学出版社防伪标签，无标签者不得销售。
版权所有，侵权必究。举报：010-62782989，beiqinquan@tup.tsinghua.edu.cn。

图书在版编目(CIP)数据

Python 全栈安全/(美)丹尼斯 • 伯恩(Dennis Byrne)著；赵宏伟等译. —北京：清华大学出版社，2022.5

(网络空间安全丛书)

书名原文： Full Stack Python Security: Cryptography, TLS, and Attack Resistance

ISBN 978-7-302-60568-3

Ⅰ. ①P… Ⅱ. ①丹… ②赵… Ⅲ. ①软件工具－程序设计 Ⅳ. ①TP311.561

中国版本图书馆 CIP 数据核字(2022)第 064086 号

责任编辑：王 军
装帧设计：孔祥峰
责任校对：成凤进
责任印制：曹婉颖

出版发行：清华大学出版社
网 址：http://www.tup.com.cn，http://www.wqbook.com
地 址：北京清华大学学研大厦 A 座 邮 编：100084
社 总 机：010-83470000 邮 购：010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质 量 反 馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
印 装 者：小森印刷霸州有限公司
经 销：全国新华书店
开 本：148mm×210mm 印 张：11 字 数：347 千字
版 次：2022 年 7 月第 1 版 印 次：2022 年 7 月第 1 次印刷
定 价：98.00 元

产品编号：094326-01

译 者 序

根据 IEEE Spectrum 在 2021 年 8 月发布的年度编程语言影响力指数，以及知名软件质量跟踪与评估机构 TIOBE 的 10 月编程语言流行度指数排名，Python 均一骑绝尘，成为最受欢迎的编程语言，尤其在人工智能、数据科学、Web 开发、系统运维等领域的应用更为广泛。正因为如此，关于 Python 开发的书多如牛毛。从另一个角度看，广泛的应用带来了无处不在的攻击面，特别是在防护不到位的情况下，使组织信息环境面临的威胁明显增加。

诚如作者所言，关于 Python 的书很多，但关于 Python 安全的书并不多。即使有，也多是使用 Python 作为工具为安全自动化服务，而与 Python 开发安全相关的书几乎没有。本书正是在这种情景下由作者及其同仁共同努力实现的结果，向读者详细介绍了更多 Python 语言本身以外的安全知识，以及如何使用 Python 实现安全开发。这主要是通过充分利用 Python 自身及其他库、框架和工具的安全免疫因子，使得开发的产品具有更强的抗攻击性。

全书覆盖网络安全领域中的几个基础主题：密钥及其应用、密码、身份验证、授权和注入、跨站、同源、劫持等攻击抵御，简要介绍它们的技术原理以及相应问题的成因，并且对在应用 Python 开发时如何依赖 Python、第三方框架或工具规避对应的问题作了讲解，从而确保开发的代码具有安全免疫性。在纵深防御模型中，这种方式属于应用层防御策略，也是解决应用自身固有脆弱性的一种技术路线。希望读者朋友在阅读本书后，能够构建以软件生命周期为主轴的大防御观，在今后的编程实践中能够借鉴这种防御思维。不但要开发优秀的产品，更要开发安全的产品，从产品的设计、开发之初就重视安全开发，充分理解从 DevOps 向 DevSecOps

的演进背景，这些均有助于缓解软件产品全生命周期的综合风险。

一本书或许需要具备解决现实世界问题的能力，方能为读者创造价值。本书即为你解决现实世界中的安全问题而生，花点时间，它定不负光阴不负卿。美好世界将如你所愿。

本书由赵宏伟、姚领田、冯少栋、唐进翻译，参与翻译的还有王旭峰、贺丹、韩春侠，在此一并表示感谢。译者研究领域涉及威胁情报、威胁建模仿真、网络靶场技术及应用、武器系统网络安全试验与评估等方面，欢迎读者就本书中涉及的具体问题及上述领域内容与译者积极交流，共同学习进步，我们将不胜感激。

姚领田

2021 年 10 月

作 者 简 介

Dennis Byrne 是 23andMe 架构团队的成员，负责保护 1000 多万客户的基因数据和隐私。在 23andMe 之前，Dennis 是 LinkedIn 的软件工程师。Dennis 是一名健美运动员和 GUE 洞穴潜水员。他目前住在硅谷，远离阿拉斯加(他在那里长大并求学)。

致　谢

写作需要承受大量孤独和辛苦，因此很容易看不清是谁帮助了你。我要感谢以下帮助我的人。

感谢 Kathryn Berkowitz，你是世界上最好的高中英语老师。我很抱歉自己是这样的麻烦制造者。感谢 Amit Rathore，我的伙伴 ThoughtQuitter，把我介绍给 Manning。感谢 Jay Fields、Brian Goetz 和 Dean Wampler，在我寻找出版商的过程中提供了建议和意见。感谢 Cary Kempston 对作者团队的支持。如果没有现实生活的经验，我不会写这样一本书。感谢 Mike Stephens 看了我的原稿，发现了潜力。感谢开发编辑 Toni Arritola 教给了我诀窍。非常感谢你的反馈，我从中学到了很多关于技术写作的知识。感谢我的技术编辑 Michael Jensen 周到的反馈和快速的流转时间。你的评论和建议帮助本书获得了成功。

最后，我要感谢 Manning 的所有审阅人员，在这项工作的开发阶段占用了他们宝贵的时间并向我提供反馈：Aaron Barton、Adriaan Beiertz、Bobby Lin、Daivid Morgan、Daniel Vasquez、Domingo Salazar、Grzegorz Mika、Havard Wall、Igor van Oostveen、Jens Christian Bredahl Madsen、Kamesh Ganesan、Manu Sareena、Marc-Anthony Taylor、Marco Simone Zuppone、Mary Anne Thygesen、Nicolas Acton、Ninoslav Cerkez、Patrick Regan、Richard Vaughan、Tim van Deurzen、Veena Garapaty 和 William Jamir Silva，你们的建议使本书的质量更上一个台阶。

关于封面插图

本书封面插图的标题是 *Homme Touralinze*，即《西伯利亚某地区的秋明人》。这幅插图摘自 Jacques Grasset de Saint-Sauveur(1757—1810)所著的不同国家服饰作品集，名为 *Costumes de Différents Pays*，1797 年出版于法国。其中的每一幅插图都由手工绘制和着色。Jacques Grasset de Saint-Sauveur 收集的丰富多样的藏品生动地展示了 200 年前世界上的城镇和地区在文化上是多么的不同。人们彼此隔绝，说着不同的方言和语言。无论是在街头还是在乡下，只要通过他们的着装就可以很容易地辨别出他们住在哪里，以及他们的职业或生活地位是什么。

从那时起，人们的着装方式发生了变化，不同地区的多样性(在当时是如此丰富)已经消失。现在很难区分不同地区的居民，更不用说不同的城镇、地区或国家了。也许我们已经用文化的多样性换来了更多样化的个人生活——当然也换来了更多样、更快节奏的科技生活。

在一个计算机书籍难以被区分的时代，Manning 将图书封面以两个世纪前丰富多样的地区生活为基础，通过 Jacques Grasset de Saint-Sauveur 的图片使其重焕活力，颂扬了计算机行业的创造性和主动性。

序　言

几年前，我曾在亚马逊上搜索基于 Python 的应用程序安全书籍。我以为会有很多本书可供选择，因为关于性能、机器学习和 Web 开发等主题的 Python 书籍已有很多。

令人惊讶的是，我竟然找不到想要的书，找不到一本关于我和同事们正在解决的日常问题的书。我们如何确保所有网络流量都是加密的？我们应该使用哪些框架来保护 Web 应用程序？我们应该使用什么算法对数据进行散列或签名？

在接下来的几年里，我和同事们在确定一套标准的开源工具和最佳实践的同时，找到了这些问题的答案。在此期间，我们设计并实施了几个系统，保护了数百万新终端用户的数据和隐私。与此同时，3 名竞争对手遭到黑客攻击。

和世界上其他所有人一样，我的生活在 2020 年初发生了变化。每个头条新闻都与新冠肺炎(COVID-19)有关，突然之间远程工作成为新常态。我认为，公平地说，每个人都有自己特有的应对流行病的方式；对我来说，这百无聊赖。

写本书让我一举两得。首先，在一年的疫情大流行封锁期间，这是一种避免无聊的很好方式。作为硅谷居民，这一希望在 2020 年秋天被放大。此时，附近接二连三的野火污染了加州大部分地区的空气，使许多居民只能待在家里。

其次，也是更重要的一点，写这本我买不到的书非常令人满意。我可以通过本书为他人创造价值。

我希望本书能帮助你解决现实世界中的许多安全问题。

前　言

我使用 Python 来讲授安全，而不是反过来。换句话说，当阅读本书时，你将学到比 Python 更多的安全知识。这有两个原因：首先，安全很复杂，而 Python 并非如此。其次，编写大量自定义的安全代码不是保护系统的最佳方式；繁重的任务几乎总是应该委托给 Python、库或工具。

本书涵盖初中级安全概念。这些概念用初学者级别的 Python 代码实现，无论是安全还是 Python 的资料都非高级水平。

本书读者对象

本书中的所有示例都模拟了在现实世界中开发和保护系统的挑战。因此，将代码推送到生产环境的程序员将学到最多的知识。本书要求读者具备初级 Python 技能或具有其他主要语言的中级经验。当然，你不一定非要成为一名 Web 开发人员才能从本书中学到知识，但对 Web 的基本了解会让你更容易理解本书的后半部分。

也许你不构建或维护系统，而是需要测试它们。如果是这样，你会对要测试的内容有更深入的理解，但我甚至不会尝试教你如何测试。如你所知，这是两套不同的技能。

与一些关于安全的书籍不同，这里的示例都非从攻击者的角度分析问题。因此，这帮人学到的知识最少。如果说对他们有什么安慰，那就是在个别章节里，他们可能真会学到有用的内容。

本书的编排方式

本书分为 3 部分。

第Ⅰ部分“密码学基础”用几个密码学概念奠定基础。这些概念在第Ⅱ部分和第Ⅲ部分中会反复出现。

- 第 1 章简要介绍安全标准、最佳实践和基本原则，进而设定预期。
- 第 2 章直接介绍使用散列和数据完整性的密码学。在此过程中，介绍在本书中出现的一小群人物。
- 第 3 章介绍使用密钥生成和密钥散列进行数据身份验证。
- 第 4 章涵盖任何安全书籍的两个必备主题：对称加密和机密性。
- 第 5 章涵盖非对称加密、数字签名和不可否认性。
- 第 6 章结合前面几章中的许多主要思想，形成了泛在网络协议，即传输层安全(Transport Layer Security)。

第Ⅱ部分“身份验证和授权”包含本书中最具商业价值的内容，特点是提供大量与安全相关的常见用户工作流程的实际操作说明。

- 第 7 章介绍 HTTP 会话管理和 cookie，为后续章节中讨论的许多攻击做好准备。
- 第 8 章是关于身份的内容，介绍用户注册和用户身份验证的工作流程。
- 第 9 章介绍密码管理，也是最有趣的一章。内容在很大程度上建立在前几章的基础之上。
- 第 10 章通过另一个关于权限和组的工作流程从身份验证过渡到授权。
- 第 11 章以 OAuth 作为第Ⅱ部分的结束。OAuth 是一种行业标准授权协议，旨在共享受保护的资源。

第Ⅲ部分“抵御攻击”可以说是本书中最具敌意的部分，但更容易消化，也更令人兴奋。

- 第 12 章深入研究操作系统，主题包括文件系统、外部可执行文件和 shell。

- 第 13 章教你如何使用各种输入验证策略防御大量的注入攻击。
- 第 14 章完全集中在最臭名昭著的注入攻击上，即跨站脚本。你可能已经预见到这一点。
- 第 15 章介绍内容安全策略。在某些方面，这可以被视为关于跨站脚本的附加章节。
- 第 16 章涉及跨站请求伪造。该章将前几章中的几个主题与 REST 最佳实践相结合。
- 第 17 章解释同源策略，以及为什么我们会不时地使用跨源资源共享来放宽这一策略。
- 第 18 章以关于点击劫持的内容和一些使你的技能保持最新的资源结束本书的讨论。

目　录

第 I 部分　密码学基础

第 1 章　纵深防御……3
1.1　攻击面……4
1.2　什么是纵深防御……6
1.2.1　安全标准……7
1.2.2　最佳实践……8
1.2.3　安全基本原则……9
1.3　工具……11
1.4　小结……15

第 2 章　散列……17
2.1　什么是散列函数……17
2.2　原型人物……22
2.3　数据完整性……23
2.4　选择加密散列函数……24
2.4.1　哪些散列函数是安全的……24
2.4.2　哪些散列函数是不安全的……25
2.5　Python 中的加密散列……27
2.6　校验和函数……30
2.7　小结……31

第 3 章　密钥散列……33
3.1　数据身份验证……33
3.1.1　密钥生成……34

3.1.2 什么是密钥散列 37
3.2 HMAC 函数 39
3.3 时序攻击 43
3.4 小结 45

第 4 章 对称加密 47
4.1 什么是加密 47
4.2 cryptography 包 50
4.2.1 危险品层 50
4.2.2 配方层 51
4.2.3 密钥轮换 53
4.3 什么是对称加密 54
4.3.1 分组密码 54
4.3.2 流密码 56
4.3.3 加密模式 57
4.4 小结 61

第 5 章 非对称加密 63
5.1 密钥分发问题 63
5.2 什么是非对称加密 64
5.3 不可否认性 69
5.3.1 数字签名 70
5.3.2 RSA 数字签名 71
5.3.3 RSA 数字签名验证 72
5.3.4 椭圆曲线数字签名 73
5.4 小结 76

第 6 章 传输层安全 77
6.1 SSL、TLS 和 HTTPS 77
6.2 中间人攻击 78
6.3 TLS 握手 80

6.3.1 密码套件协商 80
6.3.2 密钥交换 81
6.3.3 服务器身份验证 84
6.4 Django 与 HTTP 88
6.5 Gunicorn 与 HTTPS 91
6.5.1 自签名公钥证书 92
6.5.2 Strict-Transport-Security 响应头 94
6.5.3 HTTPS 重定向 95
6.6 TLS 和 requests 包 96
6.7 TLS 和数据库连接 97
6.8 TLS 和电子邮件 99
6.8.1 隐式 TLS 99
6.8.2 电子邮件客户端身份验证 100
6.8.3 SMTP 身份验证凭据 100
6.9 小结 101

第 II 部分　身份验证和授权

第 7 章 HTTP 会话管理 105
7.1 什么是 HTTP 会话 105
7.2 HTTP cookie 107
7.2.1 Secure 指令 108
7.2.2 Domain 指令 108
7.2.3 Max-Age 109
7.2.4 浏览器长度的会话 110
7.2.5 以编程方式设置 cookie 110
7.3 会话状态持久化 111
7.3.1 会话序列化程序 111
7.3.2 简单的基于缓存的会话 113
7.3.3 基于直写式缓存的会话 116

7.3.4 基于数据库的会话引擎 116
7.3.5 基于文件的会话引擎 117
7.3.6 基于 cookie 的会话引擎 117
7.4 小结 122

第 8 章 用户身份验证 123
8.1 用户注册 124
8.1.1 模板 128
8.1.2 Bob 注册账户 131
8.2 什么是用户身份验证 132
8.2.1 内置 Django 视图 133
8.2.2 创建一个 Django 应用 134
8.2.3 Bob 登录和注销 137
8.3 简明要求身份验证 139
8.4 测试身份验证 140
8.5 小结 142

第 9 章 用户密码管理 143
9.1 密码变更工作流程 143
9.2 密码存储 149
9.2.1 加盐散列 152
9.2.2 密钥派生函数 154
9.3 配置密码散列 158
9.3.1 原生密码散列器 159
9.3.2 自定义密码散列器 160
9.3.3 Argon2 密码散列 160
9.3.4 迁移密码散列器 161
9.4 密码重置工作流程 166
9.5 小结 169

第 10 章 授权 ····· 171
10.1 应用程序级授权 ····· 172
10.1.1 权限 ····· 173
10.1.2 用户和组管理 ····· 175
10.2 强制授权 ····· 180
10.2.1 低级困难方式 ····· 180
10.2.2 高级简单方式 ····· 183
10.2.3 条件渲染 ····· 185
10.2.4 测试授权 ····· 186
10.3 反模式和最佳实践 ····· 187
10.4 小结 ····· 188

第 11 章 OAuth 2 ····· 189
11.1 给予类型 ····· 190
11.2 Bob 授权 Charlie ····· 195
11.2.1 请求授权 ····· 196
11.2.2 给予授权 ····· 197
11.2.3 令牌交换 ····· 197
11.2.4 访问受保护的资源 ····· 198
11.3 Django OAuth Toolkit ····· 200
11.3.1 授权服务器职责 ····· 201
11.3.2 资源服务器职责 ····· 204
11.4 requests-oauthlib ····· 209
11.5 小结 ····· 213

第III部分 抵御攻击

第 12 章 使用操作系统 ····· 217
12.1 文件系统级授权 ····· 217
12.1.1 请求许可 ····· 218
12.1.2 使用临时文件 ····· 219

12.1.3 使用文件系统权限……220
12.2 调用外部可执行文件……222
12.2.1 用内部 API 绕过 shell……224
12.2.2 使用 subprocess 模块……226
12.3 小结……228

第 13 章 永远不要相信输入……229
13.1 使用 Pipenv 进行包管理……229
13.2 YAML 远程代码执行……232
13.3 XML 实体扩展……235
13.3.1 二次爆炸攻击……235
13.3.2 十亿笑攻击……236
13.4 拒绝服务……238
13.5 Host 标头攻击……239
13.6 开放重定向攻击……243
13.7 SQL 注入……246
13.7.1 原始 SQL 查询……247
13.7.2 数据库连接查询……248
13.8 小结……249

第 14 章 跨站脚本攻击……251
14.1 什么是 XSS……251
14.1.1 持久型 XSS……252
14.1.2 反射型 XSS……253
14.1.3 基于 DOM 的 XSS……254
14.2 输入验证……256
14.3 转义输出……263
14.3.1 内置的渲染实用程序……264
14.3.2 HTML 属性引用……266
14.4 HTTP 响应标头……267

14.4.1 禁用对 cookie 的 JavaScript 访问 ····· 267
14.4.2 禁用 MIME 类型嗅探 ····· 270
14.4.3 X-XSS-Protection 标头 ····· 271
14.5 小结 ····· 272

第 15 章 内容安全策略 ····· 273
15.1 编写内容安全策略 ····· 274
15.1.1 获取指令 ····· 276
15.1.2 导航和文档指令 ····· 280
15.2 使用 django-csp 部署策略 ····· 281
15.3 使用个性化策略 ····· 283
15.4 报告 CSP 违规 ····· 286
15.5 CSP Level 3 ····· 288
15.6 小结 ····· 289

第 16 章 跨站请求伪造 ····· 291
16.1 什么是请求伪造 ····· 291
16.2 会话 ID 管理 ····· 293
16.3 状态管理约定 ····· 295
16.4 Referer 标头验证 ····· 298
16.5 CSRF 令牌 ····· 301
16.5.1 POST 请求 ····· 301
16.5.2 其他不安全的请求方法 ····· 303
16.6 小结 ····· 304

第 17 章 跨源资源共享 ····· 307
17.1 同源策略 ····· 307
17.2 简单 CORS 请求 ····· 309
17.3 带 django-cors-headers 的 CORS ····· 311
17.4 CORS 预检请求 ····· 314
17.4.1 发送预检请求 ····· 314

17.4.2 发送预检响应……318
17.5 跨源发送 cookie……319
17.6 CORS 和 CSRF 抵御……321
17.7 小结……322

第 18 章 点击劫持……323
18.1 X-Frame-Options 标头……326
18.2 Content-Security-Policy 标头……327
18.3 与 Mallory 同步……329
18.4 小结……330

第 I 部分

密码学基础

我们每天都依赖散列、加密和数字签名。在这 3 种技术中，加密技术通常会抢占风头。它在会议、报告厅和主流媒体上得到了更多关注。程序员通常也对学习它更感兴趣。

本书的第 I 部分反复说明了为什么散列和数字签名与加密一样重要。此外，本书的后续部分展示了这 3 种技术的重要性。因此，第 1~6 章本身很有用，也可以帮助你理解后面的许多章节。

第1章

纵深防御

本章主要内容

- 定义攻击面
- 纵深防御简介
- 坚持标准、最佳实践和基本原则
- 确定 Python 安全工具

你现在比以往任何时候都更信任拥有你个人信息的组织。遗憾的是，其中一些组织已经将你的信息泄露给攻击者。如果你觉得这很难相信，请访问 https://haveibeenpwned.com。该网站允许你轻松地搜索包含数十亿受攻击账户的电子邮件地址的数据库。随着时间的推移，这个数据库只会变得更大。作为软件用户，我们通过这种常见的体验培养了对安全的认识。

既然你已经打开了本书，那么我打赌你会因为另一个原因而欣赏安全。和我一样，你不仅希望使用安全系统，还希望创建它们。大多数程序员重视安全，但他们并不总是具备实现安全的背景。我写本书旨在为你提供一个构建这个背景的工具集。

安全是防御攻击的能力。本章从攻击开始由外而内分解安全。随后的章节将介绍在 Python 中从内到外实现防御层所需的工具。

每一次攻击都始于一个入口点。特定系统的所有入口点的总和称为攻击面。在安全系统的攻击面之下是安全层，这是一种被称为纵深防御的架构设计。

防御层遵循标准和最佳实践，以确保安全基础。

1.1　攻击面

信息安全已经从几件应该做和不应该做的事情演变成一个复杂的学科。是什么导致了这种复杂性？安全的复杂源于攻击的复杂;它是出于必要而复杂的。今天的攻击有如此之多的形式和规模，因此在开发安全系统之前，我们必须培养对攻击的认识。

正如我在前面指出的那样，每个攻击都从一个易受攻击的入口点开始，所有潜在入口点的总和就是你的攻击面。每个系统都有一个独特的攻击面。

攻击和攻击面处于稳定的变化状态。随着时间的推移，攻击者会变得更加老练，并且会定期发现新漏洞。因此，保护攻击面是一个永无止境的过程，组织对此过程的投入应该是持续的。

攻击的入口点可以是系统用户、系统本身或两者之间的网络。例如，攻击者可能通过电子邮件或聊天将用户作为某些形式攻击的入口点。这些攻击旨在诱使用户与利用漏洞的恶意内容进行交互。这些攻击如下。

- 反射式跨站脚本(XSS)；
- 社会工程学(例如网络钓鱼、短信诈骗)；
- 跨站请求伪造；
- 开放重定向攻击。

攻击者也可以将系统本身作为入口点。这种形式的攻击通常被设计成利用输入验证不充分的系统。这些攻击的典型示例如下。

- 结构化查询语言(Structured Query Language，SQL)注入；
- 远程代码执行；
- Host 标头攻击；
- 拒绝服务。

攻击者可能会将用户和系统作为攻击的入口点，例如永久跨站脚本或点击劫持。最后，攻击者可能使用用户和系统之间的网络或网络设备作为入口点。

- 中间人攻击；
- 重放攻击。

本书教你如何识别和防御这些攻击，其中一些攻击有整一章的篇幅专门介绍它们(XSS 可以说有两章)。图 1.1 描述了典型软件系统的攻击面。4 名攻击者同时向这个攻击面施压，如虚线所示。尽量不要让细节使你不知所措，这里只是为了让你对预期的内容有一个概括性了解。在本书结束时，你将了解这些攻击的工作方式。

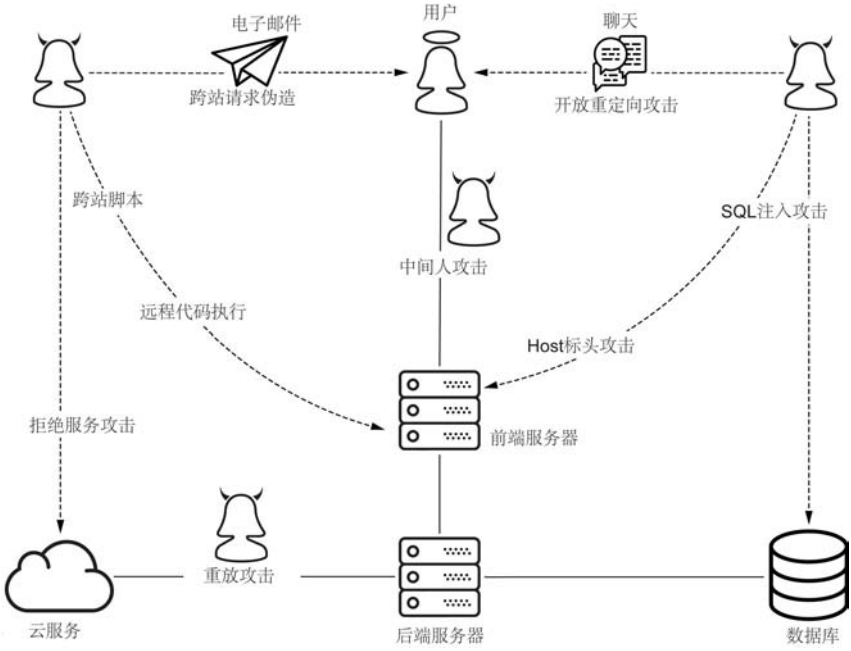

图 1.1　4 名攻击者通过用户、系统和网络同时向一个攻击面施压

在每个安全系统的攻击面之下都有多个防御层，我们不只是保护边界的安全。正如本章开头所述，这种分层的安全方法通常称为纵深防御。

1.2　什么是纵深防御

纵深防御是诞生于美国国家安全局(National Security Agency)内部的一种哲学，它主要认为一个系统应该通过层层安全来应对威胁。每一层安全都有双重用途：既可以防御攻击，又可以在其他层失效时充当备份。我们从不把鸡蛋放在一个篮子里；即使是优秀的程序员也会犯错，而且会定期发现新漏洞。

让我们先从比喻的角度探讨纵深防御。想象一座只有一层防御的城堡，也就是它只有一支军队。这支军队经常保卫城堡不受攻击者的攻击。假设这支军队有 10%的失败概率。尽管军队力量强大，但是国王对目前的风险水平并不满意。你或我会对一个不适合防御 10%的攻击的系统感到舒服吗？我们的用户会对此感到舒服吗？

国王有两个降低风险的选项。一个选项是加强军队建设。这是可能的，但不符合成本效益。消除后 10%的风险比消除前 10%的风险要昂贵得多。国王没有加强军队建设，而是决定在城堡周围修建护城河，以增加另一层防御。

护城河降低了多少风险？在城堡被攻占之前，军队和护城河的防护都必须失效，因此国王用简单的乘法计算风险。如果护城河像军队一样有 10%的失败概率，那么每次攻击都有 10%×10%(即 1%)的成功机会。不过，建立一支有 1%失败概率的军队比仅在地上挖个坑并注满水要昂贵得多。

最后，国王在城堡周围筑起了一堵墙。就像军队和护城河一样，这堵墙有 10%的失败概率。每一次攻击现在都有 10%×10%×10%(即 0.1%)的成功概率。

纵深防御的成本效益分析归结为算术和概率。添加另一层总是比试图完善单层更具成本效益。纵深防御认识到完美是徒劳的；这是一种优势，而不是弱点。

随着时间的推移，防御层的实现会变得比其他实现更成功、更受欢迎；而挖护城河的方法只有这么多。一个共同问题的共同解决方案出现了。安全界开始认识到一种模式，一种新技术从试验性走向标准化。标准团体会评估模式、讨论细节和定义规范，然后安全标准就诞生了。

1.2.1　安全标准

国家标准与技术研究所(National Institute of Standards and Technology，NIST)、互联网工程任务组(Internet Engineering Task Force，IETF)和万维网联盟(World Wide Web Consortium，W3C)等组织已经制定了许多成功的安全标准。通过本书，你将学习如何用以下标准来防护一个系统。

- 高级加密标准(Advanced Encryption Standard，AES)—— 一种对称加密算法；
- 安全散列算法 2(Secure Hash Algorithm 2，SHA-2)—— 一系列加密散列函数；
- 传输层安全(Transport Layer Security，TLS)—— 一种安全网络协议；
- OAuth 2.0—— 一种用于共享受保护资源的授权协议；
- 跨源资源共享(Cross-Origin Resource Sharing，CORS)—— 一种面向浏览器的资源共享协议；
- 内容安全策略(Content Security Policy，CSP)—— 一种基于浏览器的攻击缓解标准。

为什么要标准化？因为安全标准为程序员提供了构建安全系统的公共语言。公共语言允许来自不同组织的不同人员使用不同的工具构建可互操作的安全软件。例如，Web 服务器向每种浏览器提供相同的 TLS 证书；浏览器可以理解来自每种 Web 服务器的 TLS 证书。

此外，标准化促进了代码重用。例如，oauthlib 是 OAuth 标准的通用实现。这个库由 Django OAuth Toolkit 和 flask-oauthlib 包装，允许 Django 和 Flask 应用程序使用它。

不过，标准化并不能神奇地解决所有问题。有时，在每个人都接受标准几十年后才会发现漏洞。2017 年，一组研究人员宣布他们已经破解了 SHA-1(https://shattered.io/)，这是一种加密散列函数，此前已被业界采用了二十多年。有时供应商不会在相同的时间框架内实现标准。每个主流浏览器都花了数年时间才支持某些 CSP 功能。然而，标准化在大多数情况下是有效的，我们不能忽视它。

目前，一些最佳实践已发展为对安全标准的补充。纵深防御本身就是一种

最佳实践。与标准一样，安全系统也遵循最佳实践；但与标准不同，最佳实践没有规范。

1.2.2　最佳实践

最佳实践不是标准团体的产物；相反，它们是由风格、口碑和像本书这样的书籍所定义。这些都是必须要做的事情，有时你只能靠自己。通过阅读本书，你将了解如何识别和执行以下最佳实践。

- 传输中和静态加密；
- 不要使用你自己的密码；
- 最小权限原则。

数据要么在传输中，要么在处理中，要么处于静止状态。当安全专业人士说“传输中和静态加密”时，他们是在建议其他人当数据在计算机之间移动和写入存储时对其进行加密。

当安全专业人员说“不要使用你自己的密码”时，他们是在建议你重用经验丰富的专家的工作成果，而不是尝试自己实现某些东西。依赖工具并不只是为了满足紧迫的最后期限和编写更少的代码而变得流行起来，真正的原因是为了安全。遗憾的是，许多程序员通过艰苦的方式才认识到这一点。你将通过阅读本书来学习它。

最小权限原则(Principle of Least Privilege，PLP)保证用户或系统只被授予履行其职责所需的最低权限。在本书中，PLP 应用于许多主题，如用户授权、OAuth 和 CORS。

图 1.2 说明了典型软件系统的安全标准和最佳实践的安排。

没有哪层防御是灵丹妙药。任何安全标准或最佳实践都不会单独解决每个安全问题。因此，像大多数 Python 应用程序一样，本书的内容包含了许多标准和最佳实践。我们可以把每一章都看作增加一层防御的蓝图。

安全标准和最佳实践看起来可能不同，但实际上，它们只是应用相同基本原则的不同方式。这些基本原则代表了系统安全的原子单元。

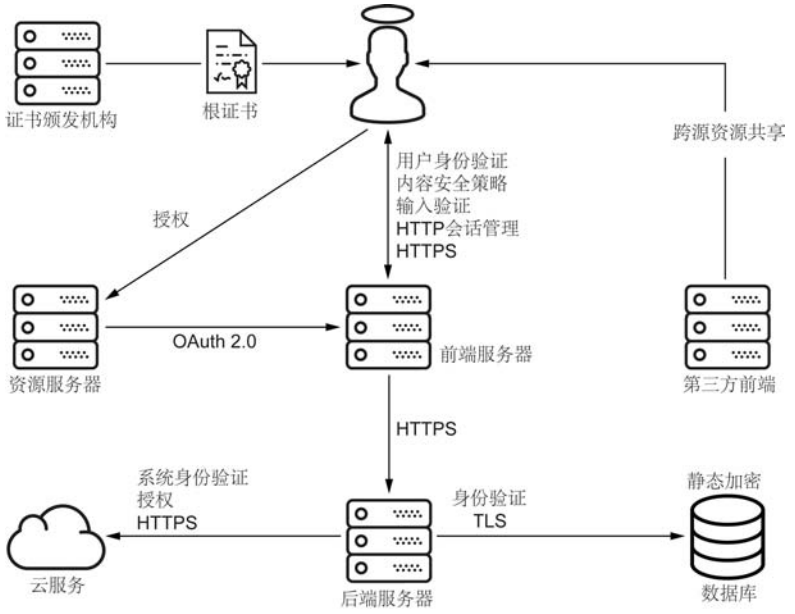

图 1.2　应用于具有安全标准和最佳实践的典型系统的纵深防御

1.2.3　安全基本原则

安全基本原则在安全系统设计中反复出现，并且在本书中同样反复出现。算术和代数或三角学之间的关系类似于安全基本原则和安全标准或最佳实践之间的关系。通过阅读本书，你将学习如何结合以下基本原则来保护系统。

- 数据完整性——数据更改了吗？
- 身份验证——你是谁？
- 数据身份验证——谁创建了此数据？
- 不可否认性——谁做了什么？
- 授权——你能做什么？
- 机密性——谁可以访问此数据？

数据完整性(有时也称为消息完整性)可确保数据不会意外损坏(位损坏)，它回答了这样一个问题“数据更改了吗”。数据完整性可确保数据按写入方式读取。无论数据的作者是谁，数据读取器都可以验证其完整性。

身份验证回答了这样一个问题“你是谁”。我们每天都在从事这项活动；这是验证某人或某物身份的行为。当一个人成功响应用户名和密码质询时，身份将得到验证。不过，身份验证不只适用于人；机器也可以进行身份验证。例如，持续集成服务器在从代码库中提取更改之前进行身份验证。

数据身份验证(通常称为消息身份验证)确保数据读取器可以验证数据写入器的身份。它回答了这样一个问题“谁创建了此数据”。与数据完整性一样，当数据读取器和写入器是不同的一方时以及当数据读取器和写入器相同时，数据身份验证也适用。

不可否认性回答了这样一个问题“谁做了什么”。它是个人或组织无法否认其行为的保证。不可否认性适用于任何活动，但对于在线交易和法律协议至关重要。

授权(有时称为访问控制)通常会与身份验证混淆。这两个术语听起来相似，但代表不同的概念。如前所述，身份验证回答了“你是谁”问题，而授权回答了这样一个问题“你能做什么”。阅读电子表格、发送电子邮件和取消订单都是用户可能被授权或未被授权执行的操作。

机密性回答了这样一个问题“谁可以访问此数据”。这一基本原则确保了两方或多方可以私下交换数据。未经授权的各方不能以任何有意义的方式读取或解释以机密方式传输的信息。

本书教你如何用这些构件块创建解决方案。表 1.1 列出了每个构件块及其映射的解决方案。

表 1.1　安全基本原则

构件块	解决方案
数据完整性	安全网络协议、版本控制、包管理
身份验证	用户身份验证、系统身份验证
数据身份验证	用户注册、用户登录工作流程、密码重置工作流程、用户会话管理
不可否认性	在线交易、数字签名、受信任的第三方
授权	用户授权、系统到系统授权、文件系统访问授权
机密性	加密算法、安全网络协议

安全基本原则是相辅相成的。每一种方法本身都不是很有用，但当它们组合在一起时就非常强大。让我们考虑一些例子。假设电子邮件系统提供数据身

份验证，但不提供数据完整性。作为电子邮件接收者，你可以验证电子邮件发件人的身份(数据验证)，但不能确定电子邮件是否在传输过程中被修改。不是很有用，对吧？如果你无法验证实际数据，那么验证数据写入者的身份又有什么意义呢？

设想有一种无须身份验证即可保证机密性的新奇网络协议。窃听者无法访问你使用此协议发送的信息(机密性)，但你也无法确定要将数据发送给谁。事实上，你可能正在向窃听者发送数据。你上一次想在不知道自己在跟谁说话的情况下和某人进行私人谈话是什么时候？通常，如果想交换敏感信息，你也只会想和你信任的人或物交换。

最后，考虑一个支持授权但不支持身份验证的网上银行系统。这家银行会确保你的资金由自己管理，只是它不会挑战你以首先确定你的身份。在不知道用户是谁的情况下，系统如何授权用户？很明显，我们都不会把钱放在这家银行。

安全基本原则是安全系统设计的最基本构件块。如果反复应用同样的方法，那么我们什么也得不到。取而代之的是，我们必须混合应用它们，以建立防御层。对于每一个防御层，我们都希望将繁重的任务转移到工具上。其中一些工具是 Python 原生的，而其他工具可以通过 Python 包获得。

1.3　工具

本书中的所有示例都是用 Python(准确地说是 3.8 版本)编写的。为什么是 Python？因为 Python 很受欢迎，而且只会越来越受欢迎。

编程语言流行度(PopularitY of Programming Language，PYPL)指数是基于 Google Trends 数据的编程语言流行度的衡量标准。截至 2021 年中期，Python 以 30%的市场份额在 PYPL 指数榜(http://pypl.github.io/PYPL.html)中排名第一。在过去 5 年里，Python 的受欢迎程度超过了任何其他编程语言。

为什么 Python 如此受欢迎？这个问题有很多答案，大多数人似乎都同意两个原因。首先，Python 是一种对初学者友好的编程语言，它很容易学、读和写；

其次，Python 生态系统已呈现爆炸式增长。2017 年，Python 包索引(Python Package Index，PyPI)达到 10 万个软件包；仅用了两年半的时间，这个数字就翻了一番。

我不想写一本只涉及 Python 网络安全的书，因此有些章节还介绍了密码学、密钥生成和操作系统等主题。我将通过几个与安全相关的 Python 模块来探讨这些主题。

- hashlib 模块(https://docs.python.org/3/library/hashlib.html)——Python 对加密散列的回答；
- secrets 模块(https://docs.python.org/3/library/secrets.html)——安全随机数生成；
- hmac 模块(https://docs.python.org/3/library/hmac.html)——基于散列的消息身份验证；
- os 和 subprocess 模块(https://docs.python.org/3/library/os.html 和 https://docs.python.org/3/library/subprocess.html)——通向操作系统的入口。

有些工具有自己专门的章节，有些工具通篇都出现了，还有一些工具只是简单提及。你将或多或少地学到关于以下方面的知识。

- argon2-cffi(https://pypi.org/project/argon2-cffi/)——用于保护密码的函数；
- cryptography(https://pypi.org/project/cryptography/)——用于常见加密函数的 Python 包；
- defusedxml(https://pypi.org/project/defusedxml/)——一种更安全的 XML 解析方法；
- Gunicorn(https://gunicorn.org) ——用 Python 语言编写的 Web 服务器网关接口；
- Pipenv(https://pypi.org/project/pipenv/)——具有许多安全特性的 Python 包管理器；
- requests(https://pypi.org/project/requests/)——易于使用的 HTTP 库；

- requests-oauthlib(https://pypi.org/project/requests-oauthlib/) —— 客户端 OAuth 2.0 实现。

Web 服务器代表了典型攻击面的很大一部分。本书连续有多个章节致力于保护 Web 应用程序。对于这些章节，我不得不问自己一个许多 Python 程序员都很熟悉的问题：Flask 还是 Django？这两个框架都很不错；它们之间的最大区别是极简主义与开箱即用的功能。相对于彼此，Flask 默认为最基本的功能，Django 默认为全功能。

作为一个极简主义者，我喜欢 Flask。遗憾的是，它将极简主义应用于许多安全功能。有了 Flask，你的大部分防御层将被委托给第三方库。

相反，Django 对第三方支持的依赖较少，具有许多默认启用的内置保护。在本书中，我将使用 Django 演示 Web 应用程序的安全。当然，Django 不是万能的；我还使用以下第三方库。

- django-cors-headers(https://pypi.org/project/django-cors-headers/) —— CORS 的服务器端实现；
- django-csp(https://pypi.org/project/django-csp/)——CSP 的服务器端实现；
- Django OAuth Toolkit(https://pypi.org/project/django-oauth-toolkit/)——服务器端 OAuth 2.0 实现；
- django-registration(https://pypi.org/project/django-registration/) —— 用户注册库。

图 1.3 显示了由该工具集组成的栈。在此栈中，Gunicorn 通过 TLS 中继往返于用户的流量。用户输入通过 Django 表单验证、模型验证和对象关系映射(Object Relational Mapping，ORM)进行验证；系统输出通过 HTML 转义进行处理。django-cors-headers 和 django-csp 分别确保使用适当的 CORS 和 CSP 头锁定每个出站响应。hashlib 和 hmac 模块执行散列；cryptography 包执行加密。requests-oauthlib 与 OAuth 资源服务器交互。最后，Pipenv 可保护包库中的漏洞。

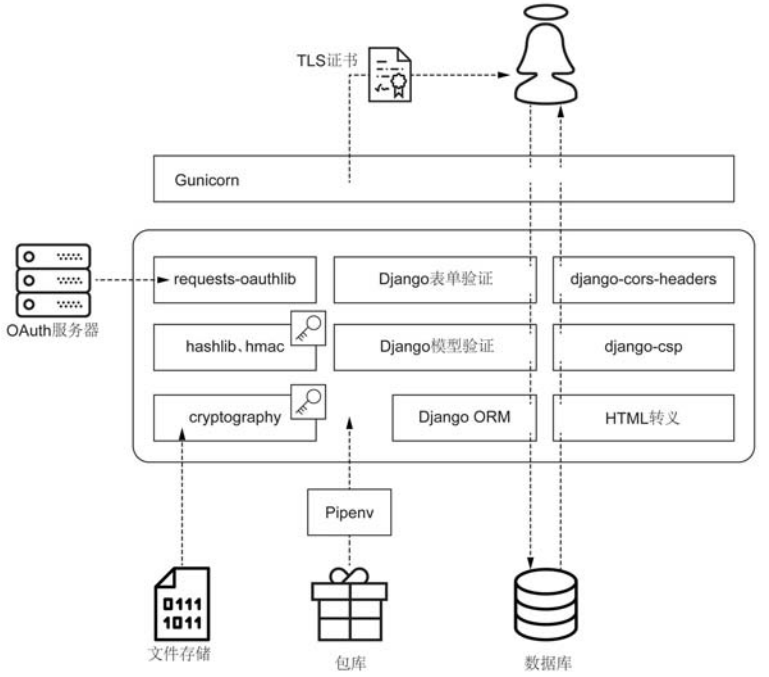

图 1.3　Python 公共组件全栈(在每个层次都能防御某种形式的攻击)

本书对框架和库并不固执己见，它不偏袒其中任何一个。如果你最喜欢的开源框架被另一个框架替代了，请尽量不要把它放在心上。本书中介绍的每个工具都是通过提出两个问题来选择的。

- 工具成熟了吗？我们任何一个人都不应该做的最后一件事是把我们的职业生涯押在一个过去诞生的开源框架上。我故意不介绍尖端工具，因为我们称它为尖端工具是有原因的。根据定义，处于发展阶段的工具不能被视为安全的。因此，本书中的所有工具都是成熟的；这里的所有工具都经过了战斗测试。
- 这个工具受欢迎吗？这个问题更多的是与未来有关。具体而言，读者未来使用该工具的可能性有多大？无论我使用哪种工具来演示一个概念，请记住最重要的是概念本身。

实用性

这是一本现场手册，不是教科书；我优先考虑专业人士而不是学生。这并不是说安全的学术方面不重要，它非常重要，但安全和 Python 是广泛的主题。本书的深度仅限于对目标受众最有用的内容。

在本书中，我介绍了几个用于散列和加密的函数。我不会讨论这些函数背后的繁杂数学。你将了解这些函数是如何运行的，但不会了解这些函数是如何实现的。我将展示如何以及何时使用它们，以及何时不使用它们。

阅读本书会让你成为一名更好的程序员，但仅凭这一点并不能让你成为一名安全专家。没有一本书能做到这一点。不要相信一本做出这种承诺的书。试着阅读本书并编写一个安全的 Python 应用程序，使现有系统更安全。你要满怀信心地将代码推向生产，但不要将你在 LinkedIn 上的个人头衔设置为密码员。

1.4　小结

- 每个攻击都从一个入口点开始，单个系统的这些入口点的总和称为攻击面。
- 攻击的复杂性推动了对纵深防御的需求，这是一种以分层为特征的架构方法。
- 出于互操作性、代码重用和安全的考虑，许多防御层都遵循安全标准和最佳实践。
- 安全标准和最佳实践实质上是应用相同基本原则的不同方式。
- 你应该努力将繁重的任务委托给诸如框架或库的工具；许多程序员已经通过艰苦的方式认识到这一点。
- 阅读本书会让你成为一名更好的程序员，但它不会让你成为密码学专家。