网络空间安全丛书

灰帽渗透测试技术

(第6版)

艾伦·哈珀　瑞安·林　斯蒂芬·西姆斯
[美] 迈克尔·鲍科姆　瓦斯卡尔·特赫达　著
丹尼尔·费尔南德斯　摩西·弗罗斯特

徐　坦　赵超杰　栾　浩
牛承伟　余莉莎　译

上海珪梵科技有限公司　审校

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：01-2023-5811
Allen Harper, Ryan Linn, Stephen Sims, Michael Baucom, Daniel Fernandez, Huáscar Tejeda, Moses Frost.
Gray Hat Hacking: The Ethical Hacker's Handbook, Sixth Edition
978-1-264-26894-8

图书在版编目(CIP)数据
灰帽渗透测试技术 : 第6版 / (美) 艾伦·哈珀等著 ; 徐坦等译. -- 北京 : 清华大学出版社, 2025. 6.
(网络空间安全丛书). -- ISBN 978-7-302-69331-4
Ⅰ. TP393.08
中国国家版本馆CIP数据核字第2025Y2E558号

责任编辑：王　军
封面设计：高娟妮
版式设计：恒复文化
责任校对：成凤进
责任印制：宋　林

出版发行：清华大学出版社
网　　址：https://www.tup.com.cn，https://www.wqxuetang.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：大厂回族自治县彩虹印刷有限公司
经　　销：全国新华书店
开　　本：170mm×240mm　　**印　　张**：40　　**字　　数**：921千字
版　　次：2025年6月第1版　　**印　　次**：2025年6月第1次印刷
定　　价：158.00元

产品编号：097947-01

译 者 序

近年来，随着数字化转型的加速，我国在数字科技领域迅速发展，带来了无数机遇与挑战。数字化转型不仅推动了以数据为核心要素的新质生产力的发展，也使得数字化水平和数据化水平成为提升各类组织竞争力的关键因素。然而，数据的广泛应用及其重要程度也导致了网络和数据安全问题愈发突出，恶意攻击事件频繁发生，严重威胁到个人、组织乃至国家的安全。

在这样的背景下，网络和数据安全技术逐渐成为各界关注的焦点，《灰帽渗透测试技术》系列丛书作为网络与数据安全领域的经典著作之一，在安全行业中拥有举足轻重的地位。《灰帽渗透测试技术》系列丛书与《CISSP信息系统安全专家认证All-in-One》系列丛书作为CISSP考试和CCSP考试的经典教材，以及数字安全人才评价的知识图谱标准而广为流传。在此，《灰帽渗透测试技术(第6版)》的译者序，旨在向高校师生、CISSP和CCSP考生、诸位安全专家、人才选拔与评价专家，以及广大读者介绍本书的主要内容与知识要点。

首先，《灰帽渗透测试技术(第6版)》的内容相较于之前的版本做出了更为全面的补充、更新和重构。本书针对当前网络安全与数据安全态势展开了深入的分析，并全面梳理和讲解了各类网络和数据安全的攻击与防御技术，包括MITRE ATT&CK框架、网络攻击、漏洞利用、逆向工程、指挥与控制(Command and Control，C2)技术、物联网安全、云计算安全、威胁狩猎实验室等各个方面。同时，针对最新的数字安全技术和工具，本书也充实了相应的章节和内容，用以帮助师生、考生和读者能够及时掌握最新的研究成果和实践应用场景。

其次，《灰帽渗透测试技术(第6版)》更为注重理论与实践相结合。在深入剖析各种攻击技术的同时，本书也给出了大量具体的实战案例和实践指导，帮助师生、考生和读者更好地理解攻击方实施攻击的原理和方法，并提供了相应的防护策略和安全控制措施。这部分内容也是CISSP和CCSP认证的重点考查内容。

再次，《灰帽渗透测试技术(第6版)》更加关注学习效果，措辞深入浅出，语言通俗易懂。尽管网络和数据安全攻防技术是一个极其复杂的专业纵深领域，但是，本书借用通俗易懂的语言和生动的案例，帮助师生、考生和读者轻松理解和掌握其中的知识和技巧，无论是初学者，还是具备一定基础的专业人士都能从中受益。本书是数字安全从业人员的重要参考文献。

总之，《灰帽渗透测试技术(第6版)》一书的出版对于提升我国的网络安全与数据安全水平、推动网络与数据安全产业发展具有重要意义。衷心希望本书能够为广大师生、考生和读者提供有益的知识和实用的方法，成为各位师生、考生、读者与安全从业人员在网络安全与数据安全领域的参考工具和学习资料。同时，希望广大师生、考生和读者能够加强对网络安全与数据安全的重视，提高自身的安全意识，共同为我国构建一个安全可信的网络环境。

有鉴于此，清华大学出版社引进并主持翻译了《灰帽渗透测试技术(第6版)》一书。本书

全面深入地探索了灰帽黑客的世界，从历史渊源开始，系统地介绍了灰帽黑客的基本概念和原则；通过案例和技术解析，详细描述了灰帽黑客的工作方法和技术手段；深入探讨了灰帽黑客与网络安全技术和数据安全技术的交融，以及对网络安全与数据安全的影响和启示，全方位地展现了灰帽黑客的多面形象。通过探讨灰帽黑客的道德观和行为准则，分享了在数字安全(覆盖业务安全、数据安全、人工智能安全、网络安全、信息安全等)领域中的最佳实践和建议，引导师生、考生和读者思考如何运用灰帽黑客的知识和经验，帮助社会、组织和个人提升网络安全与数据安全方面的能力水平和防护效果。

本书的翻译工作历时11个月全部完成。翻译过程中译者团队力求忠于原著，尽可能传达作者的原意。在此，感谢参与本书翻译的安全专家，正是有了他们的辛勤付出才有本书的出版。同时，感谢参与本书校对的安全专家，他们保证了本书稿件内容表达的一致性和文字的流畅。感谢栾浩、徐坦、赵超杰、牛承伟、余莉莎在翻译、校对和统稿等工作上投入的大量时间和精力，他们保证了全书在技术上符合网络与数据安全工作实务要求，以及在内容表达上的准确、一致和连贯。

同时，还要感谢本书的审校单位上海珪梵科技有限公司(简称“上海珪梵”)。上海珪梵是一家集数字化转型、数字化软件技术与数字科技风险管理于一体的专业服务机构，专注于数字化、数据化、软件技术与数字安全领域的研究与实践，并提供数字化转型、数字科技建设、数字安全规划与建设、软件研发技术、网络安全技术、数据与数据安全治理、软件项目造价、数据安全审计、信息系统审计、人工智能安全、数字安全与数据安全人才培养与评价等工作。上海珪梵是数据安全人才培养运营中心单位。在本书的译校过程中，上海珪梵投入了多名专家助力本书的译校工作。

在此，一并感谢北京金联融科技有限公司、江西首赞科技有限公司、河北新数科技有限公司、江西立赞科技有限公司在本书译校工作中给予的大力支持。

最后，感谢清华大学出版社和王军等编辑的严格把关，悉心指导，正是有了他们的辛勤努力和付出，才有了本书中文译本的出版发行。

译 者 简 介

栾浩，获得美国天普大学IT审计与网络安全专业理学硕士学位，马来亚威尔士大学(UMW)计算机科学专业博士研究生(人工智能研究方向)，持有CISSP、CISA、注册数据安全审计师(CDSA)、CISP、CISP-A和TOGAF等认证证书，曾供职于思科中国、东方希望集团、华维资产集团、京东集团、包商银行等企业，历任软件研发工程师、安全技术工程师、安全架构师、CTO\CISO职位，现担任CTO职务，并担任中国卫生信息与健康医疗大数据学会信息及应用安全防护分会委员、中国计算机行业协会网络和数据安全产业专家委员会专家、中关村华安关键信息基础设施安全保护联盟团体标准管理委员会委员、数据安全人才之家(DSTH)技术委员会委员，负责数转智改建设、数字安全治理、人工智能安全治理、数据安全治理与审计、个人隐私保护；安全赋能项目的架构咨询、安全技术与运营、IT审计和人才培养等工作。栾浩先生担任本书翻译工作的总技术负责人，承担全书的组稿、翻译、校对和定稿工作。

徐坦，获得河北科技大学理工学院网络工程专业工学学士学位，持有注册数据安全审计师(CDSA)、CDSE、DSTP、CISP-A等认证。现任安全技术总监职务，负责数据安全技术、渗透测试、代码审计、安全教育培训、云计算安全、安全工具研发、IT审计和企业安全攻防等工作。徐坦先生承担本书文前、第1~3、7~14、16~21、23~28和30章的翻译工作，全书的校对、统稿、定稿工作，为本书撰写了译者序，并担任本书的项目经理工作。

赵超杰，获得燕京理工学院计算机科学与技术专业工学学士学位，持有注册数据安全审计师(CDSA)、DSTP-1等认证。现任安全技术经理职务，负责数字化转型过程中的数字科技风险治理、数据安全管理、渗透测试、攻防演平台研发、安全评估与审计、安全教育培训、数据安全课程研发等工作。赵超杰先生承担全书的审校和统稿工作。

牛承伟，获得中南大学工商管理专业管理学硕士学位，持有注册数据安全审计师(CDSA)、CISP等认证。现任广州越秀集团股份有限公司数字化中心技术经理职务，负责云计算、云安全、数据安全、虚拟化运维安全、基础架构和资产安全等工作。牛承伟先生承担全书的通读和校对工作。

余莉莎，获得南昌大学工商管理专业管理学硕士学位，持有注册数据安全审计师(CDSA)、DSTP-1、CISP等认证。负责数字科技风险、数据安全评估、咨询与审计、数字安全人才培养体系等工作。余莉莎女士承担全书的通读和校对工作。

姚凯，获得中欧国际工商学院工商管理专业管理学硕士学位，高级工程师，持有CISSP、CDSA、CCSP、CEH、CISA等认证。现任首席信息官职务，负责数字化转型战略规划与落地实施、数据治理与管理、IT战略规划、策略程序制定、IT架构设计及应用部署、系统取证和应急响应、数据安全、灾难恢复演练及复盘等工作。姚凯先生担任DSTH技术委员会委员。

姚凯先生承担本书部分章节的校对工作。

万雪莲，获得武汉大学计算机技术专业工程硕士学位，持有CISSP、CISM、CISA等认证。现任信息安全专家职务，负责安全管理、安全运营和数据安全等工作。万雪莲女士担任本书第4和第5章的翻译工作。

林科辰，获得湖北理工学院软件工程专业工学学士学位，持有CISP等认证。现任高级工程师职务，负责安全事件处置与应急、DevSecOps、威胁情报和紫队行动等工作。林科辰先生承担本书第6章的翻译工作。

张伟，获得天津财经大学国际贸易专业经济学学士学位，持有CISSP、CISA等认证。现任技术总监职务，负责安全事件处置与应急服务、安全咨询服务、安全建设服务等工作。张伟先生承担本书第15章的翻译工作。

刘玉静，获得北京大学工程管理专业工学硕士学位。现任隐私合规专家职务，负责安全事件处置与应急、数据安全治理、隐私合规方案设计与落地等工作。刘玉静女士承担本书第22章的翻译工作。

刘建平，获得华东师范大学计算机科学专业工学学士学位，持有CISP等认证。现任信息技术运营经理，负责信息技术基础架构、信息技术运维、安全技术实施、安全合规等工作。刘建平先生承担本书第29章的翻译工作。

蒋昭华，获上海立信会计金融学院计算机科学与技术专业工学学士学位，持有 CISSP、CEH等认证。现任信息安全经理，负责安全技术架构设计、事件响应、安全合规等工作。蒋昭华先生承担本书部分章节的校对工作。

马嘉，获得悉尼大学商务大数据专业商学硕士学位，持有CISSP等认证。现任IT风险经理职务，负责信息科技风险管理、云平台等保安全建设和ISO体系维护等工作。马嘉女士承担本书部分章节的校对工作。

伏伟任，获得东华理工大学环境工程专业工学学士学位，持有CISSP、CCSP和CISP等认证。现任IT经理和信息安全负责人职务，负责IT运维、信息安全相关工作。伏伟任先生承担本书部分章节的校对工作。

周可政，获得上海交通大学电子与通信工程专业工学硕士学位，持有CISSP、CCSP和CISA等认证。现任信息安全经理职务，负责数据安全、SIEM平台规划建设和企业安全防护体系建设等工作。周可政先生承担本书部分章节的校对工作。

罗进，获得澳大利亚南昆士兰大学信息技术专业工学硕士学位，持有CISSP等认证。现任中国区信息安全负责人职务，负责企业信息安全架构设计、网络安全规划与建设、数据安全治理、云平台安全，隐私保护与数据跨境安全等工作。罗进先生承担本书部分章节的校对工作。

余礼龙，获得徐州工程学院计算机科学与技术专业工学学士学位，持有CISP等认证。现任实验室负责人职务，负责安全事件处置与应急、云平台安全和安全工具研发等工作。余礼龙先生承担本书部分章节的校对工作。

以下专家参加本书各章节的校对、通读和统稿等工作，在此一并感谢：

罗　春 先生，获得电子科技大学电子工程专业工学学士学位。

刘竞雄 先生，获得长春工业大学计算机技术专业工学硕士学位。

谷陟军 先生，获得中南大学工商管理专业管理学硕士学位。

齐力群 先生，获得北京联合大学机械设计与制造专业工学学士学位。

谢鲲鹏 先生，获得郑州轻工业大学信息安全专业工学学士学位。

梁龙亭 先生，获得北京理工大学计算机科学与技术专业工学学士学位。

刘国强 先生，获得内蒙古工业大学信息与计算科学专业工学学士学位。

曾大宁 先生，获得南京航空航天大学飞行器环境与安全专业工学学士学位。

江 榕 先生，获得伦敦大学信息安全专业工学硕士学位。

本书原文涉猎广泛，内容涉及诸多技术难点，在本书译校过程中，数据安全人才之家(Data Security Talent Home，DSTH)技术委员会、$(ISC)^2$上海分会的诸位安全专家给予了高效且专业的解答，在此，衷心感谢数据安全人才之家(DSTH)技术委员会以及广大会员、$(ISC)^2$上海分会理事会以及广大会员的参与、支持和帮助。

关 于 作 者

Allen Harper博士持有CISSP认证，曾担任海军陆战队军官，并于2007年退役。Allen拥有超过30年的IT和安全经验，在Capella大学荣获信息技术专业的信息安全方向博士学位，在海军研究生学院获得计算机科学专业硕士学位，在北卡罗莱纳州立大学获得计算机工程专业学士学位。2004年，Allen为蜜网(Honeynet)项目研发了GEN III Honeywall CD-ROM，名为roo。从那时起，Allen一直担任许多世界500强企业和政府单位的安全顾问。Allen的兴趣包括物联网、逆向工程、漏洞探查和各种形式的道德黑客活动。Allen是N2NetSecurity, Inc.的创始人，曾担任Tangible Security的执行副总裁和首席黑客，Liberty University的项目主管，现在担任位于 Greenbelt, Maryland的霸王龙有限责任公司(T-Rex Solutions，LLC)的网络安全执行副总裁。

Ryan Linn持有CISSP、CSSLP、 OSCP、OSCE和GREM认证，在安全行业有超过20年的从业经验，他从事过的工作包括系统研发、企业安全到领导一家全球网络安全咨询公司。Ryan曾为许多开源项目做出贡献，包括Metasploit、浏览器漏洞利用框架(BeEF)和Ettercap。Ryan的Twitter ID是@sussurro，他在多个安全主题会议上展示了研究成果，包括Black Hat、DEF CON、Thotcon和Derbycon，并在全球范围内提供安全攻击技术和取证方面的培训。

Stephen Sims是一名在信息安全领域拥有超过15年经验的行业专家，目前在旧金山湾区担任信息安全顾问，Stephen Sims花了多年时间为世界500强的公司提供安全架构、漏洞利用应用程序研发、逆向工程和渗透测试服务，Stephen Sims发现并披露了商业产品中的大量漏洞。Stephen Sims拥有Norwich University信息安全专业硕士学位，目前在SANS研究所负责安全攻击作战课程研发。Stephen Sims是SANS研究所唯一的700级课程SEC760的作者：从事服务于渗透测试人员的高级漏洞利用应用程序研发工作，主要关注复杂的栈溢出、补丁差异和客户端漏洞等方向。Stephen Sims拥有GIAC安全专家(GSE)认证，以及CISA、Immunity NOP等多项认证。在业余时间，Stephen Sims喜欢滑雪和创作歌曲。

Michael Baucom拥有超过25年的行业经验，从嵌入式系统研发到负责Tangible Security公司的产品安全和研究部门。凭借超过15年的安全从业经验，Michael Baucom 评估了多个行业的大量系统的安全水平，包括医疗、工业、网络和消费电子产品。Michael是Black Hat的培训讲师，曾多次在安全会议上发表演讲，同时也是《灰帽渗透测试技术》的作者和技术编辑。Michael Baucom目前致力于嵌入式系统安全和研发。

Huáscar Tejeda是F2TC网络安全公司的CEO(联合创始人兼首席执行官)。Huáscar Tejeda是一名具有丰富网络安全经验的专业人士，在IT和电信领域拥有超过20年的经验和显著成就，为多个宽带提供方研发运营商级的安全解决方案和业务关键组件。Huáscar Tejeda擅长安全研

究、渗透测试、Linux内核破解、软件研发、嵌入式硬件设计。Huáscar Tejeda是SANS拉丁美洲咨询小组、SANS紫队峰会顾问委员会的成员，也是SANS研究所最先进的课程SEC760的特约作者：从事服务于渗透测试人员的高级漏洞利用应用程序研发工作。

Daniel Fernandez是一位拥有超过15年行业经验的安全研究员。在Daniel Fernandez的职业生涯中，他曾发现并利用了大量产品的漏洞。在过去的几年里，Daniel Fernandez将工作重点转移到了虚拟机管理程序(Hypervisor)之上，在云安全领域，Daniel Fernandez发现并报告了Microsoft的Hyper-V等产品的多项漏洞。Daniel Fernandez曾在多个信息安全公司工作，包括Blue Frost Security GmbH and Immunity Inc.。最近，Daniel Fernandez联合创办了TACITO Security公司。在休息时间，Daniel Fernandez喜欢训练工作犬。

Moses Frost 在2000年左右开始安全职业生涯，最初是做大型网络的设计与实施。自20世纪90年代初以来，Moses Frost就以某种形式与计算机打交道。Moses Frost以前的雇主包括TLO、思科系统(Cisco Systems)和McAfee。在思科公司工作期间，Moses Frost是网络防御小组的首席架构师。这个免费的信息安全训练营(Dojo)用于培训高中、大学以及诸多企业的专业人士。在思科，Moses Frost参与关键的安全项目，例如行业认证。Moses是一位作家和高级讲师，Moses的技术兴趣包括Web应用程序渗透测试、云渗透测试和红队攻防。Moses现在是GRIMM的红队负责人员。

免责声明:本书仅代表作者个人观点，不代表美国政府或文中提及的任意公司的观点。

关于贡献人员

Jaime Geiger 目前担任GRIMM网络公司的高级软件漏洞研究工程师职务，是一名SANS认证讲师，他还是一位滑雪、登山、航海和滑板的爱好人士。

关于技术编辑

Heather Linn 历任红队队员，渗透测试工程师，威胁猎人和网络安全策略工程师，在安全行业有超过20年的经验。在Heather Linn的职业生涯中，曾担任渗透测试工程师和数字取证调查人员，并在《财富》50强公司环境中担任高级红队工程师。作为一名卓有成就的技术编辑，Heather还曾为多个安全会议和组织发表演讲，包括Black Hat USA和Girls Who Code，Heather Linn持有多项技术认证证书，包括OSCP、CISSP、GREM、GCFA、GNFA以及CompTIA Penetest+。

前　言

本书由多名安全专家共同创作，致力于遵守道德和负有责任的工作，改善国家、组织和个人的整体安全态势。

致　谢

本书的每一位作者都应感谢McGraw Hill的工作人员。作者团队特别要感谢Wendy Rinaldi和Emily Walters，没有两位编辑的大力支持，作者团队不可能完成本书的出版，编辑们的专业知识、孜孜不倦的奉献精神和对细节的关注帮助这本书获得了成功。感谢McGraw Hill出版社帮助作者团队步入正轨，并且，McGraw Hill极具耐心，与作者团队共同成长。

作者团队还要感谢技术编辑Heather Linn。作为一名技术编辑，Heather Linn在很多方面提升了本书的质量。Heather Linn不知疲倦地运行书中的所有代码，并经常亲自与作者团队一起修复代码的问题。在整个过程中，Heather Linn一直保持着幽默感，鼓励作者团队做到最好。作为一名有成就的作家，Heather Linn努力帮助作者团队做到至善至美。

Allen Harper特别感谢出色的妻子Corann和美丽的女儿们Haley与Madison，感谢家人们在Allen Harper追逐另一个梦想的过程中所给予的支持和理解。每个版本中，作者都能很好地看到家庭成长，Haley和Madison为Allen Harper的生命带来欢乐。Allen Harper为Haley和Madison感到骄傲，也为Haley和Madison的未来感到兴奋。作者团队现在分散开来，因为作者们生活在美国的多个州，感谢在T-Rex(霸王龙)的同事们，让Allen Harper发挥出最好的一面，并激励Allen Harper取得更多的成就。

Ryan Linn十分感谢Heather的支持、鼓励和建议，也感谢家人和朋友们的支持，感谢安全专家们在写书的过程中忍受了漫长的独立工作期。

感谢Jeff、Brian、Luke、Derek、Adrian、Shawn、Rob、Jon、Andrew、Kelly、Debbie和所有帮助Ryan Linn在技术、职业和生活各方面成长的人士。

Stephen Sims非常感谢妻子Leanne和女儿Audrey在研究、写作、工作、教学和旅行所需的时间上给予的持续支持。

Stephen Sims也要感谢父母George 和 Mary，还有妹妹Lisa，感谢家人们的远程支持。最后，特别感谢所有通过出版物、讲座和工具为社区作出贡献的优秀的安全专家们。

Michael Baucom非常感谢妻子Bridget和女儿Tiernan的付出和支持，让Michael Baucom能够追求自己的职业目标。感谢父母的爱和支持，以及父母在Michael Baucom身上培养出的工作态度，帮助Michael Baucom走到今天。

此外，感谢美国海军陆战队给了Michael Baucom勇气和信心，明白一切皆有可能。最后，Michael Baucom要感谢同事Allen Harper。没有一个伟大的团队，任何事情都无法完成。

Huáscar Tejeda非常感谢妻子Zoe和子女Alexander和Axel一直以来的支持和鼓励。Huáscar Tejeda感谢母亲Raysa以身作则地教导了他对热爱学习和努力工作的重要程度的认知，并指导他幼年时就开始接触音乐、绘画和数学。此外，特别感谢哥哥Geovanny，在Huáscar

Tejeda 13岁时，在他学习了强大的计算机编程技能后，哥哥Geovanny邀请他到大学学习计算机科学课程。最后，感谢弟弟Aneudy一直以来的关心和支持。

Daniel Fernandez非常感谢妻子Vanesa的爱和支持。Daniel Fernandez感谢前同事和老朋友Sebastian Fernandez、Gottfrid Svartholm和Bruno Deferrari。最后，特别感谢Rocky，一位多年前曾帮助他获得最好的职业经历机会的前辈。

Moses Frost非常感谢妻子Gina和女儿 Juliet，感谢多年以来的爱、支持和奉献。

Moses Frost感谢父母允许自己去追求梦想。要想挣脱束缚，抓住机会并不容易。最后，感谢他的前同事、导师和朋友——Fernando Martinez、Joey Muniz、Ed Skoudis和 Jonathan Cran，以及许多帮助他的人。

本书的所有参与方和作者团队在此共同感谢Hex-Rays公司的慷慨，允许安全专家们免费使用IDA Pro工具。

最后，特别感谢Jaime Geiger撰写了关于Windows内核漏洞利用的章节。

引　言

没有任何国家能够从长期的战争中受益。

——Sun Tzu

备战是守护和平最有效的手段之一。

——GeorgeWashington

如果这是事实，就不会称为情报了。

——DonaldRumsfeld

与之前的版本一样，本书的目的是向组织和个人提供曾经只有政府和少数黑帽攻击方持有的信息。在每个《灰帽渗透测试技术》版本中，作者团队都努力为组织和个人介绍最新的安全技术内容。越来越多的个人网站处于网络战争破坏的阴云之下，不仅要面对黑帽攻击方，有时还要面对政府。如果安全专家发现自己处于防守方的角色，无论是独自一人还是作为安全团队的防御方，本书都希望组织和个人尽可能多地了解攻击方的攻击技术和手段。为此，作者团队将展示灰帽黑客的心理，灰帽黑客是道德黑客，使用攻击技术达到防御的目的。道德黑客是一个光荣的角色——尊重法律和他人的权力。道德黑客赞同以下观念，即安全专家通过自行测试，进而挫败攻击方的恶意行为。

本书的作者团队希望为组织和个人提供行业和社会普遍需要的内容：全面审查道德黑客活动，以确定灰帽黑客的意图和材料是负责任的和真正道德的。这也是为什么创作团队不断发布本书的新版本，明确定义哪些是道德黑客，哪些不是道德黑客——这是目前社会中区分灰帽黑客非常困惑的事情。

作者团队更新了第5版的材料，并试图提供最全面和最新的技术、工作程序和材料的技术内容，以及实践靶场，组织和个人可根据自身需求自行复制。

本书新增了第18章，其他章也有更新。

第Ⅰ部分将讨论为学习本书其他部分奠定基础所需的主题。请记住，安全专家需要的所有技能在任何一本书籍中都是无法完全涵盖的，但本书试图列出一些主题，帮助新入门的从业人员更容易学习本书。第Ⅰ部分预备知识中涵盖以下主题：

- 灰帽黑客的角色
- MITRE ATT&CK框架
- 掌握C语言、汇编语言和Python语言的基本编程技能

- Linux漏洞利用工具
- Ghidra逆向工程工具
- IDA Pro逆向工程工具

第Ⅱ部分将探讨道德黑客的话题，并介绍安全专家在开展网络攻击和防御时所使用的技能。在第Ⅱ部分涵盖以下主题：

- 红队和紫队
- 指挥与控制(Command and Control，C2)技术
- 在用户的主机和云端构建威胁狩猎实验室
- 威胁狩猎的基础知识

第Ⅲ部分将围绕如何入侵系统展开讲解。在第Ⅲ部分中，组织和个人将掌握入侵Windows和Linux系统所需的技能。第Ⅲ部分是安全专家普遍关注的领域，涵盖了以下主题：

- Linux漏洞利用基础技术
- Linux漏洞利用高级技术
- Linux内核漏洞利用技术
- Windows漏洞利用基础技术
- Windows内核漏洞利用技术
- PowerShell漏洞利用技术
- 无漏洞利用获取shell技术
- 现代Windows环境中的后期漏洞利用技术
- 下一代补丁漏洞利用技术

第Ⅳ部分将介绍入侵物联网(Internet of Things，IoT)和硬件设备。安全专家会从网络安全领域的概述开始，然后步入更高级的主题，包括以下内容。

- 物联网概述
- 剖析嵌入式设备
- 嵌入式设备攻击技术
- 软件定义无线电(SDR)攻击技术

第Ⅴ部分将介绍入侵虚拟机管理程序，第Ⅴ部分提供了软件定义网络(SDN)、存储和虚拟机处理等内容，这些都是目前大多数业务系统的基础。在本节中，将围绕以下主题展开探讨：

- 虚拟机管理程序(Hypervisor)概述
- 创建用于测试虚拟机管理程序的研究框架
- 剖析Hyper-V内核技术
- 入侵虚拟机管理程序的案例研究

第Ⅵ部分将介绍如何入侵云计算平台。除了通常在私有数据中心运行的标准虚拟机管理程序，还描述了公有云、涉及的技术以及此类技术的安全影响。在本节中，将围绕以下主题展开探讨：

- 亚马逊Web服务(Amazon Web Services)攻击技术
- Azure攻击技术

- 容器攻击技术
- Kubernetes攻击技术

作者团队希望组织和个人喜欢新增和更新的章节。本书适合网络安全领域的新手，或者准备进一步推进和深入理解道德黑客行为的读者。无论如何，请牢记永远使用所学的技术去做有益于社会的事情！

拓展阅读和参考文献请扫描封底二维码下载。

注意：请确保个人系统的正确配置，便于正常运行实验环境，此外，本书也提供了运行实验所需的文件。安全专家可以从GitHub存储库下载实验资料和勘误表，存储库地址为：https://github.com/GrayHatHacking/GHHv6。

目　录

第 I 部分

预 备 知 识

第 1 章

灰 帽 黑 客

本章涵盖以下主题：

- 灰帽黑客概述
- 漏洞披露(Vulnerability Disclosure)
- 高级持续威胁(Advanced Persistent Threat，APT)
- 网络杀伤链(KillChain)
- MITRE ATT&CK框架

灰帽黑客(Gray Hat Hacker)是什么？安全专家为什么要关心这个问题？本章将试图定义什么是灰帽黑客，以及灰帽黑客为何对于网络安全(Cybersecurity)领域如此重要。简而言之，灰帽黑客介于白帽黑客和黑帽黑客之间，业界将灰帽黑客称为道德黑客(Ethical Hacker)，灰帽黑客几乎从不违法犯罪，而是通过所掌握的安全技术，帮助世界变得更加和谐。当今时代，灰帽黑客的概念饱受争议，人们可能并不赞同此观点。因此，本章试图澄清事实，并呼吁大家行动起来——共同加入灰帽黑客的行列，以负责的态度进行道德黑客活动。此外，本章也将为本书所讨论的其他关键主题奠定基础。

1.1 灰帽黑客概述

长久以来，“灰帽黑客”这个术语一直饱受争议。对于组织而言，灰帽黑客偶尔可能违反法律法规或者从事一些不道德的活动以达到预期目的。然而，作为真正的灰帽黑客，我们拒绝接受这种观点。本书稍后将介绍灰帽黑客的定义。大部分安全专家都读过各式各样的书籍，进一步混淆了灰帽黑客这一术语的含义，安全专家已经逐渐认识到各类书籍的作者不了解情况，也从未认为自己是灰帽黑客，因为书籍的作者从未真正了解灰帽黑客的本质，因而试图诋毁灰帽黑客群体。因此，作为灰帽黑客主题的创始作者，理应澄清事实。

1.1.1　黑客的历史

长期以来，业界并未将道德黑客视为一种合法职业。曾经有一段时期，任何形式的入侵行为，无论意图如何，业界都将其视为纯粹的犯罪行为。在技术持续发展并深度融入人们生活的背景下，针对黑客攻击行为的法律法规监管合规要求也变得更加全面和深入。对安全专家而言，了解这段发展历程至关重要。正是安全领域先驱者的不懈努力，才让道德黑客这一职业获得社会认可。本章提供的信息不仅是为了告知社会和组织，也是为了提供保护安全专家的能力，促使安全专家能够以合乎道德的方式运用安全技术，进而帮助世界更加和谐。

曾经有一段时间，在网络系统高速发展的历程中，由于立法机构和执法人员的技能与知识较为滞后，很少有法律法规能够适用于管理计算机世界的运转模式。执法滞后导致出于好奇心和恶作剧而攻击系统的恶意攻击方发现了一个充满机遇的新世界。并非所有组织和个人追求好奇心的过程都是无害的。然而，这也意味着组织和个人与那些无法真正理解灰帽黑客领域的权威人士发生冲突，导致世界上大多数软件供应方(Software Vendor)和政府将许多善良、聪明、有才华的安全专家标记为犯罪分子，无论其有何意图。大家能够看到，社会总是对于无法理解的事物感到恐惧，普通大众只是看到了攻击方未经许可而入侵系统，而并不关注攻击方的行为是否存在恶意(https://www.discovermagazine.com/technology/the-story-of-the-414s-the-milwaukee-teenagers-who-became-hacking-pioneers)。

1986年，美国通过了《计算机欺诈和滥用法案》(Computer Fraud and Abuse Act，CFAA)，以加强现有的计算机欺诈法律，CFAA法案明确禁止在未经授权或者超过授权的情况下访问计算机系统，旨在保护关键的政府系统。此后不久，《美国数字千年版权法案》(Digital Millennium Copyright Act，DMCA)于1988年颁布。DMCA法案将攻击访问控制(Access Control，AC)或者数字版权管理(Digital Right Management，DRM)的行为视为犯罪。在这一时代，民众误解了计算机黑客，并恐惧黑客，在这种环境下，对于安全研究人员而言是非常不利的。黑客社区的合法研究人员担心在发现漏洞(Vulnerability)并报告之后，可能触犯法律，甚至入狱。由于代码是受版权保护的，因此，逆向工程(Reverse Engineering)属于非法行为，同时，未经授权访问任意系统(不仅是政府系统)也将导致犯罪(参考Edelman v. N2H2, Feltonetal. v. RIAA和https://klevchen.ece.illinois.edu/ pubs/gsls-ccs17.pdf)。上述情况仍然在某些地区不断发生(https://www.bleepingcomputer.com/news/security/ethical-hacker-exposes-magyar-telekom-vulnerabilities-faces-8-years-in-jail/)。

随着黑客将自己与罪犯区分开来的压力与日俱增，许多研究人员自行定义了一套不会引发法律问题的道德准则，而其他研究人员则质疑法律的寒蝉效应(Chilling Effect)和对于安全研究的整体反应。第一类阵营的人士称之为“白帽黑客”(White Hat Hackers)，白帽黑客通常选择使用较少的细节讨论已知的弱点(Weakness)，从而试图解决问题。在研究过程中，白帽黑客通常也可能选择避开可能对系统造成损害的安全技术，只执行经过充分许可的操作。而任何可能质疑法律公正性的群体，则称之为“黑帽黑客”(Black Hat Hackers)。

然后，第三类群体出现了。那些希望改善安全环境而非造成伤害的黑客们发现，在面对各种限制时，通常无法做出积极的转变，并感到沮丧。哪里有法律要求软件制造方和提供方需要为那些对于消费方造成负面影响的安全决策负责？黑客并未真正停止漏洞探查行动；只是被迫转入地下，而白帽技术由于法律法规监管合规要求的约束，所能探查的漏洞存在一定的局限和限制。对于部分黑客群体而言，并不完全是为了遵守法律法规，但也不是为了追求个人利益或者制造伤害。

“灰帽黑客”这一术语最早由Peiter Zatko(绰号Mudge)在1997年的第一届Black Hat会议中提及[1]，当时，Mudge宣布将开始与Microsoft合作共同解决安全漏洞问题[2]。在同一场活动中，与Mudge同为黑客组织L0pht成员的Weld Pond贴切地谈及：“首先，灰色并不意味着参与或者纵容任何犯罪活动。我们当然不会触犯法律法规。大家都要对自己的行为负责。灰色，意味着人们认识到的世界不再是非黑即白”[3]。后来，在1999年，L0pht在一篇文章中使用了“灰帽黑客”这一术语[4]。(顺便说一下，我们最初决定撰写《灰帽黑客》时，我们使用“grey hat”这一短语，但是，出版方告知“grey”是在英国更为常见的拼写方式，因此，本书决定改用在美国更为常用的“Gray”的拼写方式。)

L0pht组织和安全领域的其他先驱利用其知识引导当权方，包括在国会前作证。这种教育方式有助于改变人们对于黑客行为和安全研究的态度，帮助今天合法从业人员更好地开展工作，提高计算机安全，减少因误解而受到起诉的恐惧。然而，这是一种微妙的平衡，在每一起新案件、每一项新技术和每一位灰帽黑客身上维持平衡的战斗仍将继续。

1. 道德与黑客

读者可能发现术语“道德黑客”(Ethical Hacker)一词在本节和其他章节中反复出现。这个术语有时可能受到质疑，因为道德、伦理和法律在个人、社会群体和政府之间有不同的标准，存在理解上的差异。在大多数情况下，“道德黑客”一词用于区别犯罪行为和合法行为——区分为了更大的利益和支持职业追求而实施攻击的群体和追求个人利益、主动犯罪或者利用攻防技能实施违法活动的群体。关于如何成为道德黑客的指导方针有时甚至会编撰成册，供认证机构以及那些依据行为准则来规范成员行为的计算机安全组织使用。

2. 灰帽黑客的定义

正如大家所见，术语“灰帽黑客”来自早期的认知，这意味着，相比单纯的黑与白，实际上存在更多复杂的“灰色地带”(Shades of Gray)，而不是黑色和白色的极端术语。当然，有关黑帽黑客和白帽黑客的术语源自美国老式西部电视剧的象征意义，西部电视剧中戴白帽的牛仔通常是正义的化身，而戴黑帽的则代表着邪恶。因此，灰帽黑客是介于两者之间的黑客群体。灰帽黑客选择在法律和道德范围内游走，通过研究和运用安全攻防知识以提升组织和个人的技术防御水平，并致力于构建一个更加安全、和谐的世界。

需要明确的是，作为本书的作者团队，无法代表所有的灰帽黑客群体，甚至不认为所有从事灰帽黑客的人士都可能认同本书的定义。然而，当本书展开技术主题的阐述时，本书希望首先描述灰帽黑客从何而来，站在道德黑客的立场上，灰帽黑客的工作是有益的，而不是有害的。许多灰帽黑客(但不是所有人员)使用攻防技术谋生，并对自身的技术实力和职业精神感到自豪。本书希望安全专家也能够采纳这个观点，将全部力量用于对社会有益的方面。黑帽黑客的数量已经足够多了；社会需要更多的灰帽黑客填补空缺，保护他人。如果喜欢本书，希望安全专家能够和灰帽黑客共同澄清关于灰帽黑客群体的困惑。当获悉有人错误地指责灰帽黑客时，请大声反驳。请安全专家站在正义和善良的立场上，批判和反击那些越界的恶意人士。

1.1.2　道德黑客的历史

在本节中，安全专家将概述道德黑客领域的历史，从漏洞披露(Vulnerability Disclosure)的话题开始，之后转移到漏洞赏金。这些内容将为本章后面的主题奠定基础，例如，高级持续威胁(Advanced Persistent Threat，APT)、Lockheed Martin公司的网络杀伤链、MITRE ATT&CK、渗透测试(Penetration Testing)、威胁情报(Threat Intel)、威胁狩猎(Threat Hunting)和安全工程(Security Engineering)。

1.1.3　漏洞披露的历史

软件漏洞(Vulnerability)的历史和软件本身一样久远。简而言之，软件漏洞的定义是由攻击方能够利用的软件设计或者代码编写方面的弱点(Weakness)。安全专家应该注意的是，并非所有的缺陷(Bug)都是漏洞。安全专家可通过采用可利用因素(Exploitability Factor)区分缺陷和漏洞。2015年，Synopsys发布了一份报告，展示了对于100亿行代码的分析结果。研究表明，商业代码每千行代码(Lines of Code，LoC)存在0.61个缺陷，而开源软件每千行代码(LoC)存在0.76个缺陷。然而，同样的研究表明，与诸如OWASP Top 10等行业标准相比，商业代码做得更好[5]。此外，业界已经证明了1%~5%的软件缺陷实际上都是漏洞[6]。由于现代应用程序的LoC计数通常以数十万行代码计算(如果不是数百万的话)，一款典型的应用程序可能有几十个安全漏洞。有一件事是肯定的：只要是人为研发的软件，漏洞就可能存在。此外，只要存在漏洞，用户就可能面临风险。因此，安全专家和研究人员有责任在攻击方利用漏洞入侵用户系统之前，预防(Prevent)、发现(Find)和修复(Fix)软件漏洞。这就是灰帽黑客的最终使命。

在软件漏洞披露过程中，通常可能出现诸多需要考虑的因素。对于攻击方而言，考虑因素包括联系谁、如何联系、提供什么信息，以及如何在披露的过程中明确各方的责任归属。对于供应方而言，考虑因素包括诸如跟踪漏洞报告、执行风险分析、获取修复漏洞所需的正确信息、为漏洞修复工作执行成本和效益分析，以及管理使用方和漏洞报告人员的沟通等。当攻击方和供应方在上述考虑因素上的目标不一致时，可能产生摩擦。关键问题

随之出现，例如，供应方需要多少时间才能修复漏洞？攻击方和供应方是否同意修复漏洞的重要程度？漏洞报告人员是否应该得到补偿或者认可？在攻击方或者供应方发布相关漏洞的细节之前，客户有多少时间能够通过部署补丁的方式确保自身安全水平？披露多少细节是合理的？如果客户未充分理解未部署补丁的危险，那么客户还会部署补丁吗？

上述问题的答案往往是热烈争论的焦点。如果供应方选择不对漏洞采取任何行动，部分研究人员可能难以接受不披露漏洞。面对持续存在的漏洞，消费方所面临的潜在危险可能是令人无法接受的，尤其是当没有其他司法当局权威机构能够追究供应方的安全责任时。然而，即使是致力于安全的供应方，也可能在许多研究人员、预算、产品经理、消费方和投资方的要求下工作，这需要重新平衡进度的优先级，而这不能总是满足所有研究人员的要求。目前，业界还未就是否披露漏洞的问题达成正式的共识。

常见的漏洞披露方法包括完全向供应方披露(Full Vendor Disclosure)、全面公开披露(Full Public Disclosure)和协调披露(Coordinated Disclosure)。本着道德黑客的精神，灰帽黑客倾向于协调披露的概念；然而，本书希望以一种引人注目的方式呈现各种选项，并交由安全研究人员自行决策。

注意：这些术语可能存在争议，有些专家可能更加喜欢将“部分向供应方披露”(Partial Vendor Disclosure)作为一种选择，以处理在保留概念验证(Proof of Concept，POC)代码和其他各方参与披露流程的情况。为了简化问题，在本书中将继续使用上述术语。

1. 完全向供应方披露

大约从2000年开始，部分安全研究人员更倾向于与供应方合作，使用“完全向供应方披露”的做法，即安全研究人员将漏洞全面披露给供应方，而不将漏洞披露给第三方。这在一定程度上是由于供应方愈发开放地接受公众反馈，而不是诉诸法律行动。然而，计算机安全的概念已经开始更加彻底地渗透到供应方领域，这意味着更多的公司开始采用正式的披露渠道。

大多数的披露要求研究人员禁止向公众公开，或者研究人员出于白帽精神而选择不公开披露漏洞细节。然而，由于没有正式的处理漏洞披露报告的流程，也没有外部问责的来源，组织往往要花费相当漫长的时间修补漏洞。民众认为供应方缺乏修补漏洞的动机，反而剥夺了研究人员的权力，有时导致攻击方愈发喜欢全面披露方式。另一方面，软件供应方不仅需要查询新的流程以帮助组织修补漏洞，还需要解决向客户分发和更新补丁方面所遇到的困难。软件供应方在短时间内发布太多的补丁，可能削弱消费方对于产品的信心。而软件供应方拒绝公布漏洞的修复细节可能导致消费方无法修补漏洞。消费方可能面临着环境庞大且复杂的问题，因此，在修补漏洞时可能遇到组织管理问题。在遇到问题的情况下，安全研究人员逆向工程一个补丁并创建一个新的漏洞利用应用程序需要多长时间？这个时间是比所有用户自行保护所需的时间更多还是更少？

2. 全面公开披露

多年来，有无数的杂志、邮件列表和新闻组讨论了各式各样的漏洞，其中包括于1993年创建的臭名昭著的Bugtraq邮件列表。许多披露的信息中皆是为了打响黑客的名声。其他信息的披露则源于安全专家希望看到问题得以解决却缺乏有效的正式沟通渠道而产生的挫败感。有些系统所有方和软件供应方可能根本不理解什么是安全；法律上也并未强制要求组织关注安全风险。然而，多年来，黑客社区抱怨供应方未公平对待或者不尊重安全研究人员。2001年，安全顾问Rain Forest Puppy做出决定，表示仅为供应方提供一周的时间以响应漏洞，否则将全面公开漏洞细节[7]。2002年，臭名昭著的完全披露邮件列表(Full Disclosure Mailing List)诞生了，作为一个运营了十多年的漏洞披露平台，无论是否有供应方通知，研究人员都能够自由地发布漏洞细节[8]。

漏洞披露领域的一些著名创始人，例如Bruce Schneier，则认为全面公开披露策略是取得成效的唯一方法，并声称软件供应方在羞愧难当时，最有可能解决安全漏洞问题[9]。其他创始人，例如Marcus Ranum，则不同意这一观点，认为现在的处境并没有改善，也没有更加安全[10]。同样，在漏洞披露的问题上，诸方几乎没有达成一致意见；孰对孰错，应当由大众自行判断。全面披露的方法也意味着在急于满足任意期限的情况下，供应方可能没有适当地解决实际问题[11]。当然，其他研究人员很快就发现了这种小招数，进而不断重复这个流程。当软件供应方在处理由其他团队所研发的库代码中的漏洞时，可能存在其他困难。例如，当OpenSSL出现心脏出血(Heartbleed)问题时，成千上万的网站、应用程序(Application)和操作系统发行版本都受到了攻击。所有软件研发人员都应当快速掌握信息，并在应用程序中加入Heartbleed库的更新版本。这需要耗费大量时间，而且一些供应方的处理速度比其他供应方更快，在此期间，许多消费方都处于不安全的状态，因为恶意攻击方可能在应用程序发布新版本的几天内利用OpenSSL Heartbleed漏洞发起攻击。

全面公开披露(Full Public Disclosure)的另一个益处是向公众发布警告信息，以帮助组织和个人在修复方案公布之前采取缓解措施。全面公开披露的概念是，基于黑帽黑客可能已经掌握漏洞细节的前提下，促使组织和个人提前做好准备工作，这无疑是一件值得庆幸的成就，并在某种程度上平衡了攻击方和防守方之间的信息差距。

尽管如此，组织和个人可能遭受伤害的问题仍然存在。向公众完全披露漏洞后，组织和个人是更加安全？还是更加危险？要想充分理解这一问题，组织应该认识到攻击方也在自行研究漏洞，攻击方可能发现了某个漏洞，并且在漏洞披露之前就已经在使用漏洞攻击组织和个人了。同样，孰优孰劣，这个问题将留给安全专家自行判断。

3. 协调披露

到目前为止，本章已经讨论了两种极端的披露方式：完全向供应方披露和全面公开披露。现在，安全专家将讨论介于两者之间的一种披露方式：协调披露(Coordinated Disclosure)。

2007年，Microsoft的Mark Miller正式呼吁业界采用“负责任的披露”(Responsible Disclosure)。Miller阐述了许多原因，包括需要给予供应方(例如Microsoft)足够的时间修复问题，包括底层代码，以最大限度地减少补丁数量[12]。Miller提出了一些有力的观点，但是，其他人认为，负责任的披露方式偏向于供应方一侧，如果Microsoft和其他组织并未长期忽视补丁，那么就没有必要在一开始就采用完全披露方式[13]。很快，民众开始争论“负责任的披露”这个名称意味着试图维护供应方的责任，因此是“不负责任的”(Irresponsible)。Microsoft承认了这一论点，随即也改变了自身立场，并在2010年再次呼吁使用术语协调漏洞披露(Coordinated Vulnerability Disclosure，CVD)[14]。与此同时，Google宣布，在披露漏洞细节之前解决安全问题的截止期限为60天，这一说法引起了人们的关注[15]。这一举措似乎是针对Microsoft的，Microsoft有时可能需要60天以上才能修复漏洞。后来，在2014年，Google成立了名为Project Zero的团队，旨在利用90天的宽限期发现并披露安全漏洞[16]。

协调披露的特点是在一段合理时间后，通过披露漏洞迫使供应方承担责任。计算机应急响应小组(Computer Emergency Response Team，CERT)协调中心(CC)成立于1988年，以应对Morris蠕虫病毒，并在近30年的时间内一直作为漏洞和补丁信息的协调方(Facilitator)[17]。CERT/CC在处理漏洞报告时规定了45天的宽限期，在此期间，除非有特殊情况，否则CERT/CC将在45天后发布漏洞数据[18]。安全研究人员可向CERT/CC或者授权实体之一提交漏洞，CERT/CC将处理安全研究人员与供应方的协调，并将在补丁可用或者45天宽限期后发布漏洞。有关美国国土安全部(DHS)网络安全和基础架构安全局对于协调漏洞披露的立场的信息，请扫描封底二维码下载“拓展阅读”部分作为参考。

4. 不再免费提供漏洞

截至目前，本章已经讨论了完全向供应方披露、全面公开披露和负责任披露(协调披露)方式。所有的漏洞披露方式都是免费的，安全研究人员花费大量时间寻找安全漏洞，不是为了获取经济补偿，而是为了公共利益而披露漏洞。实际上，在这种情况下，研究人员很难获得报酬，而且常常可能被人误解为研究人员在敲诈供应方。

2009年，情况发生了变化。在CanSecWest年度大会上，三位著名黑客Charlie Miller、Dino Dai Zovi和Alex Sotirov表明了立场[19]。在Miller、Dai Zovi和Sotirov主导的一次演讲中，他们举着一个纸板标牌，上面写着“NO MORE FREE BUGS”(不再免费提供漏洞)。此前，研究人员就已经开始呼吁。研究人员为了研究和发现漏洞通常需要花费漫长的时间，却得不到与之相对应的回报。并非安全领域的所有人员都赞同这一观点，部分人员甚至公开抨击这一想法[20]。其他人员则采取更加务实的方法，指出Miller、Dai Zovi和Sotirov三位研究人员已经建立了足够的“社会资本”(Social Capita)以寻求高额的咨询费，但是，其他人员将继续免费披露漏洞，以建立一定的地位和声望。[21]无论如何，这种新观点在安全领域掀起了一股冲击波。漏洞付费问题对于安全研究人员而言是推动力，而软件供应方则视为惊吓的来源。毫无疑问，在安全领域，天平正在从供应方侧转向研究人员侧。

1.2 漏洞赏金计划

1995年，Netscape通信公司的Jarrett Ridlinghafer首次使用术语“漏洞赏金”(Bug Bounty)一词[22]。与此同时，iDefense(后来由VeriSign收购)和TippingPoint通过充当研究人员和软件供应方之间的中间方，帮助双方完善了漏洞赏金流程，甚至促进了信息沟通和报酬支付事宜。2004年，Mozilla基金会为Firefox设立了漏洞赏金计划[23]。2007年，CanSecWest启动的Pwn2Own竞赛成为安全领域的一个热点，研究人员聚集一堂，通过漏洞挖掘和漏洞利用演示，以获得奖励和现金[24]。后来，Google于2010年启动了漏洞赏金计划，Facebook于2011年启动漏洞赏金计划，Microsoft于2014年启动了Microsoft Online Services漏洞赏金计划[25]。目前，已经有数百家公司设立了漏洞赏金计划。

软件供应方的“漏洞赏金”概念旨在以负责的方式解决漏洞问题。毕竟，在最佳情况下，安全研究人员通过挖掘漏洞为软件供应方节省大量的时间和金钱。另一方面，在最糟糕的情况下，如果不正确地处理安全研究人员的报告，贸然公开地发布漏洞报告，则供应方将不得不花费大量的时间和金钱控制其所导致的危害。因此，业界出现了漏洞赏金这种有趣而脆弱的经济模式，软件供应方和安全研发人员都有兴趣和动力来共同合作。

1.2.1 激励措施

漏洞赏金计划提供多项官方与非官方的激励措施。在早期，激励奖品包括信件、T恤、礼品卡，当然，安全测试人员也可能只是获得炫耀的资本。到了2013年，社区认为Yahoo!的奖品过于简陋而开始抨击Yahoo!，反映漏洞报告获得的回报不应该仅仅是T恤和匿名礼品卡。Yahoo!公司的“漏洞挖掘”总监Ramses Martinez在回复社区的公开信中解释，组织一直在自掏腰包资助漏洞赏金计划。从那时起，Yahoo!为有效报告提供的赏金从150美元提高到15 000美元[26]。从2011年到2014年，Facebook提供了极具特色的“白帽漏洞赏金计划”(White Hat Bug Bounty Program)VISA借记卡[27]。可充值的黑卡令人垂涎；安全人员在参加安全会议时出示“白帽漏洞赏金计划”VISA借记卡，能够获得认可，并可能受邀参加晚会[28]。现在，漏洞赏金计划仍然在提供奖励，包括荣誉(研究人员获得排名和认可的分数)、饰物和经济补偿。

1.2.2 围绕漏洞赏金计划所引发的争议

并不是所有相关方都赞成漏洞赏金计划，因为漏洞赏金计划存在一些具有争议的问题。例如，供应方可能使用漏洞赏金平台对安全研究人员进行排名，但是，研究人员无权对供应方进行排名。有些漏洞赏金计划用于收集报告，但是，供应方并未与安全研究人员建立良好的沟通渠道。此外，安全研究人员通常很难确定所谓的“重复发现”(Duplicate)是否真实准确。评分系统也可能是随意设定的，并且鉴于报告在黑市上的价值，可能无法准确

反映漏洞披露的价值。因此，每位研究人员都需要确定漏洞赏金计划是否适合自己，并权衡利弊。

1.3 了解敌人：黑帽黑客

中国古代著名的将军孙子早在2500多年前曾说过：“知己知彼，百战不殆。不知彼而知己，一胜一负。[29]”

基于这一永恒的建议，安全专家有必要充分了解敌人——黑帽黑客。

1.3.1 高级持续威胁

在讨论这个话题之前，本书认同一点，那就是并非所有的黑帽黑客都从事高级持续威胁(Advanced Persistent Threats，APT)行动，也不能将所有的APT行动都归因于黑帽黑客。此外，随着时间的推移，APT这一术语已经延伸到囊括更加基本的攻击形式，这是不恰当的描述[30]。话虽如此，高级持续威胁已成为对于高级敌对方的贴切描述，用于揭示敌对方的活动，并将注意力(诚然，有时过于关注)集中在敌对方身上。

顾名思义，APT使用高级形式的、本质上是持久化的攻击方式，可能对于受害方组织和个人将构成重大威胁。即便如此，安全专家也必须承认，攻击方通常不会在APT攻击的前端投放0-day漏洞。这有两个原因：第一，0-day漏洞通常难以获得，如果频繁使用，则非常容易泄露，这是因为白帽黑客或者灰帽黑客在发现攻击后将对于攻击行为执行逆向工程操作，然后，将入侵流程“道德地”(Ethically)报告给软件研发人员，制止0-day漏洞利用只是时间问题。其次，入侵受害方通常不需要0-day漏洞。鉴于对攻击方的首要威胁，通常只在绝对必要时才使用0-day漏洞发起攻击，大多数情况下，在企业网络中获得立足点后作为第二次攻击。

1.3.2 Lockheed Martin公司的网络杀伤链

在讨论APT的持久化时，Lockheed Martin公司在2011年研发了一套名为网络杀伤链(Cyber Kill Chain)的模型，用以展现补救成本，如图1-1所示。

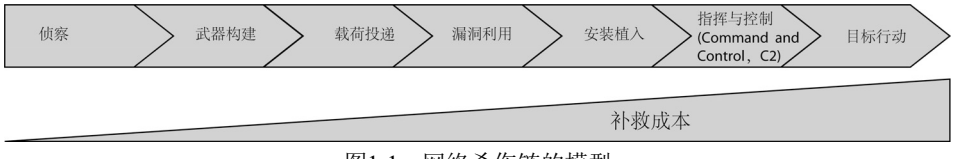

图1-1 网络杀伤链的模型

网络杀伤链模型是通过扩展美国国防部(Department of Defense，DoD)的目标制定原则并利用情报研发而成的，其核心要素是指标(Indicator)。顾名思义，网络杀伤链模型提供了敌对方行为的指标。然后，在首创论文中[31]，Hutchins等专家解释了一种常见的攻击方模式，

如图1-1所示。关键思想是敌对方通常采用重复的攻击流程，如果受害方组织能够及早发现，则能够通过多种方式反制。越早发现攻击的迹象，并“打破”(Break)杀伤链，恢复成本就越低。反之亦然。

下面将讨论网络杀伤链的各个步骤。

1. 侦察

侦察(Reconnaissance)是指攻击方在攻击之前所采取的步骤。通常涉及被动和主动的侦察技术。被动侦察技术(Passive Reconnaissance Techniques)是指在不向攻击目标发送数据包的情况下，通过公共文档、公共资源、搜索引擎和缓存的网络档案间接地收集元数据(Metadata)。另一方面，主动侦察技术(Active Reconnaissance Techniques)涉及与目标网站的交互、开放接口，甚至可能涉及端口和服务、API扫描(枚举)和漏洞扫描。

2. 武器构建

武器构建(Weaponization)包括制造或者选择现有的漏洞，以利用在侦察阶段发现的漏洞。通常，APT在攻击阶段不必做任何花哨的活动或者使用0-day漏洞。在攻击阶段，攻击方通常使用未修补的公开且已知的漏洞。然而，在极少数情况下，攻击方可能自定义有效载荷以执行特殊的利用(Exploit)，其中，可能包含一个特洛伊木马(Trojan)或者其他后门，用于提供指挥与控制(C2)以及进一步的能力。

3. 载荷投递

在载荷投递(Delivery)阶段，攻击方将漏洞利用工具和有效载荷发送给目标，以利用已发现的漏洞。这可能涉及利用已发现的Web漏洞、电子邮件漏洞或者开放的API接口。但是，进入受害方组织系统通常存在更加容易的方法，例如简单的网络钓鱼攻击(Phishing Attack)，经过数十亿美元的安全意识宣贯和培训(Security Awareness and Training，SAT)之后，仍然是有效的。当然，恶意攻击方也可能使用其他形式的社交工程攻击(Social Engineering Attacking，SEA)。

4. 漏洞利用

在漏洞利用(Exploitation)阶段，恶意攻击方启用并执行网络武器，例如，由某一“热心”(Helpful)的用户执行，或者由诸如电子邮件客户端或者网络浏览器插件等类型的应用程序自动执行。此时，目标主机上已经运行了攻击方的恶意代码。当攻击方直接攻击端口或者服务时，载荷投递和漏洞利用阶段是相同的。

5. 安装植入

在安装植入(Installation)阶段，攻击方通常执行两个操作，(1)获得持久化，(2)下载并执行辅助有效载荷(Payload)。在涉及持久化时，对于攻击方而言，最为糟糕的情况是用户关

闭正在运行恶意代码的应用程序，甚至是重启计算机，切断所有连接。因此，攻击方的首要意图是迅速获得某种形式的持久化状态。

辅助的有效载荷通常是必需的环节，这是因为主要有效载荷的体积必须很小，用以规避杀毒软件，并且通常必须适应载体文档或者文件的限制。然而，辅助有效载荷的体积可能更大，能够完全在内存中执行，进一步规避多种反病毒技术。辅助有效载荷可能包含一种标准的和成熟的攻击框架，例如远程访问木马(Remote Access Trojan，RAT)。一些攻击方甚至使用安全研究人员所研发的工具攻击组织或者个人，例如Metasploit平台。

6. 指挥与控制(C2)

在安装辅助有效载荷后，攻击方通常可能执行多种类型的指挥与控制(Command and Control，C2)行动。C2是一个军事短语，攻击方可以根据辅助有效载荷指挥远程访问工具(RAT)或者攻击框架的行动。C2的行为可能是一种简单的通信形式，也许先休眠一天(或者更长时间)，然后唤醒并联系指挥中心，检查所要执行的命令。此外，C2可能利用更加复杂的隧道方案，通过常规流量、自定义加密或者通信协议与攻击方通信。

7. 目标行动

最后，在经过所有这些努力之后，攻击方可能只需要仅仅几秒钟就能完成入侵，敌对方将对目标采取行动，这也是一个军事短语，意思是指，获取目标系统的权限，完成需要执行的任务。这通常涉及跨组织横向移动、发现敏感信息、获得企业管理特权、建立更多形式的持久化状态和访问权限，并最终泄露敏感数据、通过勒索软件勒索、比特币挖矿或者其他获利动机。

1.3.3　网络杀伤链的行动路线

在网络杀伤链的每个阶段，都有处理主动攻击和破坏敌对方的网络杀伤链的方法，如下一步所述。

1. 检测

在每个阶段，组织和安全专家都能够检测到攻击方，但是，在攻击方发起攻击的早期阶段往往更加易于检测。随着攻击方深入了解网络，攻击方开始表现得越来越像普通用户，因此，防守方也越来越难以发现。但是，这里有一个特殊的例外情况，即“欺骗”(Deceive)方法，本书稍后将讨论。

2. 拒绝

抵御攻击方的一种有效方法是“拒绝”(Deny)攻击方访问敏感资源。然而，事实证明，这比听起来更加困难。同样，如果攻击方只是利用已知漏洞攻击，例如，绕过内置的访问控制机制(Access Control Mechanism，ACM)，组织可能无法拒绝攻击方访问系统，特别是

针对面向互联网的系统。但是，对于辅助系统而言，安全专家应该进一步部署网络设置和访问控制措施以阻止攻击方。这种防御的极端形式是零信任(Zero Trust)[a]，零信任正在逐渐流行，如果部署得当，将大大改进防御方法。

3. 扰乱

通过新形式的反病毒或者操作系统更新带来的内存保护等方式能够扰乱攻击方的行为，可增加攻击方实施攻击的成本，例如，数据执行预防(Data Execution Prevention，DEP)技术、地址空间布局随机化(Address Space Layout Randomization，ASLR)和栈预警(Stack Canaries)[b]技术。随着攻击方技术的进化，防御方手段也应随之提升。内存保护技术对于面向外部的系统而言尤其重要，但不能止步于此：组织应该将所有系统和网络的内部分段都视为易受攻击的组件，并采用各种方法扰乱攻击方的攻击行动，从而减缓攻击方的进攻速度，争取宝贵的时间检测攻击方。

4. 降级

降低攻击方的操作或者行为等级意味着限制其攻击成功的能力。例如，组织能够在出站数据超过某个阈值时限制攻击方的数据回传。此外，组织还可以阻止已通过批准和经过身份验证的代理之外的所有出站流量，这能够为组织在发现攻击方并使用代理通道之前争取时间以检测攻击行为。

5. 欺骗

欺骗(Deceive)敌对方的历史和战争本身一样久远。欺骗是网络行动的基本元素之一，对于那些已经突破了所有其他防御措施，但是正在内部网络中潜伏和刺探的攻击方而言，欺骗敌人是最有成效的。欺骗的目的是诱惑攻击方触发组织为探测攻击行为而部署的数字捕鼠器(Digital Mouse Trap)，即蜜罐(Honeypot)。

6. 毁坏

除非组织碰巧为国家级的网络团体工作，否则可能无法“反击黑客”(Hack Back)。然而，当组织发现攻击方已经在网络中站稳脚跟时，组织和安全专家通常能够清除攻击方在组织网络中的立足点(Foothold)。这里要提醒一句：组织需要执行审慎的反击方案，并确保能够将攻击方连根拔起；否则，组织可能进入危险的捉迷藏游戏，从而激怒攻击方，攻击方可能隐藏在组织的网络中——比组织想象的更加深入。

---

a 译者注：零信任技术是安全防御体系中的重要组件，有关零信任技术的内容，请参考清华大学出版社引进并出版的《零信任安全架构设计与实现》一书。

b 译者注：Stack Canaries(取名自地下煤矿的金丝雀预警机制，是指金丝雀能够比矿工更早地发现煤气泄漏，起到预警的作用)是一种用于对抗栈溢出攻击的技术，即ssp安全机制，有时也叫做Stack cookies。Canary的值是栈上的一个随机数，在程序代码启动时随机生成并保存在比函数返回地址更低的位置。由于栈溢出是从低地址向高地址覆盖，因此，攻击方若想控制函数的返回指针，就一定要先覆盖到Canary。程序代码只需要在函数返回前检查Canary是否受到篡改，就可以达到保护栈的目的。

1.3.4 MITRE ATT&CK框架

现在，安全专家已经基本理解了APT和网络杀伤链(Cyber Kill Chain)，下面将讨论MITRE ATT&CK框架。MITRE ATT&CK框架比网络杀伤链更加深入且全面，允许组织了解攻击方的底层战术、技术和工作程序(Tactic, Technique, and Procedure，TTP)，从而，受害方组织和个人有一个更高细粒度的方法在TTP层面挫败攻击方。正如MITRE的威胁情报主管Katie Nickels所说的那样[32]，MITRE框架是“攻击方行为的知识库”。MITRE ATT&CK框架是由顶部的战术组织起来的，安全专家可能将注意到其中包含若干网络杀伤链的步骤，但还有更多其他的环节。然后，在每种战术下展示对应的安全技术，为了表述清晰，已在图1-2中做了删减。

图1-2 MITRE ATT&CK框架

注意：虽然示例工作程序与多项子技术相关联，但是ATT&CK框架并未包含一套全面的工作程序列表，也不打算覆盖全部工作程序列表。有关更多信息，请查看MITRE ATT&CK网站的常见问题解答(FAQ)。

工作程序(Procedure)展示了APT使用的技术变体，并与技术页面相互链接。例如，对于鱼叉式钓鱼附件(Spear Phishing Attachments)技术(T1566.001)，众所周知，APT19组织通过发送RTF和XLSM格式的文件以传递初始攻击。

Lockheed Martin公司经常更新MITRE ATT&CK框架，并提供框架的最新版本。请参见网站上的最新列表[33]。

1. 战术

表1-1提供了ATT&CK版本8的战术列表。

MITRE ATT&CK框架包含了大量有价值的信息，可运用于整体网络安全领域。本章只强调部分内容的用法。

表1-1　MITRE ATT&CK框架的战术列表

战术	描述
侦察	类似于网络杀伤链
资源研发	包括研发攻击所需的基础架构，包括受害账户、研发能力和发起攻击的系统
初始访问	类似于网络杀伤链的投递阶段；描述了在网络中获得初始访问权限的技术
执行	类似于网络杀伤链的漏洞利用阶段
持久化	类似于网络杀伤链的安装植入阶段；描述了在网络中获得权限和保持持久化(Durability)的技术
权限提升	描述攻击方用于提升其在网络中的特权的已知技术
免杀	描述已知的免杀(Evasion)技术。由于免杀战术的技术种类非常丰富，因此攻击方需要花费大量的精力试图规避组织的安全设备检测
凭证访问	描述用于窃取账户名和口令的技术
探查	描述攻击方用于发现网络中的敏感信息和资源的技术。通过蜜罐等骗局(Deception)很容易检测到本阶段和下一阶段的攻击行为
横向移动	描述攻击方用于在组织中移动、攻击和获得访问新系统权限的技术，通常是指通过以前获得的特权访问
收集	描述攻击方用于收集和发布敏感信息的技术
指挥与控制	类似于网络杀伤链的C2阶段；描述用于攻击方和恶意软件之间通信的技术
数据回传	描述了攻击方用于从网络中删除敏感信息的技术。这可能是几个文件，但通常是千兆字节的信息
影响	类似于网络杀伤链的目标行动阶段；描述了攻击方用于操纵、中断或者破坏受害方系统和数据的技术

2. 网络威胁情报

MITRE ATT&CK框架可用于以通用的语言和术语来描述攻击方行为。就其性质而言，网络威胁情报(Cyber Threat Intel)的有效期很短，因此，正确和充分地识别活动(指标)，然后及时分享(传播)信息(情报)至关重要，这样，组织和个人就能够在Internet中查找到指标。MITRE ATT&CK框架能够在全球范围内实现这一目标。值得庆幸的是，MITRE ATT&CK框架已纳入结构化威胁信息表达(Structured Threat Information Expression，STIX)语言，并能够在可信的情报信息自动交换(Trusted Automated Exchange of Intelligence Information，

TAXII)服务器上分发。TAXII服务允许组织和个人实时地获取并使用框架数据。

3. 网络威胁仿真

一旦安全专家掌握了敌对方的行动方式，就能够模拟TTP，并确定以下几点：(1)传感器是否正确配置，以检测到攻击行为；(2)组织的事故持续监测能力是否“警觉”(Awake)，以及攻击响应工作程序是否足够充分和可用。例如，如果安全专家确定APT28对组织构成威胁，可能是APT28对于组织和个人所在的行业感兴趣，那么，安全专家可使用为APT28确定的工作程序并开展一场受控的演习，以评估组织的预防(Prevent)、检测(Detect)和抵御(Withstand)APT28攻击的能力。通过这种方式，网络威胁仿真(Cyber Threat Emulation，CTE)在衡量防御安全功能的有效程度和保持警觉方面是相当成功的。

在这方面的一个有效的工具是红色金丝雀的Atomic Red Team工具。将在第9章中探讨这个工具。

警告：在开展网络威胁演习之前，一定要同上级领导协调。对此，本书已经警告！如果组织已经设立安全运营中心(Security Operations Center，SOC)，则安全人员可能也需要与SOC团队的领导协调，但是，建议不要告知安全运营中心的分析人员正在开展的演习，因为分析人员的响应工作也是测试的组成部分之一。

4. 威胁狩猎

威胁狩猎(Threat hunting)是网络安全领域的一种新趋势，本书将在第9章中详细探讨，但在现阶段，了解其与MITRE ATT&CK框架的联系是大有裨益的。使用MITRE ATT&CK框架，威胁猎人可以通过类似于CTE练习的方式选择一组APT，但在这种情况下，出现多个攻击假设。然后，威胁猎人可能将网络威胁情报与对网络环境的态势感知(Situational Awareness)相互结合，以证明或者驳斥这些假设。大多数安全专家都清楚，最为优秀的防御方是攻击方(也就是灰帽黑客)。现在，有一种能够运用框架中包含的知识库来系统地追踪攻击方的工具，用以捕获入侵的攻击方。

5. 安全工程

作为一名安全工程师，可以研发一套基于MITRE ATT&CK框架的威胁模型。自研威胁模型可使用MITRE ATT&CK导航器开展研发工作(请扫描封底二维码下载“拓展阅读”部分用来参考)。导航器可用于选择一组特定的APT，安全工程师可将其作为电子表格下载。然后，使用电子表格来开展差距评估工作，利用CTE演习的结果，并使用特定技术的颜色标记与特定技术来记录工程师与该APT相关的覆盖水平。最后，自研威胁模型以及相关的覆盖地图可用于设计未来的控制措施，以缩小上述差距。

1.4　总结

本章提供了灰帽黑客(Gray Hat)主题的概述，将灰帽黑客定义为道德黑客(Ethical Hacker)——将攻击技术用于防御目的。从灰帽黑客这一术语的背景和历史开始。随后，讨论了漏洞披露(Vulnerability Disclosure)的历史，以及这是如何与道德黑客相互关联的。最后，将重点转移到黑帽黑客，学习使用MITRE ATT&CK框架讨论、描述、分享和寻找关于黑帽黑客的活动。

第2章

编程必备技能

本章涵盖以下主题：

- C程序设计语言
- 计算机存储器
- Intel处理器
- 汇编语言基础知识
- 运用gdb调试
- Python 编程必备技能

安全专家学习编程的缘由很多，特别是对于道德黑客(Ethical Hacker)而言，道德黑客应该尽可能多地了解编程领域，能够帮助自己更加方便地挖掘程序代码中的漏洞(Vulnerability)，从而在不道德黑客和黑帽黑客利用漏洞之前完成漏洞修复工作。许多安全专业人员从非传统的角度开始接触编程，通常，安全专业人员在开始安全职业生涯之前没有任何编程经验。漏洞挖掘在很大程度上像是一场速度竞赛：如果存在漏洞，谁能够首先发现呢？本章旨在帮助安全专业人员掌握必备的编程技能，以便理解后续章节，从而先于黑帽黑客发现软件漏洞。

2.1 C程序设计语言

C程序设计语言由AT&T贝尔实验室的Dennis Ritchie于1972年研发。C语言在UNIX环境中大量使用，几乎无处不在。事实上，许多主要的网络程序和操作系统以及大型应用程序，如Microsoft Office套件、Adobe Reader和浏览器，都是综合运用C、C++、Objective-C、汇编语言和其他低级语言编写的。

2.1.1 C语言程序代码的基本结构

尽管每套C程序代码都是各不相同的，但是，在大多数程序代码中都存在着一些通用结

构。本书将在接下来的几节中讨论这些结构。

1. main 函数

所有的C程序代码都“应该”(例外情况，请扫描封底二维码下载“拓展阅读”部分参考)包含一个main()函数(小写)，其格式如下所示。

```
<optional return value type> main(<optional argument>) {
  <optional procedure statements or function calls>;
}
```

其中，返回值类型和参数都是可选的。如果没有指定返回值类型，则使用int的返回类型；但是，如果未能将返回值指定为int或者尝试使用void，则有些编译器可能发出警告。如果安全专家在main()函数中使用命令行参数，则可以使用以下格式。

```
<optional return value type> main(int argc, char * argv[]){
```

其中，整型值argc用于保存参数的数量，而argv数组用于保存输入的参数(字符串)。程序代码名称始终存储在偏移量argv[0]处。圆括号和大括号是必填项。大括号用于表示代码块的开始和结束。尽管过程与函数调用(Function Call)属于可选语法结构，但若完全缺失这些调用机制，程序代码将丧失实际执行能力。过程语句(Procedure Statement)实际上是一系列对于数据或者变量执行操作的命令，通常以分号结束。

2. 函数

函数(function)是自包含的代码块，通常通过main()函数或者其他函数调用执行。函数是非持久性的，安全专家能够根据需求多次调用，从而避免在整套程序代码中重复编写类似的代码。其格式如下：

```
<optional return value type> function name (<optional function argument>){
}
```

函数名称和可选参数列表组成了函数签名(Signature)。通过查看函数签名，安全专家能够判断在处理函数中的过程时是否需要添加实参。安全专家还要注意可选的返回值，这决定了函数在执行后是否返回一个值以及返回值的数据类型。

函数调用的格式如下所示：

```
<optional variable to store the returned value> = function name (arguments
if called for by the function signature);
```

下面是一个简单示例：

```
#include <stdio.h>
#include <stdlib.h>
int foo(){❹
```

```
    return 8; ❼
}
int main(void){ ❸
    int val_x; ❺
    val_x = foo();❻
    printf("The value returned is: %d\n", val_x); ❷❽
    exit(0); ❶
}
```

在上述程序代码示例中，引入了合适的头文件，头文件中包括exit和printf的函数声明。exit❶函数定义在stdlib.h中，printf❷函数定义在stdio.h中。如果研发人员不确定程序代码中使用的动态链接函数需要引入哪些头文件，可以查看手册，例如man sscanf，并参考顶部的概要。然后，程序代码中定义了返回值为int类型的main❸函数。我们在括号内的参数位置指定了void❹，表示不允许将参数传递给main函数。安全研发人员创建一个名为val_x的变量，数据类型为int❺。接下来，调用函数foo❻，并将返回值赋值给变量val_x。foo函数返回值为8❼。然后，使用printf函数将值打印到屏幕上，并使用格式化字符串%d，将变量val_x转换为十进制值❽。

函数调用(Function Call)能够修改程序代码的执行流程。当调用函数时，程序代码的执行将暂时跳转至该函数。在调用的函数执行完毕后，控制权返回到调用指令正下方的虚拟内存地址处。在第10章讨论栈操作时，函数调用的流程将更有意义。

3. 变量

变量(Variable)是程序代码中用于存储可能发生变更并用于动态影响应用程序的信息片段。表2-1显示了一些常见的变量类型。

在安全研发人员编译程序代码时，大多数变量都是根据系统预先定义的特定长度分配固定长度的内存。表2-1中的长度是典型值；实际长度可能有所不同。硬件的实现方式决定了具体的变量长度。然而，C语言中使用sizeof()函数以确保编译器分配了正确的内存长度。

通常，在代码块的起始位置定义变量。当编译器遍历代码并构建符号表时，应该在代码中使用变量之前先声明变量。“符号”(Symbol)只是一个名称或者标识符。变量的正式声明方法如下：

```
<variable type> <variable name> <optional initialization starting with "=">;
```

表2-1　变量类型

变量类型	用途	典型大小
整型(int)	存储诸如314或-314的有符号整型值	在64位计算机中为8字节 在32位计算机中为4字节 在16位计算机中为2字节
浮点型(float)	存储诸如-3.234的有符号浮点型值	4字节
双精度型(double)	存储较大的浮点型值	8字节
字符型(char)	存储像d这样的单个字符	1字节

例如，在代码行中输入：

```
int a = 0;
```

上述语句在内存中声明了一个名为a、初始值为0的整型变量(通常是4字节)。

变量声明后，安全研发人员将使用赋值结构更改变量的值。如以下代码所示：

```
x=x+1;
```

以上代码是一条赋值语句，语句中包含变量x，安全研发人员通过+操作符对变量x值予以修改之后，又将新值存入变量x中。以下是一种常见的语句格式：

```
destination = source <with optional operators>
```

其中，destination是存储最终结果的物理位置。

4. printf

C语言通过libc标准库集成了诸多实用功能组件。最常用的一种结构是printf命令，通常用于向屏幕输出结果。printf命令具有两种格式：

```
printf(<string>);
printf(<format string>, <list of variables/values>);
```

第一种格式非常直接，用于在屏幕显示一串简单字符串；而第二种格式则通过借助一种格式化类型提供更强大的灵活性，格式化类型由常见字符和充当占位符的特殊符号组成，占位符的具体值由逗号后面的变量列表提供。常见的格式化符号如表2-2所示。

以下格式化类型允许软件研发人员通过使用printf系列函数指定数据在屏幕的显示方式，以及写入文件或者其他可能的操作方式。例如，假设已知一处变量是浮点型值，如果希望打印变量，并限制变量的显示宽度，可以在浮点型值之前和之后包含其他内容。在这种情况下，软件研发人员通常能够使用Kali执行以下实验中的代码，首先，将shell变更为Bash，然后，使用git clone从GitHub获取代码。

表2-2　printf格式化类型

格式化类型	含义	示例
%n	不打印任何内容	printf("test %n");
%d	十进制值	printf("test %d", 123);
%s	字符串值	printf("test %s", "123");
%x	十六进制值	printf("test %x", 0x123);
%f	浮点值	printf("test %f", 1.308);

实验2-1：格式化字符串

在该实验中，安全研发人员可以下载本章所有实验的代码，并将重点关注格式化字符串，

这将允许软件研发人员根据需求格式化程序代码的输出。

```
┌──(kali㉿kali)-[~]
└─$ bash
┌──(kali㉿kali)-[~]
└─$ git clone https://github.com/GrayHatHacking/GHHv6.git
Cloning into 'GHHv6'...
remote: Enumerating objects: 509, done.
remote: Total 509 (delta 0), reused 0 (delta 0), pack-reused 509
Receiving objects: 100% (509/509), 98.11 MiB | 21.29 MiB/s, done.
Resolving deltas: 100% (158/158), done.
Updating files: 100% (105/105), done.
┌──(kali㉿kali)-[~]
└─$ ls
Desktop    Downloads  GHHv6  Pictures   Templates
Documents  gh6         Music  Public     Videos
┌──(kali㉿kali)-[~]
└─$ cd GHHv6/ch02
┌──(kali㉿kali)-[~/GHHv6/ch02]
```

现在，请观察以下代码。

```
└─$ cat fmt_str.c
#include <stdio.h>

int main(void){
  double x = 23.5644;
  printf("The value of x is %5.2f\n", x);❶
  printf("The value of x is %4.1f\n", x);❷

  return 0;
}
```

在第一个printf函数调用❶处，总宽度为5，小数点后保留两位。在第二个printf函数调用❷处，总宽度为4，小数点后保留一位。

现在，运用gcc编译并运行代码。

```
┌──(kali㉿kali)-[ ~/GHHv6/ch02]
└─$ gcc fmt_str.c -o fmt_str
┌──(kali㉿kali)-[ ~/GHHv6/ch02]
└─$ ./fmt_str
The value of x is 23.56
The value of x is 23.6
```

注意：本章以2020.4 64位Kali Linux为例说明。如果用户正在使用32位的Kali Linux，可能需要更改编译器选项。

5. scanf 命令

scanf命令的功能与printf正好相反，scanf命令通常用于获取用户的输入，具体的命令格式如下。

```
scanf(<format string>, <list of variables/values>);
```

其中，format string可以包含与表2-2所示的printf类似的格式化符号。例如，以下代码将从用户输入处读取一个整型值，并将值存入变量number中。

```
scanf("%d", &number);
```

实际上，&符号用于表示将值存入变量number所指代的内存物理位置；在讨论完“指针”部分后，研发人员将更加清楚地理解此处的含义。目前，安全研发人员只需要知道，在使用scanf命令时应该在变量名前添加&符号。scanf命令足够智能，能够自动转换变量的类型。因此，假如在上述命令中输入一个字符，scanf命令能够自动将字符转换为十进制(ASCII码)值。但是，scanf命令没有对字符串长度执行边界检查，因此可能导致一些安全风险(将在第10章进一步介绍)。

6. strcpy/strncpy 命令

strcpy命令是C语言中最为危险的函数之一，其格式如下。

```
strcpy(<destination>, <source>);
```

strcpy命令的功能是将源字符串(一个以空字符\0结尾的字符序列)的每个字符复制到目标字符串中。如果在执行复制操作时未检查源字符串长度，可能导致非常危险的后果。实际上，此处正在讨论的是内存物理位置覆盖，稍后将对此做进一步描述。一旦源字符串的长度超过为目标字符串分配的空间，则可能产生缓冲区溢出漏洞，进而影响程序代码的执行。strncpy命令相对strcpy命令而言更加安全，命令格式为：

```
strncpy(<destination>, <source>, <width>);
```

<width>字段用于确保仅有一定数目的字符能够从源字符串复制到目标字符串，攻击方能够借助<width>字段获得更大的控制权。width参数应当基于目标缓冲区的长度(如已分配的空间)而定。另一个函数snprintf可控制字符长度并处理错误。总而言之，由于研发人员需要手工处理内存分配的问题，因此，C程序语言在处理字符串时一直饱受争议，在使用C语言时需要特别注意。

警告：使用类似strcpy无边界检查的函数是不安全的。然而，大多数编程教程并没有提及这些函数可能导致的潜在风险。事实上，如果软件研发人员能够简单地正确使用更加安全的替代函数(如snprintf)，那么缓冲区溢出类型的攻击将会因此而减少很多。显而易见的是，目前缓冲区溢出攻击仍是最为常见的攻击方式，因此，能够推断出软件研发人员依然在不断地使用这些危险的函数。含有危险函数的遗

留代码是另一项常见问题。幸运的是，大多数编译器和操作系统支持各种漏洞补救(Exploit Mitigation)保护措施，有助于阻止针对此类漏洞发动的攻击。此外，即便使用了具有边界检查的函数，也可能由于未正确地计算宽度值而受到攻击。

实验2-2：循环

在编程语言中，循环(Loop)用于多次重复遍历一系列命令。常见的两种类型是：for和while循环。

for循环从一个起始值开始计数，不断计算测试值是否满足条件，执行循环体内的语句，然后递增循环变量的值，准备下一次迭代。格式如下：

```
for(<beginning value>; <test value>; <change value>){
   <statement>;
}
```

因此，for循环类似于：

```
for(i=0; i<10; i++){
   printf("%d", i);
}
```

将在同一行(因为没有使用\n)中输出0到9这10个数字，即：0123456789。

对于for循环而言，每次在执行循环体中的语句之前都将优先检查测试值是否满足条件，因此，有可能循环体一次也不会执行。当条件不满足时，程序代码将从循环结束处向后继续执行。

注意：小于号(<)和小于或等于号(<=)的作用是不同的，后者可能导致循环多执行一次，即一直执行到i=10。这一点非常重要，若不注意，则可能导致循环次数存在一次偏差(Off-By-One)错误。此外，请注意，计数是从0开始的。这一点在C语言中非常普遍，需要习惯。

while循环用于重复执行一系列语句，直到满足某个条件为止。下面是一个简单示例：

```
┌──(kali㉿kali)-[~/GHHv6/ch02]
└─$ cat while_ex.c
#include <stdio.h>

int main(void){
  int x = 0;

  while (x<10) {
    printf("x = %d\n", x);
```

```
    x++;
  }
  return 0;
}
┌──(kali㉿kali)-[~/GHHv6/ch02]
└─$ gcc while_ex.c -o while_ex
┌──(kali㉿kali)-[~/GHHv6/ch02]
└─$ ./while_ex
x = 0
x = 1
x = 2
x = 3
x = 4
x = 5
x = 6
x = 7
x = 8
x = 9
```

循环也支持嵌套。

实验2-3：if/else

if/else结构用于在满足某个条件时执行一系列语句；如果不满足条件，则执行可选的else语句块。如果没有else语句块，程序代码流将在if闭大括号(})后继续执行。以下代码是嵌套在for循环中的if/else结构的一个示例：

```
┌──(kali㉿kali)-[~/GHHv6/ch02]
└─$ cat ifelse.c
#include <stdio.h>

int main(void){
  int x = 0;
  while(1){❶
    if (x == 0) {❷
      printf("x = %d\n", x);
      x++;
      continue;
    }
    else {❸
      printf("x != 0\n");
      break;❹
    }
    return 0;
  }
```

```
}

┌──(kali㉿kali)-[~/GHHv6/ch02]
└─$ gcc ifelse.c -o ifelse
┌──(kali㉿kali)-[~/GHHv6/ch02]
└─$ ./ifelse
x = 0
x ≠ 0
```

在上述示例中，使用while❶循环遍历if/else语句。在进入循环之前，安全专家设置变量x为0。因为x等于0，所以满足if语句❷中的条件。安全专家调用printf函数后，x递增1，然后执行continue语句。现在x的值是1，在循环的第二次迭代中，并未满足if语句的条件。因此，执行else语句❸，调用printf函数，然后通过break❹跳出循环。只有一条语句时可以省略大括号。

7. 注释

为提高源代码的可读性且更加易于共享，软件研发人员通常可以在源代码中添加注释。有两种注释方法：//或者/*和*/。//注释表示该行剩余的所有字符都视为注释，当程序代码执行时，不会作为代码执行。/*和*/这对符号表示多行注释。在本例中，/*表示注释的开始，*/表示注释的结束。

2.1.2　程序代码示例

现在准备第一个程序代码实验的练习。

实验2-4：hello.c

下面将先展示包含“//”注释的程序代码，然后讨论程序代码。

```
┌──(kali㉿kali)-[~/GHHv6/ch02]
└─$ cat hello.c
// hello.c                      // customary comment of program name
#include <stdio.h>              // needed for screen printing
int main(){                     // required main function
  printf("Hello haxor!\n");     // simply say hello
}                              // exit program
```

上述的简单程序代码使用了stdio.h库中的printf函数，在屏幕上输出“Hello haxor!”。如果已经掌握程序代码的编译方式，那就编译并运行第一段代码段吧!

实验2-5：meet.c

现在尝试更加复杂的程序代码。下述程序代码将接收输入、存储，然后输出。

```
┌──(kali㉿kali)-[~/GHHv6/ch02]
└─$ cat meet.c
// meet.c
#include <stdio.h>        // needed for screen printing
#include <string.h>       // needed for strcpy
void greeting(char *temp1,char *temp2){ ❷// greeting function to say hello
   char name[400];        // string variable to hold the name
   strcpy(name, temp2);    // copy argument to name with the infamous strcpy
   printf("Hello %s %s\n", temp1, name); ❸// print out the greeting
}
int main(int argc, char * argv[]){❶   // note the format for arguments
   greeting(argv[1], argv[2]); ❷  // call function, pass title & name
   printf("Bye %s %s\n", argv[1], argv[2]); ❹ // say "bye"
}❺                              // exit program
```

上述程序代码中有两个命令行参数❶，并调用greeting()函数❷，用于输出“Hello”和给定的名称，以及回车符❸。当greeting()函数结束时，控制权回到main()函数，打印出“Bye”和输入的名称❹。最后，退出应用程序❺。

2.1.3　使用gcc编译

编译(Compiling)是将人类可读的源代码转换为计算机可执行的二进制文件的过程，帮助计算机执行程序代码。更加具体而言，编译器获取源代码，并将源代码转换为称为目标代码(Object Code)的中间文件。目标代码文件可能包含对其他源代码文件中定义的符号和函数的引用，因此无法直接执行。通过链接(Linking)的流程将每个目标代码文件链接成可执行的二进制文件，以解决符号和函数的引用问题。这里对编译过程的描述做了简化，但给出了这个过程的主要步骤。

在UNIX系统上使用C语言编程时，大多数软件研发人员更加喜欢使用GNU的C编译器(gcc)。gcc在编译时提供了大量选项。最常用的标志如表2-3所示。

表2-3　常用的gcc标志及其说明

选项	描述
–o <filename>	使用指定的文件名保存编译后的二进制文件。默认将输出结果保存为a.out
–S	生成一份包含汇编指令的文件，文件扩展名为.s
–ggdb	生成额外的调试信息，在使用GNU调试器(gdb)时比较有用
–c	编译但不链接。生成一份带有.o扩展名的目标文件

(续表)

选项	描述
–mpreferred-stack-boundary=2	使用DWORD长度的栈编译程序代码，简化了研发人员学习时的调试流程
–fno-stack-protector	禁用栈保护(Stack Protection)，此选项于GCC 4.1版本引入。当研发人员学习缓冲区溢出时(例如，第11章所述)，这个选项非常实用
–z execstack	启用可执行栈(Executable Stack)。当研发人员学习缓冲区溢出时(例如，第11章所述)，这个选项非常实用

实验2-6：编译meet.c

编译meet.c程序代码，软件研发人员可在Kali 2020.4 64位中输入以下内容。

```
┌──(kali㉿kali)-[~/GHHv6/ch02]
└─$ gcc -o meet meet.c
```

然后，要执行新程序代码，可以输入以下内容。

```
┌──(kali㉿kali)-[~/GHHv6/ch02]
└─$ ./meet Leet Haxor
Hello 1337 Haxor
Bye 1337 Haxor
$
```

软件研发人员能够使用各种编译器选项编译本书和其他的程序代码；关于使用gcc的更多信息，请扫描封底二维码下载“拓展阅读”部分作为参考。

2.2　计算机存储器

简而言之，计算机存储器(Computer Memory)是一种具有存储和取回数据能力的电子设备。最小的存储单位是1位(Bit)，在内存中以1或者0表示。4位组成一个半字节(Nibble)，表示从0000到-1111的16个二进制值，在十进制中表示为0到15。将两个半字节，即8位放在一起时，将得到一个字节(byte)，能够表示0到(2^8–1)的值，即0到255的十进制值。将两个字节放在一起时，将得到一个字(Word)，可表示0到(2^{16}–1)的值，即0到65 535的十进制值。同理，将两个字放在一起，将得到一个双字(DWORD)，能够表示0到(2^{32}–1)的值，即0到4 294 967 295的十进制值。将两个双字放在一起，将得到一个QWORD，可表示0到(2^{64}–1)的值，即0到18 446 744 073 709 551 615的十进制值。在64位AMD和Intel处理器的内存寻址方面，只使用低48位，这提供了256TB内存的可寻址能力。上述内容能够在大量的在线资源中找到。

计算机存储器[a]有很多种类型；安全专家通常需要关注随机存取存储器(Random Access Memory，RAM)和寄存器(Register)。寄存器是嵌在处理器内部的特殊形式的内存，在2.3.1节“寄存器”中将对此讨论。

2.2.1　随机存取存储器

在随机存取存储器(RAM)中，数据可在任何时间以随机顺序读取和写入——这就是随机存取(Random Access)名称的由来。然而，RAM属于易失存储器，这意味着当计算机断电关闭时，RAM中的所有数据都将丢失。当讨论基于Intel和AMD的现代产品(x86和x64)时，内存按照32位或者48位寻址，意味着处理器用于选择特定内存地址的地址总线是32位或者48位的位宽。因此，x86处理器的最大寻址内存是4 294 967 295字节或者281 474 976 710 655字节(即256TB)。在x64 64位处理器上，未来可通过增加更多晶体管以扩展寻址范围，现在2^{48}字节的内存长度已能够满足当前系统的需求。

2.2.2　字节序

Danny Cohen在1980年的Internet实验备忘录(Internet Experiment Note，IEN)137“On Holy Wars and a Plea for Peace”中讨论字节序(Endian)时，对Swift的《格列佛游记》总结如下：

格列佛发现当今国王的祖父颁布了一项法令，要求Lilliput的所有公民在打破鸡蛋时必须从较小端开始。因此，那些习惯从较大端打破鸡蛋的公民对此非常生气，于是小端派公民和大端派公民之间爆发了内战。导致大端派公民流亡到附近的岛屿Blefuscu古国避难。[1]

Cohen的论文的重点是描述向内存写入数据时产生的两种不同思路。一些研发人员认为低位字节应该优先写入(称之为“小端序”，低位优先)，另一些研发人员则认为高位字节应该优先写入(称之为“大端序”，高位优先)。两种思路的差异与其使用的硬件有关。例如，基于Intel技术的处理器使用小端序方法，而基于Motorola技术的处理器使用大端序方法。

2.2.3　内存分段

分段(Segmentation)主题的内容足以占用一整章的篇幅论述。然而，分段的基本概念非常简单。每个进程(Process)(简化为正在执行的程序代码)都需要访问进程自身在内存中占有的区域。毕竟，用户不希望进程A覆盖进程B的数据。因此，内存划分成了多个小的分段，并根据实际需要分配给进程。本章稍后将讨论寄存器，寄存器是用于存储和记录进程维护的当前分段的信息。偏移寄存器(Offset Register)用于记录保存的关键数据在分段中的具体位置。分段还描述了进程的虚拟地址空间中的内存布局。段(例如代码段、数据段和栈段)将专门分配到

a 译者注：计算机存储器技术是信息系统体系中的重要组件，有关计算机存储器技术的内容，请参考清华大学出版社引进并出版的《CISSP信息系统安全专家认证All-in-One (第9版)》一书。

进程中虚拟地址空间的不同区域，以避免冲突并能够设置相应的权限。每个正在运行的进程都有各自的虚拟地址空间，空间的大小取决于架构(如32位或者64位)、系统设置和操作系统。默认情况下，基本的32位Windows进程将获得4GB的内存，2GB分配给进程的用户模式(User Mode)端，2GB分配给进程的内核模式(Kernel Mode)端。每个进程只有小部分虚拟空间映射到物理内存，根据架构的不同，可使用内存分页和地址转换等多种方式执行虚拟内存到物理内存的映射。

2.2.4　内存中的程序代码

当进程加载到内存中时，通常被分为多个小段。安全专家通常只需要关注6种主要的段(Section)，下面详细介绍具体内容。

1. .text 段

.text段也称为代码段(Code Segment)，基本上对应于二进制可执行文件中的.text部分，主要包含完成任务所需执行的机器指令。.text段通常标记为可读(Readable)和可执行(Executable)，如果执行写入操作，将导致访问冲突。当首次加载进程时，.text段的长度在运行时就已经固定了。

2. .data 段

.data段用于存储已初始化的全局变量，例如：

```
int a = 0;
```

.data段的长度在运行时也是固定的，应该标记为可读。

3. .bss 段

栈下段(Below Stack Section，.bss)用于存储未初始化的全局变量，例如：

```
int a;
```

.bss段的长度在运行时是固定的。.bss分段应该赋予可读权限(Readable)和可写权限(Writable)，但是，不应该赋予可执行权限(Executable)。

4. 堆段

堆段(Heap Section)用于存储动态分配的变量，所分配的内存空间采用从低地址内存到高地址内存的增长方式。内存分配通过malloc()、realloc()和free()函数控制。例如，声明一个整数并在运行时动态分配内存，可以使用如下代码：

```
int i = malloc (sizeof (int)); // dynamically allocates an integer, contains
                               // the preexisting value of that memory
```

堆段通常应该具有可读写权限，但是，不应该具有可执行权限。否则，控制了进程的攻击方可以在栈和堆等区域轻而易举地执行shellcode。

5. 栈段

栈段(Stack Section)用于记录(递归)函数调用，并且在大多数系统中采用从高地址内存向低地址内存的增长方式。如果进程是多线程的，每个线程都将拥有一个唯一的栈。正如研发人员所看到的那样，栈从高地址内存向低地址内存的增长方式可能导致缓冲区溢出漏洞。这也是局部变量也存在于栈段中的原因。第10章将进一步解释栈段(Stack Segment)[a]。

6. 环境/参数段

环境/参数段(Environment/Arguments Section)用于存储进程在运行时可能需要的系统变量，包括但不限于运行中进程的可访问路径、shell名称和主机名称等变量。环境/参数段是可写的，常用于格式化字符串和缓冲区溢出的漏洞利用方面。此外，命令行参数存储在环境/参数段区域中。内存的各个段将按以下顺序存放。进程的内存空间如图2-1所示。

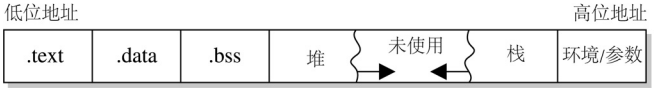

图2-1　进程的内存空间

2.2.5　缓冲区

缓冲区(Buffer)是指用于接收和保存数据，直到进程提取数据的一块存储区域。由于每个进程都有各自的一组缓冲区，因此关键是保持缓冲区的连续不间断；在进程内存的.data段或者.bss段中分配内存实现连续不间断。记住，缓冲区一旦分配，长度就是固定的。缓冲区可保留任意预定义类型的数据；但是，本章将重点关注字符串类型的缓冲区，字符串类型的缓冲区通常用于存储用户输入和基于文本的变量。

2.2.6　内存中的字符串

简单地说，字符串只是内存中连续的字符数组。字符串在内存中通过首个字符的地址引用。字符串以空字符(C语言中的\0)结束。\0是转义序列的一个示例。研发人员使用转义序列指定特殊操作，例如，使用\n指定换行符或者用\r指定回车。反斜杠用于确保后续字符不视为字符串的一部分。如果字段需要显示反斜杠字符，可简单地用转义序列\\，此时将只显示单个\。软件研发人员可从在线文档中找到各种有关转义序列的内容。

a 译者注：Stack Section 和 Stack Segment 是栈段的两种表述形式。

2.2.7　指针

指针(Pointer)是用于存放其他内存地址的特殊内存片段。在内存中移动数据属于相对缓慢的操作。事实证明，移动数据不如通过指针跟踪数据项在内存中的物理位置并更改指针容易。指针通常保存在4字节或者8字节的连续内存中，具体取决于应用程序是32位还是64位的。例如，如前所述，字符串是通过字符数组中首个字符的地址来引用的，首个字符地址值称为指针。C语言中字符串的变量声明如下所示：

```
char * str; // This is read. Give me 4 or 8 bytes called str which is a
            // pointer to a Character variable (the first byte of the
            // array).
```

请注意，虽然指针的大小设置为4字节或者8字节，但是，前面的命令并没有设置字符串的长度；因此，以上字符串数据通常被视为未初始化状态，并存储在进程内存的.bss段中。

再举个例子：如果希望在内存中存储指向某个整数的指针，可在C程序代码中发出以下命令。

```
int * point1; //this is read, give me 4 or 8 bytes called point1, which is a
            //pointer to an integer variable.
```

要读取指针所指向的内存地址的值，需要使用*符号引用指针。因此，如果希望打印以上代码中point1所指向的整数的值，可执行以下命令。

```
printf("%d", *point1);
```

其中符号“*”用于解引用名称为point1的指针，并使用printf()函数打印整数的值。

2.2.8　存储器知识小结

现在，安全专家已经掌握了基础知识，接下来，请观察一项简单的示例，示例将演示在程序代码中关于内存的使用情况。

实验2-7：memory.c

首先，使用cat命令查看程序代码的内容。

```
┌──(kali㉿kali)-[~/GHHv6/ch02]
└─$ cat ./memory.c
#include <stdlib.h>
#include <string.h>
  int _index = 5;       // integer stored in data (initialized)
  char * str;           // string stored in bss (uninitialized)
  int nothing;          // integer stored in bss (uninitialized)
void funct1(int c){ ❸// bracket starts function1 block with argument (c)
  int i=c; ❸                                    // stored in the stack region
  str = (char*) malloc (10 * sizeof (char)); ❹ // Reserves 10 characters in
```

```
                                                    // the heap region */
  strncpy(str, "abcde", 5); ❺  // copies 5 characters "abcde" into str
}                               // end of function1
void main (){❶              // the required main function
  funct1(1); ❷              // main calls function1 with an argument
}❻                          // end of the main function
```

以上程序代码功能较少。首先，在进程内存的不同段中分配几块内存。当程序执行main函数❶时，将调用funct1()函数并传入实际参数1❷，一旦调用funct1()函数，参数将传递给函数变量c❸。然后，在堆上为10字节长度的字符串str❹分配内存。最后，将5字节长度的字符串“abcde”复制到新变量str❺中。funct1函数调用结束，随后，main()函数调用结束。

警告：在继续阅读本书后续内容之前，请务必熟练掌握以上知识。如果需要回顾本章的任何部分，请在继续学习后续内容之前反复复习。

2.3　Intel处理器

目前行业中存在几种常用的计算机架构(Computer Architecture)。本章将关注Intel系列处理器(Processor)或者架构。术语“架构”(Architecture)是指特定制造方实现处理器的方式。当今最常用的架构仍是x86(32位)和x86-64(64位)架构，但是，ARM等其他架构每年也都在发展。每个架构都使用独特的指令集。不同类型的处理器架构无法理解来自另一种类型处理器架构中的指令。

寄存器

寄存器(register)通常用于临时存储数据。安全专家可以将寄存器看作处理器内部使用的8位到64位的快速内存块。寄存器一般划分为4种类别(32位寄存器前缀是E，64位寄存器前缀是R，例如EAX和RAX)。表2-4列出了类别说明。

表2-4　x86和x86-64处理器的寄存器类别

寄存器类别	64位寄存器名称	32位寄存器名称	16位和8位寄存器名称	作用
通用寄存器	RAX，RBX，RCX，RDX，R8-R15	EAX，EBX，ECX，EDX		用于操作数据
			AX，BX，CX，DX	“RAX、RBX、RCX”等32位寄存器的16位版本
			AH，BH，CH，DH，AL，BL，CL，DL	“RAX、RBX、RCX”等32位寄存器的8位高位字节和低位字节

(续表)

寄存器类别	64位寄存器名称	32位寄存器名称	16位和8位寄存器名称	作用
段寄存器			CS，SS，DS，ES，FS，GS	16位寄存器，存放内存地址的前半部分；存放指向代码、栈和额外数据段的指针
偏移寄存器				用于指示与段寄存器相关的偏移量
	RBP(基址指针寄存器)，64位基址指针寄存器的使用取决于帧指针遗漏、语言支持和寄存器R8-R15的使用	EBP		指向栈帧的底部，即函数在栈上的本地环境的起始位置
	RSI(源变址寄存器)	ESI		用于在使用内存块的操作中保存源数据的偏移量
	RDI(目的变址寄存器)	EDI		用于在使用内存块的操作中保存目的数据的偏移量
	RSP(栈指针)	ESP		用于指向栈顶的指针
专用寄存器				仅供CPU使用
	RFLAGS	EFLAGS		CPU用于跟踪逻辑结果和处理器状态。需要了解的关键标志是ZF=零标志，IF=中断允许标志和SF=符号标志
	RIP(指令指针寄存器)	32位：EIP		用于指向要执行的下一条指令的地址

2.4　汇编语言基础

虽然业界已经出版了许多关于汇编(ASM)语言的书籍，但是，本节将帮助安全专家们轻松掌握关于汇编语言的基础知识，从而成为一名更加高效的道德黑客。

2.4.1　机器语言、汇编语言和C语言

计算机只能理解机器语言(Machine Language)，机器语言由1和0组成。而人类大多难以理解由1和0组成的字符序列，因此计算机专家设计了汇编语言，用于帮助软件研发人员记住这一系列数字。后来，研发人员设计出了更加高层次的高级语言，例如C语言，高级语言帮助软件研发人员避免需要直接阅读1和0的窘境。作为一名优秀的道德黑客，必须抵制社会潮流，回归汇编的基本原理。

2.4.2　AT&T与NASM

汇编语法的两种主要形式是AT&T和Intel。GNU汇编器(gas，包含在gcc编译器套件中)使用AT&T语法，Linux研发人员也通常采用GNU汇编器的AT&T语法。采用Intel语法形式的汇编器中，最常使用Netwide汇编器(Netwide Assembler，NASM)。许多Windows汇编器和调试器都采用NASM格式。这两种格式有效地生成了相同的机器语言；但是在风格和格式上存在以下差异。

- 源操作数和目的操作数颠倒，并使用不同的符号标记注释的开始位置。
  - **NASM格式**CMD <dest>, <source><;comment>
  - **AT&T格式**CMD<source>,<dest><#comment>
- AT&T格式在寄存器前面使用%符号，而NASM不需要%符号。%表示“间接操作数”。
- AT&T格式在字面值前面使用$符号，而NASM不需要$符号。$表示“立即操作数”。
- AT&T处理内存引用的方式与NASM不同。

本章采用NASM格式给出每条命令的语法和示例。此外，本节也将给出显示类似命令的AT&T格式示例作为对比。一般而言，所有命令均采用以下格式：

```
<optional label:> <mnemonic>  <operands> <optional comments>
```

操作数(参数“Argument”)的数量取决于命令(助记符“Mnemonic”)。尽管汇编语言中有许多指令，但是只需要精通其中几个即可。接下来详细描述常见的指令。

1. mov 命令

mov命令用于将源操作数复制到目的操作数。而原数据值不会从源位置删除，如图2-2所示。

NASM语法	NASM示例	AT&T示例
mov <dest>, <source>	mov eax, 51h ;comment	movl $51h, %eax #comment

图2-2　mov命令

数据不能直接从内存移动到某个段寄存器中。相反，安全专家必须使用通用寄存器作为中间跳板。如下所示：

```
mov eax, 1234h  ; store the value 1234 (hex) into EAX
mov cs, ax      ; then copy the value of AX into CS.
```

2. add 和 sub 命令

add命令用于将源操作数和目的操作数相加，并将结果存储在目的操作数中。sub命令用于从目的操作数减去源操作数，并将结果存储在目的操作数中，如图2-3所示。

NASM语法	NASM示例	AT&T示例
add <dest>, <source>	add eax, 51h	addl $51h, %eax
sub <dest>, <source>	sub eax, 51h	subl $51h, %eax

图2-3　add和sub命令

3. push 和 pop 命令

push和pop命令分别用于入栈和出栈，如图2-4所示。

NASM语法	NASM示例	AT&T示例
push <value>	push eax	pushl %eax
pop <dest>	pop eax	popl %eax

图2-4　push和pop命令

4. xor 命令

xor命令执行逐位逻辑“异或”(Exclusive Or，XOR)运算，例如，11111111 xor 11111111 = 00000000。因此，XOR value,value命令能够用于将寄存器或者内存位置清空。另一个常用的按位运算符是AND。通过执行逐位AND能够确定是否已设置寄存器或者内存位置中的特定位，或者确定对于malloc等函数调用是否返回指向内存块的指针而非null，如图2-5所示。

这可以通过汇编代码来实现，例如在调用malloc后使用test eax, eax。如果对于malloc的调用返回null，则test操作将把FLAGS寄存器中的零标志(Zero Flag)设置为1。test操作后面的条件跳转指令(如jnz)中所遵循的路径可基于AND操作的结果。以下是汇编语言代码：

```
call malloc(100)
test eax, eax
jnz loc_6362cc012
```

NASM语法	NASM示例	AT&T示例
xor <dest>, <source>	xor eax, eax	xor %eax, %eax

图2-5　xor命令

5. jne、je、jz、jnz 和 jmp 命令

jne、je、jz、jnz和jmp命令根据标志寄存器中的零标志位eflag将程序代码流跳转到另一个位置。如果零标志位等于0，jne/jnz将执行跳转；如果零标志位等于1，则je/jz可能执行跳转；而jmp将始终执行跳转操作，如图2-6所示。

NASM语法	NASM示例	AT&T示例
jnz <dest> / jne <dest>	jne start	jne start
jz <dest> / je <dest>	jz loop	jz loop
jmp <dest>	jmp end	jmp end

图2-6　jne、je、jz、jnz和jmp命令

6. call 和 ret 命令

call指令将执行重定向到另一个函数。首先，将call指令后面的虚拟内存地址压入栈中，作为返回指针，然后执行重定向到被调函数。ret命令用于程序代码末尾处，将控制流返回到call后面的命令，如图2-7所示。

NASM语法	NASM示例	AT&T示例
call <dest>	call subroutine1	call subroutine1
ret	ret	ret

图2-7　call和ret命令

7. inc 和 dec 命令

inc和dec命令用于将目的操作数递增或者递减，如图2-8所示。

NASM语法	NASM示例	AT&T示例
inc <dest>	inc eax	incl %eax
dec <dest>	dec eax	decl %eax

图2-8　inc和dec命令

8. lea 命令

lea命令用于加载源操作数的有效地址到目的操作数中(见图2-9)。这在将目的参数传递给字符串复制函数时经常可以看到；例如，在下面的AT&T语法与gdb反汇编示例中，将目的缓冲区地址写入栈顶，作为gets函数的参数。

```
lea -0x20(%ebp), %eax
mov %eax, (%esp)
call 0x8048608 <gets@plt>
```

NASM语法	NASM示例	AT&T示例
lea <dest>, <source>	lea eax,[dsi +4]	leal 4(%dsi), %eax

图2-9　lea命令

9. 系统调用：int、sysenter 和 syscall 指令

系统调用(System Call)是进程请求执行特权操作的一种机制，通过系统调用机制，上下文和代码执行从用户模式切换到内核模式。传统的x86指令采用int 0x80执行系统调用，现在已不建议使用，但是，32位操作系统仍然支持int 0x80。在32位应用程序中，使用sysenter指令。

在64位Linux操作系统和应用程序中，则需要使用syscall指令。在编写shellcode和其他专用程序代码或者有效载荷时，必须充分理解用于执行系统调用和设置适当参数的各种方法。

2.4.3　寻址模式

在汇编语言中，可能有多种方法能够实现同一个功能。例如，有多种方法能够指示内存中要操作的有效地址。这些可选的方式称为寻址模式，如表2-5所示。请记住，以e开头的寄存器是32位(4字节)，以r开头的寄存器是64位(8字节)。

表2-5　寻址模式

寻址模式	说明	NASM示例
寄存器寻址	寄存器中存放着要操作的数据。不必访问内存。两个寄存器的长度必须一致	mov rbx, rdx add al, ch
立即寻址	源操作数是一个数值。默认为十进制，使用h表示十六进制	mov eax, 1234h mov dx, 301
直接寻址	第1个操作数是要操作的内存地址，使用方括号标出	mov bh, 100 mov[4321h], bh
寄存器间接寻址	第1个操作数是方括号中的一个寄存器，存放着要操作的地址	mov [di], ecx
寄存器相对寻址	要操作的有效地址通过使用ebx/ebp加上一个偏移值计算出	mov edx, 20[ebx]
基址加变址寻址	与寄存器相对寻址一样，但使用edi和esi存放偏移量	mov ecx,20[esi]
相对基址加变址寻址	通过组合寄存器相对寻址和基址加变址寻址来计算有效地址	mov ax, [bx][si]+1

2.4.4　汇编文件结构

汇编源文件通常由以下几个段组成。

- .model：.model指令用于指示.data段和.text段的长度。
- .stack：.stack指令标记栈段的起始位置，并用于指示栈的长度(以字节为单位)。
- .data：.data指令标记数据段的起始位置，并用于定义变量(包括已经初始化的和未经初始化的)。
- .text：.text指令包含用于存放程序代码的命令。

实验2-8：简单汇编程序代码

下面的64位汇编程序代码可在屏幕上打印字符串“Hello, haxor!”。

```
┌──(kali㉿kali)-[~/GHHv6/ch02]
└─$ cat ./hello.asm
section .data                   ; section declaration
msg  db "Hello, haxor!",0xa     ; our string with a carriage return
len  equ  $ - msg               ; length of our string, $ means here
section .text                   ; mandatory section declaration
                                ; export the entry point to the ELF linker or
   global _start                ; loaders conventionally recognize
                                ; _start as their entry point
_start:

                               ; now, write our string to stdout
                               ; notice how arguments are loaded in reverse
   mov    edx,len              ; third argument (message length)
   mov    ecx,msg              ; second argument (ptr to message to write)
   mov    ebx,1                ; load first argument (file handle (stdout))
   mov    eax,4                ; system call number (4=sys_write)
   int    0x80                 ; call kernel interrupt and exit
   mov    ebx,0                ; load first syscall argument (exit code)
   mov    eax,1                ; system call number (1=sys_exit)
   int    0x80                 ; call kernel interrupt and exit
```

汇编过程(Assembling)的第一步是将汇编文件转换为目标代码(32位示例)。

```
┌──(kali㉿kali)-[~/GHHv6/ch02]
└─$ nasm -felf64 hello.asm
```

接下来，调用链接器以创建可执行应用程序。

```
┌──(kali㉿kali)-[~/GHHv6/ch02]
└─$ ld -s -o hello hello.o
```

最后，运行可执行应用程序。

```
┌──(kali㉿kali)-[~/GHHv6/ch02]
└─$ ./hello
Hello, haxor!
```

2.5　运用gdb调试

在UNIX系统上使用C语言编程时，首选调试器为gdb。gdb提供了强大的命令行接口，允许软件研发人员在运行程序代码的同时保留全部控制权。例如，软件研发人员可在程序代码执行过程中设置断点，从而在所希望的任何位置监测内存或者寄存器的内容。因此，对于软件研发人员和攻击方而言，像gdb这样的调试器当属无价之宝。如果希望在Linux上获得更多的图形化调试体验，可采用ddd和edb等方案或者扩展。

gdb基础

gdb的常用命令及其说明如表2-6所示。

表2-6　常用的gdb命令及其说明

命令	说明
b <function>	在<function>处设置一个断点(Breakpoint)
b *mem	在指定的绝对内存物理位置设置一个断点
info b	输出有关断点的信息
delete b	移除断点
run <args>	在gdb内使用给定参数开始调试程序代码
info reg	输出有关当前寄存器状态的信息
stepi or si	执行一条机器指令
next or n	执行一个函数
bt	回溯命令，输出栈帧的名称
up/down	在栈帧中向上或者向下移动
print var print /x $<reg>	分别打印变量的值和寄存器的值
x /*NTA*	检查内存，其中*N*表示要显示的单位数，*T*表示要显示的数据类型(x：十六进制，d：十进制，c：字符，s：字符串，i：指令)，*A*表示绝对地址或者诸如“main”的符号名称
quit	退出gdb

实验2-9：调试

为了调试示例程序代码，首先需要在Kali实例中安装gdb。

```
┌──(kali㉿kali)-[~/GHHv6/ch02]
└─$ sudo apt-get update
Get:1 http://kali.download/kali kali-rolling InRelease [30.5 kB]
…truncated for brevity…
Reading package lists... Done
┌──(kali㉿kali)-[~/GHHv6/ch02]
└─$ sudo apt install gdb
Reading package lists... Done
…truncated for brevity…
Do you want to continue? [Y/n] y
Get:1 http://kali.download/kali kali-rolling/main amd64 libc6-i386 amd64 2.31-9
[2,819 kB]
…truncated for brevity…
```

现在，执行以下命令，用于调试前面的示例程序代码。第一条命令将使用调试符号和其他有用选项(参考表2-3)重新编译meet程序代码。

```
┌──(kali㉿kali)-[~/GHHv6/ch02]
└─$ gcc -ggdb -mpreferred-stack-boundary=4 -fno-stack-protector -o meet meet.c
┌──(kali㉿kali)-[~/GHHv6/ch02]
└─$ gdb -q meet
Reading symbols from meet...
(gdb) run l337 Haxor
Starting program: /home/kali/GHHv6/ch02/meet l337 Haxor
Hello l337 Haxor
Bye l337 Haxor
[Inferior 1 (process 17417) exited normally]
(gdb) b main
Breakpoint 1 at 0x5555555551ab: file meet.c, line 10.
(gdb) run l337 Haxor
Starting program: /home/kali/GHHv6/ch02/meet l337 Haxor

Breakpoint 1, main (argc=3, argv=0x7fffffffe488) at meet.c:10
10      greeting(argv[1], argv[2]);      // call function, pass title & name
(gdb) n
Hello l337 Haxor
11      printf("Bye %s %s\n", argv[1], argv[2]);  // say "bye"
(gdb) n
Bye l337 Haxor
12    }                                 // exit program
(gdb) p argv[1]
$1 = 0x7fffffffe719 "l337"
(gdb) p argv[2]
$2 = 0x7fffffffe71c "Haxor"
(gdb) p argc
$3 = 3
(gdb) info b
Num     Type          Disp Enb Address            What
1       breakpoint    keep y   0x00005555555551ab in main at meet.c:10
        breakpoint already hit 1 time
(gdb) info reg
rax            0x0                0
rbx            0x0                0
rcx            0x0                0
rdx            0x0                0
rsi            0x5555555592a0     93824992252576
…truncated for brevity…
(gdb) quit
A debugging session is active.
Do you still want to close the debugger?(y or n) y
$
```

实验2-10：使用gdb反汇编

为了使用gdb生成反汇编代码，需要输入以下两条命令。

```
set disassembly-flavor <intel/att>
disassemble <function name>
```

第一条命令用于在Intel(NASM)和AT&T两种格式之间切换。默认情况下，gdb使用AT&T格式。第二条命令用于反汇编给定的函数(包括main函数)。例如，要采用两种格式反汇编greeting函数，则输入以下命令。

```
┌──(kali㉿kali)-[~/GHHv6/ch02]
└─$ gdb -q meet
Reading symbols from meet...
(gdb) disassemble greeting
Dump of assembler code for function greeting:
   0x0000000000001145 <+0>:     push   %rbp
   0x0000000000001146 <+1>:     mov    %rsp,%rbp
   0x0000000000001149 <+4>:     sub    $0x1a0,%rsp
   0x0000000000001150 <+11>:    mov    %rdi,-0x198(%rbp)
   0x0000000000001157 <+18>:    mov    %rsi,-0x1a0(%rbp)
   0x000000000000115e <+25>:    mov    -0x1a0(%rbp),%rdx
   0x0000000000001165 <+32>:    lea    -0x190(%rbp),%rax
   0x000000000000116c <+39>:    mov    %rdx,%rsi
   0x000000000000116f <+42>:    mov    %rax,%rdi
   0x0000000000001172 <+45>:    call   0x1030 <strcpy@plt>
   0x0000000000001177 <+50>:    lea    -0x190(%rbp),%rdx
   0x000000000000117e <+57>:    mov    -0x198(%rbp),%rax
   0x0000000000001185 <+64>:    mov    %rax,%rsi
   0x0000000000001188 <+67>:    lea    0xe75(%rip),%rdi        # 0x2004
   0x000000000000118f <+74>:    mov    $0x0,%eax
   0x0000000000001194 <+79>:    call   0x1040 <printf@plt>
   0x0000000000001199 <+84>:    nop
   0x000000000000119a <+85>:    leave
   0x000000000000119b <+86>:    ret
End of assembler dump.
(gdb) set disassembly-flavor intel
(gdb) disassemble greeting
Dump of assembler code for function greeting:
   0x0000000000001145 <+0>:     push   rbp
   0x0000000000001146 <+1>:     mov    rbp,rsp
   0x0000000000001149 <+4>:     sub    rsp,0x1a0
   0x0000000000001150 <+11>:    mov    QWORD PTR [rbp-0x198],rdi
   0x0000000000001157 <+18>:    mov    QWORD PTR [rbp-0x1a0],rsi
   0x000000000000115e <+25>:    mov    rdx,QWORD PTR [rbp-0x1a0]
   0x0000000000001165 <+32>:    lea    rax,[rbp-0x190]
   0x000000000000116c <+39>:    mov    rsi,rdx
   0x000000000000116f <+42>:    mov    rdi,rax
   0x0000000000001172 <+45>:    call   0x1030 <strcpy@plt>
```

```
…truncated for brevity…
   0x000000000000119b <+86>:    ret
End of assembler dump.
(gdb) quit
```

以下是更加常用的两条命令:

```
info functions
disassemble /r <function name>
```

info functions命令显示所有动态链接的函数，以及所有内部函数(除非已将程序代码删除)。安全专家使用disassemble函数以及/r <function name>选项以输出操作码、操作数和指令。操作码本质上是预汇编的汇编代码的机器码表示形式。

2.6　Python编程必备技能

Python是一门流行的解释型、面向对象的编程语言。很多黑客工具(以及许多其他应用程序)采用Python编写的原因是Python易学易用，功能非常强大，并且语法清晰，易于阅读。本节内容仅涵盖安全专家必须掌握的最基本的Python技能。但是，几乎能够肯定的是，安全专家应了解更多与Python相关的内容，可查阅众多优秀的Python专业书籍，或者浏览www.python.org网站上的丰富文档。Python 2.7于2020年1月1日停止更新。多年以来，许多安全从业人员都认为，如果想学习Python，从而能够使用、修改或者扩展现有的Python项目，那么首先应该学习Python 2.7。如果目标是研发Python新项目，则应该重点学习Python 3，Python 3解决了Python 2.7中存在的许多问题。目前，仍有大量程序代码依赖于Python 2.6或者Python 2.7，因此，要特别注意Python使用的版本。

2.6.1　获取Python

安全专家能够从www.python.org/download/下载与操作系统对应的Python版本，然后就可跟随本节练习。或者，只需要试着在命令提示符中输入python即可启动Python，因为许多Linux发行版和Mac OS X 10.3及其后续版本中都已经默认安装了Python。

macOS 用户和 Kali 用户使用 Python

对于macOS用户，Apple没有包含Python的IDLE用户界面，IDLE用户界面对于Python研发非常方便。安全专家能够从www.python.org/download/mac/获取IDLE，或者选择从Xcode(Apple的开发环境)中编辑和启动Python，请遵循http://pythonmac.org/wiki/XcodeIntegration的说明。如果已经有了Python，但是需要升级到Python 3，并将其设置为默认版本，正确的方法是使用pyenv，请扫描封底二维码下载“拓展阅读”部分，以获得一个优秀的教程链接。

对于Kali用户，在编写本章时，Kali 2020.4是最新版本，在这一版本中，为了实现向后兼容性，python2仍然是默认的链接版本，直到所有脚本都更新为python3。

Python属于解释型(非编译型)语言，因此，能够通过交互式环境立即获得反馈。在接下来的几节中，将使用Python的交互式环境，现在请输入python命令启动环境。

实验2-11：启动Python

如果拥有Kali 2020.4，仍然需要通过运行python3命令手动启动版本3，如下所示。

```
┌──(kali㉿kali)-[~/GHHv6/ch02]
└─$ python3
Python 3.8.6 (default, Sep 25 2020, 09:36:53)
[GCC 10.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

实验2-12：Python的“Hello,World!”

通常而言，每种语言都可能以“Hello, world!”的示例开始学习，这里展示的是在Kali 2020.4上使用Python 3.8.6版本，通过前面提到的python3命令启动。

```
>>> print("Hello, world!")
Hello, world!
>>>
```

注意，在Python 3中，print是一个正式函数，需要括号[2]。如果想退出这个Python shell，请输入exit()。

2.6.2　Python对象

安全专家需要真正理解的主要内容是Python用来存储数据的不同类型的对象，以及Python是如何操作这些数据的。本小节将介绍五大数据类型：字符串(string)、数值(number)、列表(list)、字典(dictionary)和文件(file)，然后将介绍一些基本语法，以及需要掌握的关于Python和网络的基本知识。

实验2-13：字符串

在实验2-12中已经使用了一个字符串对象。字符串在Python中用于保存文本。如下所示，使用Python 3 shell展示了使用和操作字符串的方法。

```
>>> string1 = 'Dilbert'
>>> string2 = 'Dogbert'
```

```
>>> string1 + string2
'DilbertDogbert'
>>> string1 + " Asok " + string2
'Dilbert Asok Dogbert'
>>> string3 = string1 + string2 + "Wally"
>>> string3
'DilbertDogbertWally'
>>> string3[2:10]  # string 3 from index 2 (0-based) to 10
'lbertDog'
>>> string3[0]
'D'
>>> len(string3)
19
>>> string3[14:]   # string3 from index 14 (0-based) to end
'Wally'
>>> string3[-5:]   # Start 5 from the end and print the rest
'Wally'
>>> string3.find('Wally')   # index (0-based) where string starts
14
>>> string3.find('Alice')   # -1 if not found
-1
>>> string3.replace('Dogbert','Alice')  # Replace Dogbert with Alice
'DilbertAliceWally'
>>> print('AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA')  # 30 A's the hard way
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
>>> print ('A' * 30)   # 30 A's the easy way
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
```

上述都是基本的字符串操作函数，可用于处理简单字符串，语法简明直观。值得注意的一个特性是，每个字符串(程序代码中名为string1、string2和string3)都仅是一个指针(对于熟悉C语言的研发人员而言)，或者是一个指向内存中某个数据块的标签。一个标签(或者指针)指向另一个标签的概念有时可能让新手研发人员感到困惑。以下代码和图2-10演示了这一概念。

```
>>> label1 = 'Dilbert'
>>> label2 = label1
```

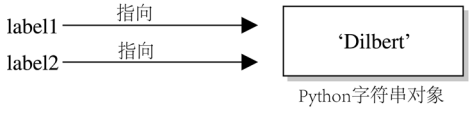

图2-10　两个标签指向相同的字符串内存地址

此时，内存中的某个地方存放着Python字符串‘Dilbert’。还有两个标签同时指向这块内存。接下来，即使修改label1的赋值，label2也不会改变。

```
... continued from above
>>> label1 = 'Dogbert'
>>> label2
'Dilbert'
```

从图2-11可看出，修改label1的赋值后，label2并没有指向label1，而是依然指向label1在重新赋值前所指向的同一个字符串。

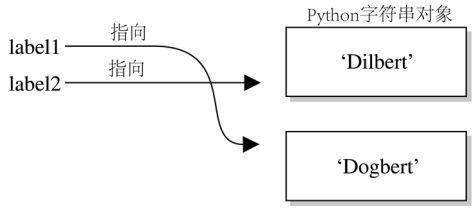

图2-11　label1经过重新分配并指向一个不同的字符串内存地址

实验2-14：数值

与Python字符串类似，数值指向一个能包含任何类型数字的对象。这种数据类型能存放较小的数值、较大的数值、复数、负数以及任何可想象的其他类型数值。语法正如软件研发人员所期望的：

```
>>> n1=5   # Create a Number object with value 5 and label it n1
>>> n2=3
>>> n1 * n2
15
>>> n1 ** n2     # n1 to the power of n2 (5^3)
125
>>> 5 / 3, 5 % 3    # Divide 5 by 3, then 5 modulus 3
(1.6666666666666667, 2)
# In Python 2.7, the above 5 / 3 calculation would not result in a float without
# specifying at least one value as a float.
>>> n3 = 1       # n3 = 0001 (binary)
>>> n3 << 3      # Shift left three times: 1000 binary = 8
8
>>> 5 + 3 * 2    # The order of operations is correct
11
```

在理解了数值的工作原理后，就能够开始组合对象。如果将字符串和数值相加，将获得什么结果？

```
>>> s1 = 'abc'
>>> n1 = 12
>>> s1 + n1
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: Can't convert 'int' object to str implicitly
```

显示错误消息！需要帮助Python理解研发人员希望执行的操作。这里，能够将'abc'和12组合在一起的唯一方法是将12转换为字符串。研发人员可随时执行转换操作：

```
>>> s1 + str(n1)
'abc12'
>>> s1.replace('c',str(n1))
'ab12'
```

如果上述方法能够成功，就能够将不同的类型组合在一起使用。

```
>>> s1*n1   # Display 'abc' 12 times
'abcabcabcabcabcabcabcabcabcabcabc'
```

关于对象还有一点需要注意，简单地在对象上操作并不会改变对象。对象(数值、字符串以及其他类型的对象)本身通常只有在显式地将对象的标签(或者指针)设置为新值时才发生改变，如下所示。

```
>>> n1 = 5
>>> n1 ** 2           # Display value of 5^2
25
>>> n1                # n1, however is still set to 5
5
>>> n1 = n1 ** 2      # Set n1 = 5^2
>>> n1                # Now n1 is set to 25
25
```

实验2-15：列表

接下来要介绍的内置对象类型是列表(List)。软件研发人员能够将任何类型的对象存放到列表中。通常，在某个对象或者某组对象的前后添加“[”和“]”就可创建列表。软件研发人员能够使用类似于操作字符串的方法对列表执行同样灵活的“切片”(Slicing)操作。所谓字符串切片，指的是只返回字符串对象值的一个子集，例如，label1[5:10]只返回第5到第10个值。以下代码将演示列表类型的工作方法。

```
>>> mylist = [1,2,3]
>>> len(mylist)
3
>>> mylist*4          # Display mylist, mylist, mylist, mylist
[1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3]
>>> 1 in mylist       # Check for existence of an object
True
>>> 4 in mylist
False
>>> mylist[1:]        # Return slice of list from index 1 and on
[2, 3]
>>> biglist = [['Dilbert', 'Dogbert', 'Catbert'],
... ['Wally', 'Alice', 'Asok']]    # Set up a two-dimensional list
>>> biglist[1][0]
```

```
'Wally'
>>> biglist[0][2]
'Catbert'
>>> biglist[1] = 'Ratbert'   # Replace the second row with 'Ratbert'
>>> biglist
[['Dilbert', 'Dogbert', 'Catbert'], 'Ratbert']
>>> stacklist = biglist[0]   # Set another list = to the first row
>>> stacklist
['Dilbert', 'Dogbert', 'Catbert']
>>> stacklist = stacklist + ['The Boss']
>>> stacklist
['Dilbert', 'Dogbert', 'Catbert', 'The Boss']
>>> stacklist.pop()         # Return and remove the last element
'The Boss'
>>> stacklist.pop()
'Catbert'
>>> stacklist.pop()
'Dogbert'
>>> stacklist
['Dilbert']
>>> stacklist.extend(['Alice', 'Carol', 'Tina'])
>>> stacklist
['Dilbert', 'Alice', 'Carol', 'Tina']
>>> stacklist.reverse()
>>> stacklist
['Tina', 'Carol', 'Alice', 'Dilbert']
>>> del stacklist[1]        # Remove the element at index 1
>>> stacklist
['Tina', 'Alice', 'Dilbert']
```

接下来，将快速理解字典类型和文件类型，并组合使用所有元素类型。

实验2-16：字典

字典(Dictionary)与列表类似，二者之间的差别在于，在使用字典时通过键(而不是对象索引)引用存放在字典中的对象。这是一种非常方便的存储和取回数据的机制。通过在键-值对前后添加“{”和“}”就能够创建字典，如下所示。

```
>>> d = { 'hero' : 'Dilbert' }
>>> d['hero']
'Dilbert'
>>> 'hero' in d
True
>>> 'Dilbert' in d     # Dictionaries are indexed by key, not value
False
>>> d.keys()     # keys() returns a list of all objects used as keys
```

```
dict_keys(['hero'])
>>> d.values()    # values() returns a list of all objects used as values
dict_keys(['Dilbert'])
>>> d['hero'] = 'Dogbert'
>>> d
{'hero': 'Dogbert'}
>>> d['buddy'] = 'Wally'
>>> d['pets'] = 2      # You can store any type of object, not just strings
>>> d
{'hero': 'Dogbert', 'buddy': 'Wally', 'pets': 2}
```

在下一节中，将更多地使用字典。字典是存储可与键建立关联的数据的极佳方式，使用键取回数据比使用列表索引更加有效。

实验2-17：Python文件操作

文件访问与使用其他Python语法一样简单。文件可打开(用于读取或者写入)、写入、读取和关闭。下面列举一个示例，将上述所讨论的几种不同的数据类型(包括文件)组合在一起。本示例首先假设存在一个名为targets的文件，然后将文件内容转移到单独的漏洞目标文件中(希望能够听到，“Dilbert的示例终于结束了!”)，注意代码格式中的缩进。在本示例中，使用Python 3 shell解析文件并将文件内容移动至另外两个文件中。在Kali中使用两个shell，每个都在同一目录中。代码中以#符号开始的部分是注释内容，安全专家当然不需要输入注释内容。

```
┌──(kali㉿kali)-[~/GHHv6/ch02]
└─$ cat targets
RPC-DCOM       10.10.20.1,10.10.20.4
SQL-SA-blank-pw 10.10.20.27,10.10.20.28
# We want to move the contents of targets into two separate files
┌──(kali㉿kali)-[~/GHHv6/ch02]
└─$ python3
# First, open the file for reading
>>> targets_file = open('targets','r') ❶
# Read the contents into a list of strings
>>> lines = targets_file.readlines()
>>> lines
['RPC-DCOM\t10.10.20.1,10.10.20.4\n',
'SQL-SA-blank-pw\t10.10.20.27,10.10.20.28\n']
# We can also do it with a "with" statement using the following syntax:
>>> with open("targets", "r") as f:
...     lines = f.readlines()
...
>>> lines
['RPC-DCOM      10.10.20.1,10.10.20.4\n', 'SQL-SA-blank-pw
10.10.20.27,10.10.20.28\n', '\n']
```

```
# The "with" statement automatically ensures that the file is closed and
# is seen as a more appropriate way of working with files..
# Let's organize this into a dictionary
>>> lines_dictionary = {}
>>> for line in lines: ❷      # Notice the trailing : to start a loop
...     one_line = line.split()     # split() will separate on white space
...     line_key = one_line[0]
...     line_value = one_line[1]
...     lines_dictionary[line_key] = line_value
...     # Note: Next line is blank (<CR> only) to break out of the for loop
...
>>> # Now we are back at python prompt with a populated dictionary
>>> lines_dictionary
{'RPC-DCOM': '10.10.20.1,10.10.20.4', 'SQL-SA-blank-pw':
'10.10.20.27,10.10.20.28'}
# Loop next over the keys and open a new file for each key
>>> for key in lines_dictionary.keys():
...     targets_string = lines_dictionary[key]      # value for key
...     targets_list = targets_string.split(',')     # break into list
...     targets_number = len(targets_list)
...     filename = key + '_' + str(targets_number) + '_targets'
...     vuln_file = open(filename,'w')
...     for vuln_target in targets_list:      # for each IP in list...
...           vuln_file.write(vuln_target + '\n')
...     vuln_file.close()
...
>>> exit()
┌──(kali㉿kali)-[~/GHHv6/ch02]
└─$ ls
RPC-DCOM_2_targets            targets
SQL-SA-blank-pw_2_targets
┌──(kali㉿kali)-[~/GHHv6/ch02]
└─$ cat SQL-SA-blank-pw_2_targets
10.10.20.27
10.10.20.28
┌──(kali㉿kali)-[~/GHHv6/ch02]
└─$ cat RPC-DCOM_2_targets
10.10.20.1
10.10.20.4
```

本示例引入了两个新概念。首先，可看出文件使用是非常方便的；open()函数包含两个参数❶：第一个参数是需要读取或者创建的文件名称，第二个参数是访问类型。软件研发人员可通过读取(r)、写入(w)或者追加(a)的方式打开文件。在字母之后添加“+”符号可提升权限，例如，r+可获取对文件的读写访问权限。在权限后添加b将以二进制模式打开文件。

其次，下面是一个for循环示例❷，for循环的结构如下所示。

```
for <iterator-value> in <list-to-iterate-over>:
    # Notice the colon at the end of the previous line
```

```
    # Notice the indentation
    # Do stuff for each value in the list
```

警告：在Python中，空格是非常重要的，而缩进则用以标记代码块。大多数Python软件研发人员坚持缩进4个空格。在整片代码块中，缩进数量应该保持一致。请参阅“拓展阅读”中的“Python编程风格指南”(请扫描封底二维码下载)。

减少一级缩进或者在空行回车，将关闭循环。这里并不需要C语言格式的大括号。if语句和while循环也采用类似的结构。例如：

```
if foo > 3:
    print('Foo greater than 3')
elif foo == 3:
    print('Foo equals 3')
else:
    print('Foo not greater than or equal to 3')
...
while foo < 10:
    foo = foo + bar
```

实验2-18：Python套接字编程

本章将讲解的最后一个主题是Python的套接字(Socket)对象。为演示Python套接字，以下代码构建了一个简单客户端，客户端连接到远程(或者本地)主机，然后发送“Say something:”。为测试下述代码，需要一个“服务器”(Server)以监听客户端的连接请求。安全专家可使用如下语法，将netcat监听应用程序绑定到4242端口，从而模拟服务器端(需要在新的shell窗口中启动nc)。

```
┌──(kali㉿kali)-[~/GHHv6/ch02]
└─$  nc -l -p 4242
```

客户端代码(应该在单独的shell中运行)如下所示。

```
#client.py
import socket
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect(('localhost', 4242)) ❶
s.send(b'Say something: ') ❷ # b tag added in python3 to indicate bytes not str
data = s.recv(1024) ❸
s.close()❹
print('Received', data) ❺
```

请记住导入socket库，在套接字实例化行中有一些套接字选项也需要记住，此外，其他部分非常简单。安全专家连接到主机和端口❶，发送想要的内容❷，使用recv接收数据并存

储到指定对象❸，然后关闭套接字❹。安全专家当在某个单独的shell中执行命令python3 client.py时，在netcat监听应用程序中将显示“Say something:”消息，而在监听程序中输入的任何内容都将返回到客户端❺。至于安全专家如何使用Python语言的bind()、listen()和accept()等语句模拟netcat监听应用程序，此处留作练习。

2.7　总结

本章简要介绍了编程概念和安全注意事项。道德黑客必须掌握足够多的编程技能以便编写漏洞利用程序或者审查源代码；当逆向恶意软件或者发现漏洞时，道德黑客需要理解汇编代码；最后，同样重要的是，为了能分析恶意软件运行时的行为或者跟踪shellcode在内存中的执行，调试是必备的技能。为了掌握编程语言或者逆向工程，唯一的途径是不断练习，从这里启航吧！