游戏开发技术系列丛书

EXPLOITING ONLINE GAMES
CHEATING MASSIVELY DISTRIBUTED SYSTEMS

网络游戏安全揭密

（美）Greg Hoglund
Gary McGraw 著

姚晓光 等译

机械工业出版社
China Machine Press

本书主要介绍了游戏被入侵的机理、防范手段及找到软件漏洞的方法。本书以《魔兽世界》等 MMO-RPG 游戏为典型案例进行分析，非常清楚完整地展示了入侵代码，从简单的宏指令到复杂的系统后门入侵方法，都毫不保留地进行了展示。本书中的网络游戏就像是软件安全方面一个相当有趣的实验品。

本书适合软件开发、游戏开发人员等阅读。

图书在版编目(CIP)数据

网络游戏安全揭密/(美)霍格伦德(Hoglund, G.)，(美)麦克格劳(McGraw, G.)著；姚晓光等译. —北京：机械工业出版社，2009.1

(游戏开发技术系列丛书)

书名原文：Exploiting Online Games

ISBN 978-7-111-25522-2

Ⅰ. 网…　Ⅱ. ①霍…　②麦…　③姚…　Ⅲ. 计算机网络－游戏－安全技术－基本知识

Ⅳ. G899　TP309

中国版本图书馆 CIP 数据核字(2008)第 172477 号

机械工业出版社(北京市西城区百万庄大街 22 号　邮政编码　100037)

责任编辑：曾　珊

北京诚信伟业印刷有限公司印刷

2009 年 2 月第 1 版第 1 次印刷

186mm×240mm · 15 印张

标准书号：ISBN 978-7-111-25522-2

定价：45.00 元

凡购本书，如有倒页、脱页、缺页，由本社发行部调换

本社购书热线：(010)68326294

关于本书的评论

“在足球比赛中，如果只一味防守却不去关注对方的进攻会发生什么情况？你将无法得知对方什么时候会杀过来，不知道如何去防守对方的传接渗透，也不知道如何发动一次闪电战般的进攻。在计算机网络领域也是这样，要做好防护工作，必须先清楚如何去进行攻击。我常常在我的课程里说到，你应该是第一个去攻击自己计算机系统的人，而不是最后一个。”

“这个世界的网络化进程越来越快。当我还在警惕网上的投票行为时，网络游戏已经暴风骤雨般地铺开了。新时代到来了，游戏中的虚拟道具也可以具备真实货币的价值，因此财富可以通过这些虚拟物品而累积或消失，这对于玩家来说，构成了一定威胁。为了知道如何防护这种威胁，你必须理解这些危害是怎么回事。本书是我见过的最深入最详尽地介绍如何入侵网络游戏的图书，我相信每一个 White Hat 都应该阅读此书。这可以让你们紧紧跟随着那些黑客们的步伐。”

——*Aviel D. Rubin* 博士

约翰霍普金斯大学信息安全研究所计算机科学 教授

“所有人都在讨论虚拟世界，但是却没有人去讨论虚拟世界里的安全。Greg Hoglund 和 Gary McGraw 在本书中完美地叙述了目前网络游戏中的安全问题。”

——*Cade Metz*

《PC 杂志》资深主编

“在如何改进安全隐患方面，本书指出了一条正确的道路。正如本书作者所说的，当你面对着这些可憎的恶魔时，你需要有经验的同伴一起作战，就像持起亚瑟王之剑那样披荆斩棘。”

——*Edward W. Felten* 博士

普林斯顿大学信息技术中心主任

计算机科学与公共事业 教授

“游戏经常用于现代军事演习来衡量技术进步带来的成果，尤其是空军演习。本书的讨论方式也类似这一概念，将目标定为“入侵游戏”，来衡量一个游戏的安全性。对于目前较为严重的计算机系统安全问题来说，本书意义甚广。”

“大规模分布式系统将会是未来四分之一个世纪里软件发展的方向。我们需要去熟悉它们是如何运作的，并且更重要的是理解它们是如何因为安全隐患被入侵的。本书堪称此学术领域的奠基之作。”

——*Daniel McGarvey*

美国空军信息防护委员会主席

“就像孩子一样，Gary 和我都是通过游戏而深入了解了计算机(而后是计算机安全)。起初我们沉溺于玩 Apple][s 上的游戏，但后来发现游戏太少又很贵。于是我们互相拷贝对方的游戏，但发现这些游戏有版权保护措施，于是我们开始研究如何破解这些保护措施。但后来发现，破解这些游戏远比玩它们更有乐趣了。”

“随着如今网络游戏行业的蓬勃发展，人们不再仅仅是因为兴趣而去入侵游戏，而是为了现实的金钱。这是无法阻止的一个趋势。但首先，本书揭示了目前黑客们常用的一些入侵技术。”

——*Greg Morrisett* 博士
哈佛大学工程学与应用科学学院 计算机科学 教授

“如果你现在是一名网络游戏的玩家但你却不了解网游中的安全，那你的游戏经历是不完整的；如果你正在编写一个大规模分布式软件系统但你又不参考网游，那你就落伍了。”

——*Brian Chess* 博士
Fortify Software 创建者、首席技术师
Static Analysis 中安全模块的制作者之一

“这本书以一种全新的视角来看待软件安全问题：去攻击一个网络游戏。新手一定会觉得本书充满了新奇之感，而老手们也会享受到独特的感觉，比如发现自己以前编程的一些错误，而且只有大规模的多人在线的网络游戏才可以揭示到这一点！太棒了！”

——*Pravir Chandra*
Cigital 首席顾问
OpenSSL 网络安全模块的制作者之一

译 者 序

入侵游戏的诱惑

如果有一天，突然有人对你说“你要交200元的罚款才可以离开”，千万不要以为你是因违犯交通规则而被交警抓了，而是因为你在网游中使用“外挂”，被游戏程序自动“抓捕”了，要交200元的罚款。

如果有机会，很多人会作弊，尤其是认为自己不会被抓到的时候。网络上可以很容易匿名，这就让作弊更具吸引力。游戏是软件，是软件就有缺陷，这是来自人类开发者的DNA。每天都有很多人使用网络游戏的外挂和辅助工具，其中部分玩家是因为太忙，虽然很想玩但没有办法投入太多时间，不得不采用作弊的方式。为了改变网络游戏岌岌可危的安全局势，游戏开发人员必须在软件安全方面做得更好，毕竟任何公司都没有权利通过对玩家罚款来制止外挂。提高网络游戏安全性的第一步，就是掌握游戏被入侵时实际发生的情况。那么

- 游戏是如何被入侵的呢？
- 我们又该如何防范呢？
- 有哪些方法可以找到软件的漏洞呢？

这就是本书所要说的。书中以《魔兽世界》等MMO-RPG游戏为典型案例进行分析，非常清楚完整地展示了入侵的代码，从简单的宏指令到复杂的系统后门入侵方法，都毫不保留地展示。在本书中网络游戏就像是软件安全方面一个相当有趣的实验品。

网游的安全问题阻碍游戏的成功

随着竞争越来越激烈，游戏研发、代理、运维、营销的成本越来越高，信息和电子对抗本来就是一场精彩的较量，网络游戏的设计有着很多缺陷，例如：“两种技能的叠加”就是一种找漏洞的策略。在《魔兽世界》中魔法交互作用的结果会很惊人。在有的宠物身上使用“豹群守护”魔法，然后让它攻击远处的怪物，紧接着在“飞行点”搭乘飞行坐骑。通过这两种技能的组合，有趣的状态发生了。在这个例子中，你并没有开始自动飞行，而是获得了飞行坐骑的控制权，可以自己驾驶坐鹰了。这类问题并不限于《魔兽世界》，状态组合攻击对大部分的MMO-RPG都有效，利用状态时间差甚至可以复制网络游戏中的装备和金钱。当然除了找到游戏设计缺陷，本书的精华部分更在于第6、7、8章所介绍的入侵技术和手段。其中谈到的破解技术：自动控制键盘鼠标、像素采样、数据包传递和截获、覆盖率测试工具、反编译、反汇编、故障注入引擎、逆向代码工程、动态跟踪、注入DLL文件、内存伪造等技术。这些内容不管是针对游戏，还是其他软件都非常重要。这些技术的掌握程度是区分初级破解者和资深破解人员的标志。如果读者真的

对破解软件感兴趣，学习逆向这些技巧绝对是值得的。

“看雪”是一位知名破解者的网名，他曾说过：“很多人都想做破解者，好像破解了一个程序就很风光，人人都很佩服。”[0Day]是国际上著名的破解者组织。据说，美国的一些大学给成为[0Day]成员的学生加分，表示认同他们的技术水准。这个组织的成员有能力在一天内破解软件。这些年轻人喜欢穿着印有破解组织标志的外套招摇过市，FBI（中央情报局）官方网站曾被美国的网络破译者们改为 FBF(中央愚笨局)。他们不赚钱，但追求绝对技术。“看雪”也说：“当破解者的目的是，在破解中跟踪软件，了解程序思路，这样提高自我，使自己能写出更完美的程序。”在美国和欧洲，破解者被厂商告上法庭，但被判无罪或仅被处以轻微罚款，因为有些破解者并不谋利。表面上看，破解者主观上是在和游戏技术较劲，客观上是和游戏的厂商对抗。游戏厂商往往财大气粗，破解者常常钱少势单，这使破解具有了几分以弱抗强、行侠仗义的色彩。所以游戏公司的研发人员必须用自己的技术手段保证大量的付费玩家不受到外挂等非法软件的侵害，维护游戏世界的公平环境。

恭喜你拥有了本书

本书作者是具有传奇色彩并为美国国防部服务的破解天才 Greg Hoglund 和全球公认的软件安全权威 Gary McGraw。本书除了针对软件破解者，对游戏研发者更具参考价值，Greg 也希望有朝一日能够将自己的反破解技术运用到自己研发的一款网络游戏，在实战中去完善游戏安全方面的知识。对于经验不够丰富的软件安全人员而言，两位大师著作的此书可能有些晦涩，如能细心研读并多加参考相关书籍，定能“入宝山而不空手而归!”

再次恭喜你选择了本书，请记住你是个天才。当其他人只看一个台阶的时候，天才看到了两个。现在你做到了，在网络游戏安全问题日渐突出的时候，你选择了本书，破解和反破解本来就是一个神奇的行业，对那些胸怀大智的游戏设计者们，本书非常有参考意义。欢迎你选择来到天才的世界!

笔者陆续翻译了《网络游戏开发》、《游戏关卡设计》、《游戏角色和剧本创造》游戏行业书籍，但是拿到本书才发现翻译游戏技术书籍原来如此困难，书中涉及了太多的专业名词和代码中的注释。很有幸在翻译过程中得到了来自腾讯互动娱乐“业务安全组”同事们的支持，感谢李俊明、单晖、曹赟为本书翻译付出的心血，并感谢上海外国语大学张蔚女士提供专业支持。

腾讯公司游戏项目总监

姚晓光

序　言

聪明人懂得从错误中吸取经验；更聪明的人懂得从别人的错误中吸取经验。像我们这些从事计算机安全方面的人，由于拒绝谈论错误，或是对错误视而不见，从而往往错失了宝贵的学习机会。

此书开诚布公地讨论了很多游戏公司犯过的错误以及它们所带来的严重后果。对于广大游戏公司而言，这是从自身和同行们的错误中学习经验的一次机会；对于广大读者而言，我们能了解到哪些环节可能有安全问题以及如何改善并提高。

关于是否要和盘托出网络安全方面的所有案例，不管是在行业道德方面还是法律方面都久有争论，在此无需赘述。对于从事计算机安全方面的人而言，这个问题的答案很简单——安全技术对所有人都十分重要，学习这方面的技术无疑会让全社会受益匪浅。

在现如今的社会上，我们的金钱、时间、个人隐私甚至生命都依赖于某些系统的正常运作。有时为了做出正确的选择，我们必须争取一切能争取到的帮助。在有些领域，例如航空领域，我们对安全问题很有信心，因为没有人能接受飞机制造方掩盖真相、阻碍调查，或是误导大众，因此所有的安全问题得到了很好的研究和改善。这种信息公开、细致研究、及时矫正的态度是我们的航空事业一直保持高度安全系数的原因。

游戏开发行业也许没有航空行业的利害关系那么严重，但问题的性质是相似的。和航空行业一样，产品的表现直接关系到游戏公司的利益。一款成功的游戏，尤其是像《魔兽世界》那样的游戏，可以形成一种经济——游戏中的每样东西和人物都具有实际的经济价值。每天有越来越多的人依靠游戏交易产生的经济收益谋生。在虚拟世界中究竟产生了多少 GDP？这个问题经济学家们也没有统一定论，但可以确定的是，虚拟经济和纳斯达克股票交易一样，是能够带来实际经济效应的。

除了游戏公司，还有很多人与游戏的表现息息相关：投资方给游戏公司投的那一大笔钱究竟结果如何，程序员做游戏能换来怎样的营生，甚至游戏公司对面的餐馆生意如何也受到影响。所有的人都深切地关心这个行业是否能稳健发展。还有那些潜在的客户们——在他们把辛苦赚来的钞票用来购买游戏包月和增值服务之前，也想知道游戏究竟给他们带来怎样刺激的体验。

如果说航空行业向我们展示了信息公开的益处，另外一个例子——网络投票则充分体现了信息封闭带来的害处。那些进行网络投票的普通用户不了解计算机是如何运作的，他们只知道计算机经过认证，但对认证的过程又一无所知。别人告诉他们，细节不重要……信息封闭的后果就是：研发力量薄弱，问题长期得不到解决，研发进程缓慢，有问题出现时，研究解决十分困难……

这本书的优点在于，它不仅告诉你存在的问题，还给出大量细节信息：有的安全漏洞只存在

于理论当中，实际操作时很少发生；有的问题看上去很严重，实际并没有那么夸张；还有的问题实际处理起来比以前书上写的要棘手得多。要甄别个中差异，我们必须了解细节，必须清楚地知道黑客是如何攻击的，破解的是哪些安全屏障。所有的这些必要信息，都能在此书中找到。

此书触及的领域是热门的大规模多人在线游戏，为游戏安全提供高水平多细节的视角，为想要学习游戏安全的学生提供丰富的案例资料。只有在实践中得到有效的运用，理论这一工具的价值才能得到最高体现。在网游这个行业，我以一名实践者的身份起家，先是学习如何解决问题，系统是如何运作的，然后才从层次上提高自己，开始正式的计算机知识培训。我想这行的高手基本都是这样的成长过程。在我们学习的时候，类似书籍是根本没有的（即使有，我也从来不知道）。比较起来，现在的学生要幸运得多。

也许有的游戏公司看到这本书会很不高兴，怪作者批判他们的软件做得不安全。别犯傻了，如果我们要进步要发展，这样的开放式讨论是唯一出路。

每个人都会犯错，让我们从自身和他人的错误中吸取经验教训，下次做到更好！这是我们共同的愿望。

Edward Felten（爱德华·费尔顿）教授

写于美国新泽西州普林斯顿

2007 年 6 月

前　言

目前，网络游戏(包括魔兽世界、无尽的任务、第二人生和在线扑克)已经席卷了计算机世界。游戏技术一直跑在计算机技术的前端，渗入消费市场的各个方面。过去十多年来，计算机游戏与因特网以同样快的速度发展，现在可以在数以千万计的家庭中发现游戏的踪迹。

互联网正在飞速发展，也正在经历很多青少年成长过程中所能遇到的烦恼——这些烦恼主要是无处不在的计算机安全问题。网络游戏，特别是大型多人在线角色扮演游戏(简称为MMORPG)，直接受到这些安全问题的影响。

MMORPG 游戏是一种非常复杂的大规模分布式客户端—服务器结构软件。游戏推动软件技术发展的极限，特别是在处理状态和时间方面(数以十万计的用户需要实时互动)，作为软件安全的个案研究，该问题特别值得关注。事实上，MMORPG 游戏遇到的安全问题可以用来预测计算机软件将会碰到的安全问题。

现代软件(不只是游戏软件)正在演变成为大规模分布式系统，需要与服务器发生交互；服务器应当能够同时提供服务给数千用户。转向 Web 服务和面向服务的架构所采用的技术，如 AJAX 和 Ruby，正是在线游戏技术发展的结果。目前在游戏这里学习到的技术将来一定会广泛适用于其他软件的发展中。

多人在线游戏是个大买卖，无时无刻不在赚钱，这更添加了该类软件的安全性需求。目前世界上最流行的网络游戏，暴雪的魔兽世界，用户已超过 800 万，每个人每月支付 14 美元去玩游戏。分析师估计，2009 年的游戏市场将达到 120 亿美元。

在虚拟世界中创造的 MMORPG 游戏，简单的数据结构具有价值，主要是因为该数据反映了游戏玩家花在玩游戏上的时间。玩家积累并交易虚拟财产。许多虚拟经济的人均国民生产总值比不少小国家都要大。游戏的虚拟经济和现实经济之间存在直接联系。最近，在 eBay 中可以购买游戏点卡，游戏点卡相当于玩游戏的权利，另外，目前市场上还存在很多中间形式的游戏业务。这导致部分玩家对将游戏中的虚拟财富带到现实中来更感兴趣。

只要有利益存在，作弊等行为就会存在。在 MMORPG 游戏中，作弊者由于利益驱动，会采用破坏游戏安全的手段来积累虚拟物品和经验点，用于交换现实世界中的金钱。许多非常有特色的游戏道具会通过拍卖，出售给出价最高者。

不少经验丰富的黑客在利润丰厚的 MMORPG 游戏领域工作多年，其中一些人以游戏作弊技术谋生。本书清晰地描述游戏黑客常用的攻击技术和技巧。

本书目标

正如你想象的那样，游戏公司对游戏作弊行为深恶痛绝。如果游戏中作弊行为猖獗，那些不

作弊的玩家将会非常不满，从而改玩其他游戏。游戏开发商已经采取了一系列措施来改善游戏安全，其中有些方法尚有争议(在后台监测玩家的电脑)，有些通过法律来约束玩家(实行严格的软件许可协议和使用条件)，有些通过加密手段(使用对称加密技术，但在游戏客户端代码包括密钥)。本书希望通过介绍各种攻击和黑客技术，让游戏开发商更好地做好在线游戏安全工作。

作者认为本书的话题非常重要。第一，游戏涉及现实的金钱；第二，很多玩家不关注游戏安全，只关注游戏本身是否好玩；第三，网络游戏安全有许多重要的经验教训可以直接适用于其他软件。另外，网络游戏安全很有意思。

例如，一些游戏公司已经使用 rootkits 技术来监测玩家的电脑。游戏公司也采用强硬手段制裁黑客，甚至制裁那些没有试图以任何方式恶意赚钱的黑客。其他软件厂商是否会在将来采用类似的手段呢?

不仅是技术问题，法律问题(如网络游戏及其创造性的软件许可条款)也预示着今后其他软件可能的发展模式。游戏公司，学术界和用户之间的法律争议才刚刚开始。

最终，网络游戏安全带来一些有趣的问题，最紧迫问题是：如何平衡玩家的隐私权利以及保护游戏不被破解?

法律界限

首先，游戏一定不能纵容恶意黑客攻击行为或任何其他游戏作弊行为。最有意思的是深刻认识和讨论目前的网络游戏安全问题。作为安全专家，我们认为，只有理解游戏被分析时的具体技术细节，才能真正开始建立游戏安全保护系统，游戏才能够经受真正的攻击。因为利益的驱动，可以肯定，攻击都是有组织的行为。

笔者认为，了解游戏如何工作以及怎么失败是非常必要的。做到这一点的唯一办法是仔细研究游戏技术。本书未从安全角度显示网络游戏客户端失败的细节，仅说明了可以用来破解网络游戏的技术。本书目标不是为了创造网络游戏黑客大军，游戏黑客人数已经非常多，已经做这行的人员不可能从这本书获取更多信息。本书主要目的是通过描述游戏破解的各种工具和方法，让游戏开发人员了解游戏是如何被破解的。

本书研究的内容，不与任何法律冲突。笔者也希望读者同样不要使用本书介绍的技术进行违法行为。

本书内容

本书先介绍游戏安全的总体情况，并逐步在后续的章节讲解更多的技术细节。

第 1 章介绍了为什么选择游戏这个主题? 回答了一些简单的问题：网络游戏规模，人数，为什么有人想破解游戏，游戏作弊的动机等。这些问题的答案可能会出乎您的意料，1000 万人玩在线游戏，金额达数十亿美元，其中有部分人则以欺骗为生。第 1 章还简单介绍了游戏架构，描述了大多数游戏使用的客户—服务器模式。

第 2 章介绍了基本的游戏分析技术，本章描述 6 个基本内容：制作“外挂”类程序，用户界面利用，使用代理，内存修改，使用调试器，预测技术。笔者特别关注外挂制作，因为大多数游戏都存在外挂。本章后半部分中，笔者展示了一个非常简单的外挂，让读者能有更直观的了解。

然后，笔者描述一个游戏制造商为了对付外挂所采用的有争议的行为：安装类似 rootkit 的间谍软件，用于跟踪玩家的 PC 行为。笔者对此行为持有不同意见，并开发了一个用于检测玩家计算机行为的程序。笔者认为游戏厂商应当投入资源开发更安全的游戏，而不是把资源投入在开发监测技术上。

第 3、4 章主要讲解金钱和法律相关内容。第 3 章描述了在利益驱使下，部分玩家试图作弊。最近 Julian Dibbell（Basic Books，2006）的 Play Money 一书中描述一个可怜男子进军职业联赛的情况。游戏企业已经成长，有足够的钱提供中介服务，为玩家购买和出售虚拟物品提供平台。甚至，对于互联网游戏娱乐公司（Internet Gaming Entertainment，IGE）来说，值得为该公司专门开辟一章来介绍。

第 4 章介绍有关法律内容。游戏公司（实际上包含其他软件制造商）建立了一系列许可证，最终用户许可协议（下载内容）和使用条款文件等。笔者不是律师，读者也不应依靠笔者的意见，这里仅简要介绍了美国版权法与数字千年版权法（DMCA）。本章还介绍了索尼的 rootkit 相关内容。

第 5 章重新开始描述技术方面的内容，主要讲述在线游戏的漏洞。本章描述在许多场游戏中发现的各种漏洞及其攻击手段。特别关注时间和状态相关的漏洞，该类漏洞更容易在游戏中出现。

第 6 章开始描述真正的技术内容，主要关注游戏客户端破解。随着我们更深入地分析游戏，笔者不得不使用一两个游戏作为特定的目标。其目的不是为了挑某些游戏的刺，而是讲解技术的需要。本书选暴雪的魔兽世界作为目标，主要是因为它是世界排名第一的网络游戏。使用魔兽世界上的技术可以类推到几乎所有的在线游戏（甚至现代 Web 2.0 软件）。第 6 章的开始部分讨论了攻击者的工具包（笔者描述的为标准软件测试工具）。然后讨论了游戏黑客技术同样重要的四个方面：利用游戏用户界面突破游戏，操纵游戏内存，游戏之下，通过底层服务如视频驱动程序突破游戏，拦截和操纵游戏的网络流量。第 6 章是本书的核心内容，介绍了大量游戏分析者常用的技术。

第 6 章包含太多的数据，读者通过实践会更容易理解这些技术。为此，在第 6 章的基础上，第 7 章构建了一个外挂。第 7 章的技术性较强，用大量的例子代码展示了第 6 章的想法如何实现。

第 8 章主要介绍逆向工程技术，技术性更强。本章内容比较难，但也不是什么新的技术，读者可以参考 Exploiting Software（Addison-Wesley，2004）来获取更多信息。反汇编器和反编译器本身只是一个工具，无所谓好坏，主要看是好人还是坏人使用。

第 9 章介绍最终的技术内容：先进的游戏破解技术。本章讨论游戏金钱系统和游戏资源，描述了从游戏数据文件中获取资源构建私服及混和不同的游戏特性等内容。游戏公司一直在努力试图制止该行为的发生，但是该类行为一直存在。

本书的目的是帮助游戏开发者把安全工作做得更好。第 10 章介绍软件安全的新领域。游戏开发商应当采取一些更好的做法，使游戏安全做得更出色。笔者还描述了玩家可能向游戏公司咨询的一系列安全问题。笔者最希望的是，本书能推动更多安全的软件的开发，无论是在游戏界或者其他领域。

软件安全系列图书

本书是 Addison-Wesley 出版社软件安全系列图书的一部分，该系列还包括以下图书：

- *Exploiting Online Games*: *Cheating Massively Distributed Systems*
- *Secure Programming with Static Analysis*
- *Software Security*: *Building Security In*
- *Rootkits*: *Subverting the Windows Kernel*
- *Exploiting Software*: *How to Break Code*
- *Building Secure Software*: *How to Avoid Security Problems the Right Way*

本系列将会有更多的书籍。想获取更多信息，可以联系 Addison-Wesley 出版社或者 Gary McGraw。您也可以访问网址 <http://buildingsecurityin.com>。

联系作者

如果有任何意见、建议、BUG 修复和其他问题，欢迎读者通过电子邮件和作者联络。您还可以通过该书的网站 <http://www.exploitingonlinegames.com> 联系我们。

作译者简介

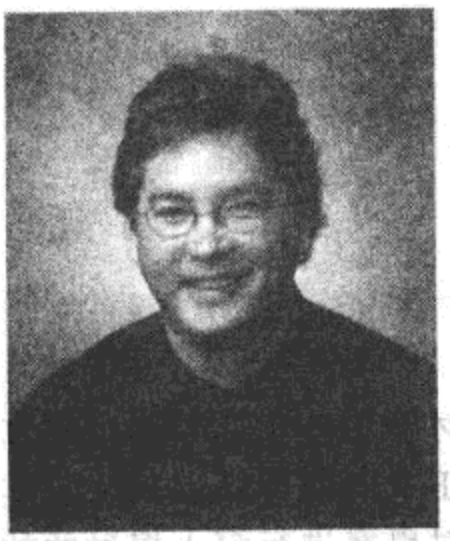
Greg Hoglund　一位依靠自学成才的天才黑客。十几年前，计算机安全还是一个全新的领域，没有任何正式的课程可以用来借鉴和开发成学习课件。事实上直到今天，对黑客进行深入探讨的技术材料依然很少。Greg Hoglund 和许多出生在他那个时代的年轻人一样，在恰当的时间进入了计算机安全领域，并且成就了一番事业。Greg 发现自己能够将黑客行业复杂的信息通过解释和重组让人理解清楚，便自然而然地成为了专业的培训师和作家。

他一直用“做自己的事情，让别人说去吧”来激励自己。Greg 成立了一系列的从事计算机安全方面的公司，其中包括著名的 Cenzic 和 BugScan。他的座右铭也成为了这些公司企业文化的精髓。目前 Greg 正打理他自己创立的第三家公司——HBGary <http://www.hbgary.com>，专门致力于如何在黑客正在入侵和破解的时候迅速地捕捉到黑客信息，其主要的服务对象是美国国防部。

Greg 在计算机安全方面的兴趣点集中在找到 bug 和反编译底层代码上。许多年来，他一直都在与低级的机器代码打交道，直到他后来迷上了破解网络游戏。他对黑客事业的激情因此被重新点燃，近几年来他都一直在研究暴雪娱乐的《魔兽世界》(在本书以及他的其他网络出版物中非常明显)，同时研究的游戏还有《阿瑟龙的召唤 II》、《EVE Online》、《Lineage》系列以及《Vanguard》等。

Greg 曾与人合著了两本技术含量非常高的书籍并且畅销一时。这两本书分别是与 Jamie Butler 合著的《*Rootkits: Subverting the Windows Kernel*》(Addison-Wesley, 2005)，以及与 Gary McGraw 合著的《*Exploiting Software: How to Break Code*》(Addison-Wesley, 2004)。他还同时运营了热点网站 <http://www.rootkit.com>。

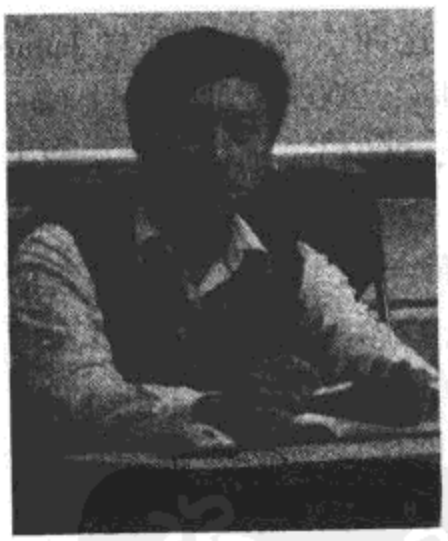
在生活中，Greg 不仅与美国政府签订了百万美金的合同，同时每年还举办了多次著名的“rootkit”开发高级课程。Greg 希望有朝一日能开发一款网络游戏，实战了解一些游戏安全方面的知识。他和妻子 Penny Hoglund 有个 11 岁的女儿，还养了三条狗。不忙于安全方面工作的时候，Greg 喜欢待在他位于加州 Carmel 沿海的别墅里，捣腾自己的小帆船，即使偶尔修理不利让他倍感挫败，但依旧乐此不疲。



Gary McGraw 全球公认的软件安全权威，频频出席各类国际大型研讨会担任主要发言人，为财经、游戏、电子商务等行业的高层、技术和运营专家们提供策略咨询。目前他的身份是 Cigital 公司的 CTO、董事会成员，掌管 Fortify 软件公司的技术顾问团，同时担任 Raven White 公司的高级顾问。Gary 还积极参加学术活动，是维吉尼亚大学和加州大学的计算机系学术顾问，以及印第安纳大学信息学院的顾问团成员。在 2005 年的美国国家网络安全峰会上，Gary 发表了他著名的研究著作《*Security Across the Software Development Lifecycle*》。

Gary 在网络安全方面著有多本著作，其中六本常居畅销书排行榜前茅：2001 年和 John Viega 联名著有《*Building Secure Software*》(Addison-Wesley，2001)，是软件安全行业的奠基之作。这本书中介绍的理念在他的后一本著作《*Software Security*：*Building Security*》(Addison-Wesley，2006)中得以进一步深化和延伸。除此之外的其他著作有《*Java Security*》(Wiley，1996)、《*Securing Java*》(Wiley，1999)、《*Software Fault Injection*》(Wiley，1998)和《*Exploiting Software*》(Addison-Wesley，2004)。他是著名的 Addison-Wesley 网络安全丛书的编辑，出版了 90 多篇文章。他还是 darkreading. com 网站的专栏作家，精彩的言论被广泛引用。他拥有维吉尼亚大学认知学的博士学位，是美国国家电气和电子工程师协会安全隐私小组和计算机安全小组成员，并为协会的月刊编写文章。

在专家、科学家、作家、发言人等这些辉煌的头衔之后，Gary 还有一个头衔不为众人所知，那就是——音乐家。Gary 三岁就开始学习小提琴，大学开始就能够即兴创作，弹起曼陀铃和吉他也是一把好手，经常和乐队一起演奏并曾经录制自己原创音乐的光碟。他的音乐风格广泛，从怀旧民歌到现代爵士样样精通。Gary 和他的妻子 Amy Barley 以及两个儿子一起，住在 Shenandoah 河畔的庄园里，旁边是个野生动物园，从房间的窗户望去，便是美丽的 Blue Ridge 山脉。



姚晓光 npc6. com 创始人，现任腾讯公司游戏项目总监。曾任盛大网络盛锦游戏公司常务副总经理，游戏首席制作人：监制中国第一款回合 MMORPG《幻灵游侠》；监制中国第一款真 3D 商业网游《神迹》；带领研发《QQ 飞车》，创 50 万人同时在线；编译的《网络游戏开发》等书被选为游戏研发教材。

目　录

第1章　为什么选择游戏这个主题

电脑游戏的世界很庞大。全世界有数以千万计的用户在玩电脑游戏。微软公司的报告显示，玩游戏是操作系统平台上第三种最常见的活动，仅次于浏览网页和阅读电子邮件。在2005年，游戏市场的年增长率超过10%，全世界游戏市场规模超过60亿美元。几乎每一个人都喜欢玩游戏。

在1952年，当A. S. 道格拉斯为英国的“爱达赛克”(EDSAC，是英文Electronic Delay Storage Automatic Calculator的简称，第一台存储程序的“诺依曼机器”)制作了连珠游戏时，第一款电脑游戏诞生了。今天，电脑上有各种类型的游戏，从单机游戏(例如《小精灵》)到大型多人在线游戏WoW《魔兽世界》，以及介于这两种游戏类型之间的其他游戏(比如在线扑克以及DOOM《毁灭战士》)。2006年，全世界有一千万以上的玩家在玩大型多人在线游戏(MMO)，并且这个数字以每两年翻一倍的速度增长[⊖]。IDG的分析家估计，美国有290万人在玩大型多人在线游戏。

最流行的大型多人在线游戏莫过于角色扮演游戏(RPG)。RPG十分符合“龙与地下城”的规则。现在，全世界有超出200款大型多人在线角色扮演游戏(MMORPG)。在这些游戏里，玩家居住在虚拟世界里的角色，完成各种任务，和其他玩家成为敌人或朋友。计算机里的数据也拥有真实的价值，在eBay快速搜索“魔兽世界金币”<http//search. ebay. com/world-of-warcraft-gold>，结果可以证实这一点。同时，这也证明了虚拟游戏中的金钱和真实的货币事实上已经在互相流通了！

在线扑克游戏牵涉的金钱甚至比MMO中的更多。每天都有上万人在很多流行的扑克网站中互相对战，玩美式扑克之类的游戏。扑克游戏网站通过收取手续费或是收取赌注抽水比例来盈利。(玩家自己也可以通过击败对手赚钱。一些专业玩家以此为生)。一些比较受欢迎的在线扑克网站的年度利润高达数十亿美元。2005年至少有两家这类公司上市。<http：//www. businessweek. com/bwdaily/dnflash/feb2005/nf20050225_9794_db039. htm>。

所有的电脑游戏都有一个就像“阿基里斯腱”这样的弱点——它们都是计算机软件。通过修改一些配置和参数，以及其他攻击计算机软件的方法，恶意的玩家因而能够作弊。当黑客只是修改你家书房电脑里俄罗斯方块的最高成绩文件时，这似乎没什么影响。但是当这种行为变成在网络扑克等游戏中骗你入圈套来赢钱，或是把你珍贵的游戏道具通过eBay卖给别的玩家来赚钱，或是把虚拟货币卖给第三方交易中心，这件事情就严重了。

每天都有很多人侵入网络游戏。某些是以此为生的罪犯，其他则是普通玩家的恶作剧。另外还有部分玩家是因为太忙，虽然想玩，但没有办法投入很多时间去玩，不得不采用作弊的方式。为了改变网络游戏岌岌可危的安全局势，游戏开发人员和游戏公司必须在软件安全方面做得更好。提高网络游戏安全性的第一步，就是掌握入侵时实际发生的情况。那么游戏是如何被入侵的呢？

这就是本书所要说的——入侵网络游戏。我们会非常明确地、毫无保留地向你展示作弊的

⊖ 关于MMO玩家数量成长曲线图可以参考<http：//mmogchart. com/Chart4_files/Subscriptions_21524_image001. gif>。

代码，从简单的游戏中的宏指令到复杂的系统后门入侵方法。

我们认为，网络游戏就像是软件安全方面一个相当有趣的实验品。软件渗透在现代生活中，已经远远超越出娱乐的范围。但即使只是娱乐，游戏世界中的大量金钱让游戏安全危如累卵。从某种意义上来说，游戏世界中的安全问题已经走在其他网络应用前面了。

1.1 全世界的网络游戏

目前电脑游戏中全球最流行的游戏是《魔兽世界》，熟悉的人都称之为 WoW < http: //www. worldofwarcraft. com > 。WoW 由暴雪娱乐制作发行的(a division of Vivendi)。WoW 是一款 MMORPG，共有超过八百万的不同国家用户[⊖]。WoW 客户端分有英语、韩语、德语、法语、中文等版本，最近还出了西班牙文版本。

在游戏服务器的各个分区，任何时候都有 500 000 以上玩家同时在线。他们的角色居住在一个叫做艾泽拉斯的虚幻世界(请参考 < http: //www. blizzard. com/wow/townhall/worldmap. shtml >)。如图 1-1 所示，到调查截止为止，WoW 的用户达到全球 MMO 用户总量的一半。

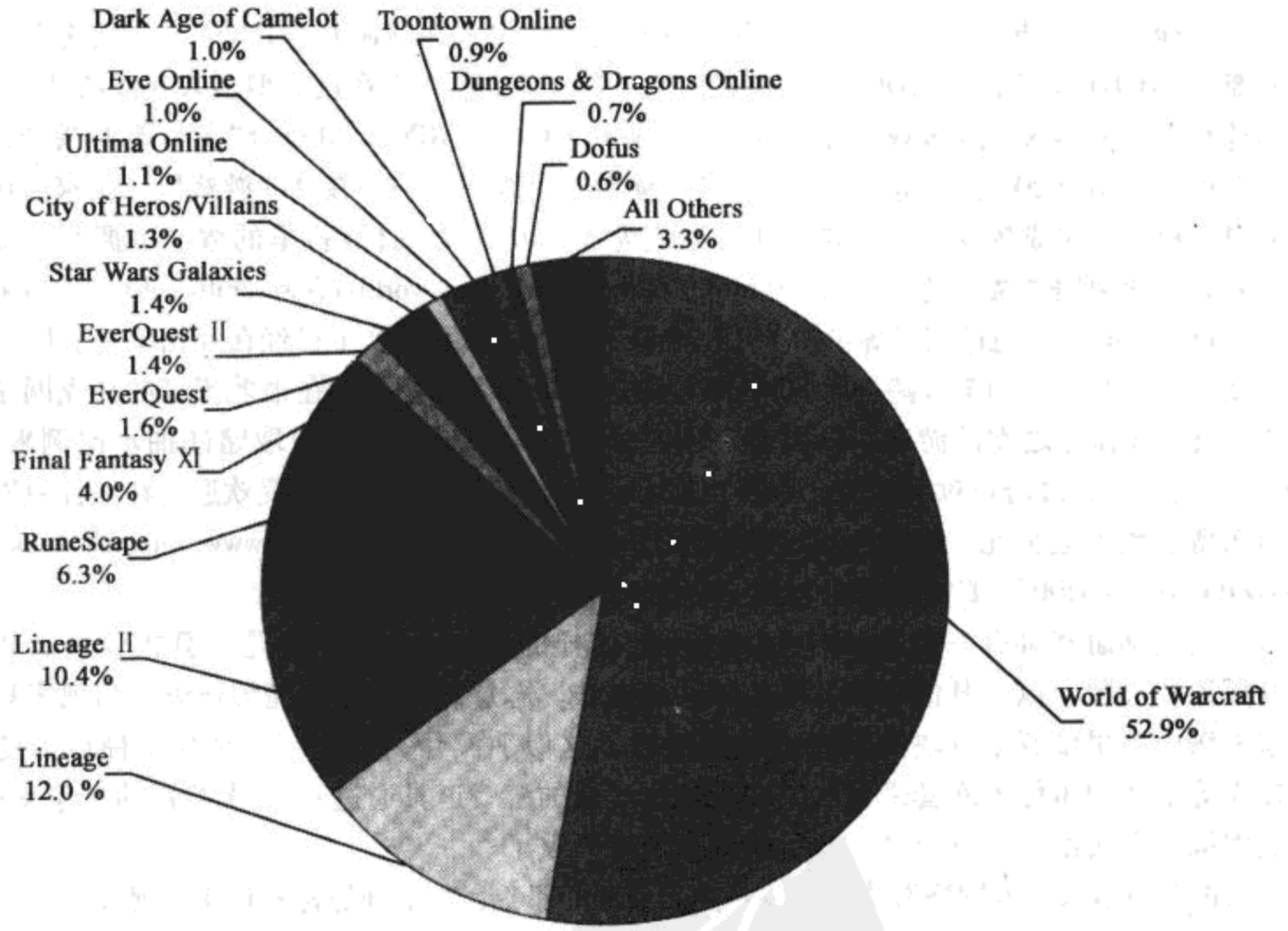


图 1-1 2006 年 6 月各 MMO 的市场占有率(出自 Woodcock，Bruce Sterling。“*An Analysis of MMOG Subscription Growth*，*Version*21. 0” < http: //www. mmogchart. com > ；已获使用许可。)

⊖ 八百万这个数字是来自于暴雪公布的信息 < http: //www. blizzard. com/press/070111. shtml > ，这些用户中，200 万在美国，350 万在中国。

从安全的角度来看，WoW 的两样东西特别有趣。第一个就是，游戏是在中央服务器组上运行的，玩家用自己的电脑通过网络来联接。现在每一个 MMORPG 都采用这种结构了。这种网络结构几乎是不言而喻地存在着固有的安全隐患。第二个是，WoW 的经济体系十分庞大，有很多方法可以把游戏虚拟财产转出游戏换取真实货币。举例来说，一直到最近为止，你都可以在 eBay 上出售你的角色或是一些虚拟道具(见第 3 章)。这让 WoW 相比其他不那么普及的 MMORPG 而言更有意思，也是我们重点说明它的原因。

很多玩家对待 MMORPG 的态度很严肃。在 2005 年 6 月，一名中国上海的玩家由于一把虚拟宝刀“屠龙刀”而谋杀另一玩家。根据 MSNBC 报导“邱诚伟，41 岁，在得知朱某将属于自己的网游《传奇》装备屠龙刀擅自出售之后，将其刺杀。”㊀

专业玩家和游戏打钱公司也出现了，他们寻找游戏漏洞，制作入侵工具，编写宏指令，然后建立起一种称为“农场”(*farm*)的电脑工作室，来创造虚拟财富，然后转移出去或是出售给别的玩家㊁。通过在游戏中获取虚拟货币和虚拟道具，财富被创造出来。农场提供装好游戏的电脑，由组织者带领雇来的人有组织地打钱，再由交易中心把这些虚拟货币和道具出售给其他玩家。在极端的情况下，这会构成有组织的犯罪行为。

有时，玩家会发现游戏中的一些漏洞，然后入侵游戏。这些漏洞可能很有用，加拿大的一个玩家曾经在《星球大战：星系》(一款在美国流行的 MMORPG)中通过复制游戏货币赚取了 700 000 美金。然后他用这些钱买了栋大房子，为了炫耀自己的财源，他的门铃也使用《星球大战》的主题曲。㊂

黑客们对提前进入虚拟世界未开放区域也很有兴趣，这些区域在正式开放前是禁止玩家进入的。入侵软件就是用来强行进入这些区域。

就像是军备竞争一样，游戏的管理者也在监视着作弊行为，有时还会将个别玩家封号。网络游戏正在飞速成长着。正如前面提到过，MMO 的用户每两年翻一番，但是更令人震撼的是美国的在线扑克类游戏的成长速度，到 2006 年底其收入应该超过 10 亿美金(第 3 章有详细说明)㊃。随着更多用户选择玩游戏，更多的金钱牵涉其中，入侵、欺诈、网络钓鱼等行为的动机会随之增强，入侵游戏软件的行为数量也会同时增长。

虚拟世界中已经存在着犯罪，并且种种犯罪将越来越常见。

1.2 在 MMORPG 中作弊的诱惑

如果有机会，很多人会作弊，尤其是他们认为自己不会被抓的时候。网络上可以匿名，这就让作弊更有吸引力。有了快速赚钱的能力，你也同时有了触犯法律的可能。金钱在所有作弊动机里可能排第一位。(第 3 章所有内容都是有关金钱和网络游戏。)

但是，并不是所有的作弊行为都是为了赚钱。那些只是单纯想玩游戏的玩家也有作弊的理由。在 MMORPG 中，想要拥有一个真正高级别的角色，让游戏玩得更有意思，就必须花上好几

㊀ 完整故事请参见 <http://www.msnbc.msn.com/id/8143073/>。
㊁ 关于天堂 2，WoW 等更多的打币工作室的有趣文章请参见 <http://www.1up.com/do/feature?cId=3141815>。
㊂ 更多请参见第 3 章。
㊃ 美国的在线扑克游戏将会好景不长，因为 2006 年美国颁布了相关发令限制了网络赌博行为。

个月的时间认真练级。举例来说，在游戏《EVE Online》中，要练习一个技能需要花上6个月的真实时间。快速升级是作弊的另一个诱因(尤其对生活忙碌的人而言)。这一诱因又触发了对稀有道具或是用来购买这些道具的游戏货币的需求。

1.2.1 作弊代码

作弊有几种模式。有时，作弊可能只是简单地把你的角色转移到怪物的出生点，这样它们每次刷新时就可以立刻杀到。在不那么极端的情况下，你可能会用你的角色反复制作道具或者食物，这也是作弊。如第2章所述，当这些行为是通过程序自动执行的时候，这种作弊行为也变得更加复杂。

有时，作弊与否很难界定。有的人认为，编写简单的宏指令来执行重复性的动作也带有一点作弊的性质。一连串动作的宏指令，从寻找敌人到瞄准再到开火，在网上到处可见(在 <http://www.cheatcc.com/> 和 <http://cheatcodes.com> 上可以看到两个例子)。

更强大的宏指令基本可以让你在做其他事情的同时玩游戏，就像我们在第2章会说明的宏指令那样。大部分玩家认为用这些程序算是作弊。很明显，很多核心型玩家痛恨作弊行为。

在第2章我们会说明很多基本的作弊技巧。更高级的技巧见稍后的章节。

1.2.2 犯罪型作弊

在本章的前面，我们提到过专业打钱打装备卖给其他玩家的“农场”。代练角色的服务也很常见。虽然这些行为从表面看都不像是作弊，但是滥用这类服务对游戏本身的玩法也是一种伤害(详见第4章)。尽管并没有什么针对这类行为的法律，有些人还是会认为这类行为属于犯罪。最糟的情况就是游戏公司和此类服务的提供者(最常被起诉的罪名是破坏游戏和游戏中的经济)之间发生法律纠纷。使用这些服务的玩家通常不会被起诉，只是简单地被封账号。

当作弊是为了赚取大量金钱时，这种作弊就很接近真正的犯罪行为了。就在寻找漏洞、利用漏洞、然后把作弊的方法出售之间，巨大的地下经济诞生了。有传言说，当一款新网游发行的第一年，一些人利用漏洞(秘密复制道具)可以在一个周末赚到几千美元，小团队甚至可以赚到七位数。在线纸牌的赌博游戏的玩家经常会用数据库追踪其他玩家的打牌水平，包括赢了几次，输了几次。然后他们利用这些信息寻找战绩较弱的玩家对战，这样他们就能稳操胜券了。Cigital公司的安全研究员破解了一款在线扑克游戏的洗牌运算规则，可以预先知道发牌的顺序，进而可以预知所有玩家的输赢(见第2章)。

1.2.3 将虚拟转换为货币：从入侵到道具

要把MMO中的虚拟物品转变为现实的金钱是很容易的。我们在第3章中会说到，一些合法的贸易公司例如IGE对道具的来源采取熟视无睹的态度。这就意味着，只要获得道具，然后把它卖给中间人，就可以轻松通过MMO赚钱。

玩家也可以跳过中间人，直接彼此交易。很多传统的玩家都用eBay做交易，直到2007年eBay停止了一切虚拟道具的交易。玩家可以在拍卖网站拍卖游戏道具，先通过PayPal账户收钱，

然后在游戏里把道具交给买家的角色。当然，买家要谨慎。

利用软件漏洞入侵游戏，在游戏的地下经济中，这比游戏货币和虚拟道具更有价值。这种攻击行为可以编写成宏指令，然后卖给打钱农场或是其他想用宏来赚钱的人。用入侵游戏漏洞的方式赚钱简直就像自己印钱一样快捷。

1.3 游戏也是软件

MMO 是游戏，游戏也是软件，所有的软件都会有漏洞。正如本书所言，技术上的漏洞和设计上的缺陷都会被黑客利用(见本书第 5 章)。攻击者的所有常见工具和技术都能用在 MMO 游戏上。我们在本书中记录的很多方法都是极其古老的，但是它们都有一个共同点，就是把 MMO 游戏当成一种软件程序。

1.3.1 基本游戏结构

大部分 MMO 的基本结构都一样。它们通过一个中心数据服务器(有时是多个)让玩家通过网络实时交互。玩家在自己的电脑上运行的软件叫做客户端。客户端软件有时需要购买，有时免费赠送。图 1-2 是基本游戏结构图。

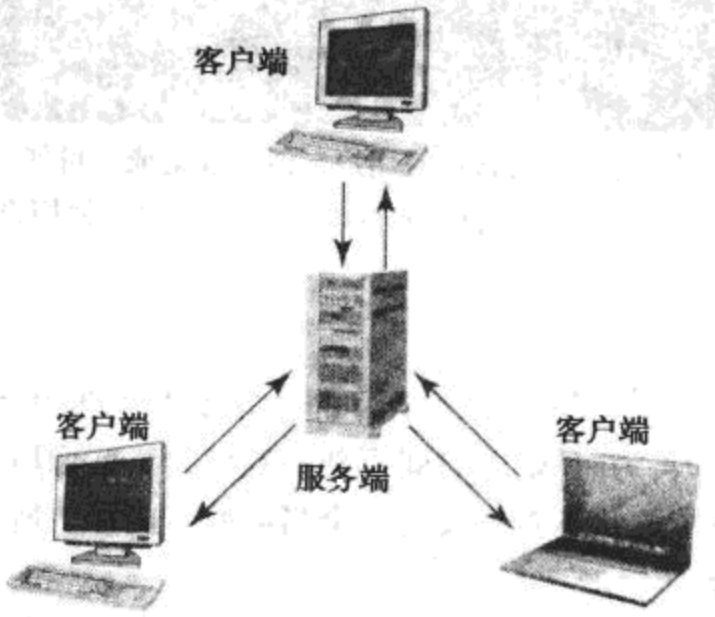


图 1-2 MMO 游戏的基本架构图

要让客户端能够和中心服务器正常通信，大部分 MMO 都是一个请求- 回馈的模型。

1.3.2 游戏客户端

游戏客户端是运行在玩家电脑上的软件程序。客户端接收到玩家的游戏输入，然后通过网络传输给服务器。客户端软件通常会显示出一个虚拟世界，包括场景，其他玩家、信息等。图 1-3 展示的是 WoW 的截图。这就是典型的客户端画面，提供玩家直接进行与客户端的交互。客户端在玩家的电脑上运行，并且与一个中央服务器进行通信。

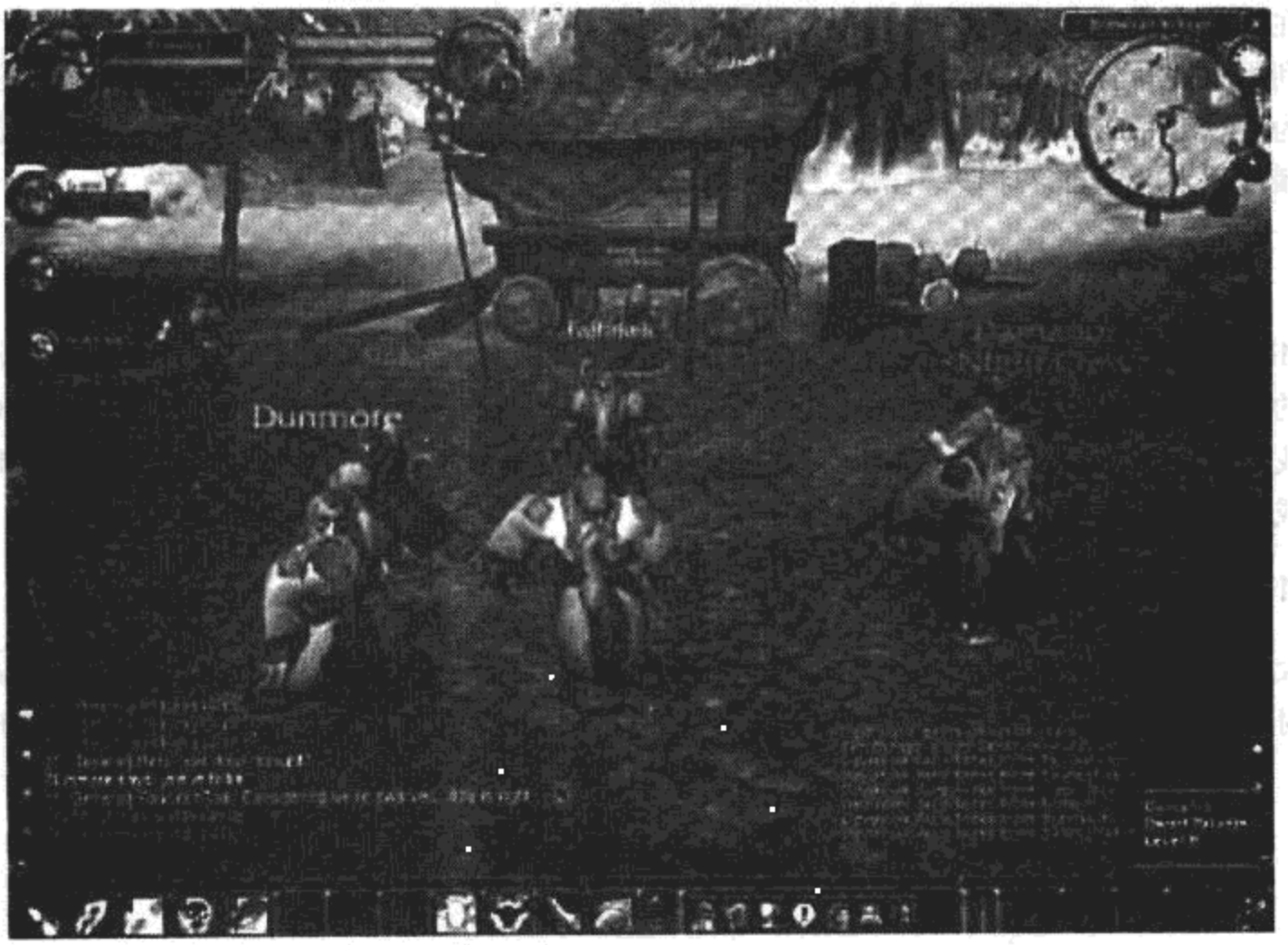


图 1-3 WoW 的客户机软件使用户与游戏互动成为可能(本图来自<http://compactiongames.about.com>，已获使用许可。)

1.3.3 客户端状态

游戏中发生着各种事情，尤其是当成千上万的玩家实时发生互动的时候。游戏有一个中央服务引擎，接收所有玩家输入，并且随着时间更新游戏世界。就像所有电脑程序都有一个状态一样，游戏也有一个状态。系统的状态被定义为一个当前所有内存、所有二级存储、所有寄存器以及系统其他组成部分的集合[⊖]。

问题是，当前的网络不够快，无法让所有的游戏状态都存放在服务器端，并且同步迅速发送到所有客户端。为了解决这个问题，也为了让游戏更顺畅地运行，游戏设计师往往允许客户端保存和管理某些“不重要”的状态。

虽然让客户端来保存状态理由很充分，但是任何客户端上的状态都会带来严重的安全风险。如我们第 2 章所言，客户端状态是被攻击的首要目标之一。

另一个严重的安全问题跟时间有关。对分布式系统的一种高级攻击方式是利用状态的临界时间点[⊜]。当一个系统是以与服务器通信作为基础的时候，“竞态”(临界状态)条件和其他利用时间点的攻击都很有效。最常见的一种攻击 WoW 的方式就是利用服务器例行重启时产生的“竞态”条件。

⊖ 见 Matt Bishop(Addison-Wesley，2005)的《Computer Security：Art and Science》。

⊜ 见 Gary McGraw(Addison-Wesley，2006)的《Software Security：Building Security In》的第 12 章。

1.3.4 和其他软件类似的功能

很多软件包含有客户端和服务器的持续的复杂交互。其中最重要的例子是国债之类的金融机构进行的在线拍卖，以及平常在 eBay 上进行的普通拍卖。在过去的几年里，分布式实时应用经受了苛刻的安全分析，并且为了降低风险而进行了扎实的重新设计。拍卖系统的构架师所学到的一些教训同样适用于 MMO。

1.4 入侵游戏

现在我们已经清楚入侵网络游戏的动机。虽然游戏可以带来大量金钱，游戏的传播也很广泛，游戏软件的设计却从未真正全面考虑过安全问题。不过幸运的是，一些游戏开发者已经注意到了这个问题，并且在设计自己的游戏时开始考虑安全问题。下面是“红客角”网站宗师级的游戏开发者 Matt Pritchard 提出的关于游戏的开发与安全方面的 9 条定律。

1.4.1 谁在入侵网络游戏

入侵游戏的历史几乎和游戏本身存在的时间一样久远。随着 20 世纪 90 年代网络开始普及的同时，网络游戏首次出现了，与此同时，游戏黑客也崭露头角。第一批游戏程序员中有许多都是来自像麻省理工学院学生这样的电脑黑客，他们对游戏是怎样运行的很感兴趣。事实上，这些黑客创建了用来作弊的系统。

在维基百科网站中记载了在游戏《反恐精英》(Counter-Strike)中作弊的历史经典案例[<http://en.wikipedia.org/wiki/Cheating_in_Counter-Strike>。]这篇文章记录了一个自称 Vasily 的俄罗斯黑客(Vasily 是著名的 hook.dll 程序的作者)和一个使用“XQZ”工具(并制作了“XQZII”的木马系列软件)的瑞典大学生。这两个人是第一批游戏入侵者的代表。

供您参考 红客角(破解与反破解交流沙龙)

宗师级游戏开发者 Matt Pritchard 写过一篇名为“How to Hurt the Hackers: The Scoop on Internet Cheating and How You Can Combat It”《如何反击黑客：深度挖掘网络入侵并与之对抗》的文章，在游戏网站 gamasutra 可以看到<http://www.gamasutra.com/features/20000724/pritchard_pfv.htm>。在文章里，他提出 9 条关于网络入侵的定律，供游戏开发者参考。

定律 1：只要有游戏，就会有入侵和作弊。

定律 2：游戏越成功，入侵者也会越来越多。

定律 3：入侵者总是努力不让开发者了解到自己的入侵方法。

定律 4：只要你的游戏在入侵者的本地机器上运行，就不可能安全。

文件不安全，内存不安全，服务应用和驱动程序都是不安全的。

定律 5：简单地把系统隐藏起来是不可能安全的。

定律 6：基于开放式网络的通信都存在可以被破解、分析和修改的天生缺陷。

定律 7：作弊和入侵不可能无害。作弊者在利用漏洞入侵获得好处的时候是非常有创意的。

定律 8：对于客户端-服务器型的游戏，唯一值得信任的就是放在服务器上的程序。

定律 9：诚实的玩家总是希望游戏没有漏洞，作弊者则相反。

以上转载已获得许可。

现在，很多黑客组成派别，合作攻击游戏。其中一个组织叫做 TKC(Teamkill and Cheat 的缩写 <http://www.tkc-community.net>)。这些组织松散的玩家在他们内部开发和交易入侵工具，他们的作为大部分是为了自己的声誉。这些组织对自己作弊的方法保密，这样他们就能持续利用这个方法。公开的作弊方法都会出现对应的防范措施(包括本书中提到的例子)。

1.4.2 为什么入侵游戏

黑客制作作弊工具的理由很简单就是：金钱、名声、荣誉。

在“黑客角”里，作弊作者 Vasily 阐述了他作弊的动机和理由。在网上这样的访谈和声明随处可见。例如 TKC 的成员在 <http://tkc-community.net/Main/modules.php?name=TKFAQ> 的 FAQ 中曾经提到很多入侵游戏的原因。他们在文章开头这样说：

作弊哲学

就和现实生活一样，在网络游戏中也会有各种各样的人，有着各种不同的爱好。一些人满足于游戏开发者做出的游戏，而另一些人则不是。我们不同意这样一个观点：如果玩家对一个游戏不满意，就应该去换一个游戏来玩。如果一款游戏未能达到一个人的预期，或是失去了它从前的吸引力，那么用技术来修改游戏可以为玩家创造更好的游戏体验。利用编程中无意留下的漏洞也是基于同样的信仰。

接着他们列出了为什么有人想要入侵网络游戏(并加入他们的组织)的原因：

- 加强游戏体验
- 友谊
- 挑战自我
- 消遣娱乐
- 天生爱恶作剧
- 为了引起注意
- 不墨守成规
- 拍马屁
- 报复
- 获得优势
- 广告
- 偶然
- 窥探

随着时间流逝，网络游戏的经济发展起来，并且和现实中的经济结合起来，作弊的初始动机从获得声望变成了赚钱。早期的作弊者只是为了显示他们的黑客技术，树立起技术方面的威望。有的人甚至会试着去帮助开发者(据说大多遭到了拒绝)。大部分早期的作弊手法一旦被游戏公司发现就会立刻被根除。因此，最近的作弊手法都会在各个团体之间保密，并小心地互相交易。

1.4.3 如何入侵游戏

本书的一个目的就是尽可能详尽地说明如何入侵游戏。也就是说，书中会有大量的代码。我们不得不展示大量的代码，因为仅仅用术语来说明某个技术往往太过含糊，无法让人满意。要完全理解游戏入侵技术，做点坏事是无法避免的。

现在我们举个例子来说明。要入侵一个真正的游戏，首先需要一个目标。我们在这里多次选择 WoW《魔兽世界》作为入侵案例有两个原因。首先，WoW 是最受欢迎的 MMORPG，并且大部分玩家都不知道和他们一起玩的一部分人如何在作弊。第二个原因是，已经有很多针对 WoW 的经典游戏入侵技术，我们可以轻松地直接拿来说明。比如，如果有开源游戏入侵工具(WoW 就有很多针对型工具)，我们就会展示出来。

供您参考 黑客角

下文出自 Vasily 的访谈(hook. dll 的作者，用于入侵早期的 CS)，其中他谈到自己的动机。原文在 <http: //www. cached. net/? go = main/featurearticle/single/96 > 可以看到。> styles. no：作为 CS 作弊纠纷的背后主导者之一，你曾经有过罪恶感吗?

Vasily：说实话，我不觉得我应该感到罪恶。我没有把任何作弊方法公布出来，我只告诉了几个有名的人……，是的，我在这件事中所处的位置大家都知道，我不反对在电脑游戏中作弊。我完全不在乎这一点，我认为这完全不是个问题，如果有人认为这是问题，那他就坐下来重新思考一下他的人生。有很多比作弊更严重的事情需要你与之战斗。恐怖主义、种族主义、歧视、穷困、失业、犯罪。根据 UN 的报告，每天都有上千个孩子饿死。以色列、巴勒斯坦以及其他热点地区的人民每天死于战争。城市和国家受到恐怖分子袭击。你还想告诉我在电脑游戏里作弊是很严重的问题，应该不惜一切立刻解决? 你一定是在开玩笑。

接下来让我们快速说明一下本书中覆盖到的技术。在第 2 章我们会以黑客的基本方式的内容作为开始。首先我们要介绍的是采用最基本技术的自动打怪。第 2 章的中心是为了让大家了解一些专业术语，以及一系列的游戏入侵步骤。我们囊括了机器人、GUI、代理、内存入侵、调试器、不安全随机、后门。第 2 章的内容还包括有防范作弊的对策，以及游戏公司为此愿意做到什么地步。我们会研究一些反监控代码，这样当你在电脑上运行游戏客户端的时候就会有直观的感受。

接着，在第 5 章中会第二次出现大量的技术性内容，这些内容是关于游戏 bug 和缺陷，以及为什么它们会成为攻击游戏的最佳目标。我们在致力于游戏安全的时候，发现一件很有意思的事，我们所探讨的安全隐患问题在游戏之外的领域也越来越常见。事实上，MMORPG 的缺陷已经预兆了即将发生的事。大规模分布式软件系统总是有非常复杂的时间和状态的管理问题，因为它们使用中央服务器来实时处理上千个交互进程的状态。在美国国家科学院，网络游戏已经成为软件密集型军事系统的重要案例，用于理解和研究[⊖]。

第 6 章的内容会从技术角度进一步深入，详尽说明游戏客户端的入侵。我们会围绕游戏进行一系列技术的分类说明。在第 6 章会有大量的代码，这样我们说明的技术会更

⊖ 更多关于此内容请参见 <http: //www7. nationalacademies. org/cstb/project_ producibility. html > 。

清楚。第6章唯一的问题是它就像一个游戏入侵技术大杂烩，不过第7章解决了这个问题。在第7章的开头，我们会把第6章中介绍过的大量技术结合成一个复杂的机器人程序。第7章会一步一步深入机器人原理，并在结尾介绍以调节器为基础运行的机器人代码。

本书中最艰深、也是最黑暗的内容在第8章。第8章全部都是关于反编译游戏的内容。如果你想要根据第6章和第7章中出现过的技术建立起新技术，就必须掌握反编译技术。第8章初步介绍了反编译，告诉你一些我们经过多年发展获得的有用的启发。

从某种意义上来说，对游戏彻底的转化和所谓的游戏mod版本(改造后的版本)才是游戏入侵的最高境界。第9章会告诉你如何做到这些。再次声明，我们会用大量代码来说明我们想要表达的意思。

尽情享受本书带给你的乐趣吧，只是当你必须要逃离键盘和编译器才能让自己的神经恢复正常的时候别感到吃惊。

1.4.4 到底有多少起游戏入侵事件

没有人确切地知道网络游戏入侵有多猖獗。即使游戏公司知道，他们也不会告诉你。不过，他们会选择性地禁止一些作弊嫌疑者登录服务器。在2005年，暴雪娱乐因为这个原因将超过1000个WoW玩家封号[⊖]。Valve游戏公司也曾驱逐过玩家。但是要重新回到游戏常常很简单，只需要买一个带有不同序列号的客户端就可以重新开始了。

游戏公司对于作弊的态度本来是很含糊的，除非这种做法严重影响到了非作弊玩家正常游戏。在第3章，我们会以《网络创世纪》为例，说明对作弊的容忍极限是如何被通货膨胀打破的。有时一些游戏公司会以防作弊机制作为特色吸引玩家，因为很多玩家痛恨作弊者，会为了避开这些作弊者改玩别的游戏。不过有些情况下，危害比较小的作弊是可以容忍甚至是受到鼓励的(例如farm打钱农场)。(别忘了打钱公司的农场主必须为雇员的账号付费才能赚钱，也就是说游戏公司还是获得了收入。)综上所述，对作弊的容忍度是游戏公司必须仔细斟酌的一件事。

1.5 最大的课题：软件的缺陷

在过去10年里，软件安全作为一门重要学科出现了。我们从网络游戏的攻击和入侵中学到的教训可以直接应用到其他软件中。由于有着上千万人在玩MMO，那些隐藏的软件安全问题更容易浮出水面。

MMORPG很重要，原因之一是MMORPG软件走在其他现代软件的前面。网络游戏的形态就是现代软件的未来——大型分布式系统，几十万用户通过一组公共服务器进行实时交互。时间点和状态都是可以被直接利用的安全缺陷，它们带来的问题在网络游戏中时有发生，并且在其他类型的软件中也越来越严重(见第5章)。我们对于安全工程和大型分布式系统还有很多需要学习。MMORPG正在挑战极限，并且带给我们非常重要的启示。

很多以软件为生的大型企业已经开始着手研究在最初就内建安全机制，以此开发更好的软

⊖ 详见网址<http://www.gamesindustry.biz/content_page.php?aid=7392>。

件(见第 10 章)。最知名的软件安全行动就是微软的“可信赖计算计划”(Trustworthy Computing initiative)。大量的金融机构也在进行软件安全行动，但他们通常不会公布自己在做什么。

如果你想要学习更多的软件安全知识，可以去看看 Addison Wesley 软件安全系列的其他书籍。如果你只能选择一本，那么就选择《软件安全：内建安全机制》<http://www.swsec.com>。

MMO 和其他网络游戏完全可以比现在的情况更安全。忽视问题，或是打击威胁那些讨论入侵游戏方法的人，这些手段都不能确保制作出更安全的游戏。如果你关心自己玩的游戏的安全性，现在就可以联系这款游戏的开发公司，问问他们采取了哪些和软件安全有关的措施。

第2章　游戏黑客的基本方式

游戏从20世纪70年代独立的TV游戏机发展到20世纪80年代的电脑(PC-GAME)之后，盗版软件就一直是影响电脑游戏行业的严重问题。游戏制作者为了反盗版自然是竭尽所能。在过去，游戏制作者在软件上采取各种加密和保护措施，让游戏更难于被破解。这么做是为了阻止猖獗的盗版行为，想玩游戏的人必须去买一份正版游戏软件。虽然最后这些游戏还是会被破解，但至少有的时候防范措施还是可以让破解版本的出现拖延上几天甚至几周。对游戏公司而言，这种拖延就可以带来真实的利益，因为盗版拖延一周，正版就可以卖出几十万美金。

反盗版措施为单机游戏带来了利益，玩家到商店买游戏光盘，然后安装在自己的电脑上。但是现在事情不同了。很多现代游戏转移到了网上，并且随着游戏机和PC联网的出现，这种趋势发展得越来越快[⊖]。这就意味着，现在游戏公司有两种利润来源需要保护：零售渠道出售的游戏客户端利润，以及联网支付点卡带来的巨额收入。

在本章中，我们会详细说明大量的常用作弊技巧，并且探讨一些用于防范盗版和作弊的新技术。不幸的是，一些最新的安全监控技术对玩家暗中构成了隐私的侵犯。

2.1　网络对抗盗版

要防止类似非法拷贝这样简单的盗版有一个很简单的方法，就是不要发布任何可以被拷贝的东西。也就是说，如果你的游戏大部分都在一台中央服务器上，它就不容易被拷贝。基本上，游戏公司采用了这种策略来预防一般的游戏破解方法(回忆一下第1章中的客户端－服务器模式)。现在游戏几乎全都要求玩家玩游戏的时候登录官方的服务器才能进行游戏。这些网络服务器至少会检查游戏客户端本地版本(玩家电脑上运行的部分)的校验码是否合法。

当然，网络游戏需要一个游戏账号，也就是说，要玩游戏就必须要有某种用户或玩家认证。用这种方法把一个游戏和一个特定玩家联系起来，比之前范例中的方式要更加清晰。追踪玩家行为是与作弊的斗争中的一项重要策略。

我们在第1章中简单提到过，游戏是个大买卖。例如，开发《魔兽世界》(下文简称WoW)的美国暴雪公司，不仅获得每份游戏客户端30美金的收入，而且玩家每个月登录游戏服务器还要再付14美金。WoW拥有800万每个月付费的用户，你可以算算看其产品带来的收入规模。

2.2　另一方面

破解服务器当然并不简单。但是不久前，很多聪明的破解开发爱好者发现可以自己架设新服务

⊖ 在2006年，特别流行任天堂Wii，当年营业额翻倍，必然加速了这种趋势。

器让玩家登录，这些服务器可能是免费的，这样就避免缴纳游戏月费。问题是，这算盗版吗？

有3个程序员编写了暴雪公司的服务器软件 BnetD 的开源版本，暴雪因此起诉了他们，并且获得了胜诉，详见第4章。

2.3 作弊的诀窍与技巧

在网络游戏中作弊的方法有很多。有些方法根本不需要太多关于计算机编程方面的技术。在在线扑克游戏里纠结一群人一起对付一个普通的玩家，在这样的范例中只需要打一个电话就能作弊。另一方面，有的作弊方法则需要深厚的编程技术。

在本章中，我们会向你介绍一些基础的作弊概念：

- 制作外挂机器人
- 利用用户界面(UI)
- 使用代理
- 对内存进行修改操作
- 利用调试器
- 预测未来

这些方法的最终成果现在可以直接在作弊网站上在线买到。Pimp My Game 网站 <http://www.pimpmygame.org/> 就是其中之一。这个网站和其他很多同类网站一样，有一段申明：

> 我们为我们的用户带来更多的机会：提供所有常见 MMORPG 和我们支持的部分 FPS 游戏(第一人称射击类游戏)的入侵工具、机器人、黑客工具、宏指令、补丁以及作弊和帮助工具。从我们的下载区可以获取这些工具，在论坛可以进行讨论。你会在你的游戏中取得更大的成功！

当然，比起购买这些“入侵工具、机器人、黑客工具、宏指令、补丁以及作弊和帮助工具”，我们对它们背后隐藏的东西更感兴趣。

2.3.1 制作外挂机器人：自动进行游戏

如果你在 Google 上搜索“网络游戏，机器人”(“online game bots”)，你会看到数百万条搜索结果。大部分结果来自于出售机器人的网站。那么机器人到底是什么呢？

外挂机器人是一种独立的程序，可以替你操作游戏自动杀怪物升级等(或是结合成游戏的一部分)。机器人这个词最初来自于第一人称射击的电脑游戏(FPS)。我们所说的机器人就源自游戏中模拟另一个玩家的机器人。你可以在玩象棋游戏的时候和一个机器人对战，或是在一款 FPS 游戏(例如《DOOM》)中和一个电脑控制机器人对战。

今天，机器人这个词已经广泛应用在大量程序中，从简单的常用动作键盘映射，到像一个真正的玩家一样遵循制定的简单规则自动进行游戏的人工智能(AI)。在 FPS 游戏领域里，玩家用机器人做出超人动作(例如，精确瞄准)。在 MMORPG 的领域里，玩家利用机器人自动操作令人厌烦的那部分游戏内容。我们会在本章稍后的内容中介绍一个 WoW 中控制角色的宏指令范例，让这个被控制的角色成为一个机器人(至少在外挂运行的时间内是)。机器人执行特定任务几乎

总是比人类强。或许它们的逻辑性比较强，或许它们在操作游戏角色时对游戏状态的了解比玩家更清楚，或许它们只是做重复性任务不会感到厌烦。但无论机器人的程序设定要做什么，它们都令角色拥有了绝对优势。

机器人甚至可以用来在游戏里抢劫。下面一段文字出自《新科学家》杂志：

> 在日本一位玩家因为在网络游戏《天堂2》中使用机器人击败并抢劫玩家角色而涉嫌虚拟抢劫，现已被捕。这些抢劫到的虚拟财物已被兑换成为真实货币。

虽然不如在大型游戏里面那么流行，但是网络扑克机器人也经常用来在扑克游戏中为使用者赢钱。扑克机器人虽然没有达到专业级别的水平(这就涉及要制作出可以通过图灵测试的合理AI)，但是利用一些算法规则在新手区里赢钱已经绰绰有余。

总的来说，机器人的口碑很复杂。有些严肃的玩家担心机器人成为游戏的癌症，认为它们会毁掉游戏和游戏行业。另一些人则把机器人视作非常有用的工具，可以把玩法中令人厌烦和重复劳动的部分变成一个电脑程序。还有些人把机器人当成重要的谋生手段。

游戏公司经常会部署技术性和法规性的反机器人外挂的策略。有时，他们会对角色属性进行统计，当某些特定数值超出范围时，就会引起注意(例如，当一个角色在一个小时内财产翻4倍的时候就会给它做上重点标记)。另一个常用对策是对角色提问，看他的回答是否像是人类的回答。《韩国时代报》曾经报导，在MMORPG《天堂》中，至少有150名GM监视着游戏中是否有机器人，他们的工作是发现这些机器人，然后禁止玩家使用。报道中还说，在2004年到2006年4月间，大约有500 000个账号因为使用机器人而被暂停或者删号。

2.3.2 利用玩家界面：键盘、点击以及色区

现在的游戏都有着出色的界面。看图2-1中的WoW截图，思考其中的界面。<http://xunegamers.tripod.com/id3.html>收集有各种令人印象深刻的不同UI。

如图所示，UI包括了屏幕中玩家可以通过使用标准输入来互动的部分，有按钮、文本窗口以及图片。通过操作使用UI，你可以进行游戏，UI就是为你报告当前发生的事情的窗口。

作弊者可以利用UI来作弊。我们假设，一个游戏有三个操作按钮，A、B和C，你只能自己手动点击。在一些游戏公司的概念里，如果你安装了一个自动操作的软件工具(例如鼠标精灵键盘精灵等测试工具)，可以自动操作鼠标点击在x轴和y轴的位置上，来触发这些按钮，你就是在作弊。

在很多情况下，软件用户授权协议(EULA)和相关强制机制规定了你使用软件的方式。也就是说，你被允许自己点击按键，但是不能通过程序来按键。关于EULA的更多内容在第4章有详细说明。

类似这样控制人们进行游戏的方式看似有些极端，但请思考一下自动进行游戏对游戏中经济造成的冲击。在大部分情况下，自动玩游戏是通过使用特殊工具和宏指令这样的脚本来实现的。举例来说，WoW中，怪物会周期性地在特定地点出现。你可以很轻松地写一个宏指令，让游戏中的角色站在指定的坐标点，在怪物每次刷新出现的时候自动杀怪(然后获得经验值和金钱并且自动捡起物品)。当然，你可以自己手动做这些事，整天等着怪物出现，但如果怪物每10分钟才刷新一次，你就要经历一个漫长而无聊的夜晚了。为什么不写一个宏指令来代替你等着呢?

最后，问题就出来了，为什么自动操作这样乏味而重复性的行动会被当作作弊呢？

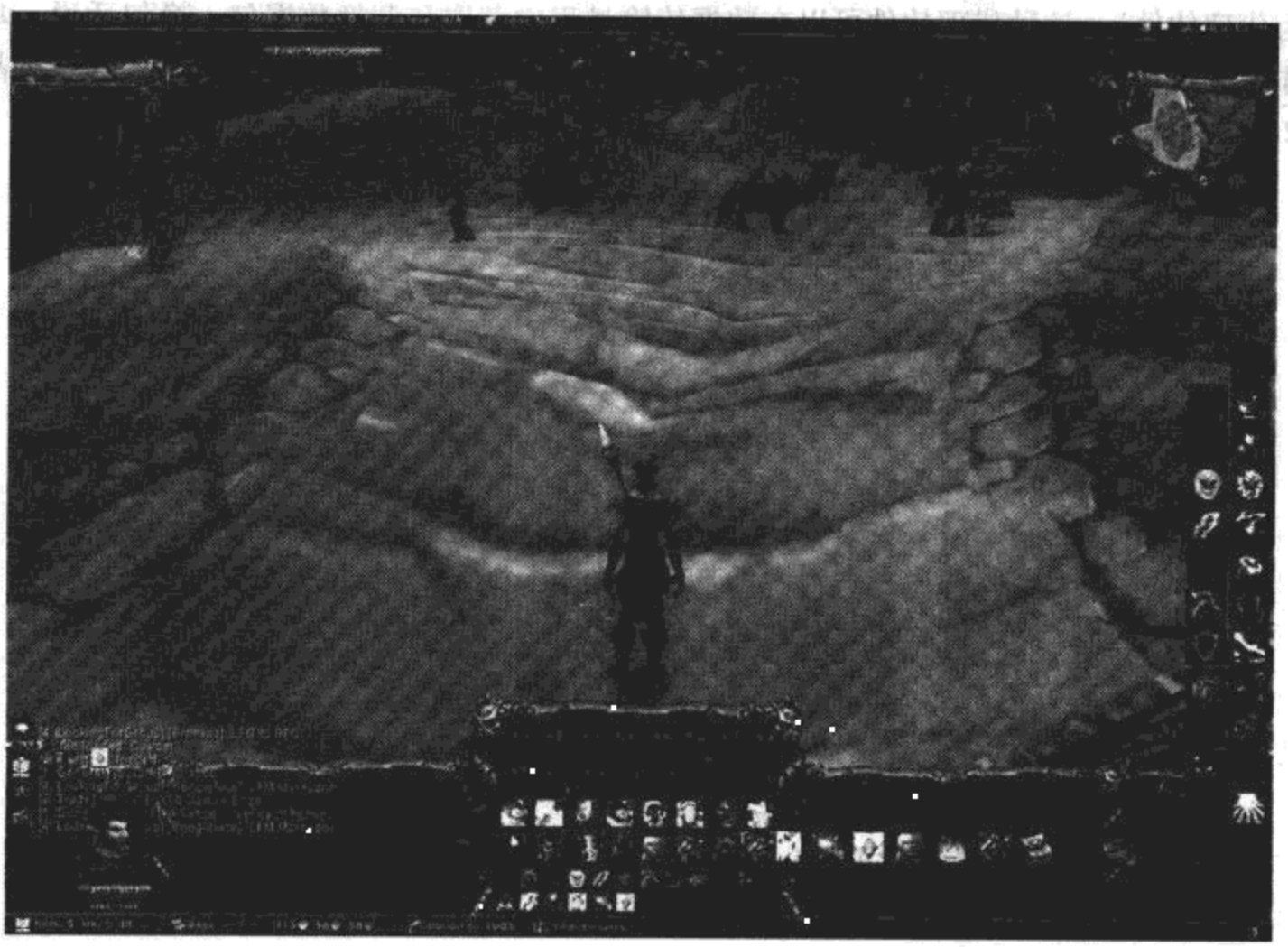


图 2-1　一张 WoW 的截图，展示了网络游戏中用户界面的美术风格

WoW 和许多类似的 MMORPG 都有着许多令人苦恼的重复性玩法，玩家为此而用这样一个词来形容：推磨。意思就是，角色整天做着重复性的讨厌事情来赚取经验，就像骡子一圈一圈推磨，天天把谷子磨成面粉。出于某些原因，玩家喜欢这种自我折磨，并且愿意为此每个月花 14 美元。为什么？

结论是，这是出于一种人类根深蒂固的心理状态，玩游戏就好像有了双重人生，并且这样的推磨带来了经济回馈。不论何时你杀死怪物，都能得到游戏中的金钱和其他奖励，例如更多的经验、技能以及最后的升级。

如果你写了一条宏指令来自动模仿键盘鼠标来操作 UI，代替你做这种推磨人的“工作”，你就可以去工作或者睡觉，稍后回来的时候，重复性杀怪得到的金钱和经验已经在那里等着你。因此，这些宏指令为你赚到了游戏中的金钱的同时还提高了角色能力属性，并且不需要你付出实际的精力去做那些让人觉得枯燥无味的事。这是多么好的主意呀！事实上，这主意真的很好，已经有成千上万的玩家在这么做了。甚至有一个专门的词汇来形容用这种方式进行游戏的玩家——“农场主”。

本章我们介绍的一个简单的机器人就是通过使用 UI 来操纵角色推磨的。

2.3.3　使用代理软件：截取数据包

通过 UI 和游戏交互是一种很直接的作弊方式，代码并不难。当然，还有很多更复杂的作弊

方式。其中一个方法就是在游戏客户端和游戏服务器端之间使用“数据包截获类代理软件”(以下简称为代理软件)。这种代理软件可以在数据传输过程中截取和调换数据包。换句话说，基于代理的作弊方式在信息安全领域被称为“中间人”截获攻击(我国称做“封包工具”。——译者注)，见图2-2(注：国内称作“封包工具”)。

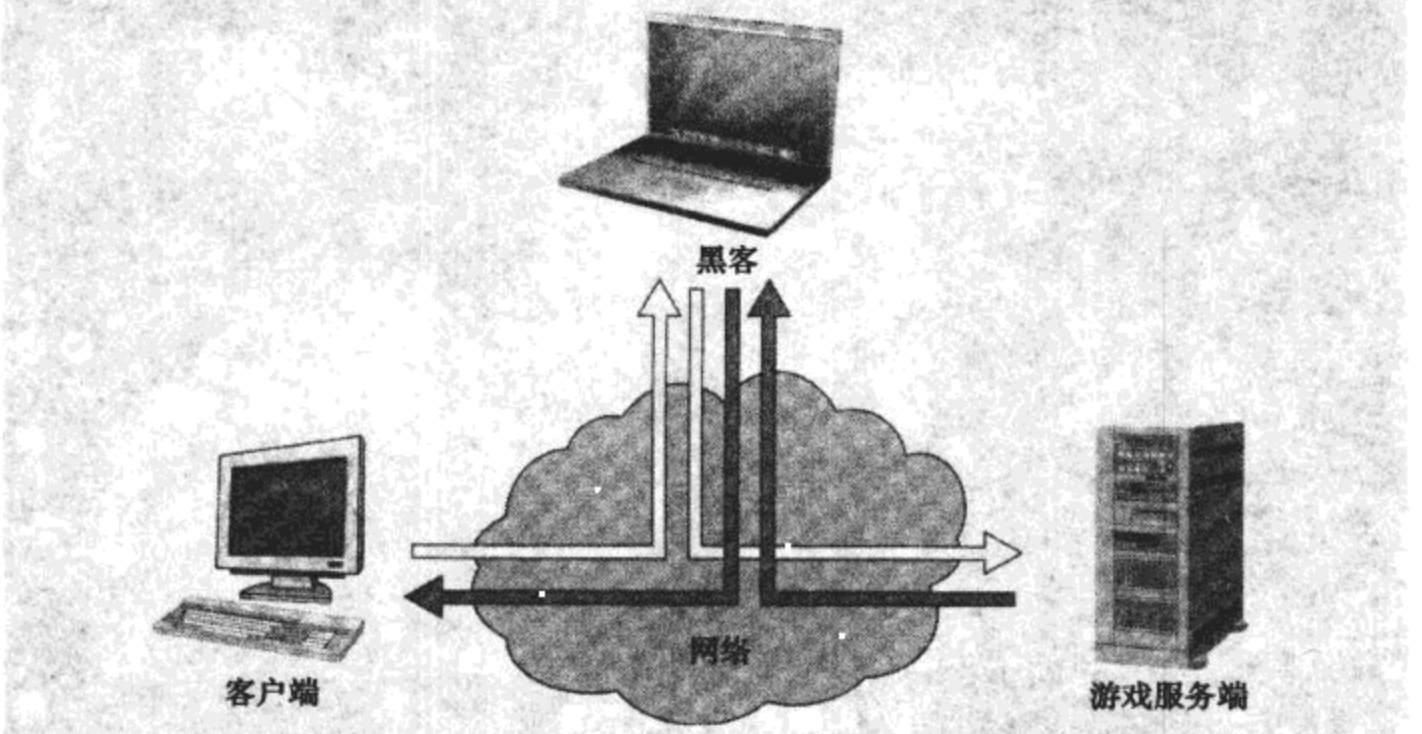


图2-2 “中间人”攻击的示意图。黑客可以在客户端和服务端之间进行自由的操控，掌握了通信的交通枢纽

要实施这样的攻击，方法有很多种。监控网络传输是其中一种。破解程序和它所使用的系统动态链接库(DLL文件)是另一种方法。基本上，目标程序发送的信息不论发送到哪里都是可以通过这种方法截获的。

代理攻击法有着悠久的历史。第一批以网络为基础的代理攻击法中有一部分应用于FPS类游戏。在这些游戏里，有相当一部分关于游戏状态的数据在客户端和服务器之间传送。有时，这些数据不会显示给玩家看到，但这些数据是玩家在游戏中必须用到的数据信息。代理软件作弊的方法是：搜寻网络数据包，分析它们，并且调整一些玩家本不该知道的游戏细节参数。在经典案例FPS游戏的CS(反恐精英)中，代理作弊法的运用结果是：提高瞄准水平(射击类游戏的重要特征)、穿墙行走、透视、隐身。

在FPS游戏中使用以代理为基础的方法作弊时要十分小心。因为那些使用者都希望不被发现，包括在复杂的实际情况中。如果在局域网游戏聚会中作弊被发现的话，可能会被其他玩家辱骂或者痛打！代理底层的自动瞄准机器人在设计的时候要非常小心，要在实际情况中应用而不被发现，就可能牵涉到统计学上的模糊计算，这样才能模拟非完美的瞄准射击。复杂的代理攻击会试着截获并且替换服务器和客户端之间传输的数据包。给数据包加密是一种很直接的对策，这样代理截获到的就只是毫无意义的杂乱信息。然而，加密会延长计算时间，这会降低游戏体验的即时感，而真实的快速反映的游戏体验是网络带宽占用的主要消耗之处。加强安全性总要牺牲掉某些东西，它永远不会是免费的。

要使用代理软件做攻击，作弊者需要为代理软件配置好所使用的服务器地址。在大多数情

况下，FPS 的客户端完全在另一台机器上运行，通过代理软件和服务器联接，这种情况实际上和标准的网络嗅探一样，唯一的不同就是，数据包在传送过程中可以被操作。

要真正实现作弊的功能就要为游戏构建一个有用的模型，可以追踪传输的数据包。这个模型可以追踪玩家来实现作弊的功能。当一个标记有类似“向那个自大的笨蛋开火”这种指令的数据包通过时，代理软件可以分析出谁在什么位置，谁必须要死，然后实现这些指令。可能还需要对武器准心稍作移动，这样才能准确瞄准要打的那个笨蛋。(记住，武器或是玩家的移动都要看起来自然合理，如果一个人忽然后空翻 3 次 360 度转体然后紧接着射击，很明显就是在作弊。)

代理攻击对于非 FPS 游戏同样有效。事实上，大部分针对 MMORPG 的比较复杂的攻击方式都出自于类似的想法，通过对网络数据包的截获和改写操作影响游戏世界。在第 6 章你会看到更多这类攻击方式的说明。

2.3.4 操作内存：改写数据

在前面我们已经说过，通过代理软件的介入，来操作游戏状态是一种类似置身“事外”的作弊技术。游戏软件本身并没有被直接操作，只是它的输入和输出被入侵的信息替换操作了。更加直接的干预型作弊技术会进入游戏程序内部，改写内存数据，更换数值，通常还会直接在客户端中扰乱游戏状态。

这类操作的一个明显标准就是在电脑中控制图像渲染的驱动程序。FPS 游戏再次引导了此类作弊之路。CS(反恐精英)是一款经典的 FPS 游戏。和很多游戏一样，它需要大量使用驱动程序来控制图像、音效以及网络连接。作弊者把目标瞄准在 OpenGL 和 Direct3D 的驱动上。一种被称为 XQZ 的 CS 黑客工具是第一批更改驱动的工具之一。CS 的开发者们创建了一种称为“作弊必死”(Cheating-Death)的反作弊工具，可以侦测出 OpenGL 是否被替换。作弊与反作弊之间的又一次军备竞赛就此开始。

从历史上来看，这些驱动之所以如此容易被操作，其原因在于微软的 Windows® 没有一种简单的方法来判断一个驱动是否合法。随着 Vista 操作系统和它的兼容性认证驱动的发行，情况终于有所改变(但是千万不要相信这就是保障安全的万全之策)。在 Vista 之前，就算是提供了全部的访问权限，要查出一台电脑上是否有作弊软件都是很困难的(事实上，即使有系统后门技术也查不出)。

另一种比较常见的客户端操作方式是用经典的 Hook(钩子)技术入侵 DLL 文件或是其他系统软件库。这些技术有很多都是软件入侵常用的技术(详细内容见本书入侵软件章节)。所有的现代操作系统都使用了动态运行库(Win32 使用 DLL，Linux 和 Mac OS X 使用标准 UNIX 共享对象库，例如 glibc. 等)，同样地，这些都很容易受到 Hook 攻击。

在有的游戏里，客户端的通信通过客户端 EXE 执行程序来调用 DLL 实现，这种架构下信息都很容易被截取。现代软件几乎总是大量使用外部的软件支持库，而这会直接导致操作内存类作弊的猖獗。

作弊与反作弊的军备竞赛于是再次开始，从经典黑客技术(例如 CS 专用的 loaddll 库)到反作弊 Hook 检验，到更加隐蔽的 Hook，再到 Hook 侦测机制。

2.3.5 利用调试器：断点

调试器是另一种入侵软件者常用的攻击工具。没有什么比能够中断运行中程序的核心级调试器更加有效了。很多作弊者利用调试器来设置断点或是其他触发点。这些触发点能够寻找特定信息(例如，一个秘密的按键)，或是使用特殊功能(例如，简单的反编译 DLL)然后实施各种恶意行动。

在第6章中我们要说明的现代攻击类型经常会用到复杂的中断－驱动状态操作。更多内容见后续章节。

2.3.6 预测未来：可预知的和随机的，如何在在线扑克中作弊

很多游戏都含有偶发性元素，扑克游戏是其中表现得最明显的。问题是，要通过计算机的工作方式创造出不可预测的随机是非常困难的。很多软件安全方面的工作最后都变成了致力于提高随机率(这对密码系统也是很重要的)。

有个很好的例子可以说明在实际中伪随机的数字生成器是如何被破解的。1999 年，Cigital 的软件安全部门发现了 ASF 软件有限公司的“德州扑克”游戏在运行中有致命的缺陷。作弊的玩家通过入侵可以实时计算出每个人手中的牌。这就意味着，使用这种入侵方式的玩家知道每个对手中的扑克牌以及桌上的翻牌(即，每一轮加注后放在台面上的翻转过来的牌)。作弊者每次都会知道何时该跟注，何时该弃牌。恶毒的攻击者可以用这种方式诈骗无辜玩家们的真实金钱而不会被抓获。

这个致命的缺陷就在每一局用来生成扑克的洗牌运算规则里。讽刺的是，洗牌的代码就在网络 FAQ 中公开展示着，用以说明游戏是多么公平，以此吸引玩家(后来该页面被撤销)，因此它的代码甚至不需要破解或是猜测。

在代码中，每次扑克生成前都会调用函数 **randomize()**（一种 Pascal IDE），来为随机数生成器重新生成种子。用 Delphi4（注：Borland 公司推出的 RAD 开发工具）写出的代码调用系统时钟自午夜开始累计的毫秒数作为随机数生成器的种子。这就意味着，随机数生成器的输出结果是可以轻易预测的。结果就是，可预测的随机数生成器成为了非常严重的安全问题。

ASF 软件使用的洗牌运算规则以固定顺序的扑克牌作为开始，然后生成一系列随机数来对牌进行重新排列。在玩真实的扑克时，扑克会有 52!（大约 2 的 226 次方）种不同的排列方式。而 32 位随机数生成器的种子必须是 32 位数字，也就是说，种子只有大约 40 多亿种的可能性。因为牌会重新初始化，生成器会在每次洗牌前重新生成种子，根据这种运算规则只能产生 40 多亿种洗牌顺序，即使种子的数字范围比系统时间要大。但是，40 亿种可能的洗牌顺序比起 52! 实在小得多。

这种具有缺陷的运算规则采用 Pascal 功能函数 **randomize()** 为随机数生成器选择种子。这个 **randomize()** 功能选择的种子出自于午夜开始累计的毫秒数。一天仅仅有 86 400 000 毫秒。因为这个数字要用来作为随机数生成器的种子，那么可能出现的洗牌顺序就减少到了 86 400 000 种——比 40 万又少得多。

简言之，这个洗牌运算规则有三个主要问题，任何一个问题都足以摧毁系统。

- 伪随机的数字生成运算规则采用的种子数字范围太小（32 位）。
- 伪随机的数字生成运算规则未加密。
- 代码种子的来源随机范围太小（事实上经常会重新播种）。

系统时钟种子带给 Cigital 成员一个新的想法，就是进一步缩小洗牌顺序的可能性。通过让程序和服务器的系统时钟同步来产生伪随机数，他们可以将洗牌顺序的可能性降低到 200 000。在这一步迈出之后，系统就已经被破解了，因为要搜索如此少量的洗牌顺序实在是微不足道，一台电脑的 CPU 即可以实时完成这项工作。

Cigital 的工作人员用来攻击程序的工具需要首先知道一副牌里的 5 张牌是什么。根据已经知道的 5 张牌，程序搜索出可能出现的几十万种洗牌顺序，推算出哪一种最符合当前情况。在德州扑克的这种情况下，程序首先输入了作弊者手中的两张牌，再加上前三张公共翻牌，在前 4 轮下注中，程序就知道了 5 张牌，足以用来确认确切的洗牌顺序（在游戏中过程中实时获得）。

图 2-3 是入侵工具的软件界面。左上角的参数显示框是用来保持时间同步的。右上方的游戏参数框是用来输入 5 张牌然后开始搜索的。这里的截图是在程序确认出所有牌之后截下的。作弊者知道谁有什么牌，剩下的翻牌是什么，以及谁最终会赢。

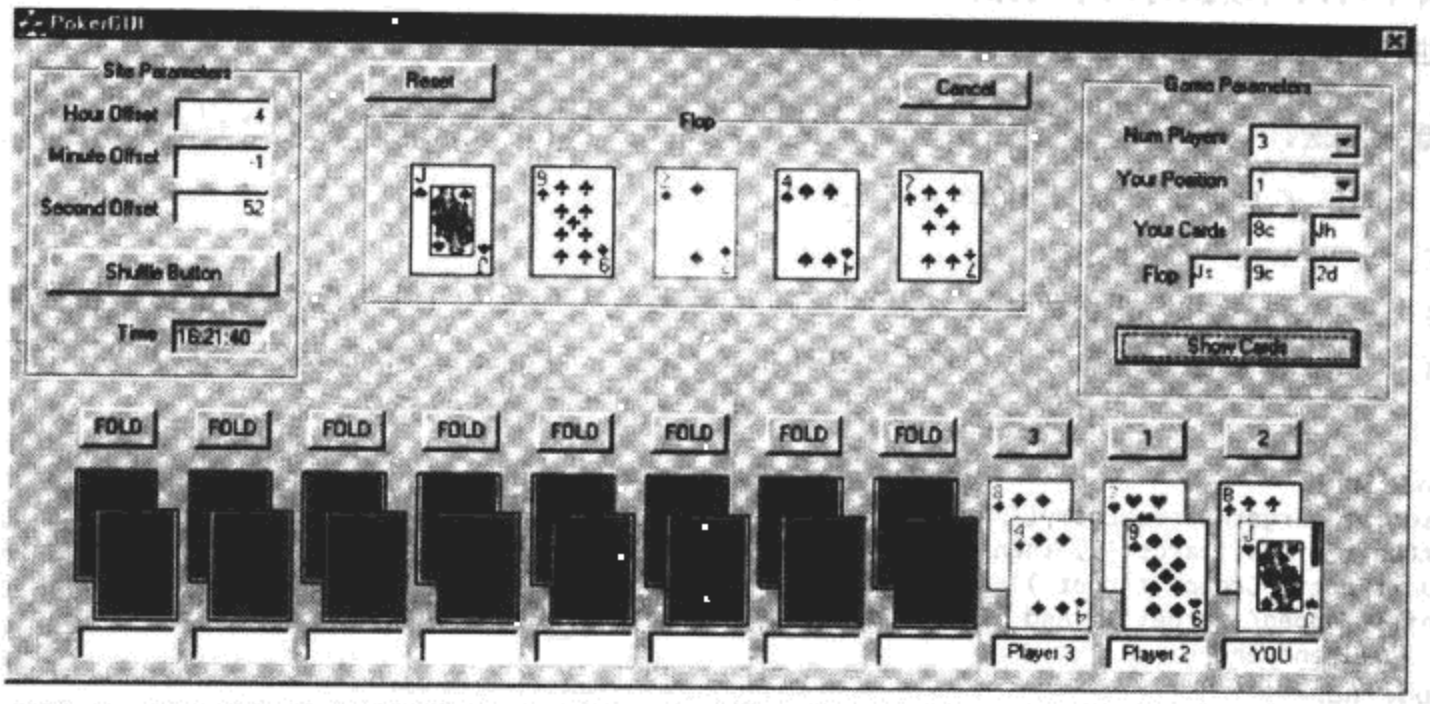


图 2-3 通过破解洗牌的运算规则，Cigital 的在线扑克入侵工具可以预测出每个人拿到的牌

程序一旦知道了 5 张牌，就会持续生成洗牌顺序，直到找出包含有 5 张符合顺序的已知牌为止。由于函数 **randomize()** 功能是建立在服务器系统时间的基础上，要在一个合理的范围内猜出开始的种子并不是很困难（你猜的越接近，要搜索的洗牌顺序就越少。）。在找到一次正确的种子之后，要让入侵程序和服务器的时间保持一致只需要花上几秒钟。这种“后同步”让程序可以确认出随机数生成器所用的种子，然后只需要一秒钟不到就可以推算出接下来所有游戏的洗牌顺序。

AST 扑克软件确实是很容易受到攻击。然而，大部分使用线性序列的伪随机数字生成器（例如 **rand()**）都会受到这种攻击。这种攻击归结起来就是，在足够的信息全部泄露之前，攻击者需要多少输出结果。即使你只是在比较小的范围内使用随机数，并且抛弃一部分调用 **rand()** 函数的输出结果，但是通常只要几次输出结果就可以把内部状态泄露无遗。

除了扑克还有很多游戏会用到随机。任何一款游戏如果可以完全被预测，那么只需要一个简单的程序就可以推断出接下来会发生的事情。随着越来越多的金钱投入到游戏中，这类随机问题也成为了非法攻击者的印钞机。

2.4 机器人大阅兵

鉴于“机器人”软件是一种无处不在的作弊工具，有必要花点时间好好举例介绍一下。机器人工具分三大类：自动战斗宏（一般用于 MMORPG 大型网游）；自动瞄准机器人（基本只用于 FPS 第一人称视角射击）；扑克机器人。

2.4.1 自动战斗宏

很多的游戏，特别是大型网络游戏，都使用与游戏底层程序交互的脚本程序语言，这些脚本可以把数个基本操作合并成一个宏。有的游戏并不包含脚本语言，但是和游戏一起运行的第三方程序提供了脚本运行空间。

脚本可以自动执行简单的操作。下面是针对 WoW 魔兽世界游戏的一个脚本例子，你可以很轻易地在网络上找到无数这样的脚本程序。

供您参考 宏

自动寻怪 最后更新时间：2006 年 6 月 22 日
在上次 WoW 上补丁时更新

这个宏指令会搜索你周围的敌人，直到找到一个未被攻击的敌人（非部落或是联盟）。在推磨打怪时非常有用！（在搜索过 30 个敌人后会停下，这样在找不到敌人时不会让你的电脑空悬着。）（这个宏指令默认你是联盟，如果你不是，只要设置“部落”替换掉“联盟”同样可以正常运行。）

```
Code Start:
/script TargetNearestEnemy();local i = 1;
if(UnitExists('target')) then while(UnitExists('target') and
(UnitFactionGroup('target') == 'Horde' or
UnitIsTapped('target')) and i < 30) do TargetNearestEnemy();
i = i+1;end;end;
Code End:
```

下面是另一个例子，用于《星球大战：星系》的宏。

供您参考

更新时间：2004 年 8 月 2 日

这是一个 AFK 战斗宏指令，专为游戏内的宏系统设计。如果你想进一步使用，可以用 AC 工具运行 HAM 检验宏（检查你的 HAM，在你需要的时候对你进行治疗）。

要运行这个宏，首先要设置一个机器人，围绕一个出生点在方形区域内巡逻。调整你的摄像机角度，让它面对出生点，然后去看电视或是睡觉吧。

宏指令：

```
/ui action targetGroup0;
/follow;
/ui action cycleTargetOutward;
(if you're a Brawler, insert /follow here)
/attack;
(insert specials here)
/pause XXX;
(amount of time to kill it)
/follow;
/pause 2;
/loot;
/harvest hide;
/macro AFKCombat;
```

这个脚本看起来比第一个有趣，因为它不需要玩家操作。事实上，用简单的话来说，很多玩家认为使用主动操作的宏和使用不需要玩家操作的宏是两种完全不同的行为。他们认为使用不需要操作的宏是一种作弊行为。

有些游戏把使用宏视为非法（见第4章，防止欺骗的法律行为）。微软的《阿圣龙的呼唤》（Asherons' Call）有如下使用条款：

> 在游戏软件中禁止使用未认证的第三方软件和宏工具。特别是当你离开你的键盘时，你的角色还能在游戏中取得经验点和道具物品的第三方软件（这种软件常被称为外挂）。在管理员出现时，或是管理员试图与你交谈时下线，将被认定为使用外挂。

Lin2Rich <http://lin2rich.com/> 是一种广为人知的常用宏系统。Lin2Rich 被认为是高端的机器人程序环境。它运行在游戏程序的旁边（例如：你把它直接和游戏客户端串联起来），根据设定直接向游戏服务器发送数据包。Lin2Rich 可以设定为，当一定的事件发生时，向服务器发送一组指令。举例来说，如果你在特定时间按某个按钮，或是在特定时间地点开始休息，你可以使用作弊程序来向游戏服务器发送相应的命令包。鉴于其工作方式，很难通过监测认定使用 Lin2Rich 的游戏玩家。

2.4.2 自动瞄准机器人

自动瞄准机器人应用于 FPS 游戏中。它的想法很简单，瞄得比你的对手准就意味着胜利。在 FPS 中对手被打中头部会一枪毙命（俗称“爆头”），使用程序取得超人般的瞄准精度，是最常见的作弊方式（尽管有些射击游戏有内置的自动瞄准工具）。

自动瞄准也有不同的偏好，有的仅仅是辅助瞄准，有的瞄准后射击，有的还能操作人物运动、瞄准、射击。最初的自动瞄准机器人，用于《雷神之锤》（Quake）和 CS。你可以在网址 <http://www.digdilem.org/ut/aimbot.php> 发现关于《虚幻竞技场》（Unreal Tournament）的有趣 FAQ 和作弊工具。

供您参考 黑客角：ZelliusBot 自述文件

ZelliusBot UT2004 版本 1.0

——简介——

本自动瞄准工具有很多有用特性：应用于 UT2004 Demo 版和 UT2004 零售版 ver. 3186。其基本特性是瞄

准，让你更容易找到哭闹的孩子，更方便地拥抱家庭成员。

——安装——

1. 把文件 ZelliusBot. u 和 ZelliusBot. ini 拷贝到你的 UT 的 system 目录下。

2. 打开 UT2004. ini 文件，搜索［XInterface. ExtendedConsole］，修改 ServerInfoMenu 的值为 ZelliusBot. ZelliusServerInfo。如果 ServerInfo 栏不存在，就加一行 ServerInfoMenu = ZelliusBot. ZelliusServerInfo。

3. 打开 User. ini 文件，把“serverinfo”绑定到你选定的按键。如：z = serverinfo。

4. 登录游戏，按设置好的 serverinfo 键，如果一切正常，ZelliusBot 状态将出现在游戏画面的 HUD（抬头显示区，即屏幕上方）。

——使用方法——

使用数字键盘上的上下左右箭头作为快捷键，绑定不同的命令。你可以像 AutoTauntKey 一样，在 UT2004. ini 中调整快捷键。

把 bShowKeyNumbers 设置为 true 可以在操作图标上看到不同的数字提示。进入游戏，注意数字提示，按正确的数字键。

——注意事项——

本工具在 UT2004 中无法瞄准交通工具。

在 UT2003 的环境下编译。

在游戏中试验或是参考 uscript，了解怎样使用。

当敌人绕过围墙进入你的视野时，出现子弹打在墙上的问题，加大开火延迟时间。

每场结束时，退出工具。如果你在加载新地图时仍然出现无效游戏文件的错误，在控制台上敲命令“reconnect”（重新连接）。

——感谢——

感谢 tenbucks，moonwolf，play2win，lamer，spunky 和其他测试本工具并反馈信息的人。

感谢［ELF］HelioS，帮助我们编写 ping 码、HUD 以及提供贴图（来自于 helios ut 机器人工具）。

感谢 DrSiN［Epic］，在 UT2004 中留下这么多漏洞。

希望你们喜欢这个工具！

~ Zellius

以上来自于一个叫做 ZelliusBot 的，通过编辑《虚幻竞技场》的初始化文档来运行的，流行的自动瞄准机器人工具。黑客角引用了 ZelliusBot 自述文件（readme 文件）中的一部分。在大多数情况下，自动瞄准常常和别的功能一起打包，比如穿墙。XQZ 是一个早期的 CS 自动瞄准机器人工具。XQZ 把几个功能整合到一个简单易用的软件包，一起发布。XQZ 在一开始代替 OpenGL 与游戏连接，然后再连接到 OpenGL。XQZ 包含的特性使得用户可以在局域网聚会中使用它而不被发现。到现在，CS 游戏中仍旧充满了各种 XQZ 的衍生品。最初的版本中，XQZ 在用户按住某个特定的键或是鼠标时被调用。按下按钮，自动瞄准机器人就控制瞄准目标。

使用自动瞄准机器人时要小心，以确保你表现出来的超人的能力不会引起别人的注意。举例来说，使用自动瞄准机器人工具，遵循完美的轨迹精确地瞄准目标，反复击中对手的同一点，肯定会被人抓住。这就有必要对瞄准和射击进行削弱。就像前面提到的，更先进的自动瞄准机器人，为自己的行为加入了统计噪声改进算法。

2.4.3 扑克机器人

既然在线扑克的最高利润已经达到了 10 亿美元，那么会出现很多自动玩扑克的机器人就一

点也不奇怪了。幸运的是，扑克是一种相当有难度的游戏，并不容易自动操作。然而，扑克机器人确实存在着，并且还在迅速进步。

当然，如果扑克机器人真的到达了非常棒的境界，那么扑克游戏本身也就被破坏了。没有人愿意一直输给机器人！特别是当金钱也卷入其中的时候。

据大量传言，新一代的复杂的扑克机器人通过一些规则可以打败新手。甚至有些传言还说，平均水平之上的玩家也会被它打败。当然，和其他类型的游戏一样，游戏公司希望能够阻止玩家使用机器人。他们使用玩家监测机制，寻找玩法可疑的人物，有时还会对游戏做出改变，以求战胜机器人。

然而，还是有一部分人通过商业化出售扑克机器人来维生。在 WinHoldEm 网站 < http：// www. winholdem. net > 可以找到大量的机器人程序，从 25 美元的手动分析标准包软件到 200 美元的机器人自动游戏版本应有尽有。

接下来，我们再次陷入了标准的军备竞赛。简单的机器人流行了，反机器人程序相应出现，机器人升级，然后反机器人继续。当然，大部分在线纸牌游戏不允许使用机器人，但是要强制推行游戏规则还是很困难。

在线扑克游戏的整体现象让很多安全专家都糊涂了。最大的问题在于玩家之间的勾结。玩家私下交流牌面，这样的作弊很难被发现。从法律的角度要禁止这种行为很容易，但要强制推行这条禁令是不可能的。

有人认为，无法控制扑克机器人和私下交流会造成在线扑克的毁灭。我们只能等着瞧了。

2.5 潜伏（数据虹吸）

通过观察其他玩家玩游戏，学习他们的行为，就可以收集到大量有价值的游戏数据。在体育运动中，这当然是一条真理，通过观察你的对手来了解他们，这对你的帮助是无价的。在网络游戏中，这同样是一条真理，从 MMORPG 中的行为到在线扑克中的打法皆可适用。

2.5.1 网络统计

像 Thottbot 这样的数据库可以帮助使用者收集和利用游戏的统计数据。举例来说，Thottbot 可以告诉你某样道具的确切坐标地点，为你提供地图路线，并且让你知道在到达之后得到这样道具的几率有多大。Thottbot 尽可能地收集其他玩家的信息，然后经过合理的组织再发布出来。更多关于 Thottbot 的信息参考本书第 3 章。

2.5.2 扑克统计

尽管美国通过的立法将要大量缩减在线扑克的市场规模，它也只会比 MMORPG 更大而绝不是更小。

有很多第三方卖家制造并出售软件包，帮助用户分析拿过的牌的数据，并且建立起一个玩家和他们的玩牌趋向的数据库。认真的扑克玩家会用这些统计来审视自己在玩牌时有什么弱点，同时寻找其他玩家的弱点。这类工具的使用很常见，所有认真对待的玩家（包括所有专业级的在线扑克玩家）都会用到它们。

很常见的，理论和数学分析发生了冲突。Jason Swanson 写过一篇很有趣的论文，题为“数据统计和在线扑克” *Mathematical Statistics and Online Poker*，这篇文章可以在网址 < http：// www. math. wisc. edu/ ~ swanson/instructional/stats_ poker. pdf > 看到。事先声明，这篇文章包含了

真正的数学!

图2-4出自一个典型在线扑克第三方软件的GUI。这款软件帮助玩家弄明白扑克统计法，以及接下来应该做什么。它会为德州扑克生成统计表、胜率、可能拿到的扑克以及有用的战术信息。

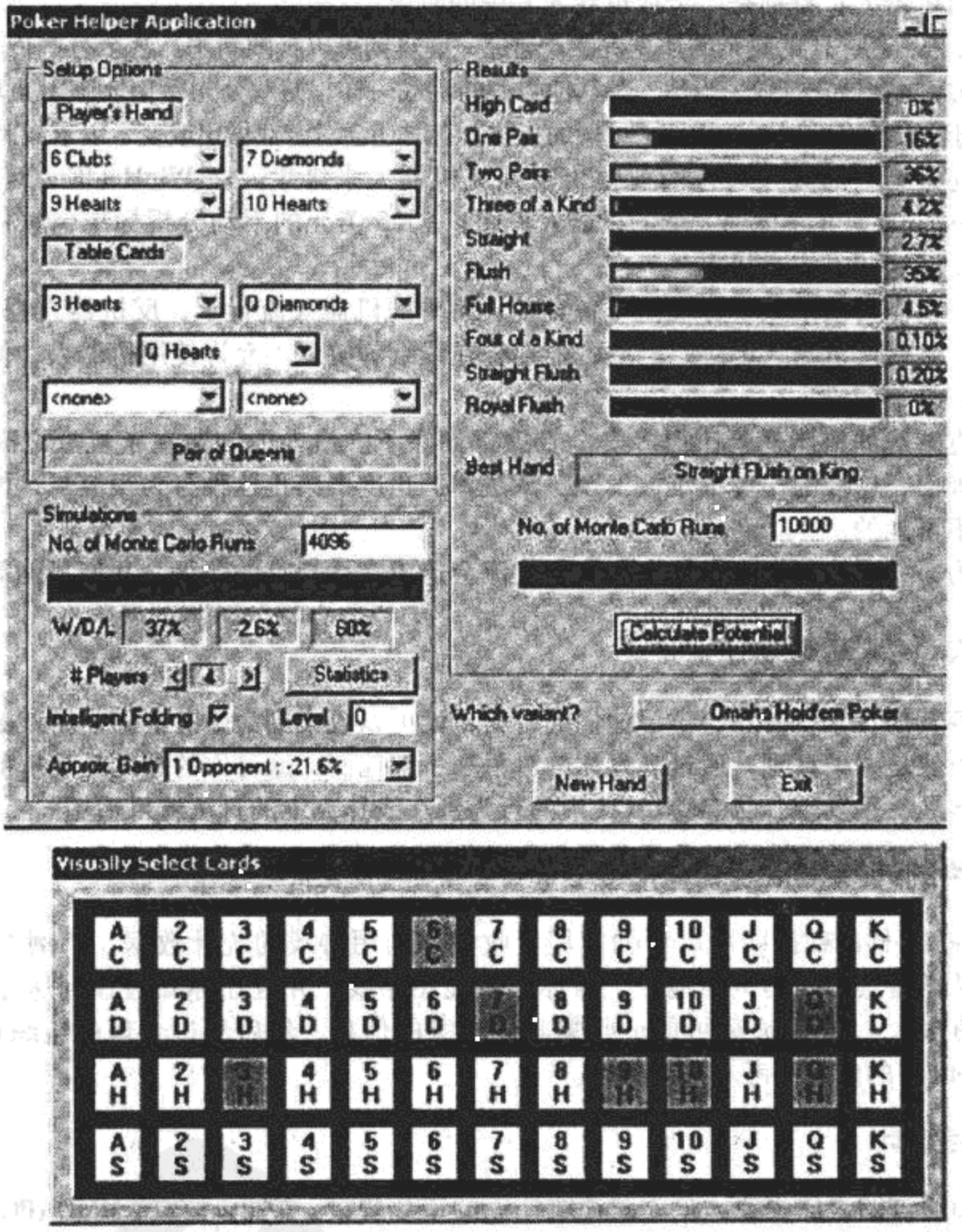


图2-4 一款在线扑克帮助软件（出自<http://www.frayn.net>；已获得转载许可权。）

在任何一款卷入了金钱的游戏中，你都应该搞清楚你的对手是不是用了外挂工具。在线扑克作弊和状态追踪必然会做得越来越好。或许有一天，机器人会好到能够一直打败最棒的人类。

2.5.3 拍卖操作

作弊者还喜欢在拍卖中作弊，尽管网络拍卖和网络游戏几乎没有什么关联性，但在“快速敛财”这一点上网络拍卖和网络游戏倒是有着很多共同之处。思考一下网络拍卖中的作弊者所

采用的策略，或许可以让我们对所有的作弊者有一个大致的了解。

在拍卖中最明显的策略或许就是用“托”。找人假装去拍一样东西的目的是为了让这件东西最终的价格上涨。当然这么做也是违法的，在美国是一项重罪（你看，托的存在远远早于网络拍卖）。像 eBay 这样的拍卖网站都会追踪 IP 数字，来试图阻止假拍。他们还会持续监视拍卖的进行，寻找是否有可疑的部分。听起来是否很耳熟？

另一种网络拍卖常见的作弊手法是中断网上的交易，改为线下交流。这种行为可以通过电子邮件或是其他渠道进行。在拍卖中串通就和在扑克游戏中串通一样，是不公平的，而且也是很难被察觉的。

第三种骗术是在拍卖即将结束的时候中断付款过程。只要匆忙发一封邮件给竞拍获胜者，装作自己是卖家，要求对方付款，有时就会有可怜的竞拍者上当，把钱付错了人。可追踪的付款系统能够减少这种骗术的发生。

最后，还有一种方法可以在抢拍时作弊，在拍卖即将结束的激烈争夺战中尤为适用。竞拍人（A 和 B）可能在最后几分钟互相竞拍，这时 A 可以对 B 做出攻击，让 B 无法登录服务器。这只是个简单的技巧，A 可以不断登录 B 的账号并且连续失败，B 的账号就会被临时锁定。

2.6 正式开工

到现在为止，我们已经介绍了“游戏作弊 101”的很多常见的攻击方法。我们甚至告诉你简单的宏指令是怎样操作的。但是，说明这些东西是如何操作的，和弄明白它们的结构是完全不同的事情。要朝着这个方向前进，我们首先呈上一个 WoW《魔兽世界》的简单宏指令。

警告：使用该宏指令属于作弊行为，违反了游戏规定。如果使用，你的角色可能被禁用。注意，这些类型的工具十分常见，在各类游戏中每天都会有人使用。作弊并不只是发生在 WoW 里。

AC Tool：巨集

在网上有大量流行的巨集工具，包括有宏记录工具（要花钱）、AC Tool、AutoHotKey、AutoIt3.0、LTool-0.3，以及 xautomation。这里提到的每一种软件都被很好地收集在 < http：//wiki. atitd. net/tale2/Macros > 。

AC Tool 特别流行。你可以从 < http：//www. actool. net > 免费下载。安装了 AC Tool 之后，你就可以使用它的宏语言为 WoW 生成宏指令。在 AC Tool 网站的 FAQ 里这样描述这个工具：

> AC Tool 是一种十分有效的工具，让你可以预先列出一系列的按键和鼠标点击，并在稍后传送给《阿圣龙的召唤》（Asheron's Call，一款经典的 MMORPG）。按键和鼠标点击的记录称为宏，或是脚本。

下面是一个简单的推磨宏（名为 Hoglund's WoW_ Agro Macro），是用 AC Tool 设计出来的。这个宏控制一个角色在野外等待怪物出现，然后杀死它们。我们会在代码中间插入注释，来帮助你理解当前的情况。

```
// ------------------------
// hoglund's WoW_Agro Macro
// ------------------------
//RESOLUTION: 1024x768
//PUT ALL FILES IN YOUR AC TOOL\MACROS FOLDER.

SETActiveWindow World of Warcraft
delay 4 sec

// put all your 'globals' here
Constants
       gPCHealth = "NoValue"
       // the current health of the PC ( HIGH | MEDIUM | LOW |
CRITICAL )
       gMobHealth = "NoValue"
       // the current health of the current targeted mob ( HIGH |
MEDIUM | LOW | CRITICAL )
       gPCPosture = "Standing"
       // current/starting posture of the PC character
       gMachineState = "START"
       // global machine state, core of the system
       gSelectedTarget = "NoTarget"
       pc_name = xanier
```

代码的第一部分设置了一些全局变量，主要是存放当前选择的目标，自动控制的角色的名字以及状态机当前的状态。这段脚本的实现是依照一个很多宏指令都遵循的标准结构，也就是说，它被设计作为一个状态机进行操作。这就意味着，脚本在自动操作角色时包含有当前唯一的状态变量。状态可以标记有类似“攻击”、“治疗”、或是“逃跑”这样的标签。同一时间只能激活一种状态，从一种状态转到另一种状态时必须遵循规则。

```
// hot keys
keyExecAttack = 1
// set your F1 key to attack before using this MACRO
keyExecPickup = {F2}
```

上面这部分脚本简要地说明了需要用到哪些键来自动操作游戏。当点击 F2 时，角色拾取道具，当按下数字 1 时，角色攻击。这种按键绑定在这样的宏指令里很常见。只用按键和鼠标移动来操作软件的简单宏算不上是对游戏的直接入侵。更高级的作弊方法远远不是这么简单。

```
// screen coordinates
coord_MobHPMin              = 262, 50
coord_MobHPFull             = 370, 50

coord_SafeHP                = 200, 50
//if green at or above this mark, you're fine
coord_HalfHP                = 146, 50
//half hp if lower than this mark
coord_LowHP                 = 116, 50
//low hp if at or lower than this mark
coord_DeadHP                = 96, 50
//you're dead :(
coord_temp                  = 0,0

// colors
color_AliveGreen            = 150
color_AttackRed             = 147
```

```
// temp stuff
TempTarget                  = NoValue
//holds the value for target before placing into list
tCount                      = 1
//used to traverse the targetList index
Total                       = 1
//used to find the ListCount
redDifference               = 0
blueDifference              = 0
greenDifference             = 0
numAttacks                  = 0
Returned                    = 0
aCount                      = 0

End
```

上面这部分脚本非常令人着迷。它制定了屏幕上 x 轴和 y 轴的位置。不仅如此，它还制定了特定颜色在各个像素的极限位置。这个信息可以用于在屏幕上的坐标位置上做像素颜色的采集。当像素用来对应屏幕上显示玩家和敌人的生命值的位置时，这条信息和颜色的极限位置一起用作判断角色是生命值全满，还是受了伤，抑或是情况危急——这些都是宏运行时通过屏幕上采集到的像素以及判断颜色是否吻合来确认的。

```
////MAIN LOOP
While 1=1
  Processmessages     //required for AC Tool operation
  Call Lazy8_Main     //the main lazy8 machine handler
End
///END MAIN LOOP

//////////////////////////////////////////////////////
// the main lazy8 machine handler
//////////////////////////////////////////////////////
Procedure Lazy8_Main

      //KeyDown /whisper $pc_name Lazy8_Main {RETURN}
      //KeyDown /whisper $pc_name current state: $gMachineState
{RETURN}
```

在宏的代码里到处都可以看到这种东西，**whisper** 命令在开发宏的过程中会在屏幕上打出调试信息。在 2-5 的截图中你可以看到 **whisper** 的调用结果。

```
// perform effective switch() on machine state
//
// START
if $gMachineState = "START"
Call LazyStart
end

// IDLE
// bored, we need to do something
if $gMachineState = "DRAWING"
Call LazyDraw
end

// END
// game over, man
if $gMachineState = "END"
call LazyEnd
end
```

```
End //end Lazy8_Main

Procedure LazyStart
        KeyDown /stand {RETURN}
        delay 100
        // go directly to draw agro mode
        Set gMachineState = "DRAWING"
End

Procedure LazyDraw
        KeyDown /whisper $pc_name LazyDraw {RETURN}

        //Call LureIfNoMonster

        //target last agro, this targets nearest also
        KeyDown {TAB}
        Delay 400
        Call EnsureAttackMode

        // hero strike
        KeyDown 2
        Delay 400

End
```

在上面这部分宏里，我们执行了一次像素测试，来确认角色是否处在攻击状态，也就是说是否在打怪。如果不是，我们会调用一套程序来寻找最近的一个怪物作为目标，然后开始攻击。脚本点击2键来执行英雄模式的攻击——也就是游戏中的一种强力攻击模式。

```
// run forward and back to lure agro
Procedure Lure
        KeyDown {UP} 3 sec
        KeyDown {DOWN} 3 sec
End

// put the PC into attack mode
Procedure EnsureAttackMode
        LoadRGB 34,64
        Compute $redDifference = Abs ( $color_AttackRed -
{RGBRED} )
        if $redDifference < 80
                KeyDown /whisper $pc_name im currently in attack
mode {RETURN}
        else
                KeyDown /whisper $pc_name im attempting to start
attack mode {RETURN}
               KeyDown $keyExecAttack 150
        end
End

// perform a random move if no monster is targeted
Procedure LureIfNoMonster
        Call util_GetMobHealth
        if $gMobHealth = "DEAD"
                KeyDown /whisper $pc_name Rest in Peace! .. luring
{RETURN}
                Call Lure
        else
                KeyDown /whisper $pc_name still going... {RETURN}
        end
End
```

上面这段宏是通过附近的怪物来触发攻击状态的。其本质就是通过靠近怪物来吸引怪物的注意，角色靠近怪物会触发怪物的攻击。一旦遭到攻击，角色就会攻击怪物并杀死它。

```
////////////////////////////////////////////////////
// END - kill the engine
////////////////////////////////////////////////////
Procedure LazyEnd
      KeyDown /whisper $pc_name LazyEnd {RETURN}

      //TODO
End

////////////////////////////////////////////////////
// Utility Function
// Get target mob health
////////////////////////////////////////////////////
Procedure util_GetMobHealth
     ' //KeyDown /whisper $pc_name util_GetMobHealth {RETURN}

      LoadRGB $coord_MobHPMin
      Compute $greenDifference = ABS ( $color_AliveGreen -
{RGBGREEN} )

      //KeyDown /whisper $pc_name sample monster {RGBRED}
{RGBBLUE} {RGBGREEN} {RETURN}

      if {RGBRED} = 0 AND {RGBBLUE} = 0 AND $greenDifference < 80
            Set gMobHealth = "ALIVE"
            //KeyDown /whisper $pc_name monster is still alive
{RGBGREEN} {RETURN}
      else
            // we never found any green
            Set gMobHealth = "DEAD"
            KeyDown /whisper $pc_name monster is dead {RETURN}
      end

End
```

上面这段宏用来监测目标怪物是否已经死亡。就像监控角色的生命状态一样，通过简单的像素采集技术就可以做到这一点。在这里，我们在显示怪物生命槽的地方采集样本。

以上就是一个完整的宏，可以在 WoW 中自动操作引怪和打怪的行为。通过运行这个宏，角色可以积攒经验值和金钱，而不需要人来干涉。

图 2-5 是 WoW_Agro 宏运行时的截图。注意角色旁边的怪物的尸体。你要注意，运行像这样的一个宏是违反规则的，可能会引起账号被封。在屏幕上的 WoW 角色的名字 Xanier 并非秘密。暴雪在 Hoglund 的角色 Xanier 即将到 60 级的时候封了他的账号（在当时这是 WoW 角色的最高级别）。在这种情况下，Hoglund 作弊得有点过分。到这个时候为止，Hoglund 花掉的游戏点卡钱已经超过 400 美元。所有的账号因为各种理由而全部被封。

现在我们要提到另一个讨厌的问题：为什么用宏来 farm 是作弊？有人会争辩说，是暴雪先让游戏如此烦人的，这是一报还一报。

图 2-5 出自 WoW 的截图，WoW_Agro 宏正在运行。
宏整夜运行着，虚拟的尸体也在不断堆积

2.7 对策

出于很明显的原因，游戏公司对于如何禁止在游戏中使用机器人非常感兴趣。科学家已经开始研究这个问题，并且公布对此的研究工作。例如，Philippe Golle 和 Nicolas Ducheneaut 的论文“防止网游中的机器人”（Preventing Bots from Playing Online Games）出现在美国计算机学会期刊《*Computers in Entertainment*》中。

有一个聪明的办法，可以确认是人在玩游戏而不是机器人，这种方法已经成功用来对付邮件炸弹。这种测试被称为“逆向涂林测验”（reverse Turing test）。涂林测验出自于 AI 领域，是通过对话来判断对方是否是人类的测试。逆向涂林测验是通过提问来确认目标（在这里是玩家）是否是人类。这种测试通常会展示一张比较费解的图片，只有人类能够解读（见图 2-6）。这种技术常常被称为 *captcha*（全自动区分计算机和人类的图灵测试，Completely Automated Public Turing Test to Tell Computers and Humans Apart 的缩写）。

通常，玩家会互相监督，查出谁在用机器人然后举报出来。只要利用游戏的聊天功能，就能做到这一点（游戏管理员也经常用这种方法找出机器人）。聊天的方法在很多游戏里都适用，从 MMORPG 到在线扑克，这种方法都起作用。扑克机器人并不擅长聊天！

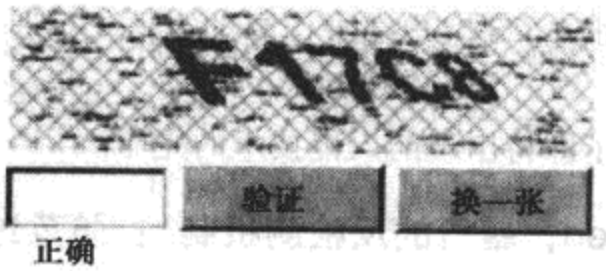


图 2-6 一个标准的基于人类感知的逆向图灵测验

（出自 <http：//www. brianpautsch. com/Blog/2005/12/1/Captcha >；已获转载授权。）

实时监控往往伴随着历史记录监控。通过查找非人的行为——可能是超人的瞄准，或是一些奇怪的行为，例如在同一地点反复杀死敌人——这些都是使用机器人的证明。利用历史记录寻找作弊者已经成为游戏管理员采用的标准策略。

2.7.1 间谍软件

反作弊工具存在的历史大概和游戏服务器一样久。第一批这类工具的出现来源于公众对开放式服务器的呼吁。那些运行私服的游戏公司将这项传统延续到了今天。

第一个提供反作弊服务的是德克萨斯一家名为 Even Balance 的公司，该服务称为 PunkBuster。PunkBuster 服务开始于 2000 年。第一批版本仅仅依赖于一些简单的技术，例如，变量检查和通过服务器进行进程确认。接下来就是大量的基于服务器的检查。现在，PunkBuster 已经整合进大量网络游戏。

PunkBuster 网站中列出如下的产品功能。

- 通过玩家电脑上的 PB 客户端实时扫描内存，寻找已知的黑客/作弊工具。
- 带有后备功能的软件自动更新系统，使用多个服务器来提供更新，防止用户拿到已经破坏的文件。
- 所有玩家的状态报告（高度加密）会不断发送到 PB 服务器，PB 服务器会在必要时发起警报，将入侵玩家踢出游戏，并且通知其他玩家入侵信息。
- PB 管理员可以手动把玩家踢出游戏一定时间，如果需要的话可以永久性封号。
- PB 服务器可以进行任意配置，随机检查玩家设置，寻找已知的游戏入侵软件。
- PB 管理员可以从特定玩家处获取截屏，或是设置服务器在玩家的游戏过程中随机截屏。
- 可选择使用“黑名单”工具，这样 PB 管理员可以禁止玩家使用曾经攻击过游戏的用户名，以免带来不必要的麻烦和人身攻击。
- PB 管理员可用搜索功能，搜索玩家的按键绑定和脚本，看是否有任何入侵游戏的内容。
- PB 玩家力量的功能可以让玩家在服务器管理员完全不在的时候自己管理游戏服务器而不需要密码。
- PB 服务器具有一个可选的内建小型 http 页面式的服务器界面，通过 web 浏览器可以对服务器进行远程管理。

注意，以上的功能中有一部分极具侵犯性，例如扫描客户端电脑！这种扫描通常会伴随着在玩家电脑上安装间谍软件。

PunkBuster 原本是一种防止在 CS 客户端中作弊的方法。但是现在 PunkBuster 致力于保护和管理其他游戏。

事实上，PunkBuster 成为了即将出现的入侵技术的先驱。

2.7.2 典狱官（the Warden，暴雪的反检测机制）：防范过度的反作弊措施

目前，暴雪公司对 WoW 采用两种不同策略对付作弊者。第一种方法是制定反作弊的规则，禁止那些被抓住作弊的角色再次登录游戏。这种方法通过公布使用协议来实施，协议在法律上紧紧围绕着软件最终用户许可协议（EULA）。这么做完全没有问题。（更多法律情况详见第 4 章。）第二种方法是，时刻留意每一台运行着 WoW 客户端的电脑，判断它是否用来作弊。这种监测行为就有问题了。

如果监控一个人的电脑听起来让你感觉就是在监控你，那是因为事实就是如此。WoW 中嵌入的“典狱官”从玩家的电脑上读取各种数据。2005 年秋天，Hoglund 发现典狱官的事件引发了一场关于隐私、安全以及为了阻止作弊安全措施应该把握的尺度的争论。这也是我们决定写本书的部分原因。

除了监控 WoW 的程序空间，追踪该空间中运行的 DLL，典狱官还会进入其他各个程序，做一些类似读取每一个窗口上的标题栏文本或是扫描每一个正在你电脑上运行的程序代码（之后它会将之和已知作弊代码进行比较）这样的事情。暴雪声称，该软件设计的目的只是为了安全，这些资料不会用作安全之外的任何目的，但事实上没有什么能阻止暴雪对玩家的电脑为所欲为。进入游戏之外的其他程序，这种做法已经过界了。我们无法信任他们。这是很明显的对隐私的侵犯，以至于“电子前沿基金会”（Electronic Frontier Foundation）也发表了自己的观点 < http：// www. eff. org/deeplinks/archives/004076. php > 。尽管在 EULA 中确实声明了暴雪可能会用典狱官监控电脑行为（但未指出典狱官实际会做什么），但这条信息被淹没在一片小字中，几乎没人会去看。在与 WoW 玩家的非正式交谈里，我们发现没有人注意到自己曾经同意接受这样的监控。他们中的一部分人十分在意，甚至不再玩游戏。

全世界有 800 万人在玩 WoW（通常有 500 000 人同时在线）。每一个玩家都同意了暴雪有权扫描自己电脑中的程序和内存，而暴雪也确实这么做了。

在游戏程序运行的时候，典狱官程序任意扫描类似公开进程、URL 等诸如此类，根据暴雪的决定进行控制，并将它找到的信息通过电子邮件传送给暴雪。图 2-7 是典狱官发回给暴雪的一个信息范例，是由一个我们写的称为“总督”（Governor）的程序监测出的。试想一下，典狱官作为一个被游戏公司称为“合法的间谍软件”安装了 800 万份。图 2-7 显示了“总督”程序截图。它会报告 WoW 典狱官的行为。典狱官就像暴雪的间谍，通过进入开放进程和读取内存来监视你的电脑，然后把它找到的信息发还给暴雪分析。

当然，暴雪不是唯一一家像这样进行监控的公司，很多其他游戏公司也这么做。索尼甚至试图在音乐 CD 放进电脑的时候安装监控软件，但它做得太差了，导致这家公司就此捅了漏子。类似 WoW 典狱官和索尼的后门这样的程序一旦被揭穿，引起的反应十分激烈。

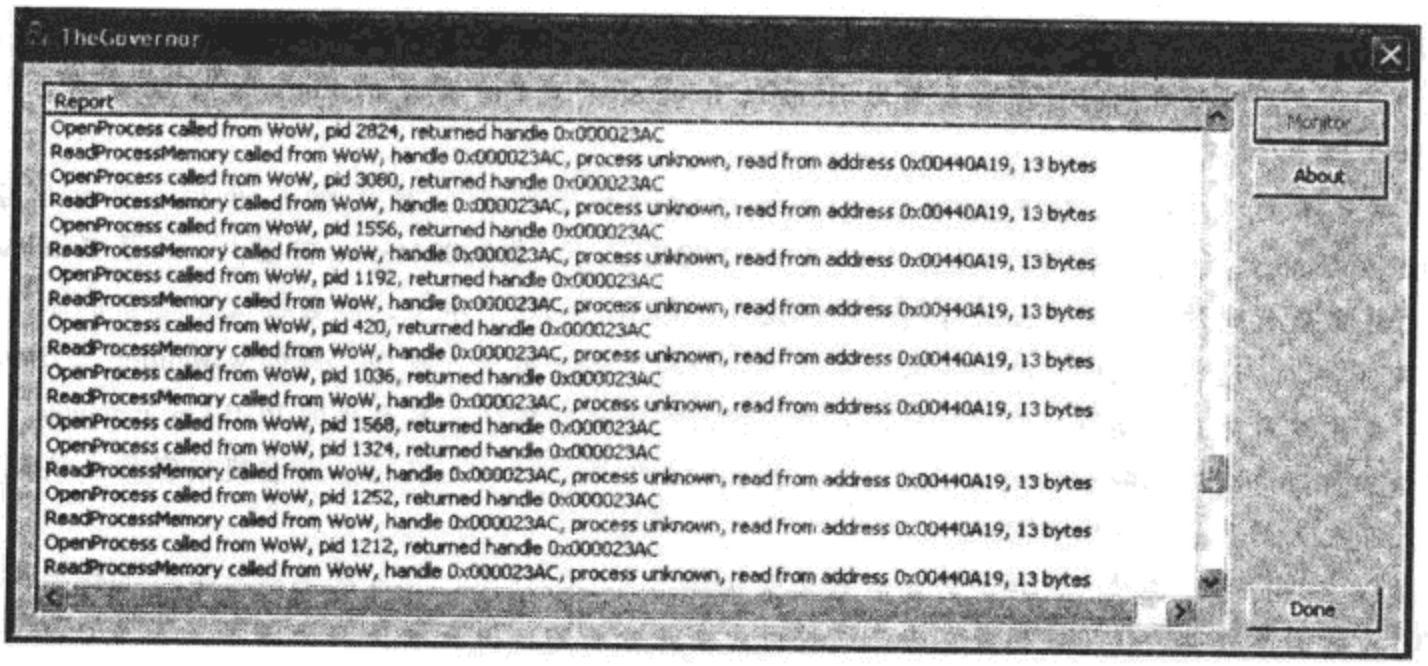


图 2-7 “总督”（Governor）程序截图。它会报告 WoW 典狱官的行为。典狱官就像暴雪的间谍，通过进入开放进程和读取内存来监视你的电脑，然后把它找到的信息发还给暴雪分析

下面的黑帽角转载了一篇博客文章，把典狱官当作间谍软件。这篇文章在全世界都传开了。

供您参考 黑帽角：WoW 典狱官是间谍软件

2005 年秋天，我们中的一人（Hoglund）发现了 WoW 典狱官，并在技术社区发帖公布。Hoglund 的原帖在 rootkit. com blog（在 <http: //www. rootkit. com/blog. php? newsid = 358 > 可以找到，还有相关线索和讨论），这引发了一场激烈的争论。下面是原帖内容。

安装了 450 万份的符合 EULA 的间谍软件

2005 年 10 月 5 日，19：07

Hoglund 发言：

我最近花了很长时间对一个暴雪娱乐公司制作的软件进行逆向分析——是的，就是制作了《魔兽争霸》以及 WoW《魔兽世界》（目前有大约 450 万用户）的暴雪。这个软件名为“典狱官客户端”，它是独立运行的，就像 shellcode 一样。它通过暴雪服务器飞速下载，每 15 秒钟运行一次。作为间谍软件，它的设计目的只是为了检查用户是否遵守了 EULA，这是最有趣的一点。下面就是它每 15 秒对大约 450 万人所做的事（其中 500 000 人是同时在线的）：

典狱官通过调用 ToolHelp API 植入所有的 DLL。它从《魔兽世界》执行程序空间的所有运行的 DLL 中读取信息。这没什么问题。

之后，典狱官用 GetWindowTextA 功能来读取每一个窗口的标题栏上的窗口文本。这些窗口并非 WoW 程序中的窗口，而是你的电脑中所有正在运行的程序。这就是个大问题了。我看到典狱官嗅探我在 MSN 上所有联络人的电子邮件地址，我当时打开的几个网站的地址，以及我所有在运行的程序的名称，包括那些最小化的或是在工具栏里的。

典狱官在获得这些字符串后，用基于哈希算法的函数把它们过一遍，然后和一串“禁用字符串”进行对比，如果你有部分特征符合，我猜你的账号会被禁。举例来说，如果你有一个窗口的标题是“WoW! Inmate”（一种魔兽外挂），不论这个窗口实际是做什么的，它都会令你的账号被禁。如果你不相信，做一个什么也不做的假窗口，给它起这个名字，然后运行 WoW。典狱官一定会因此而报告你是个作弊者。考虑

到窗口标题包含了很多个人资料，我真的认为，读取这些窗口标题侵犯了隐私。但是，我们都知道，暴雪娱乐公司是合法的。看看他们对那些试着做 BNetD、freecraft（魔兽争霸 2 的 linux 版）、或是第三方 WoW 服务器的人做了什么。

接下来，典狱官打开了你的电脑上所有运行的进程。当各个程序被打开后，典狱官会调用 ReadProcessMemory，读取一系列的地址——通常是 0X0040xxxx 或是 0x0041xxxx 范围内，这是 Windows 大部分执行程序放置代码的区间。典狱官每次测试时读取大约 10 到 20 个字节，然后再次经过哈希计算和禁用字符串进行对比。这些测试的目的很明确，就是为了检测已知的第三方程序，例如 wowglider 和 friends。每一个进程都用这种方式被读取。我看到典狱官打开我的电子邮件程序，甚至我的 PGP key manager。我再次感到这是一种对个人隐私的强烈侵害，但是你能怎么样呢？要设计一个测试，看看典狱官到底读取了哪些机密的或是私人的信息，这倒是很简单。

正是以上的行为让典狱官直接成为了一种间谍软件。有意思的是，它可能是第一个用来检测用户是否遵守 EULA 的间谍软件。我不能想象在未来这种行为会是合法的，但在现在的法律里，这样的事情还是头一遭。

你不能责怪暴雪这么做，包括其他任何一家公司，但如果我们还需要任何一点隐私，这种行为就必须停下来。不论是否赞同游戏作弊者，“个人隐私”都是一个太过巨大的问题，无论出于什么原因，暴雪都无权打开我的 excel 或是 PGP 程序。

——Greg

2.7.3 总督

Hoglund 被典狱官搞得心烦意乱，他甚至为此写了一个叫做总督的程序，玩家可以用来确认典狱官当前正在做什么。

我们相信，安装总督是有用处的，如果你对 WoW 软件可能在你的电脑上做什么感兴趣的话，总督尤为有效。这里列出总督的一部分代码。你可以在本书的网站或是 < http: // www. rootkit. com/vault/hoglund/Governor. zip > 下载该软件及其需要的库文件。和前面一样，我们会在代码中加上注释，让它更容易理解。

```
// The Governor
// Oct 16, 2005 - Greg Hoglund
// www.rootkit.com

#include "stdafx.h"
#include "GovernorDLL.h"
#include <windows.h>
#include <stdio.h>
#include <stdarg.h>
#include <process.h>
HANDLE g_hPipe = 0;
CRITICAL_SECTION g_pipe_protector;

void PatchFunctions();
```

这里的代码使用了 DLL（动态链接库）的形式，能够被直接加载到程序里。在这里，WoW 程序通过一种软件安全用语里称为 DLL injection 的进程强行加载这个 DLL。被注入的 dll 实际上只是一个普通 dll，但它在 WoW 游戏客户端启动并登录 WoW 服务器后强制加载。

```
BOOL APIENTRY DllMain( HANDLE hModule,
                       DWORD  ul_reason_for_call,
                       LPVOID lpReserved )
{
        switch (ul_reason_for_call)
        {
        case DLL_PROCESS_ATTACH:
                InitializeCriticalSection(&g_pipe_protector);
                g_hPipe = StartPipe("\\\\.\\pipe\\wow_hooker");
                SendText(g_hPipe, "GovernorDLL Loaded.");
                PatchFunctions();
                break;
        case DLL_THREAD_ATTACH:
                break;
        case DLL_THREAD_DETACH:
                break;
        case DLL_PROCESS_DETACH:
                SendText(g_hPipe, "GovernorDLL Unloaded.");
                ShutdownPipe(g_hPipe);
                break;
        }
        return TRUE;
}
```

DLL 在加载时打开一个命名过的管道（一种通信端口的形式），这个命名过的管道会被用来报告典狱官的行动。

```
//
// Send text down the pipe
//
void
SendText (HANDLE hPipe, char *szText)
{
        if(!hPipe) return;
char *c;
DWORD dwWritten;
DWORD len = strlen(szText);
DWORD lenh = 4;

 EnterCriticalSection(&g_pipe_protector);

 // send length first
 c = (char *)&len;
 while(lenh)
 {
        //char _g[255];
        //_snprintf(_g, 252, "sending header %d", lenh);
        //OutputDebugString(_g);

        if (!WriteFile (hPipe, c, lenh, &dwWritten, NULL))
        {
                LeaveCriticalSection(&g_pipe_protector);
                ShutdownPipe(hPipe);
                return;
        }
        lenh -= dwWritten;
        c += dwWritten;
}

// then string
c = szText;
while(len)
{
```

```
        //char _g[255];
        //_snprintf(_g, 252, "sending string %d", len);
        //OutputDebugString(_g);


        if (!WriteFile (hPipe, c, len, &dwWritten, NULL))
        {
                LeaveCriticalSection(&g_pipe_protector);
                ShutdownPipe(hPipe);
                return;
        }
        len -= dwWritten;
        c += dwWritten;
}
        LeaveCriticalSection(&g_pipe_protector);
}

HANDLE StartPipe(char *szPipeName)
{
        HANDLE hPipe;
        TCHAR szBuffer[300];

        //
        // Open the output pipe
        //
        hPipe = CreateFile (szPipeName, GENERIC_WRITE, 0, NULL,
                        OPEN_EXISTING, FILE_FLAG_WRITE_THROUGH,
                        NULL);
        if (hPipe == INVALID_HANDLE_VALUE)
        {
                _snprintf (szBuffer, sizeof (szBuffer),
                    "Failed to open output pipe(%s): %d\n",
                        szPipeName, GetLastError ());
                        OutputDebugString(szBuffer);
                return NULL;
        }

        return hPipe;
}

void ShutdownPipe(HANDLE hPipe)
{
        //cleanup
        if (hPipe)
        {
                FlushFileBuffers (hPipe);
                CloseHandle (hPipe);
        }
        // make sure it stops being used
        g_hPipe = 0;
}
```

下面我们会列出典狱官的一部分 hooking 文件，来加强你对典狱官的行为的了解。记住，在这里我们只说明了总督系统中最基本的部分。

```
// The Governor
// Oct 16, 2005 - Greg Hoglund
// www.rootkit.com
#include "stdafx.h"
#include <windows.h>
#include <stdio.h>
```

```
#include <string.h>
#include <winsock2.h>
#include "GovernorDLL.h"
#include "detours.h"
#include <psapi.h>
////////////////////////////////////////////////////////////////////////
// These are functions used by the Warden to spy on other processes
////////////////////////////////////////////////////////////////////////
```

下面的功能都是 WoW 典狱官用到的。在这里，我们用微软的 *Detours* 库来截取典狱官对这些功能的调用。这一步骤能够报告给我们所有试图通过我们命名的 pipe 来使用这些调用的程序，包括典狱官。

```
DETOUR_TRAMPOLINE(BOOL __stdcall Real_GetWindowTextA( HWND hWnd,
LPSTR lpString, int nMaxCount), GetWindowTextA);
BOOL __stdcall Mine_GetWindowTextA( HWND hWnd, LPSTR lpString, int
nMaxCount)
{
	int len = Real_GetWindowTextA( hWnd, lpString, nMaxCount);
	if( len != 0)
	{
		// WoW found some window text, let's report it
		char _t[255];
		_snprintf(_t, 252, "GetWindowTextA called from WoW,
returned %d bytes, %s ", len, lpString);
		SendText(g_hPipe, _t);
	}

	return len;
}
DETOUR_TRAMPOLINE(BOOL __stdcall Real_GetWindowTextW( HWND hWnd,
LPWSTR lpString, int nMaxCount), GetWindowTextW);
BOOL __stdcall Mine_GetWindowTextW( HWND hWnd, LPWSTR lpString,
int nMaxCount)
{
	int len = Real_GetWindowTextW( hWnd, lpString, nMaxCount);
	if( len != 0)
	{
		// WoW found some window text, let's report it

		char _t[255];
		_snprintf(_t, 252, "GetWindowTextW called from WoW,
returned %d bytes", len);
		SendText(g_hPipe, _t);
	}

	return len;
}
```

现在，当典狱官打开电脑中的窗口，甚至是那些属于别的程序的窗口（例如，你的即时通信程序）的时候，我们就能发觉了。总督不仅会报告典狱官打开的窗口，还能报告它所读取的文本。这种技术可以用来观察典狱官程序读取的比较敏感和私人的信息，包括你的 MSN 中的联系人的邮件地址。

```
DETOUR_TRAMPOLINE(int __stdcall Real_WSARecv(
  SOCKET s,
  LPWSABUF lpBuffers,
  DWORD dwBufferCount,
```

```
  LPDWORD lpNumberOfBytesRecvd,
  LPDWORD lpFlags,
  LPWSAOVERLAPPED lpOverlapped,
  LPWSAOVERLAPPED_COMPLETION_ROUTINE lpCompletionRoutine),
WSARecv);

int __stdcall Mine_WSARecv(
  SOCKET s,
  LPWSABUF lpBuffers,
  DWORD dwBufferCount,
  LPDWORD lpNumberOfBytesRecvd,
  LPDWORD lpFlags,
  LPWSAOVERLAPPED lpOverlapped,
  LPWSAOVERLAPPED_COMPLETION_ROUTINE lpCompletionRoutine)
{
        int res = Real_WSARecv(s, lpBuffers, dwBufferCount,
                               lpNumberOfBytesRecvd,lpFlags,
                               lpOverlapped,lpCompletionRoutine );
        char _t[255];
        _snprintf(_t, 252, "WSARecv returned %d, %d bytes
received", res, *lpNumberOfBytesRecvd);
        SendText(g_hPipe, _t);

        return res;
}

DETOUR_TRAMPOLINE(DWORD __stdcall Real_CharUpperBuffA( LPTSTR
lpString, DWORD cchLength), CharUpperBuffA);
DWORD __stdcall Mine_CharUpperBuffA( LPTSTR lpString, DWORD
cchLength)
{
        DWORD len = Real_CharUpperBuffA( lpString, cchLength );
        if( len != 0)
        {
                // WoW is processing some text, let's report it

                char _t[255];
                _snprintf(_t, 252, "CharUpperBuffA called from WoW,
string %s", lpString);
                SendText(g_hPipe, _t);
        }

        return len;
}

DETOUR_TRAMPOLINE(HANDLE __stdcall Real_OpenProcess( DWORD
dwDesiredAccess, BOOL bInheritHandle, DWORD dwProcessId ),
OpenProcess);
HANDLE __stdcall Mine_OpenProcess( DWORD dwDesiredAccess, BOOL
bInheritHandle, DWORD dwProcessId )
{
        HANDLE h = Real_OpenProcess( dwDesiredAccess,
bInheritHandle, dwProcessId );
        if( h != 0)
        {
                // WoW is opening a process, let's report it

                char _t[255];
                _snprintf(_t, 252, "OpenProcess called from WoW, pid
%d, returned handle 0x%08X", dwProcessId, (DWORD)h);
                SendText(g_hPipe, _t);
        }
```

```
        return h;
}

DETOUR_TRAMPOLINE(BOOL __stdcall Real_ReadProcessMemory(
                            HANDLE hProcess,
                            LPCVOID lpBaseAddress,
                            LPVOID lpBuffer,
                            SIZE_T nSize,
                            SIZE_T* lpNumberOfBytesRead ),
ReadProcessMemory );

BOOL __stdcall Mine_ReadProcessMemory( HANDLE hProcess, LPCVOID
lpBaseAddress, LPVOID lpBuffer, SIZE_T nSize, SIZE_T
*lpNumberOfBytesRead )
{
        BOOL ret = Real_ReadProcessMemory( hProcess,
lpBaseAddress, lpBuffer, nSize, lpNumberOfBytesRead );
        if( ret && ((DWORD)hProcess != -1) )
        {
                // WoW is reading a process, let's report it
                char szProcessName[MAX_PATH] = "unknown";


                GetProcessImageFileName(hProcess, szProcessName,
MAX_PATH);

                char _t[255];
                _snprintf(_t, 252, "ReadProcessMemory called from
WoW, handle 0x%08X, process %s, read from address 0x%08X,
%d bytes",
                        hProcess,
                        szProcessName,
                        lpBaseAddress,
                        nSize);
                SendText(g_hPipe, _t);
        }

        return ret;
}
```

上面的最后两个功能毫无疑问是关于典狱官行为的。不论典狱官何时打开其他进程，并且读取该进程的内存，它们都可以清楚地报告出来。我们的反间谍系统相当彻底。

```
void PatchFunctions()
{
        DetourFunctionWithTrampoline( (PBYTE)Real_GetWindowTextA,
(PBYTE)Mine_GetWindowTextA);
        DetourFunctionWithTrampoline( (PBYTE)Real_GetWindowTextW,
(PBYTE)Mine_GetWindowTextW);
        DetourFunctionWithTrampoline( (PBYTE)Real_CharUpperBuffA,
(PBYTE)Mine_CharUpperBuffA);
        DetourFunctionWithTrampoline( (PBYTE)Real_OpenProcess,
(PBYTE)Mine_OpenProcess);
        DetourFunctionWithTrampoline( (PBYTE)Real_ReadProcessMemory,
(PBYTE)Mine_ReadProcessMemory);

        //DetourFunctionWithTrampoline( (PBYTE)Real_WSARecv,
(PBYTE)Mine_WSARecv);
}
```

最后一部分代码实际上安装了我们定义的“绕行”函数来追踪选中的功能。你可以很轻松

地更改源代码来监视更多的调用，并且可以用这些代码监视任何你希望监视的程序，包括其他游戏。我们鼓励你多多体验这种功能。或许你会发现其他监控使用者的游戏和程序。

2.7.4 你的立场在哪里

自从 Hoglund 说明了典狱官之后，MMORPG 的论坛和网站上就展开了关于间谍软件的激烈争论。有的人认为，抱怨这类间谍软件很愚蠢，如果担心间谍软件，他可以选择不玩。我们不同意这个观点。我们还听说另外一种对微软的抱怨，因为它的操作系统让任意一种程序都可以收集到典狱官收集到的信息。不管你是否侵犯了个人隐私，我们担忧的是这种行为的意图，难道不对吗？

暴雪公司以安全之名所做的行为让人无法接受，它必须停止这种行为。个人自由和安全之间的平衡点十分根本，必须认真权衡。生活在自由社会的公民必须要警惕地守卫好自己的自由，以及防范以安全之名进行的暴政。从历史观点上说，监控行为会迅速导致滥用，并让状况迅速下跌到不可接受的程度。暴雪不需要阅读我们的电子邮件、浏览我们的网络链接或是查看一切非暴雪进程。如果我们运行一家公司这么做，那么我们放弃的太多了，和换得的完全不等价。一旦每个人都做么做，在个人电脑上将不再有任何隐私。

Hoglund 在 <http://www.rootkit.com/newsread.php?newsid=371> 的帖子里说得很好：

> 暴雪，你想用你的方式来制作电脑游戏并且抓住作弊者，这是你的权力。但是，读取非《魔兽世界》游戏客户端的其他进程的内存和窗口是一种对隐私的侵犯。即使是通过 EULA 和 TOS 把侵犯隐私合法化，它依然是不道德的。它依然是对隐私的侵犯。请重新制定你们的方针，停止扫描内存，限制典狱官只去检查游戏客户端的内存空间，并且请停止进入其他进程，不要再读取那些不属于你们的窗口。

如果我们袖手旁观，让一家游戏公司以安全之名任意进入我们的电脑，你认为接下来他们会做什么呢？

2.7.5 作弊

如我们本章说明，有很多广泛流传的网游作弊方式。我们说明的一部分技术已经有几年的历史了。随着越来越多的金钱卷入其中，越来越多的玩家想要尝试作弊。作弊不好，但作弊技术在不断提高。

然而，反作弊技术存在着很大问题。后门程序、rootkit 以及入侵式扫描真的对于盗版侦测必不可少吗？我们的对策是何时在保护正版和侵犯隐私之间过了界的呢？为了玩网游，你必须要放弃哪些使用电脑的权力？

举例思考一下，暴雪的典狱官扫描程序不是真的用来侦测盗版，它其实是用来侦测作弊者的。当然，暴雪会很好地说明它认为的作弊是什么，毕竟这是他们的游戏。在这里，作弊就是使用了任何一种自动操作工具来玩游戏，例如我们在本章说明过的 WoW_Agro 宏。

看来，作弊之战已经在某种形式上对隐私带来了额外的侵害。但是，这只是游戏世界的一个开始。

第3章 金 钱

如今全世界有数百万人在玩网络游戏，并且大部分是付费玩游戏。事实上，网络游戏经济的规模已经可以和一些小国家匹敌了。

本章花点时间思考一下网络游戏中的金钱，让我们可以更好地理解为什么会有人作弊，以及游戏公司为了维护自己的权益会用哪些资源来承担、阻止这些作弊行为。当有钱可赚的时候——玩网络游戏确实能赚钱，作弊者们出现了。当游戏收入因为猖獗的作弊或是玩家对游戏感到厌烦而流失，并且这笔钱又很重要的时候，游戏公司就会开始竭尽全力保卫自己的领土不受侵犯。

3.1 游戏公司如何赚钱

MMORPG 游戏公司赚钱的方式有两种。第一种是直接免费进行游戏，对游戏里面的增值服务收费。第二种是每月收取玩家上线游戏的费用（通常是包月制或者时间点卡）。

让我们想想看 WoW 的这两种盈利方式赚到了多少。拥有超过 800 万的玩家，如果 WoW 的客户端软件零售价平均为 20 美元，那就是 1 亿 6 千万美元。如果在过去，游戏软件一旦被破解并且免费散布出去，这将会是多大一笔金钱上的损失。“光盘游戏”模式产生的问题给了客户端 - 服务器模式一个技术上的启示。回顾第 2 章，客户端 - 服务器模式通过把大部分的状态存储在服务器上的方式，让游戏不再那么容易被破解。

迄今为止，我们已经算到 1 亿 6 千万美元了。但是如果我们把点卡的收入也算上，结果将会更惊人。以 WoW 为例，每个玩家每个月要交 14 美元的月费，也就是每个月会有 1 亿 1 千 2 百万美元的总额。把这个数额乘以 12，我们就得到了年收入 13 亿 4 千 4 百万美元的惊人数字。而这只是一款 MMORPG 的估计收入！

根据微软估计，即使把客户端零售的收入和点卡收入平均分开来计算，2005 年全世界在其公司操作平台上的游戏市场仍然达到约 60 亿美金。DFC 的分析家报告，到 2010 年，游戏市场将会翻倍，达到 120 亿美金（14% 的年增长率）。

游戏公司收入高了。很明显，他们希望保住这种收入来源。不仅如此，他们还希望能够保证大部分玩家能够一直对游戏体验感到满意，这样玩家才会继续缴费。作弊对这笔大买卖是个直接的打击。毫无疑问，游戏公司会努力清除作弊行为。

在线扑克游戏

在线扑克同样十分庞大。图 3-1 是几年间在线扑克的估计增长率（包括玩家和金钱两方面）。在线扑克公司利用这种增长上市，结果令人惊愕。

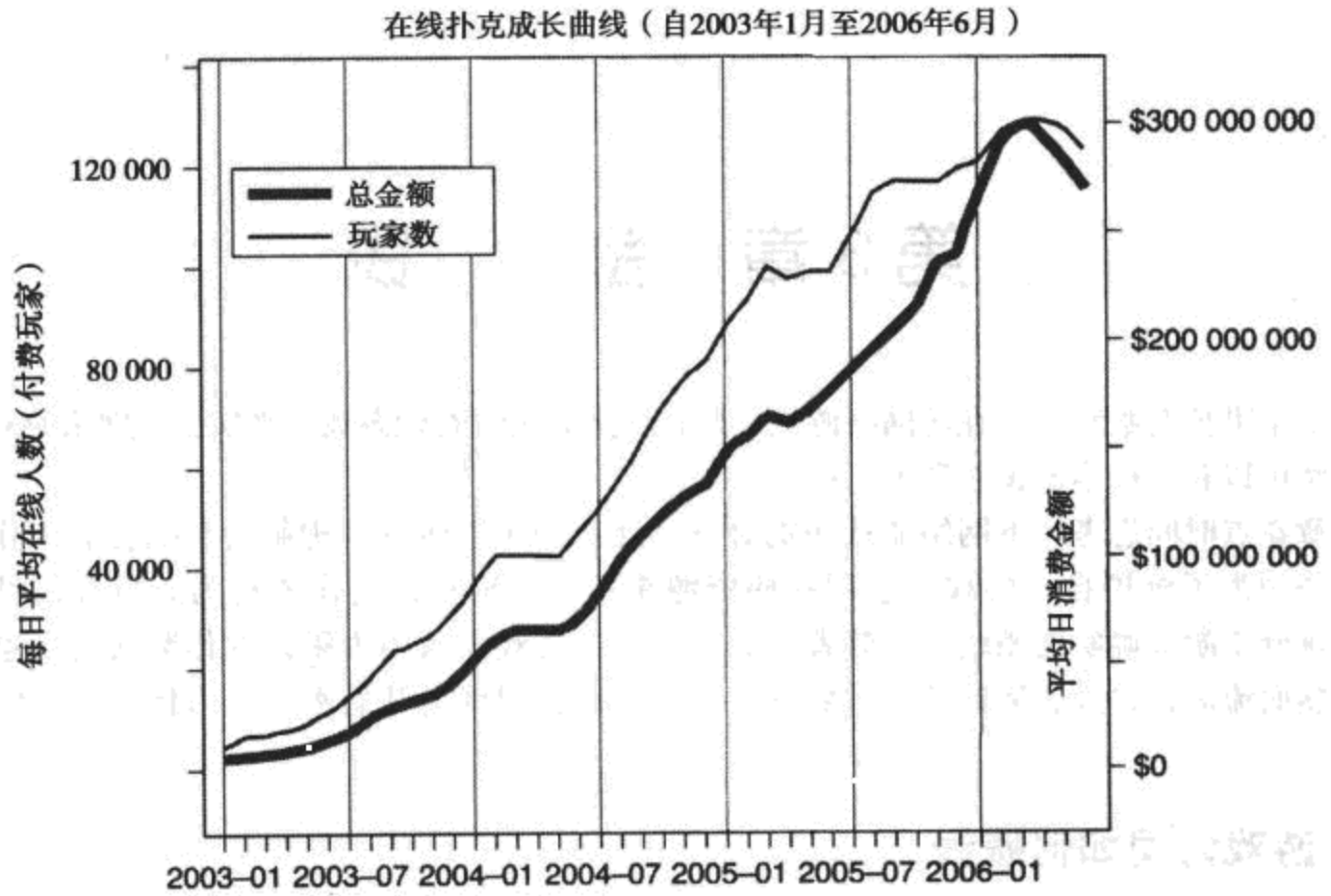


图3-1 截止于2006年6月的扑克统计（出自<http://www.pokerpulse.com>；已获转载许可）

2006年秋天，在线扑克的这种增长遭到美国议会的打击。2006年之后，根据法律，美国公民不能在线用真钱赌博。但是，世界上还有美国之外的很多国家。在图3-1你可以看到美国法律所造成的打击。2005年，在线扑克的利润估计每个月有2亿美元。

有了这种金钱诱惑，玩家们寻找一切超越对手的机会也在意料之中。因为一切都是自动操作的，要追踪在线扑克游戏的玩法很容易。大部分在线扑克网站都有便利的“出牌记录”功能，玩家可以追踪游戏中的每一个动作，获得大量有用的信息。

3.2 虚拟世界：游戏经济学与经济

在美国，学者们研究MMO已经多年。一些评论和帮助理解MMO玩家的社会学的项目现在正在进行中（PARC <http://blogs.parc.com/playon/> 上的PlayOn和<http://www.nickyee.com/daedalus/> Daedalus Project就是两个例子）。社会学固然不错，但是在我们看来，理解MMO的经济才是最有趣的部分。

在一项早期的研究中，经济学家Edward Castranova（现在印第安纳大学担任教授）曾经指出，MMO《无尽的任务》（EverQuest）的国内生产总值超过了很多真正的国家。《*Walrus*》杂志报道如下：

> 根据所有玩家每一年在游戏中创造的财富总值来计算，《无尽的任务》国民生产总值达到了人均2266美元。根据世界银行分级，《无尽的任务》比印度、保加利亚或是中国更富有，几乎和俄罗斯一样富有。
>
> 它是世界上排名第77位最富有的国家，但它甚至不是一个国家。

尽管 Castranova 关于《无尽的任务》GDP 的估计可能过高，但是，我们至少很清楚自从玩家开始投入到网络游戏的虚拟世界中开始，就从无到有地创造出了财富。

芝加哥大学的教授 Steven Levitt（同时也是《*Freakonomics*》一书的作者）利用在线扑克讲授经济学 <http：//www. pokernomics. com/ >。他用在线扑克追踪公司（和玩家手动追踪）收集到的大量资料进行大范围的分析。他的目的之一是为了了解什么样的策略有效，什么样的会失败。当然，他的研究可能会让他更擅长玩扑克，并且赚到钱。

3.2.1　和真实经济体的联系

虚拟世界的经济和真实的经济并非是完全隔绝的。在很大程度上它们存在着联系。最明显的联系是，玩家投入的时间（很明显具有真实世界的经济价值）。紧随时间而来的是游戏公司在真实经济中创造的利润。不仅如此，类似虚拟金币或是宝剑之类的虚拟道具让相应的电脑数据也有了实际价值。

大部分的大规模 MMO 都会有虚拟金币和真实货币之间的兑换率。GamsUSD 网站 <http：//gameusd. com > 实时追踪各个游戏的兑换率。图 3-2 是 2005 上半年间 WoW 金币和美元之间的兑换率。

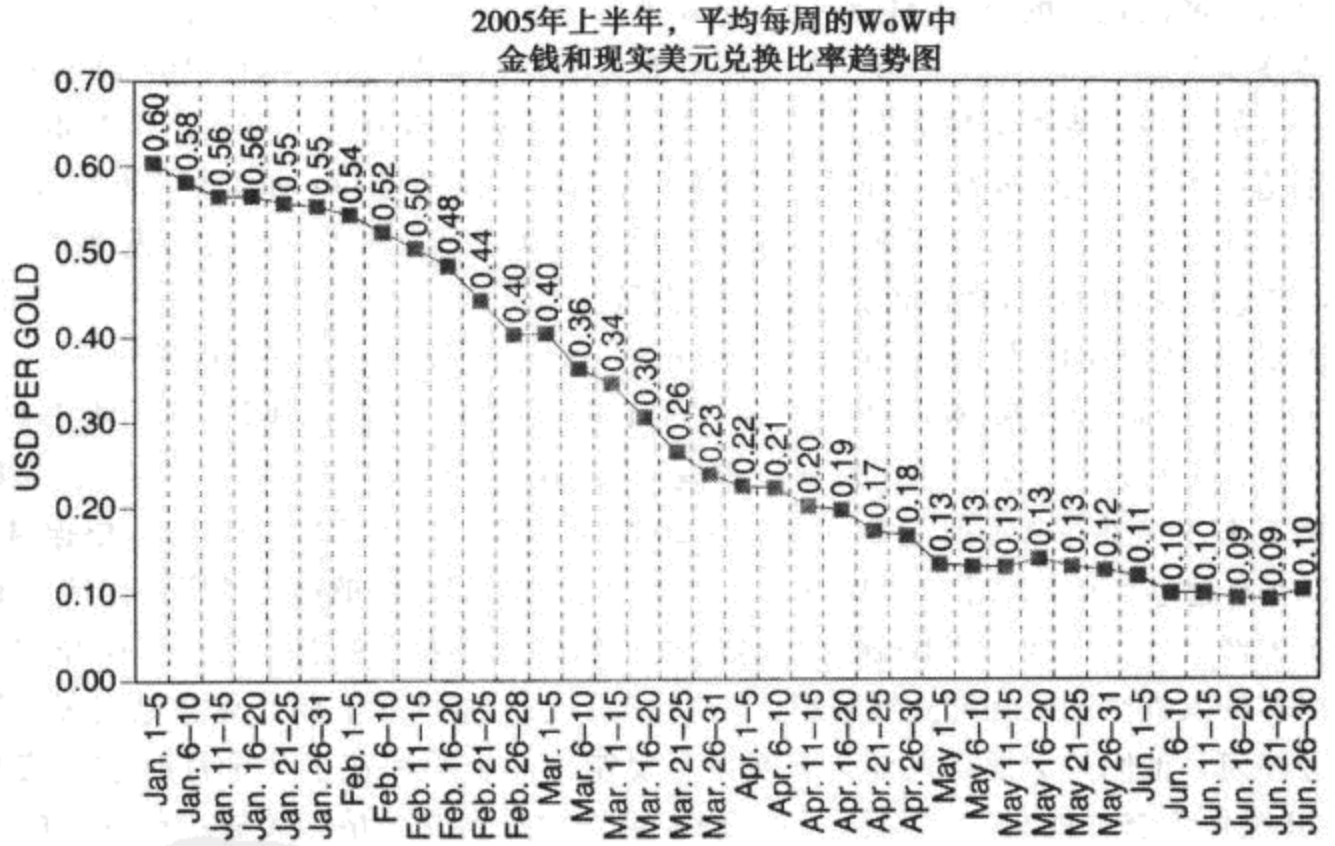


图 3-2　WoW 虚拟金币和美元之间的兑换率

（出自 GameUSD. com <http：//www. gameusd. com/ >；已获转载许可）

和真实的经济一样，MMO 经济会受到经济不稳定和其他整体经济问题的影响。一些游戏公司甚至会聘请经济学家来帮助他们了解和管理这些经济问题。最早的成功 MMO 之一，《网络创世纪》经历过大规模的通货膨胀，并由此导致了 1997 年的一场经济危机。有意思的是，这场通货膨胀源自于一个可以让玩家复制金钱的入侵软件（也就是，什么也不用做就可以从无到有创造出游戏金币）。复制和伪造差不多。

总之，成群的经济学家把 MMO 用作研究目的，重点关注什么经济模式最有意义：集权经济、自由市场、福利等。游戏设计师和游戏管理者要根据游戏的经济做出各种艰难的决定，就像真实世界中的中央银行家一样。

就和真实经济中存在着犯罪一样，虚拟经济中也有犯罪。有些行为只是边缘犯罪（或者至少是违反了游戏规则；见第 4 章）。例如，玩家经常会被禁止出售游戏世界中获得的角色或是其他道具，但这并不能制止丰富的中间市场的发展。

3.2.2 中间人

游戏经济相当地富足，以至于产生了相关企业级专家。一个早期的企业家，Robert Kiblinger，曾是 UO《网络创世纪》的玩家之一，他创建了 UO Treasures < http：//www. uotreasures. com/ > 。UO Treasures 是典型的中间人，购买虚拟角色和道具，然后再卖出去，以此获利。这种行为在业界中称为 RMT（即英文 Real Money Trade 的缩写，意为用真实的货币去购买游戏中的虚拟金币或虚拟装备）。

根据 *Play Money* 书中的记载，游戏中的虚拟道具单价的最高记录是 100 000 美元，2005 年 10 月用于购买《安特罗皮亚计划》（Project Entropia）游戏中的太空站。这本书的作者 Julian Dibbell 估计全世界虚拟商品每年的交易金额将近 8 亿 8 千万美元，并且还在持续增长。然而，最近流行的一款虚拟现实游戏 *Second Life*（第二人生）可能会刷新这些数字。*Second Life* 通过明确地允许和支持在自己的游戏世界中交易虚拟道具，将虚拟经济推到了极限。事实上，和大部分游戏世界中的虚拟创造物不同，*Second Life* 中的创造物属于创造者，而不是 Linden Lab（游戏公司）。这个新奇的点子牵涉到了知识产权。正如你可能想象到的，这会引发一些有趣的法律状况。一些关于 *Second Life* 交易和虚拟财产价值的诉讼目前仍然悬而未决。

IGE 是最大的 RMT 中间人机构（Internet Gaming Entertainment 的缩写）< http：//www. ige. com/ > 。IGE 由童星 Brock Pierce 和 Alan Debonneville 创建，2005 年 8 月拥有约雇员 420 人。有趣的是，IGE 的现金交易操作目前在香港进行。IGE 声称：“一些专家认为，虚拟资产市场将会在接下来几年里将会占据主要市场，到 2009 年预计可达到 70 亿美元。”根据 *Wired* 杂志报导，IGE 拥有超过 100 名稳定的供应者。所有的核心级玩家都会把自己的资产出售给 IGE。

和所有的经济体系一样，MMO 游戏中也隐藏有黑暗的一面。首先，像 IGE 这样的公司为所有想成为黑客的人提供了便利的洗钱渠道。“中间人”公司通常并不那么在意他们买入的不同游戏道具来源如何。他们只是很高兴地买进卖出，复制来的金币和花了几个月时间持续游戏而练成的合法 60 级角色被一视同仁。（传言说我们在第 1 章提到的《星球大战：星系》的黑客就是用 IGE 洗出了复制来的钱。）*Wired* 杂志还报告说，俄罗斯的企业花很少的钱雇来工人（每周 100 美元）在游戏内打钱，然后出售给 IGE。多产的工人还能赚到更多的真实金钱。正是发展中国家经济和第一世界经济的不一致让这种策略得以生效。

付钱请工人来创造虚拟财富和用自动操作的机器人之间的界限相当微弱，但是却很重要，*Wired* 报告：

> 自动操作……通常都被认作是作弊，是一种不被鼓励的行为。IGE 的 CEO Brock Pierce 说，公司会严惩那些可能使用了机器人的供货者。那么就只有另一个办法可选

了：廉价劳动力。

然而，其他不能容忍这类玩法的玩家曾经结队去杀那些农场主。

不那么正规的游戏服务公司来来去去，都有一些规律。有些公司，例如现在已经不存在的黑雪互动（BlackSnow Interactive），开办工作室雇用廉价小时工不断“玩”角色，以获得经验值升级。黑雪的创建是为了给《卡米洛的黑暗时代》中的角色升级。该游戏的公司 Mythic 最后通过法律途径强制关闭了这家打钱公司。

在 2007 年初，eBay 宣布禁止网络游戏道具交易。这可能对中间人市场造成了巨大的冲击。根据新闻报导，eBay 的市场份额有几百万用户。CNET 报告说，“虽然 RMT 市场还没有一个普遍认同的价值，根据推断应该达到每年大约 2 亿 5 千万到 8 亿 8 千万之间。”

3.2.3　为了赚钱玩游戏

中间人市场也包括有一群唯利是图的人想要玩游戏，或者只是为了赚钱而玩。当然，一个有钱的玩家可以在自己不能玩的时候花钱找人来练自己的角色。事实上，根据游戏网站 Gamasutra. com，IGE 已经作为想升级的玩家和想玩游戏赚钱的人之间的中间人进入了这个商业领域。

大量以 Web 为基础的游戏都会给出玩游戏的价格（包括一些类似西洋双陆棋、象棋或是跳棋这样的简单游戏的虚拟版本）。< http：//www. game4dollars. com/home. php > 有一个例子。解谜竞赛也很常见。这类游戏大部分都会有小赌注，可以使用 Paypal 或是其他的电子现金办理媒介作为支付系统。这种活动是否合法还要看你住在哪个地区。在英国，基本上任何事都是合法的。在美国，不同的区域有不同的规定。有的州宣布这样的竞赛或是赌金是不合法的，包括阿肯色州、亚利桑那州、特拉华州、佛罗里达、爱荷华州、路易斯安那、马里兰以及田纳西州。随着美国联邦政府开始逐步控制在线扑克，没人知道未来会怎样。

3.2.4　Thottbot

Thottbot 既是一种应用程序，也是一个网站 < http：//www. thottbot. com/ > ，现在和其他很多游戏联合网站一样，归 IGE 所有。Thottbot 收集了大量有关《魔兽世界》的信息。玩家安装 Thottbot 插件（称为 Cosmos），它会把未加工的信息传输给 Thottbot 程序。这些信息之后会组织进一个巨大的资料库，方便玩家游戏的时候使用。

Thottbot 资料库包含了 WoW 中所有虚拟物品的详细目录。这个资料库包括了各种信息（例如，非玩家角色、怪物、可以拥有或交易的道具）。Thottbot 通过从大量的 WoW 客户端挖掘资料来收集这些信息。

Thottbot 程序可以追踪哪个角色或是 NPC 有什么，并且让电脑上安装了帮助程序的玩家可以看到这些信息。插件程序从资料库读取信息，然后显示出你的角色相关的有用的统计数据。在某种意义上说，它就像一个很好的信息助手。当然，当它提供帮助的同时，它也在忙着收集信息并且发送回 Thottbot 服务器。

Thottbot 助手为玩家提供了很多服务，比如在地图上显示给玩家要如何取得他们想要的道具，并且在地图上标出道具的出现率，这样玩家就可以决定他们的行动方案。

比起其他没有使用这种服务的玩家来说，这些信息对想要更了解游戏的玩家当然是很有用的。这对打钱工作室的玩家也很有用，通过像 Thottbot 这样的系统的指导，他们可以最大限度地使用劳动力。

一些人因此而对 Thottbot 和 IGE 之间的联系产生了质疑。我们知道，IGE 拥有 Thottbot。我们也知道，Thottbot 读取你的财产清单和其他数据，知道使用该系统的哪个玩家拥有什么。这种信息可以有多种用途，从修理道具的价格，到市场最大化的套利交易。

IGE 还收购了 Allalhazam 和 OGaming，二者在本质上和 Thottbot 差不多。这种联合让核心级玩家怀疑 IGE 是否将会遍及整个游戏世界（很多玩家强烈反对买卖角色和道具）。

3.3 犯罪行为

“打钱工作室”，就像是 BSI 运营的一个，或是 IGE 的俄罗斯分部，在第三世界国家最常见，尤其是在远东地区。这种买卖通常和经过玩家同意的游戏使用协议背道而驰（更多法律协议见第 4 章）。但是，他们持续存在。有时，当他们的规模比较大，引起了游戏公司的注意时，游戏公司就会开始采取相关的措施。

像 IGE 这样的中间人公司看似合法得多，并且他们宣称会用正大光明的方式为玩家提供服务。但是 IGE 和其他的中间人很明显目的在于洗钱，不论是虚拟的还是真实的。通过网络游戏洗黑钱很有效果，大量的金钱通过上百万次的交易被“清洗”了。同样，出售通过作弊获取的虚拟道具，让作弊成为了黑市交易的一部分。

正式声明，我们并不赞成和宽容网络游戏中的作弊或是其他任何犯罪行为。但是，这种现象确实存在着，并且相当广泛地传播着。网络游戏越是成长，犯罪行为牵连得也会越深。我们只能通过恰当地理解这种行为是如何出现的，才能与之斗争。

第4章　进入律师视野

我们不是律师，也没有在电视上扮演过律师。当我们做专家证人的工作时，我们会和律师交谈，但那只是在被雇佣的情况下。要说明一下，本章中的观点不能当作任何法律上的建议。如果你需要法律上的观点，最好不要问那些冒牌货！

不论如何，网络游戏的世界现在充斥着网络行业相关律师。因为 MMORPG 的发行商可用两样武器来对付作弊者：用法律文案制定规则，用科技手段强制执行规则。前一种武器包括聘请大量的律师。

4.1　合法

在过去的电脑游戏中，一张光盘上装有一个游戏软件，只能在自己的电脑上单机运行。在那种情况下，法律协议是用来限制游戏制作商的责任，以及防止部分玩家制作非法拷贝（盗版）的。在那时最严重的问题是内容保护机制被破解，然后游戏出现“免费”版本。

我们已经知道了，大部分现代的网络游戏使用一种客户端/服务器的 C/S 结构（见第 1 章）。原本，以客户端为基础的大部分游戏的服务器只是用来连接玩家的，例如第一人称射击游戏（FPS），或是在线授权检验。只有到了最近，大部分游戏状态和游戏进程才放在了服务器上（例如，像 WoW 这样的 MMORPG）。

FPS 游戏的玩家多年来一直聚集在一起举行局域网聚会，他们会把电脑连接起来，然后彼此对战。当 FPS 的制作者开始用服务器端的技术保护他们的客户端游戏时，在局域网上把客户端游戏连接起来就变得更具挑战性。为了避免在线授权检验的干扰，一些程序员干脆自己制作服务器工作区。最后，类似的服务器工作区也适用于 MMORPG 了。

当三个程序员写出了暴雪的服务器软件开源版本之后，暴雪起诉了他们，并且获得了胜诉。在“电子前沿基金会”（Electronic Frontier Foundation）的网站上对此有如下总结（EFF 保护开发者）。

> 在这个案子里，起诉的是到底这三个创造了 B-netD 游戏服务器（可以和暴雪的网络游戏交互操作）的软件程序员是否侵犯了“数字千年著作权法案”（Digital Millennium Copyright Act）以及暴雪的最终用户许可协议。
>
> B-netD 是一种开源程序，让玩家可以在非暴雪的 Battle. net 服务器上玩流行的暴雪游戏（例如《魔兽争霸》《暗黑破坏神》《星际争霸》及其他）。暴雪在法庭上说，这些写 B-netD 的程序员违反了 DMCA 的反规避规定，并且还违反了暴雪的 EULA 中的几个部分，包括破解部分。

开源服务器软件 B-netD 运行良好并且易懂。暴雪所担心的问题是，游戏的购买者一旦不是必须要连接他们的商业服务器，可能不愿继续付月费的钱。暴雪认为，那很不公平，是对智力财富的滥用。

另一方面，程序员认为，他们想和过去同朋友在局域网聚会或是通过网络共享服务器玩类似的其他游戏那样玩游戏。他们认为，暴雪进行了双重剥削，即把它当成单独的游戏收取客户端的费用，同时又收取在线服务的月费。

暴雪不这么认为。对暴雪而言，开源服务器为人们开辟了另一条道路，人们可以制作非法游戏客户端，不需要验证码或是网络账户就可以玩游戏。暴雪声称，法院最终发现开源服务器制作出来是为了给盗版者使用，而不是合法的玩家。

那么这些 EULA 和 DMCA 到底都说了些什么呢？在本章，我们会开始进行一次法律丛林之旅，看看美国的著作权到底是怎样的。

4.2 公平使用和著作权法

公平使用的概念在美国是独一无二的，现在被编入了著作权法第 107 节。美国法典第 17 章的著作权法在第 107 到 118 节颁布了对著作权持有者权力的大量限定。根据著作权办公室：

> 107 节包含了一系列不同的目的，为了这些目的而进行的某作品的复制被认为是“公平”的，例如批评、评论、新闻报导、教学、学术、以及研究。107 节还宣布了 4 条要素，用来判断某种使用是否公平：
>
> 1）使用的目的和性质，包括是否用作商业目的或是非盈利的教育目的；
>
> 2）著作权认证作品的性质；
>
> 3）与著作权认证作品整体相比，被使用部分的数量和内容；
>
> 4）用于潜在市场的效果，或是著作权认证作品的价值。

当然，法律总是要经过解释的，有很多案件围绕公平使用和侵权之间的界限进行着争论。大体的著作权法存在了。事实上，美国版权局声明：“版权局不能决定某项使用是否可被认作‘公平’，也不能给出任何可能违反著作权的建议。”这真是绝好的建议，那么游戏又怎么样呢？

就非律师人员所关心的，在过去，如果你购买了一样拥有著作权的东西，你象征性地拥有了公平使用的权力——也就是说，你可以想怎么用就怎么用，只要你是作为个人使用，并且不打算转售或是和他人分享衍生物品（或是原始内容）。自然，有的玩家会争辩说，如果他们买了一个游戏，他们就有权在自己的服务器上玩。但是随着《数字信息千年著作权法》的出现，时代不同了。

4.3 数字信息千年著作权法

在 1998 年后期，美国议会颁布了《数字信息千年著作权法》（DMCA）。它的目的是“为了修正美国法典第 17 章，为了贯彻世界知识产权组织版权条约和表演、录音制品条约，并兼顾其他目的。”法律宣告，想要避开知识产权保护机制使用某技术进行生产和发行都属于犯罪行为。也就是说，它限制了围绕数字版权管理的特定行为，以加强著作权法。它还提高了网络侵权的处

罚。欧盟有一项类似的法律叫做《欧盟议会和理事会关于信息社会中版权和相关权某些方面的指令》（EU Copyright Directive）。

DMCA 并非没有引起争议。很多人认为，它太过于支持著作权所有者，甚至到了扼杀竞争的地步。具有讽刺意味的是，DMCA 存在的理由（用法律支持 DRM）反而削弱了自己。普林斯顿教授 Ed Felten 辩论说："由于每个人都越来越清楚地认识到 DRM 技术无力阻止 P2P 侵权，DRM 的基本理论改变了"，关于 DRM 的争论最终也会从著作权的强制执行转移开。

在这里 DMCA 和我们的目的息息相关，因为它和 EULA 共同约束了网络游戏中的某些特定行为。

4.4 最终用户许可协议

EULA 是一种用于软件的许可证。它实际上就是软件制作者和最终用户之间的法律合同，规定软件要如何使用。EULA 经常会在使用上做出约束，让违反者承担责任。

你可能会想到，EULA 有很多不同的种类。有的要求点击一次来激活（通常在网上）。有的是在 CD 拿出包装时生效。当然，EULA 的条款和约束也不同。

EULA 和 DMCA 都是直接和网络游戏玩家相关的。如今，DMCA 用于支持 EULA 中的使用规则，现在还可以加入使用在线服务的付费和使用要求。

你在点击 EULA 的同意按钮之前，必须要知道你同意的是什么。罚款的内容可能会让你大吃一惊。

EULA 可以对许可内容的使用提出各种不同的奇怪约束。在下面一部分，我们会探讨一些实例。

- EULA 可以规定你能够在哪里以及何时使用软件，或者是一旦安装一次就不能再次安装。
- EULA 还可以规定软件要在你的电脑上安装 rootkit 后门（见索尼的 EULA）。
- EULA 可以要求在线服务必须全权读写你的内存（见暴雪的 EULA）
- EULA 可以要求全权读写你的硬盘，允许供应商下载你所有的文件（见 Gator 的 EULA）。
- EULA 甚至可以规定，一旦你接受了 EULA，你就接受了供应商未来所有还未发行（见苹果电脑的 EULA）的 EULA。

开放式的令人惊讶的 EULA 充满了这样的内容，并且还很常见。

有些人认为，EULA 的观点并未在法院经过适当的检验，因此 EULA 不能算作有效。这是对合同法的一种误解。在过去，唯一可以挑战 EULA 的方法是找出其中的合同条款是否有不当之处。在某些情况下，一些特定的条款被认作是不当的。在另一些情况下，当然是没有不当的。结果是，EULA 有时会被法庭终止，有时则不会。

4.4.1 索尼 BMG 的 EULA：大量的 rootkit

索尼因为在毫无察觉的 CD 购买者机器上安装 rootkit 而陷入了大麻烦（见"索尼陷入黑暗面"）。但是公司的 EULA 明确地说明了它的意图。这不能让索尼所做的事情变得正确，但它至少改变了它的法律处境。

这里是索尼 CD 的 EULA 中关于安装 rootkit 的一部分。

> 一旦同意了 EULA 的条款和条件之后，本光盘将自动安装一个小型的所有权软件程序（软件）在你的电脑上。这个“软件”用于保护光盘上的音频文件，还可能对你的数码内容有一定的帮助。一旦安装，该软件会留在你的电脑中，直到被移走或删除。但是，无论安装在你的电脑里与否，该软件绝不会用来收集你的任何私人信息。

此外，EULA 接着描述了安装的软件（rootkit）。这里真正的问题是，在按下同意键开始听已经放进光驱的 CD 之前，多少人看了这个 EULA，并完全理解了它？我们可以打赌一定是很少的。

幸运的是，在 rootkit 的新闻震撼出击后，这个 EULA 未能让索尼免于被联合起诉了至少 15 个诉讼。联合起诉被归档在纽约地方法院之后很快就处理掉了。最终，这些诉讼都被合并起来，索尼 BMG 赔偿了带后门 CD 购买者（作为交换给了他们歌曲的 MP3 文件），并且为它发布的不同 rootkit 制作了卸载软件。索尼还同意不再制作发行带有两个 rootkit 的 CD。希望索尼人学到了教训。

我们能从中学到的是，即使在 EULA 里的内容也未必合法。在这个例子里，索尼过界了，法律以保护消费者告终。

供您参考　索尼陷入“黑暗面”调查

索尼真的非常想要确保用户没有偷着用它的音乐的数码版本，以至于这家公司会离谱到在你的机器上扎根，以确保你没有这么做。这是一种可怕的发展趋势，有很多原因可以证明这一点，包括：把间谍软件用作安全功能的道德问题，恶意黑客可以用这种间谍软件进一步危害电脑安全的事实，以及这种缺乏思考就传播的安全机制对隐私的侵害。但最糟糕的结果可能是，索尼的 rootkit 间谍软件就此强制成为你的 IT 成员之一。简单来说，就是这东西将会很难彻底清除掉。

Sysinternals 的创始人 Mark Russinovich 发现，XCP2（一种由英国的 First4internet 出售，Sony/BMG 许可和使用的 CD 保护系统）会监控电脑的使用，并且不允许用户从 CD 拷贝音乐 < http://www.sysinternals.com/blog/2005/10/sony-rootkits-and-digital-rights.html > 。当 XCP2 保护的光盘放进 Windows 系统的电脑时，Windows Autorun 会拷贝一个小软件到电脑里。从那一刻起，如果用户试图从受保护的光盘中盗取音乐，软件就会用噪音代替音乐。

拷贝保护软件并不新鲜，那么为什么这个就是 rootkit 呢？这种软件会通过隐藏特定进程、改写 interrupt address table，以及插入各种核心级系统调用中来掩饰自己，把自己和普通的诊断工具和一些安全产品区分来。这让 XCP2 很难被发现。同时它也很难被清除掉。

要运行一个大型的电脑网络是很困难的，但想象一下，试图管理一个每一台电脑都装了隐形 rootkit 的大型网络会如何呢？使用 rootkit 的一个最基本的动机就是要保持隐秘。也就是说，他们试图隐藏自己的存在，不让系统管理员或其他安全工具发现。事实上这让 rootkit 成为了一个危险的武器（rootkit 作为攻击者常用工具中的第一名是当之无愧的）。当一个安全的进程接受了这种坏行为，要管理一台电脑几乎是不可能的了！因为要找出像 XCP2 这样的隐藏进程很困难，而且要花很长时间。在你的公司电脑上大量安装这种程序，你就可以找到问题了。

但是事情还可以更糟。索尼的 rootkit 技术不仅隐藏了反盗取进程，还会隐藏起任何特殊标记过的进程。根据 Russinovich 的说法，XCP2 的隐藏机制会隐藏起任何文件、目录、注册表、或是名字以“ $sys $ ”开头的进程。这意味着，恶意软件可以利用索尼的 rootkit 来避开监测。事实上，在事情曝光几天后就出现了试图利用索尼 rootkit 的病毒。还有传言说玩家利用这种代码隐藏游戏入侵程序不被在线游戏 WoW 的侦测系统发现。

除此之外，迫于压力而制作的网络版卸载程序（很难找到，并且仅供索尼 CD 的实际购买者使用）被关联到个人和一台机器。也就是说，它只能卸载一台被感染的电脑。咨询部门不会对此感到满意。此外，Ed Felten 认为 <http://www.freedom-to-tinker.com/>，这个网络版的卸载程序会在它运行的电脑上打开一个严重的安全缺口。这就是有毒的废物，不是吗？所以公司咨询部门首先要判断一下，他们的机器上到底有没有这种隐藏软件存在，然后把它安全地删除掉。这两项行动都很耗时，而且在部分机器上可能会有困难。

XCP2 还可能影响电脑的正常操作。该程序每两秒扫描一次系统中所有运行的进程，查询文件的基本信息，并且消耗 2% 的系统资源（因为马虎的编码）。用户报告说，感染了 XCP2 的电脑受到很大影响，大部分影响都很不可思议，直到索尼的事情公开，问题之源的音乐 CD 经过了清理。一些防病毒机制和个人防火墙在面对 XCP2 时几乎发了疯。

《华盛顿邮报》报道说："加利福尼亚一半的消费者联合起诉，他们受到了索尼音乐 CD 安装的反盗版软件的伤害。一起起诉索尼的二级全国联合诉讼将由纽约法庭受理。"这是索尼的耻辱。希望正义获胜[⊖]。

4.4.2 暴雪的 EULA：你的内存都属于我们的监视范围

我们在第 2 章中已经说过，WoW 的玩家都被典狱官程序监控着（由暴雪安装和维护）。暴雪的 EULA 中直接说明了这个行为。特别是，暴雪的 EULA 允许暴雪一网打尽你的电脑内存。下面是出自该 EULA 的一点有趣的小片段。

> 同意接受监控。当游戏运行时，会对你的电脑的随机存取内存进行监控，以防有未经许可的第三方程序和游戏同时运行。这里的"未经许可的第三方程序"应该被解释为任何第三方软件，且不仅限于暴雪定义的任何"AddOn"、"MOD"、"黑客"、"数字逻辑演算装置"、或是"作弊"：（i）使用户可以用任何方式作弊或是便于作弊；（ii）使用户可以修改或是破坏游戏界面、环境、以及通过非暴雪认同的方式获得经验；或是（iii）截取、"挖掘"、或是从/通过游戏收集信息。当游戏发现有未经认证的第三方程序时，游戏可能会（a）把信息发送给暴雪，包括且不限于你的用户名、侦测到的未认证第三方程序的细节、以及侦测到未认证第三方程序的日期和时间；并且（b）不论是否通知用户，本协议保留所有相关权力。

记住，并不是 EULA 规定了一个行为，并且你无心地接受了它，这种行为就是合法的。一些 EULA 条款已知是在法庭上败诉了的。

就我们现在探讨的这个问题，暴雪的 EULA 中有一条关于出售游戏中道具的禁止条例值得一提：

> 游戏中或相关的所有权和知识产权以及所有的拷贝（包括，且不仅限于所有的标题、电脑代码、主题、物品、角色、角色名、故事、对话、惯用语、场所、概念、美术、角色的任何详细记载、建筑或场景的设计、动画、音效、混音、视听特效、故事情节、角色肖像、操作方式、精神权力、所有相关文件、以及合并在游戏中的小型应用程序）都属于许可人所有或由许可人特许授权。游戏受到美国著作权法律、国际著作权条约及其他法律的保护。

⊖ 这部分内容出自 2006 年 1 月的《IT Architect》专栏"Sony/BMG 是由恶意黑客运营的吗?"作者 Gary McGraw <http://www.itarchitect.com/shared/article/showArticle.jhtml?articleId=175001534>。

看起来想要建立 WoW 角色再出售来赚钱行不通了。唉!

当暴雪的 EULA 遭到违背时（通常是“典狱官”程序发现的），暴雪常常会注销你的账户。预先警告一下。

4.4.3 Gator 的 EULA：从不受欢迎的访客

Gator 公司曾经是一个恶意广告软件公司。这家公司出售网络钱包，制作填充程序追踪上网行为，并且弹出广告。这个隐藏程序使用 Gator 称为 GAIN 的技术来追踪用户行为。也就是说，Gator 使用间谍软件。Gator 公司现在变成了 Claria，不再支持 Gator 软件。

Gator 间谍软件有一个长达 63 页的 EULA。在这个 EULA 里深深隐藏着非常不一般的内容：

> 同意你不会使用、或是鼓励其他人使用任何未经认证的方法从电脑上移除 GAIN AdServer 或是任何支持 GAIN 的软件。

这真让人喜欢。你不被允许清除该软件。很好。甚至是间谍软件清除程序也不被批准用来清除 Gator AdServer。

Gator 在往回发送你的个人上网资料的同时还禁止你使用信息包嗅探器来判断它到底在做什么：

> 严厉禁止使用任何信息包嗅探器或其他设施来中断或进入 GP 与 GAIN AdServer 之间的通信。

你所有的资料都属于 Gator。很高兴看到这样的想法就此消失。

4.4.4 微软 FrontPage2002 的 EULA：友好点，因为你别无选择

> 你不能使用任何与这样的立场相关的软件：诋毁微软、MSN、MSNBC、Expedia、或是他们的产品及服务，侵犯他们的任何知识产权或其他权力，侵犯任何州、联邦、或是国际法律，宣扬种族主义、仇恨或是色情。

微软这位思想警察想要知道你用他们给你的魔法棒干什么。

4.4.5 带有 EULA 的病毒：病毒软件合法化

在 2002 年，一种叫做 FriendGreeting 电子贺卡的病毒软件广泛流行，它会把自己复制发送给受害者 Outlook 联系文件中的每一个人。但是病毒的作者在 EULA 中加入了特别允许这种行为的条款，从而在起诉中得到了相对的保护。如果用户点击了同意，病毒就会开始工作，而作者却得以脱身。

这是一个 EULA 腐败的真实故事!

4.4.6 苹果电脑的 EULA：无穷与超越

根据报道，苹果电脑的 EULA 是一个展开性的文件。同意接受了 EULA，你就同意了 EULA 在未来任何的任何添加和修改。对我们而言这听起来真是没有任何限制！下面是 EULA 中的内容：

> 苹果保留权力，在任何时候或是有时，对这份协议进行更新、修正、补充、以及其

他方式的更改，来增加新的或是附加的规则、政策、条款、或是使用服务的条件。这样的更新、修正、补充修改，以及附加规则、政策、条款和条件（在这份协议里全部归结为“附加条款”）都将立刻生效，并和当前协议结合起来。如您继续使用 iTunes 音乐商店，您的行为将被认作是接受了所有此类附加条款。所有的附加条款会根据这里的说明与此协议结合起来。

4.4.7 EULA 大阅兵

有一件事很明白了——当你同意接受了一个 EULA，你就有法律责任根据授权软件的规定来约束自己的使用行为。如果你在同意之前没有弄清它的内容，你会为此付出代价！从现在起看看那些 EULA 吧。如果你对自己要接受的 EULA 感到担心，就去看看 EFF 的文章“Dangerous Terms：A User's Guide to EULAs”《危险条款：EULA 用户指南》（见侧边栏的 EFF 对于 EULA 坏条款的意见）。

但是，真正的问题是，像我们看到的这些 EULA 是否真的对阻止软件盗版事业不可或缺。它们是否其实还有更深层的含义呢？你不得不承认，使用指定服务器来玩网络游戏是抑制盗版的最佳策略。但是对于游戏公司审查你的电脑中的个人信息，你又感觉如何呢？你对那些公司挖掘你的电脑中的资料有什么想法？你对于视频游戏安装 rootkit 有何感觉？DMCA 以及伴随的 EULA 是从何时开始无关版权保护内容了的？这是当今网游玩家面临的各种法律问题。

供您参考 EFF 对于 EULA 坏条款的意见

在《危险条款：EULA 用户指南》一文中，Annalee Newitz 稍许探讨了一下“无数可能对消费者造成伤害的条款写入了 EULA 中。”在这里我们重复一下她列出的条款（已获 EFF 转载许可），不做任何说明。原文见 <http://www.eff.org/wp/eula.php>。

会损害消费者的常见 EULA 条款：

1. “不可公开对本产品进行批判。”
2. “使用本产品即代表您将被监控。”
3. “不可破解本产品。”
4. “本产品不可和其他卖主的产品同时使用。”
5. “一旦在本合约上签字，您也同时接受了未来对其进行的任何更改。并且，EULA 的更改不会特别通知。”
6. “如果本产品把您的电脑弄砸，我们不负任何责任。”

4.4.8 禁止破解

DCMA 明确禁止为了破坏版本保护机制而进行的破解。其目的在法律中已经声明，是为了执行著作权法。问题是，破解类似游戏客户端这样的软件到底是否合法。如果隐藏 API 被曝光，著作权是否遭到侵犯？如果安全设计不完善会怎样？

注意，DMCA 中说明，为了计算机安全研究的例外，以此保护研究人员和学院。虽然如此，这仍是法律中非常模糊的一块区域。

4.4.9 禁止游戏入侵

没有人给游戏入侵建立起一个法律定义。不过，在 EULA 中或是使用限制（我们很快会进行说明）中经常会出现入侵、作弊、mod，有时甚至还有网络嗅探。

4.4.10 财产权

财产权是网络游戏的另一个法律盲区。尽管暴雪很努力地通过在公众面前摆出姿态想要弄清这件事情，事实上依然没有一个真实的处理虚拟财产的先例。暴雪做出类似这样的发言："《魔兽世界》的使用条款中声明了《魔兽世界》中的一切都属于暴雪所有，暴雪不允许用真实货币买卖游戏中的道具。因此，暴雪娱乐公司会采取任何以及所有必要的措施来制止这种行为。我们不仅相信这是违法的，而且它本身可能会伤害到游戏经济，以及数千万其他 WoW 玩家的整体游戏体验。"

但是，最近一起对 *Second Life* 的供应商 Linden Lab 提出的诉讼可能会改变这一切。来自宾夕法尼亚州的律师 Marc Bragg 最近十分沮丧，因为他的虚拟土地交易失败，而 Linden Lab 没收了他的虚拟财产。*Second Life* 以明确允许玩家合法拥有自己的创造物而知名。这导致了巨大的网络经济与现实的不动产、衣服、车、以及其他虚拟物品之间进行的大量交易。

在 *Second Life* 这个案例中，比较知名的是 Bragg 用来敛财的技术。他注意到 *Second Life* 在线拍卖系统使用 URL 中的标签来追踪被拍卖的财产。通过改变标签（一种经典的攻击方式，我们在 *Exploiting Software* 一书中有详细说明），Bragg 有效地对已经登录但尚未公开出售的物品进行竞拍。这个案例很有趣，值得观察。

在任何情况下，如果你和游戏制作商发生了冲突，你所投入的所有时间和努力都会理所当然地消失。

4.5 使用条款

游戏公司使用的另一项法律武器是使用条款协议。这些 TOU 文件（有时称为服务条款【TOS】协议）常常伴随着 EULA 出现在客户端 -- 服务器游戏中，约束游戏服务器端的使用方式。EFF 说："EULA 和 TOS 协议中有很多条款是相同的。但是标准的 TOS 协议还应该禁止定义模糊的行为和通信方式。"

和 EULA 一样，TOU 文件中也有很多有趣的东西。例如，暴雪的 Battle. net 的 TOU 中有如下声明：

> 你可以将 Battle. net 用作个人用途，但你的以下行为被禁止……
>
> （ii）对 Battle. net 进行整体或局部的复制、影印、再生产、翻译、破解、修改、分解、或是反编译……
>
> （iv）为任何暴雪软件程序架设服务器或提供匹配服务器，或是模仿或修改暴雪用作 Battle. net 一部分的通信协议，通过模仿协议，使用中介程序、修改或增加程序组件，使用帮助程序或是任何现在已知或正在开发的技术，来达成任何目的，包括且不限于通过网络玩游戏、使用商业或非商业游戏网络联网玩游戏、或是事先未经暴雪书面同意用作内容聚合网络的一部分，或是为任何商业目的利用 Battle. net，包括且不限于在

网吧、游乐场所、或是其他向用户收费的场所（不论是按小时收费或是其他方式）使用 Battle. net；

（v）使用任何第三方软件修改 Battle. net 来改变游戏玩法，包括且不限于作弊或是入侵。

注意，TOU 协议可能和 EULA 一样可任意修改。下面是微软 TOU 的一个要求范例：

使用条款。微软保留修改条款的权力，在此基础上提供支持、远端获取以及软件服务。一旦继续使用支持或是本网站，您将被默认为接受了本条款的任何变更。

4.5.1 禁令

TOU 文件经常会宣布游戏版权者在任何时候都可以终止协议的权力。下面是出自暴雪 TOU 的内容：

在违反任何前述规定的情况下，暴雪可能行使自己的权力而不必事先通知

（i）临时冻结你的 Battle. net 账号；或

（ii）立刻终止你的 Battle. net 账号；

禁令是游戏公司怀疑作弊行为时最经常采取的行动。

供您参考 《魔兽世界》地形查看器（wowmapview）的消失

2006 年 1 月 10 日，娱乐软件协会（ESA）的 Jack Snyder（一位反盗版调查员）发送了一封关于开源程序魔兽世界地形查看器的终止信函给项目的领导人 Szego Zoltan 和 SourceForge（一个流行的开源程序网站）。虽然这个项目是否哪里不合法这一点还不清楚，这封信足以让项目终止，并且让 SourceForge 从网站上将其撤掉。这种结果很常见。

在过去，铁腕政策和威胁对于游戏制造者而言很有效，所以现在再用这种政策也不会令人惊讶。

下面是摘自电子邮件的一小部分。

ESA 根据《美国千年数字版权法》和 17 USC 512（c）发信，提醒 Sourceforge 注意自己网站或系统上的内容侵犯了 ESA 的某个或更多成员的著作权。你作为 Sourceforge. net 的代理人接受这个侵权通知，同时在美国著作权办公室留下当前记录。作为违法的处罚，我们在此声明，ESA 有权为维护 ESA 成员的利益采取行动，如上文所言，我们相信他们的著作权受到了侵犯。

ESA 坚信，网站 http：//wowmapview. sourceforge. net/和 http：//wowmapview. sourceforge. net/wowmodelview/因为提供下载某个或多个受到著作权保护的游戏产品的某个或多个未经授权的接口程序，对某个或多个 ESA 成员造成了持续的著作权侵犯，其中包括且不仅限于《魔兽世界》。

出现在这类网站、或是通过网站可链接到的这类游戏产品未经授权的接口程序已被列入黑名单，或是立即会根据它们的标题、其中的变量、或是其中描述到的艺术性内容（包括游戏标题、文字说明、清单以及任何关于或借鉴此类游戏产品任何内容的描述，在下文中将被认作是“侵权材料”）。根据 ESA 在 2006 年 1 月 10 日所处理的信息，ESA 相信，本通知中的说明正确，并且恰当地描述了侵权材料的侵权本质和状况。

根据我们的资源，暴雪从未将这些案例告上法庭，除了本章之前提到过的 BnetD 案例。在那个例子里，暴雪的声明仅仅局限于在线授权检测机制。在所有其他案例中，侵权声明足以用来让游戏狂热者终止所有的运作，放弃软件项目。

4.5.2 被起诉不等于违法

游戏公司还以通过起诉来制止游戏入侵。要知道，被起诉不等于违法，这一点很重要。即使你完全没有违法，你也可以被起诉。这正是我们热爱诉讼的社会的伟大之处！

在介绍“魔兽世界地形查看器的消失”的内容里，我们简单地探讨了用起诉来威胁对方通常会发生什么事。

4.6 盗窃软件与游戏入侵

很多关于网络游戏的法律原意是为了有助于控制盗版（例如，直接拷贝并发行属于别人的软件程序）。使用客户端－服务器结构的更复杂的网络游戏，其面对的情况也更复杂。网络游戏入侵和过去的软件破解并不完全相同。而法律中的先例都是处理软件破解和盗版的。当法庭开始认识到这一点之后会怎样呢？大家都在猜测这一点。

第5章　被程序bug包围

bug和缺陷是电脑安全风险的主要原因。对银行系统的程序来说是如此，对于网络游戏来说也是如此。软件里有如此之多的安全相关bug，这些bug又具有着蔓延性，以至于像Fortify Software <http：//www. fortify. com>这样的软件商制作了专门寻找bug的工具。不仅如此，科学家们还发表了大量关于bug分类的论文，并且各持己见。

一种普遍流行的分类法是McGraw的《软件安全：内建安全系统》（*Software Security*：*Building Security In*）一书中的“邪恶七领域”。下面是这七个领域：

1．输入校验和断言
2．滥用API函数
3．安全功能
4．时间和状态
5．错误处理
6．代码质量
7．封装
*．环境（细心的读者会发现这是第八个“邪恶领域”）

注意，这里的领域（类似生物学分类法中的领域）是以普遍程度为顺序列出的。也就是说，类似缓冲溢出、SQL语句注入，和跨站式脚本（XSS）这样的输入校验和断言问题是最常见的bug类型，紧接着是语言和库的API函数滥用，再接下来是安全功能中的错误，以此类推。

或许最有趣的bug要数排第4位的时间和状态类。这类bug之所以很有意思，主要是因为它还是一种对未来的预兆。时间和同步问题已经成为一个主要问题。但是随着分布式系统和多线程语言变得越来越普遍，同步和状态追踪的问题只会随之变得越来越常见。（见摘自《软件安全》一书侧边栏中提到的“邪恶领域之四：时间和状态”中的片段。）因为网络游戏设计的方式，它们普遍存在着时间和状态问题。

供您参考　邪恶领域之四：时间和状态

分布式的计算是关于时间和状态，也就是说，为了让一个以上的组件能够相互通信，状态就必须共享，而所有这些都必须花时间。状态的时间差可算是目前最大最丰富的可供攻击资源了。

大部分程序员各自为政地进行（或者更准确地说，只是自我式地思考）开发工作。他们把自己编码设计当作是单线程的控制方式，并且认为这样就够用了，在写整个程序的时候他们也按照自己手动工作的方式来单线程进行。这真的很奇怪。现代的计算机在各个任务之间快速切换，并且有着多核、多CPU、或分布式系统，两个以上事件可以在同一时刻发生。程序员想当然的程序运行方式和实际情况之间的差距带来了很多问题。这些问题和线程、进程、时间以及信息之间的意外交互相关。这些交互发生在共享状态中：信息发送、变量、文件系统、分类储存系统，以及所有可以储存信息的东西。

很快有一天，这类 bug 将会排名第一。

已获《*Softwar Security: Building Secarity Im*》引用许可，作者 Gary McGraw（Addison Wesley 出版社，2006 年出版）。

5.1 游戏中的时间和状态 bug

我们在第 1 章说明过，大型网络游戏是大型分布式程序的先例。当成千上万的客户端进程在一台普通服务器上通过网络实时交织在一起的时候，移动状态准确地导致网络游戏软件安全第一大问题——“竞态条件”。

竞态条件和其他状态问题是网络游戏中 bug 的主要来源。网络延迟（会让时间扭曲得很有趣，有点像黑洞）会让这些问题进一步恶化。大型网络游戏由于特别的设计，需要把大量的数据存储分布在许多服务器上。有些服务器可能存储的是账号信息，其他可能存储着玩家的统计数据和财产清单，还有另一些存储着网络世界中场景和其他事件当前的状态。

从技术上来说，事情的真相是，大部分 MMO 并不是真正地拥有一个独立完整的网络世界，而是由很多重复的部分组成了看似是同一个的世界。举例来说，WoW 的每一个平行的网络世界限制每台服务器用户的人数为 50 000 人。《星战前夜 EVE Online》则是单一的网络世界，但是这个虚拟世界分布在如此巨大的一个分类储存系统里（该游戏世界是一个太阳系），没有一个服务器曾经超载过。

多重世界有边界问题。竞态条件建立在软件边界状态的基础上——例如登录和注销的状态。如果一切都按照不可分割的整体来进行，比如你从登录到注销，不需要无数步骤就可以一下子完成，那么事情就会很好。但如果需要多重步骤，并且这些步骤不受计算机科学家称之为临界区的信号量的保护，麻烦就会突然出现了。

为了让你了解得更清楚一点，请思考一下图 5-1。在这个简单的图例中，本应是一个整体的步骤被分成了三部分。在竞态条件攻击中，攻击者插入三个步骤的状态切换动作之间，来扰乱世界的状态。要防止这种方式的攻击，你可以把这三个步骤都标记为临界区，只能同时执行它们，而不允许插入。当然，临界区状态会暂停所有操作从而变成一个严重的瓶颈，所以有时开发者会走捷径。

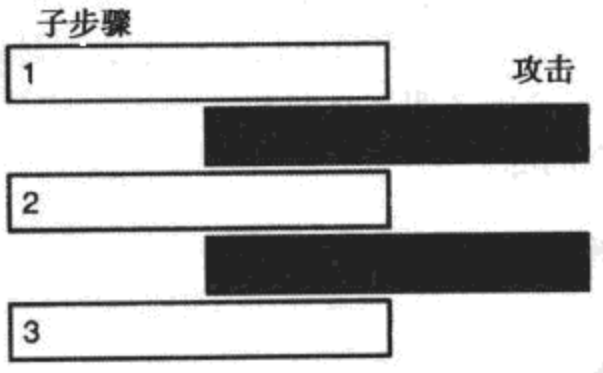


图 5-1 一个简单的竞态条件图示。白色长方形代表原本不应分开的三个步骤。因为它们在临界区域不受保护，攻击者通过可以在各个步骤之间插入攻击，把状态弄混乱（如灰色长方形所示）

“竞态条件 101”中给出了一个简单的例子，说明可能发生在真实世界中的竞态条件。或许这曾经发生在你身上，谁知道呢？你看，这些类型的错误都是有关于时间的。

供您参考　竞态条件 101

什么是竞态条件?

让我们假定，爱丽丝和鲍勃在同一家公司工作。通过 E-mail，他们决定一起吃午饭，并约定中午在大厅碰头。然而，他们没有约定他们是要在办公室的大厅，还是这座建筑下面几层的大厅。到了 12 点 15 分，爱丽丝站在公司大厅的电梯旁等鲍勃。这个时候，可能鲍勃在大楼底层的大厅等她。她找鲍勃的方法是乘电梯下楼去看鲍勃是否在一楼。

如果鲍勃在那里，那么一切还好。如果不在，那么爱丽丝是否可以推测鲍勃要么是迟到了，要么是爽约了? 不能。鲍勃可能一直坐在大厅等爱丽丝。在某一时刻，爱丽丝可能在楼上等待，而与此同时他乘电梯上楼来找她。如果爱丽丝和鲍勃同时在电梯里，除非是同一个电梯，否则他们会在途中失之交臂。

当鲍勃和爱丽丝彼此假定另一个人在别的地方并且留在原地，便都去搭乘电梯，他们就被竞态条件所控制了。当一个假定需要在一段时间内有效，而事实上可能却不对，这时竞态条件就出现了；到底对不对完全取决于确切的时间点。在每一个竞态条件中，都有一个空窗期，也就是说，在其中一段时间内，假设如果被破坏，将会导致错误的行为。在爱丽丝和鲍勃的例子里，空窗期大约是乘坐两次电梯的时间长度。爱丽丝走进电梯的时候，鲍勃的电梯恰好到达，爱丽丝仍然错过了他。鲍勃走进电梯时，爱丽丝的电梯正要到达。我们可以想象一下，爱丽丝的电梯门打开时，鲍勃的电梯门恰好关上。当假定被破坏，导致了意料外的行为，竞态条件发挥了作用。

已获《*Building Secure Software*》转载许可，作者 John Viega 和 Gary McGraw（Addison Wesley 出版社，2001 年出版）。

在本章中，我们会描述一些测试关卡和模板，你可以用来寻找并利用网络游戏中和时间相关的 bug。为了让你看清它的真实性，我们会重点说明在 WoW 中的一些已经被发现和利用的 bug。

5.1.1　如何免费玩游戏

序列图是说明按时间顺序发生的事件的好方法。图 5-2 展示的是关于一个简单状态的问题，让你可以不用付钱。

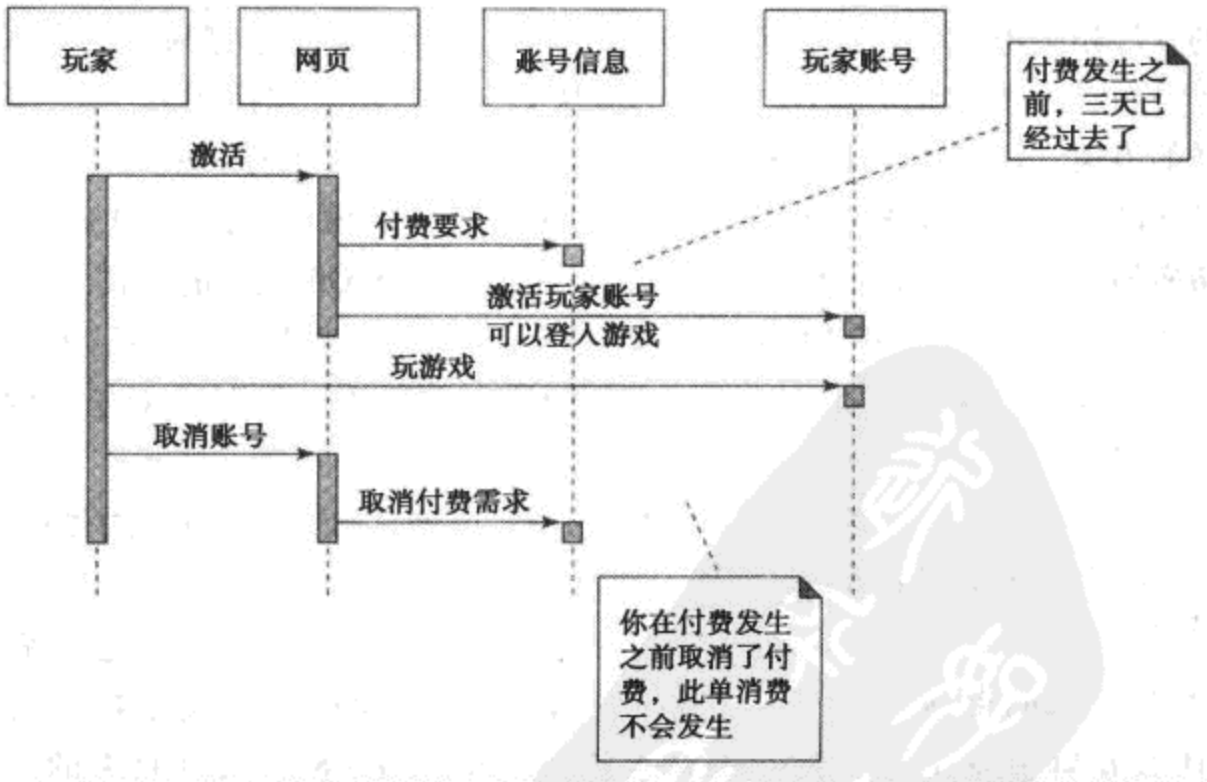


图 5-2　这个图例显示了你取消付费时发生的事情（你看到在本图中潜藏的竞态条件了吗?）

一旦你激活了自己的账号，几件事情就会发生。第一，你的账号信息会更新（你现在欠游戏公司钱）。第二，你的玩家账号会被加上一个游戏数据库编码。然而，这些事情可能不是同时发生的。事实上，在你的付费要求到达账户服务器和你实际付费的时间之间，空窗期开启了。这个时间上的延迟有时可能长达3天。然而，激活你的角色要用的时间短得多，通常只要几分钟。如果你作为黑客来思考一下，你会发现，在某种意义上你是在空窗期的时间段上免费玩游戏。你可以登入玩游戏，而你在三天内不必付账！不过等一下，下面还有更多……

让我们假设，你在一天结束的时候玩游戏（更实际一点来说，当太阳升起，房间开始变亮的时候）。现在，继续用黑客思维方式，你再次登录网络账号管理系统，并且取消付费。在这时候会发生什么？事情很有趣。你的玩家账号并没有如你想像的被删除。它立刻被置为暂停的状态。同时，账号服务器获取了一个信息，把你的账号标志为不可付费。因为这一切都发生在3天的空窗期时间内，你不会收到任何账单。

在经过一天辛苦的工作，你回到家，准备开始振奋地打一晚上游戏。你恢复了自己的账号，三天的空窗期（再次）从零开始计时。你（又）玩了一晚上，当太阳（又）升起时，你例行取消了付费。

你可以继续用这种方法免费玩上几年的游戏，而一次账单也不用付。当然，你的账号上的这种行为方式可能会引起注意。“嘿，等等，这家伙一个月取消了10次账号，发生什么事情了?”但传言说，这种免费玩游戏的攻击方式对WoW也有效，或者至少曾经有效过。在这里只是记录一下这种方法，我们并没有试过这种方法，因为我们坚信，想要玩游戏的人就应该付钱。

5.1.2 用bug扰乱状态边界

我们玩弄的取消/付费的伎俩就是状态边界的一个例子，但比起软件而言，状态边界其实和商业流程联系更加密切。不过，和时间差问题相关的软件状态边界还是很多的。

最明显的软件相关边界与数据库有关。多重数据库之间的交互很容易受到竞态条件的影响。因为虚拟世界分布在很多台服务器上，做着类似从一个地宫场景切换到另一个，从一块大陆飞到另一块这样的事情，这经常会让玩家从一台服务器切换到另一台服务器。

这种切换在游戏里是很普通的事，一定会经过游戏公司的质量保证（QA）部门（并不是多次进行实际切换的大量玩家）详细计划过的检测。下面是很多QA工作的方式。检测计划会说类似“当玩家在入口68执行行为47时，财产目录预定不变。”这样的话。然后测试员登录，走到入口68，执行行为47，看看是不是一切都好（例如，玩家的财产目录）。对于软件测试员来说，这是功能测试。问题是，这种测试既枯燥又死板!

你看，攻击者并不常常像你预计的那样做。相反，攻击者的重点放在努力做些程序员没有预料到的事情。他们做意想不到的事情，有时会有非常荒唐的结果。

我这里有个点子。不要再去温文尔雅地走过68号入口，不如在你这么做的同时注销游戏。把以太网技术的束缚抛到一边。和任务管理器一起把客户端干掉。之后，再登录游戏看看有什么新鲜事发生。你是到了新大陆还是依然在68号入口原来的一边？你的角色状态如何？

让我们再次切换到破解者的思路，逐步跟踪一些可能性。让我们假设，你最终在入口原来的一边。如果你给自己的玩家朋友一些钱，几秒之后立刻结束进程，然后朋友继续像往常一样通过68 号入口，会发生什么呢？当你在这期间登录游戏，回到 68 号入口原来的一侧，看看你的钱包吧。那些钱是从钱包里被拿走了，还是随着你的位置一起被重置了？如果它确实重置（回到给钱之前的初始数值）了，你的朋友是否在他的一侧入口仍然得到了那些钱？如果你的钱翻倍了，你就发现了复制 bug——这是网络游戏永远最被人垂涎的 bug 之一。没什么能比免费复制财产更加酷的了！

在 WoW 中，大量这样的 bug 存在于从传送点到副本实例的边界。因为 WoW 中的副本实例就像大陆或是其他场所一样，是在特定的服务器上处理的，在玩家进入副本的时候，他们实际上就是一团从一台后台服务器转移到另一台后台服务器的数据。

一般来说，一台实例服务器负责运行指定副本的所有实例。例如，所有的“矿坑”场景实例都是在同一台矿坑服务器上。然而，因为这是游戏里很流行的一个任务，服务器开始负载，并且开始延迟。对于从游戏外想要利用竞态条件的人而言，延迟的服务器是非常理想的目标。

图 5-3 是一家出售 WoW 黑客和入侵工具的公司出售的复制方法（真实货币买卖）。这是卖家利用竞态条件制作入侵工具的最好例证 < http：//www. wow-dupe. com > 。我们购买了一份看看是否能起作用。

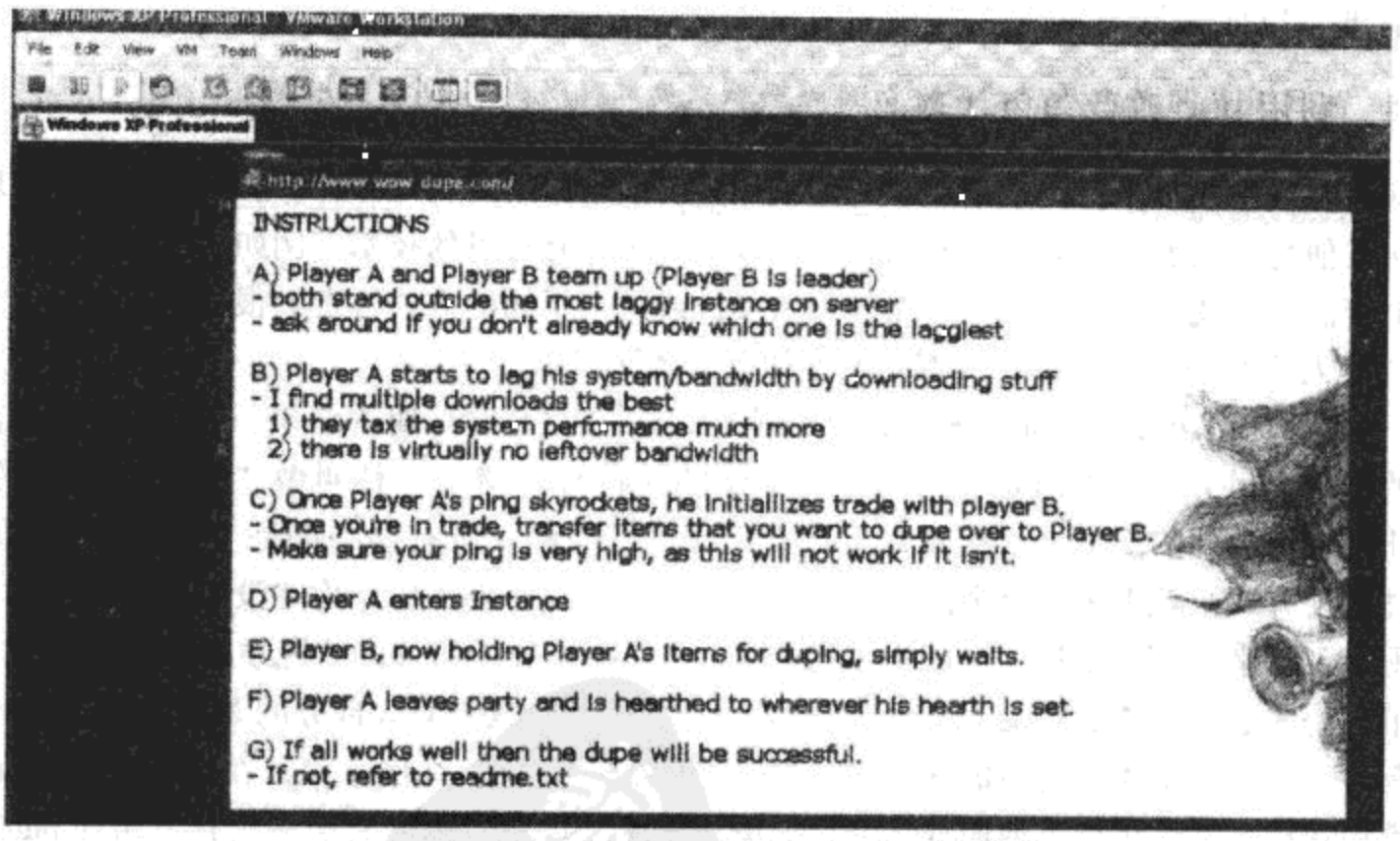


图 5-3　一个黑客入侵网站出售的复制入侵方法截图

花了 19. 95 美金，我们却被愚弄了，或是被当成了傻瓜，因为我们尝试后发现这种复制方法不起作用。不过，这个作弊的方法对我们来说听起来很合理，并且它确实示范了如何利用这些竞态条件作弊。

注意图 5-3 的截图中有一段特别说明，告诉用户如何消耗掉本地网络链接的带宽。在

WoW 中有一个 ping 值计数，用来测定你的客户端程序延迟了多少。不过，如果你真的对利用竞态条件感兴趣的话，服务器延迟比起你的本地网络连接或是客户端延迟要重要得多。并且服务器延迟不是你能控制的。你可以在自己的客户端机器上使用一个代理服务器来延迟 TCP 信息包的往返，以此模拟服务器延迟，看看会发生些什么。代理服务器连接游戏服务器，并且让你在需要的时候制造延迟。比起用下载来占用带宽制造延迟而言，这种控制客户端延迟的方法好得多。

5.1.3 使用 botnet 引发游戏服务器延迟

事实证明，botnet（俗称“机器人网络”的病毒）是引发网络延迟的一种非常有效的方法（见专门为初学者提供的 botnet 黑客角）。因此 botnet 非常严重地危害到了网络游戏。通过 botnet 引发指定的游戏服务器延迟，攻击者就可以把一切都设置得更适合于竞态条件入侵。

寻找游戏服务器的 IP 地址非常容易。有了 IP 地址，botnet 控制者就可以照例展开各种行动引发游戏服务器延迟；之后，网游小偷就可以更加成功地进行竞态条件攻击。像这样的骗术：用同一个账号多次登录服务器，重复登录会被拒绝，这当然是有原因的，但是重复登录的进程会导致计算上的延迟，这是一种延迟乘数。无数相似的延迟乘数出现了。一旦服务器开始延迟，竞态条件加剧，入侵者就可以开始复制道具或是金钱，或是任何他想要做的事，以此大捞一笔。

5.1.4 利用 bug 改变角色状态

网络游戏中的很多行为会导致角色状态的变化。在 WoW 中会改变角色状态的首先就是魔法。在其他行动之前，先对自己施加魔法，这是一项很有意思的技术。例如，你可能会对自己使用一个魔法，改变自己行走方式，然后开始飞行或是用不同的方式行走。很多这类想法，包括做一些“没人曾经做过”的事，这都是典型的攻击者即兴思维。

“一次叠加两个技能”是一种找漏洞的策略，偶尔会引发有趣的结果。在 WoW 中魔法交互作用的结果会很惊人。下面是一个例子。在你的宠物身上使用一种叫做“猎豹守护”的魔法，然后让它攻击远处的怪物，紧接着在飞行点搭乘飞行坐骑。所有这些都是 WoW 中的标准行为，但通过这种方式组合起来，有趣的状态交互发生了。在这个例子中，你并没有开始自动飞行，而是获得了飞行坐骑的控制权，可以自己驾驶了。图 5-4 是出自 Youtube 视频的一张画面，示范了这种组合。注意，这类状态交互并不限于 WoW；状态组合攻击对大部分的 MMORPG 都有效。

有时，唯一性魔法或能力的交互会引发有趣的状态变化。举例来说，在加入地面战斗之前，激活猎豹守护，然后在加入地面战斗之后，激活雄鹰守护——你可能会一次有了两种 buff。这是一个状态追踪的问题，伴随着有趣的效果。这个问题的根本是，WoW 在这个例子中有着太多的交互状态，意外的状态组合导致了有趣的行为。

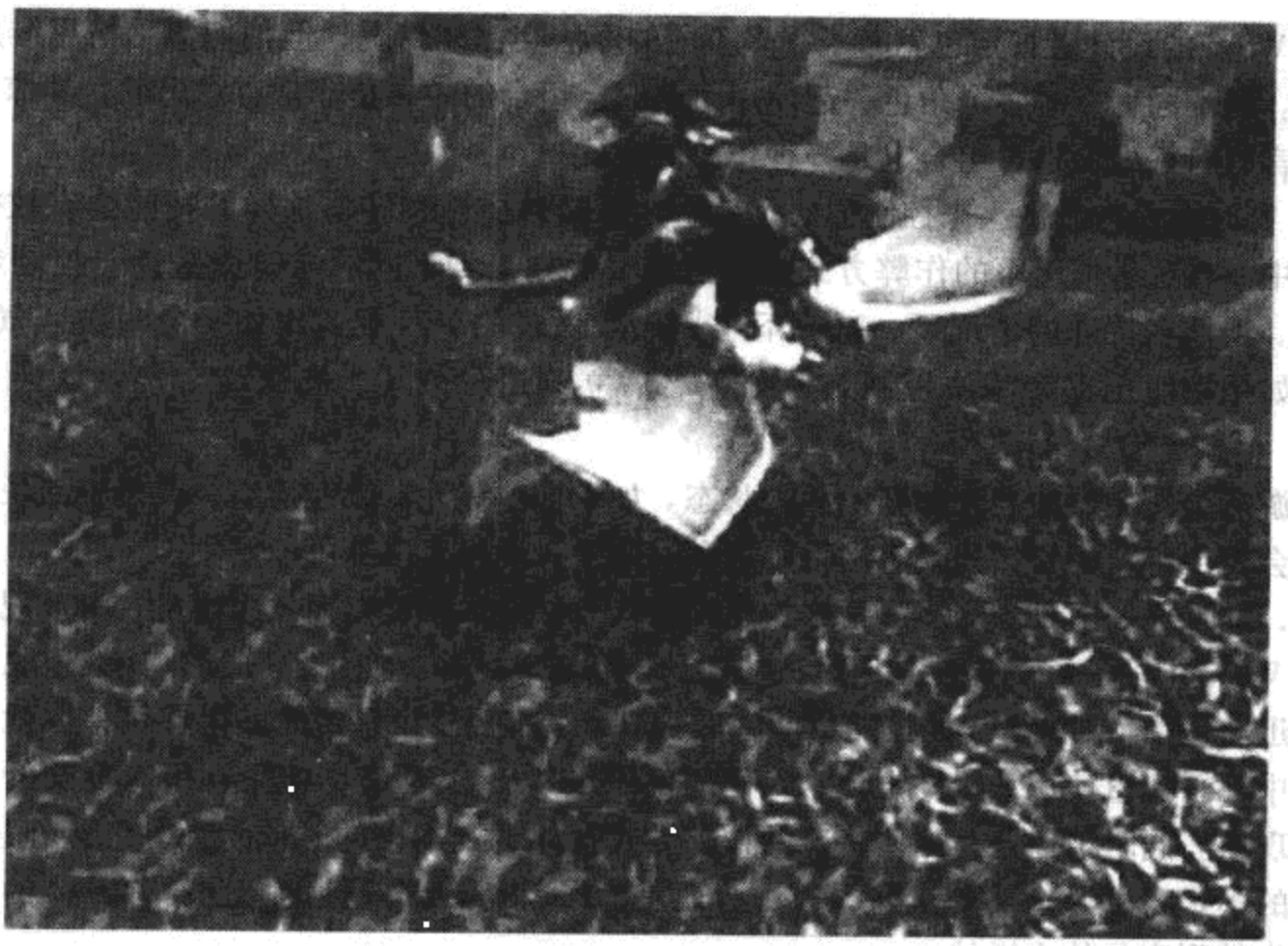


图 5-4　在这张图片中，玩家控制着飞行坐骑，驾驶它沿着地面飞行，这本来是不应该发生的。见 < http：//youtube. com/watch? v = FbEOZnUF66o > 的视频

供您参考　黑客角：botnet

恶意黑客已经控制了互联网电脑网络数十年了。要入侵一台电脑，首先要找到任何一个可远程控制的安全漏洞（通常在目标电脑上安装的软件里发掘），然后利用那个漏洞进行入侵。在早期，基于入侵某种称为缓冲区溢出的软件 bug 的攻击会为攻击者创造出一个可交互的登录副本。在这些攻击中用到的恶意代码称为（shell）外壳代码。Jack Koziol 等作者写的《壳代码编写手册》*The Shellcoder's Handbook*（Wiley 出版社 2004 年出版）是一本很棒的参考书，解释了这类攻击代码是如何运作的，我们的《入侵软件》*Exploiting Software* 一书也是如此（Addison Wesley 出版社 2004 年出版）。

更现代化的攻击方式会在被害者的电脑上安装一个交互后门，例如一个秘密的、无法察觉的 rootkit。Greg Hoglund 和 James Butler 写的《*Rootkits*》（Addison Wesley 出版社 2005 年出版）详细地说明了这种技术是如何运作的。安装 rootkit 的主要目的之一，是为了确保连接到和控制这台电脑，并在将来能够持续占据它。这样每次你想要做点什么的时候，要重新“植入”一台电脑的成本会低很多。

所有被这样植入后门的电脑都被攻击者远程控制着。通过这种方法，每一台被控制的电脑都是一个机器人，所有被控制的电脑就形成了一个机器人网络（botnet）。机器人网络一旦形成，就可以将上千台电脑分散的计算能力联合起来，执行攻击者（黑客的行话里称为牧羊人）的命令。botnet 用来做拒绝服务攻击、大型扫描攻击、发送垃圾邮件以及其他需要大型计算资源的攻击。

建立 botnet 的现代攻击工具设计都具有可自动繁衍特性，利用高效的攻击建立起大型计算机网络。

botnet 在入侵网络游戏时也可以发挥作用。一个很明显的作用就是引发服务器延迟，这样更易于利用时间差和状态相关的 bug。

尽管 WoW 的设计者肯定也思考了很多不同的魔法交互的情况，但他们似乎把思维局限在一个角色上。当不同种族的多个角色一起使用魔法时，组合的效果是非常有意思的，其惊人的结果对玩家而言很有好处。这种组合也是一种作弊的方式。

WoW 有时会允许其他人来控制你的角色。像是精神控制和转移这样的技能会有数据拷贝进你的状态中。在这种情况下的作弊方式很有意思，例如走到不同的入口或是让“控制者”快速注销下线。当你在特殊的情况下，试一下“别人没做过的事”吧。比如一个叫做“气定神闲”的魔法，如果在使用这个魔法控制某个角色之后立刻退出战场，那么这个角色就会一直被控制了。

有时游戏会使用缓冲器来保存不同的状态。如果这些缓冲器没能清空，或是中断联接，有趣的事情就会发生了。记住，只要有网络延迟和时间差，任何事情都可能发生。

有时，如果你一下子做很多事情来快速切换状态，游戏可能因而发生混乱，游戏规则也被打破了。在 WoW 中的一种骗局就是用魔法快速切换状态。比如，使用“狂暴”让战士脱离攻击状态，接着迅速使用“冲锋”（该技能只能在非战斗状态使用）。只要你做得够快，你就能在和玩家的战斗中几乎一直使用冲锋。记住，我们的例子虽然出自 WoW，但是在所有复杂的游戏中都会有这些现象。同时请注意，这些特殊的作弊方式在你看到本书的时候大部分已经经过了修正。我们的目的不是为了讨论某个作弊的方法，而是为了就一个普遍性的问题给你具体的例子，好让你理解作弊到底是什么样的。

5.2 游戏中的路线 bug

我们之前所说的像竞态条件和状态交互这样时间差相关的 bug 并不是网络游戏中唯一一类 bug。其他类型的问题也都存在着。其中一类路线 bug 和虚拟世界中的空间相关。

这里的“路线”所指的是游戏中的怪物（机器人）行走的路线（译者注：即游戏中的道路等路线）。游戏经常会让你的角色和这些怪物战斗。在 WoW 中遇到的很多怪物必须在 3D 虚拟空间里紧贴着你的角色，否则是打不到你的。具有这种缺陷的怪物很容易成为利用路线作弊的牺牲品。

下面我们来说明如何利用路线（译者注：或者叫做寻路问题）的作弊。这个策略中的要点就是站在一个怪物到不了的地方，但同时要足够靠近怪物，让你能用某种远程攻击方式攻击到它。有了正确的位置，一直到你最后打死怪物全程几乎不会受到任何伤害。那是因为，怪物在试图接近你的时候被“卡”住了。（一些读者可以回忆一下 20 世纪 80 年代的 2D 游戏《大金刚》中也有相似的问题。）

这类路线 bug 利用起来特别刺激，因为你必须要到某个路线 bug 存在的地点，可能会离怪群很近，甚至要穿越充满怪物的危险区域。路线 bug 也可能非常微妙，通常对你的位置和你自己的移动都非常敏感，所以如果你搞砸了，没有把事情都做好，那么怪物就会冲过来痛殴你。在这类例子里，就算是作弊者也会真正感到游戏的刺激性。

几乎所有的 3D 游戏都会有卡住怪物的行走路线 bug。这是因为，要设计没有导致路线 bug 出现的卡位点的场景是很困难并且枯燥的工作。但是，路线 bug 只有在和自动杀怪的宏（整晚一遍又一遍）组合起来之后才会真正有用。即使你找到了路线 bug 位，利用它来杀怪也很快就会感

到腻味和觉得枯燥难忍的。图 5-5 展示的是 WoW 中的路线 bug 示范。

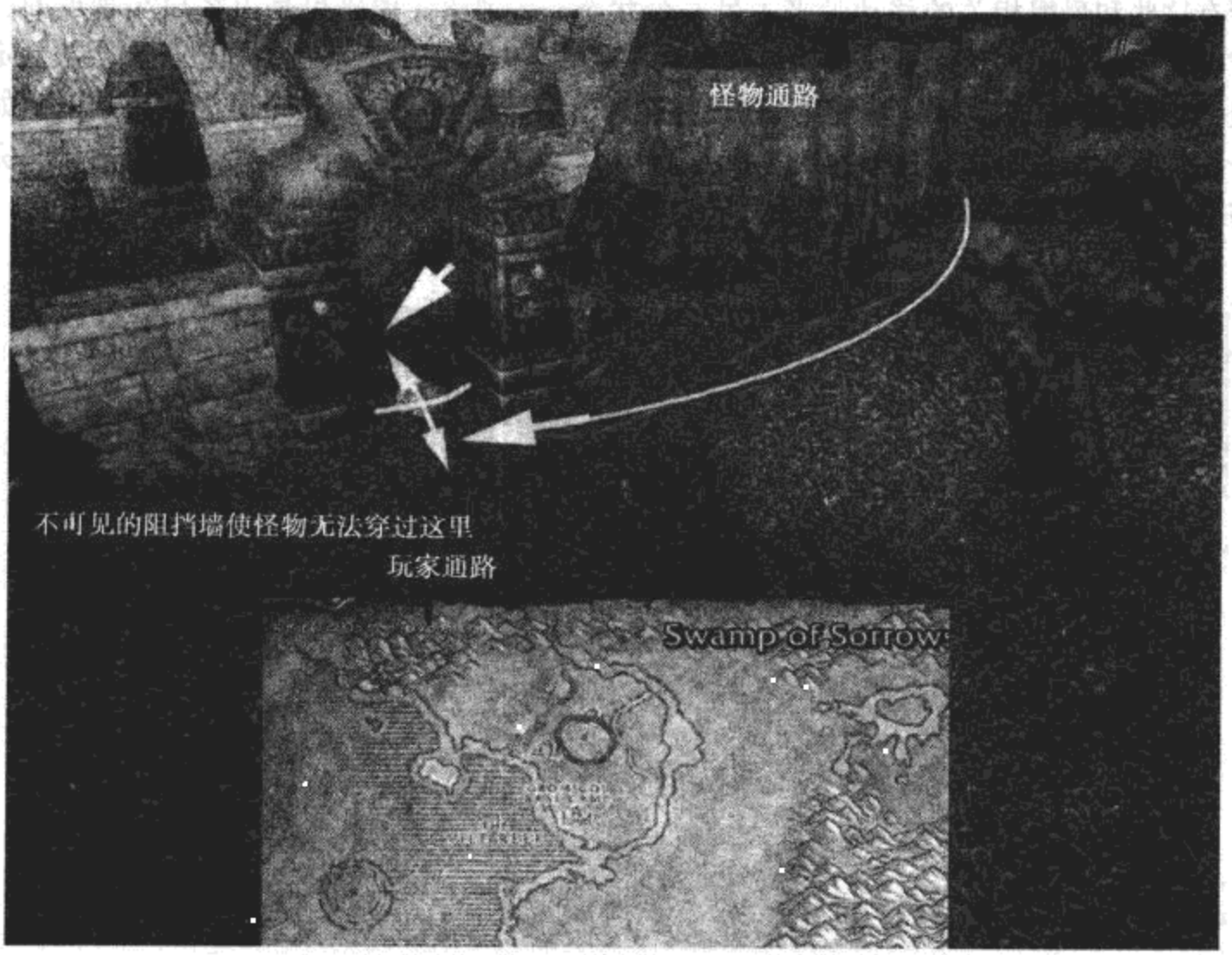


图 5-5 怪物无法穿过拱门，而玩家可以，因此玩家可以待在那个地方，不被怪物攻击。然后，怪物就会为了攻击玩家而在它的两个通道上来回行走，企图到达那个玩家所在区域，但是他到不了，于是怪物循环走动（来自 <http://www.edgeofnowhere.cc/viewtopic.php?t=323124>）

利用 bug 以有趣的方式移动

网络游戏中的虚拟世界只是看上去真实。事实上，它们只是模型建立起来的。

在游戏中移动的方法有很多种。或许你可以飞行。游泳通常也是可以的。可能你还能传送。跑和走是最常见的操作。还会有很多加强你的移动需求的魔法。管理移动的状态机制通常在客户端软件中，至少大部分的 3D 物件交互是如此。很明显，改变客户端就能改变移动的方式。因为这个原因，很多常见的入侵手法会改变客户端处理移动的方式。

仅仅通过覆盖 WoW 客户端里一个字节，角色就可以爬山甚至是穿墙（如我们在第 7 章中所说），玩家用这种入侵方式进入虚拟世界中他们不该进入的地区。利用这样的 bug，角色可以站在铁炉堡大门上，或者爬进山里寻找远处可以钓鱼的湖。更实用的方式包括从一个地点到另一处时直接穿山抄近道。

甚至不用改变游戏客户端本身的代码，也可能通过调用很多移动相关因素，对客户端的移动造成混乱。例如，如果你的角色中了恐惧魔法，像疯了一样跑离施法者。这就意味着，客户端软件很明显是根据你受到“恐惧”时所站的位置来处理这次移动状态，你可能会穿墙或是从墙

的末端跑出去，或者掉进星空中（在地下的区域，你本应不能进入的）。

所有这些和恐惧相关的移动带来了另一个状态——掉落。掉落也是由客户端管理的。事实上，有一个引力常数用于计算掉落。如果你从一定的高度掉下来，落在硬物上，你会被判定为摔死。但是，在 WoW 中，有一个魔法可以让你在掉落后不受伤害。在这个魔法的影响下，如果你让一个朋友对你施放恐惧，让你穿墙，掉入星空，你可能会在某个有趣的地方安全着陆，完全不受伤害。你也可以试试掉进水中。见图 5-6 中的示例。

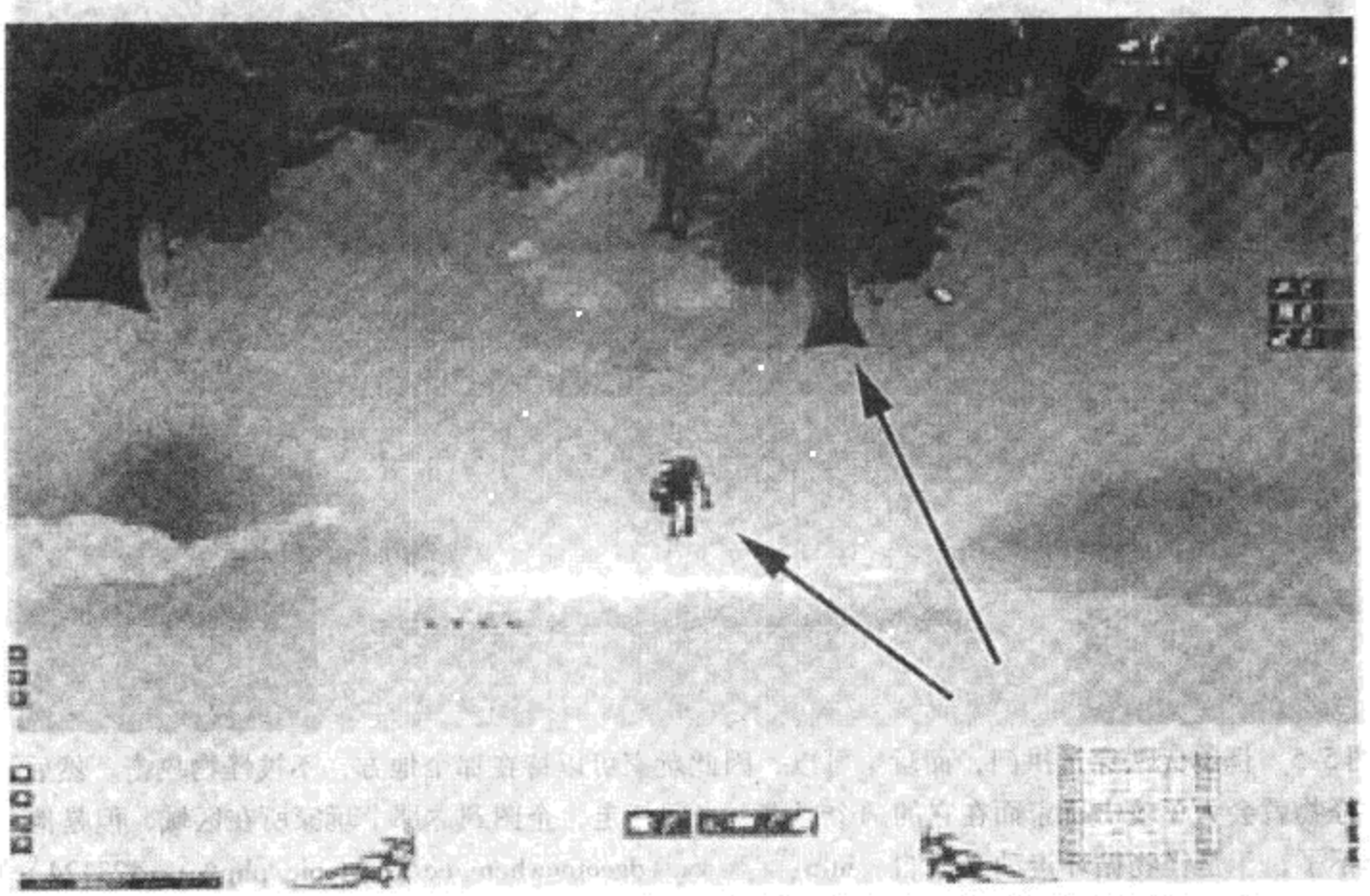


图 5-6 恐惧魔法导致了移动状态的改变，穿墙，并且进入虚拟世界地下的星空。在截图中，你可以看到角色进入了游戏世界下的开放空间，在放置有树的平面下方。要进入这里可能还需要诸如缓落术这样的魔法辅助。（出自“Booty Bay Cove Exploit”，见 <http://video.google.com/videoplay?docid=-8320552955949215666> 的视频）

通常，所有这些虚拟移动的操作理论上都是用角色 x，y，z 轴的坐标进行传送。（见第 6 章）。

5.3 改变用户界面

在大部分网络游戏中，用户界面包括菜单、颜色、列表框、枚举值、脚本、图标、图片、动画等等，所有这些都是用来处理你能够直接通过图形用户界面，或是通过脚本或特别设计过的网络信息包下达的指令。例如，聊天信息在 WoW 中是直接显示在用户界面上的。但是聊天信息也可以用网络信息包或是脚本宏来发送。

想象一下，聊天窗口中服务器信息以红色显示，玩家信息以白色显示。你作为一名普通玩家，不能改变任何聊天信息的颜色。也就是说，这些信息永远都是用白色显示。然而，如果你在

网络上放一个嗅探器，分析服务器发送的聊天信息，你可能会注意到把字体变成红色的“溢出码”。你不能用游戏内的聊天界面插入这些溢出码，但是如果你自己制作信息包，你就可以考虑自己插入溢出码了。

这种技术让你可以把聊天信息作为服务器信息发给别人。这是社会工程，不是吗？这是客户端强制约束带来的后果，可能会给一部分不知情的玩家带来错误的期待和理解。或者被用来发布非法的中奖欺骗信息和广告消息等。

举例来说，在 WoW 中，有很多不同的道具拾取规则，这些规则由队长来设置。如果你超出了拾取范围，就不能在用户界面内参与拾取，但如果你用下面这样一个脚本就可以做到了：

```
/script SetLootMethod("Master","YourCharacterNameHere","7");
```

调用这种技术产生的影响也有有趣的一面。当其他玩家想要离开团队时，用一个超出范围的数值发动用户界面事件。在 GUI 中，非法参数“7”被用于颜色数值，而颜色数值“7”超出了数值范围。这样就引发了错误，用户界面会因此拒绝玩家退出团队，可以说，这个角色被陷害了。当然，这只是用户界面的问题。只有不知道这类问题的玩家才会长时间中这种圈套。比较机灵的玩家（比如看了这本书的玩家）如果中了这种圈套用短短的一句脚本就可以脱身。

```
/script LeaveParty( );
```

这种技术以及类似这样的技术所强调的是用多种方法对网络游戏的 GUI 做同一件事。你的方法可能也不相同。

5.4 修改客户端游戏数据

游戏客户端的本地数据也具有一定的风险，可能会被修改。副本入口可能被转移，副本入口相关的数据库可能被修改。我们在接下来的章节会说明这种类型的攻击。x，y，z 轴坐标也是客户端软件控制的危险参数之一。模型控制客户端行为和游戏画面。通过干预模型，你就能改变类似进入副本的要求之类的事情。你还可以做些移开挡住你到某处的门这样的事情。

5.5 监控掉落物和重生点

WoW 中的怪物，以及其他 MMORPG 中的怪物，在被杀时经常会掉落有趣的东西，玩家可以自由拾取这些东西。这是游戏管理者产出物品给大家的方式之一。“掉落物”这个词指的本应该是怪物所携带的任务东西，但常常只被用于那些有价值的道具。怪物掉落价值低的道具的几率很高。并且，如你所想，它们掉落高价值的稀有道具的几率很低。

不同怪物的掉落几率因此而特别有趣。有几个网站致力于追踪所有知名道具和怪物的掉落几率（见 <http://www.thottbot.com> 就是一例）。图 5-7 展示的是 WoW 怪物不同掉落几率的截屏图。这几个网站还追踪了游戏中怪物出现的位置（图 5-8）。

Items dropped by Crimson Guardsman:

Name	Drops / Kills	Drop Rate
Lightforge Belt Lvl 53, 341 AC, +10 Str, +9 Sta, +15 Int, +6 Spi, +40 AP	6 / 240	2.50%
Pristine Black Diamond	1 / 240	0.42%
Six of Portals Use: Combine the Ace through Eight of Portals to complete the set.	1 / 240	0.42%
Stonegrip Gauntlets Lvl 55, 392 AC, +9 Str, +14 Sta, +10 def, Equip: Increased Defense +10.	0 / 240	0.00%
Aquamarine	1 / 240	0.42%
Arcane Gloves Lvl 52, 46 AC, +13 Sta, +12 Int	2 / 240	0.83%
Backbreaker Lvl 51, 49.4 DPS	1 / 240	0.42%
Burnside Rifle Lvl 51, 26 DPS	1 / 240	0.42%
Commander's Vambraces Lvl 54, 245 AC	1 / 240	0.42%
Councillor's Pants of the Owl Lvl 53, 66 AC, +17 Int, +17 Spi	0 / 240	0.00%
Crystal Sword of Stamina Lvl 52, 33.9 DPS	0 / 240	0.00%

图 5-7 WoW 中特定怪物的掉落几率

（出自 <http：//www. goblinworkshop. com/creatures/crimson-guardsman. html >）

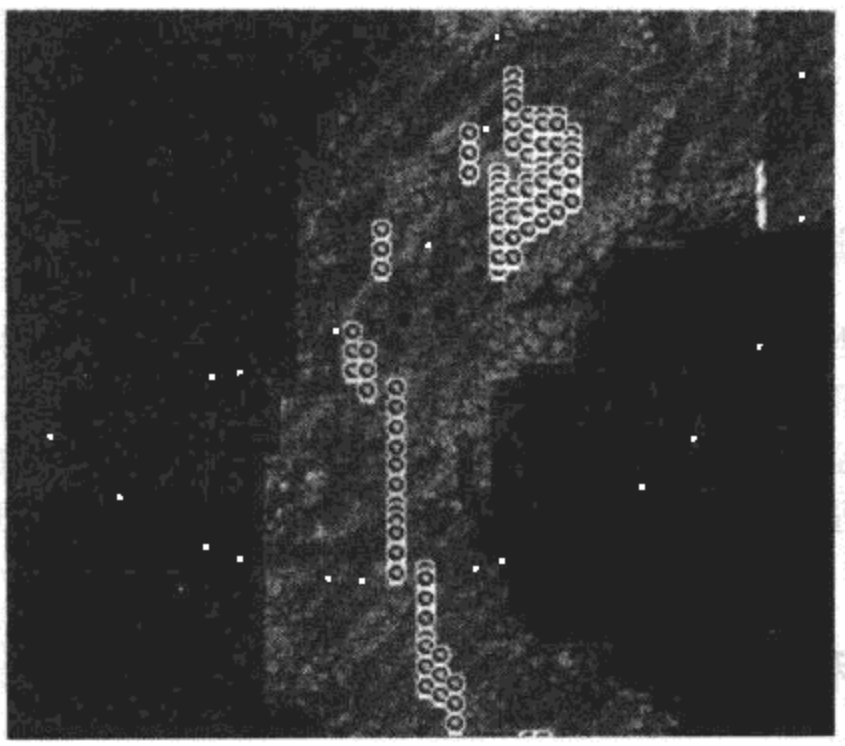


图 5-8 怪的重生点以及掉落几率（出自 <http：//www. goblinworkshop. com/creatures/arei. html >）

特殊掉落物对玩家而言尤其有趣。这些特殊的掉落物包括有优秀的武器或装备、特殊配方、或某个魔法配方的成分。配方的特定成分可能在游戏里很难找到，因此在市场上卖得很贵。大规模打落稀有道具可以让你的角色在游戏里赚到一大笔钱。

但是掉落物有个问题，就是你无法确定某个生物是否会掉某样东西。幸运的是，在一些网络游戏中，客户端程序已经知道某个怪物会掉落什么（甚至会自己进行计算，来判断掉落是否发生）。如果是这样的情况，那么作弊者在游戏中具有极大的优势，因为作弊者在事先知道了一个怪物会掉落有价值的东西时，就可以选择攻击的目标。

在 WoW 中，事情有一点不同——只有在杀死一个怪物之后，才会调用数据库来计算掉落。然而，因为怪物会掉落一个特殊道具，例如武器，游戏客户端软件可能会在怪物死之前就得到怪物携带的财产信息。在这种情况下，怪物带着武器这个信息被发送到客户端程序。有时，武器甚至会很明显地被怪物带着，提醒你它会掉落。

这种知识就是力量。

当一个资源被开采了或是怪物被杀掉时，游戏服务器通常会标记资源消失的时间。然后，它会周期性地在游戏世界中寻找一个重生点。这就类似于在服务器上执行一个任务循环，就像下面这样：

```
While(running)
{
        TASK t = GetTaskFromQueue( );
        t.Run( );
        Sleep(1);
}
```

这里的任务可以是需要服务器来执行的任何工作。这样的任务可能是扫描寻找重生点。最后，任务运行了，执行了一下像下面这样的数据库搜索：

```
"SELECT * FROM mob_instances WHERE alive=0
AND killtime > (NOW - respawn_rate)"
```

通过搜索可以得到一组数据库信息，这个任务可以一次性运行所有怪物的重生。

有时，一些怪物的重生几率非常的低。如果任务选中了这样一种怪物，重生点会因而出现很多（如果不明白可参考前面的查询代码）。高速重生点可以和快速赚经验或是打掉落道具的宏结合起来应用。

5.6 只要露个脸就可以

在 WoW 里有一种一定可以获得额外经验值的方法，就是自动加入一个搜索到的队伍里。通过这种方式，你不需要真正做什么就能获得经验值。这种行为通常只需要点点用户界面就能办到（但是现在我们知道这也可以自动进行的！）。

5.7 结论

在本章里，我们谈论了游戏在执行和设计上的弱点。这些 bug 和缺陷对于入侵网络游戏而言都非常有用。有些缺点属于大型分布式进程的副作用，有些则和用户界面的问题以及有关信任模型有关。另外还有一些则是因为做了明显不该去做的事。

接下来，我们要告诉仍然在看本书的读者，如何通过技术干扰游戏客户端、状态以及和服务器的通信来利用这些问题进行入侵。

第6章　入侵游戏客户端

本章的内容和软件破解有关，尤其是针对破解网络游戏的客户端部分。所用到的方法其实是从软件测试和软件安全领域直接借鉴而来的，包括用来试探、理解以及最终控制此类软件的各种手段。如果想了解更多关于软件破解的知识，你可以参看我们以前写的《*Exploiting Software*》一书（Addison Wesley 出版社，2004 年），还有一本非常优秀的参考书是 DavidLitchfield，Jack Koziol 等人著的《*The Shellcoder's Handbook*》。

6.1　恶意的软件扫描测试（攻击者的入口）

寻找软件的攻击入口的过程是一个恶意破坏的过程吗？并不是这样。但是最优秀的测试者能够把他们正在测试的软件置于极限情况，测试各种极端情况下软件的表现。很多时候，质量保证组里最好的测试员不是在开发者中最受欢迎的人（有时候甚至在管理层之中也是如此）。这样的测试员总是能发现可怕的 bug 而让开发者们头痛不已，并且总是指出别人犯错的地方以及通过演示证明错误的存在。他们的思维定势里认为软件必然包含错误。

有趣的事情是，最接近探究软件安全风险的边界的人，正是那些努力破坏软件运行的测试员们。这类用软件崩溃风险来驱动的测试的方法在 McGraw 所著的《*Software Security*》一书（Addison Wesley 出版社，2006 年）的第7章有描述，这种思想的出发点就是使用经典的测试技术来暴露尽可能多的安全问题。

虽然许多软件开发者并不会把软件质量保证的过程看作是恶意测试行为，但是事实上可以这样去看待。一方面来说，功能性测试是被设计来确保软件具有可信赖的稳定性以及良好的用户体验。但安全测试并不只是测试安全方面的功能，比如确认加密功能是否正常工作。安全测试必须用像扮演黑客这样的坏人角色的心态来窥探软件的运行过程。基于风险的安全测试，具有结构性风险分析以及攻击模式组合，本质上就是恶意测试。事实表明，许多软件的 bug 同时带来了大量的安全风险，从这方面来讲，软件测试确实就是恶意破坏的过程。

唯一的区别是公司的测试人员会把可怕的漏洞及时告诉开发者，而外部的破译者对他发现的漏洞总是尽量的保密。

更坏的情况是，如果坏人发现了这个 bug，他们不会去告诉好人。这意味着测试人员可能会变成坏人，从寻找 bug，转变为利用 bug。

“质量保证”人员使用的工具和技术

软件安全测试员通常拥有令人印象深刻的工具程序库，库里面工具程序可能经过数年的时间来开发完善，它们能够帮助测试员了解软件以及纠正错误。如果列个清单的话，这里的工具程序包括调试

器，代码覆盖程序，错误注入引擎，虚拟机模拟器，反编译器，反汇编程序等一大堆。即使是新手破坏分子也无需四处寻找有效的黑客工具，游戏黑客能够使用现成的同类软件测试工具达到目的。这些工具程序（有开源的，也有商业化的）每天都被全世界的质量保证部门广泛使用来寻找 bug。

为了发现并利用游戏里的 bug，黑客们会全力使用目前的这些的测试工具。这里，我们只介绍一些基本的工具程序，更多的请看我们之前出版的书《*Exploiting Software*》。

反编译器

反编译器的原理和编译器一样，只是功能相反。编译器是将比如 C++语言写的源代码编译为比如 Win32 平台上的可执行代码。所以，反编译器是将一些平台上的二值代码反编译为可读的源代码。

反编译器的执行过程比较容易理解，不过做起来没那么容易，因为从二值代码返回源代码会稍微困难一些。这也是国为编译器工作的内部原理较为复杂。所以，反编译器输出的结果，往往是不那么精准的源代码，它经常输出很多杂乱的注释，不一致的变量名，甚至有点混乱的控制流结构。因此，反编译器最后的结果对于真正的源代码来说只能达到一个近似。

但反编译器是十分有用的，因为阅读和理解源代码，远比阅读和理解二值代码来的容易。虽然有些人可以直接阅读二值代码，但是他们只是小众。如果要试着去阅读一段代码是如何运作的，那阅读源代码是比较容易的方式。

游戏黑客们用反编译器来帮助他们理解一个游戏的客户端里的二值代码是如何运行的。图 6-1 是一个叫做 Boomerang 的开源反编译器 <http://sourceforge.net/projects/boomerang/>。

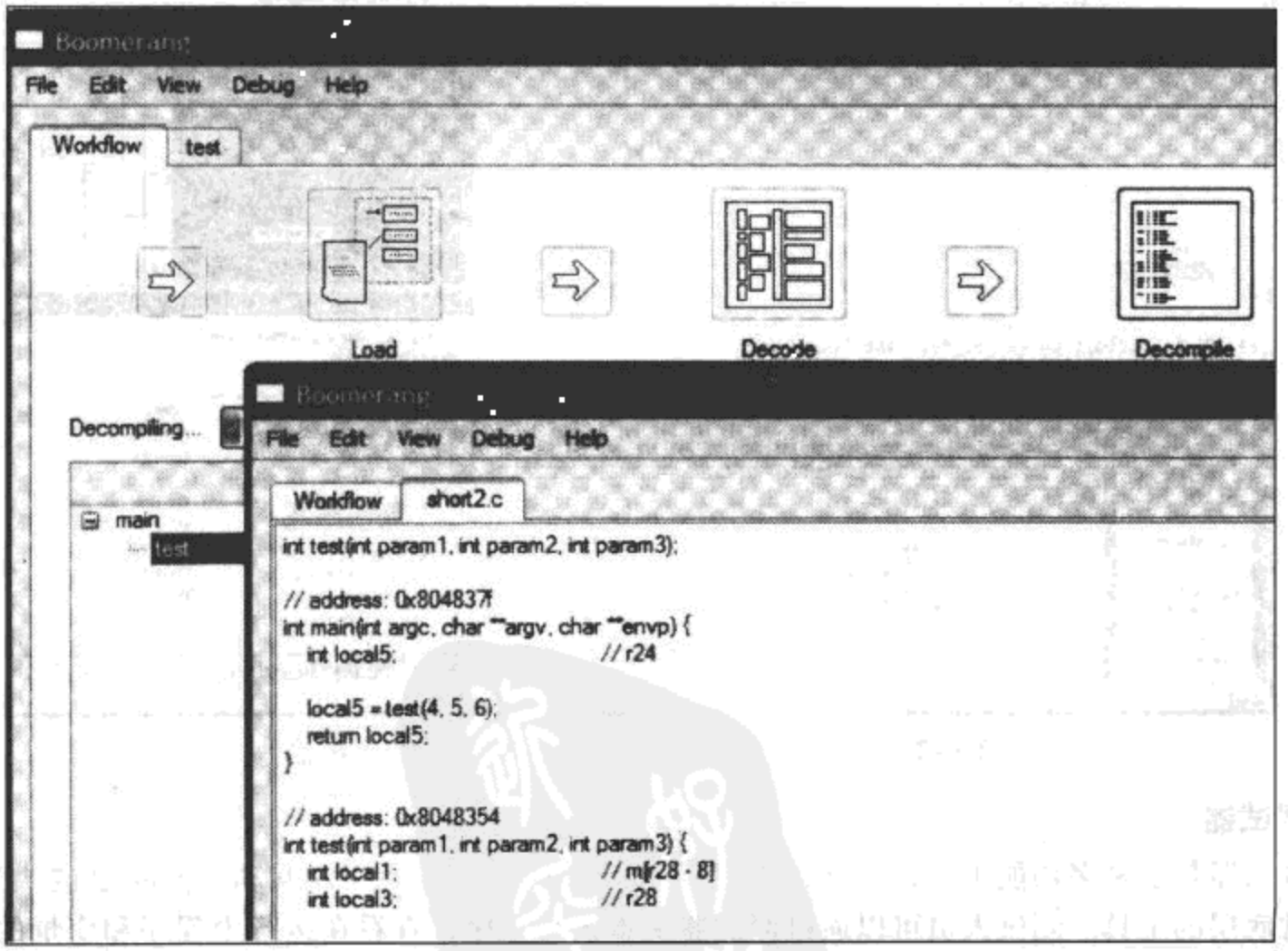


图 6-1 Boomerang 反编译器

反汇编器

“反汇编”器和“反编译”器非常相似。在很多方面都是一致的，如将二进制数据转换成“汇编语言”比起转换成源代码（一种非常底层的代码语言，但不像0和1那样底层）简单很多。很明显这里的反汇编器所要处理的就是区分二进制数据中指令和数据。记住，我们现在所讨论的是二进制数据。要在这一堆堆的二进制数据中区分出哪些是数据，哪些是指令在有些时候的确也是一种挑战。

现代反汇编器在转换过程方面所做的工作已经很不错了。一般在做指令分析工作开始时，倾向于用反编译器来解释一段可执行的二进制数都做了些什么工作，当反编译器在工作中不能很好翻译的话，反汇编器的威力就开始显现了。

图6-2展示的是一个标准商业化的反汇编器，而它最典型的应用领域是在软件逆向工程中。这款程序名叫Inspector，是HBGary公司的产品<http://www.hbgray.com>。虽然这套程序有很多先进的特性，不过对于游戏黑客而言，价格贵了点。免费的反汇编器在很多平台上虽然没有太多先进的功能，不过却可以在网络上很容易找到。

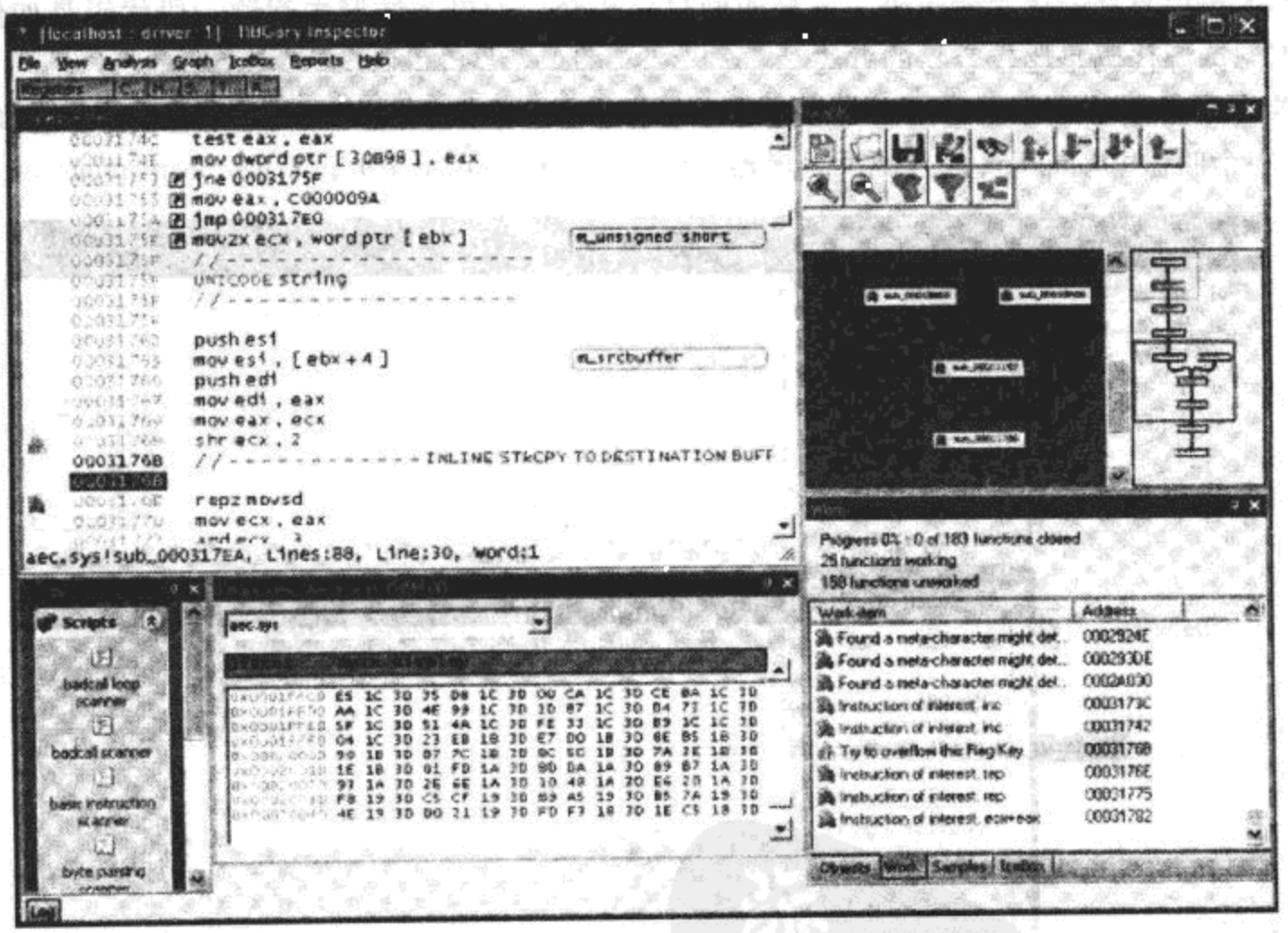


图6-2 一个用于软件逆向工程中的典型的反汇编器

调试器

调试器是一种多功能工具，它允许在程序运行过程中直接同用户互动。同时也被认为是一种非常底层的工具。测试人员可以通过调试器单步运行程序，查看在运行中程序和变量的状态，也可以在程序中插入断点——这些工作大体上可以非常全面的看到程序在运行中都具体做了些什么。

在软件开发的过程中，调试器是一个必不可少的组成部分。很多调试器非常的先进，其工作包括：保存程序状态，显示堆栈图，展示各种不同变量之间的捆绑关系，而其中最重要的部分是在分析代码中展现其强大的洞察力。所以，一个好的调试器在分析和查找 bug 中是必不可少的。

进入程序底层，你会看到一款软件往往是有着繁杂的设计体系：线程间的关系，堆栈，寄存器和内存区域等。而这些复杂关系归根到底都源自于两件事——程序的指令和所运行的机器硬件。计算机本身是由硬件工程师设计和制造的（比如 Intel 公司提供的 CPU），而通过编译器（比如 gcc）翻译并运行在机器上的软件是由软件工程师创作的。以此，很多规则的制定是由处于中间层次的操作系统（比如 Microsoft Windows）来提出的标准，由上可知，软件归根结底就是由每台机器各自带有的特殊指令和机器内存中存储的数据相互作用而组成的。

一个功能强大的调试器，允许用户与那些复杂的数据相互作用。在合理的操作下，调试器可以迫使程序去做任何有能力执行的操作，甚至有些实际上并不是出自于设计本意。这就意味着虽然调试器最初是作为寻找 bug 而被创建的，不过对于那些喜欢寻找游戏软件中漏洞的黑客来说，他们已经把调试器作为远程机器人组中的核心组成部分。事实上，我们稍后就将用到那些原理并告诉你怎样编写一个简单的调试器。

图 6-3 是现在非常流行的（并且是免费的）调试器，叫做 OllyDbg < http: //www. ollydbg. de/ >。这个工具通常用于微软 Windows 平台上，作用是反向工程和破解游戏软件。

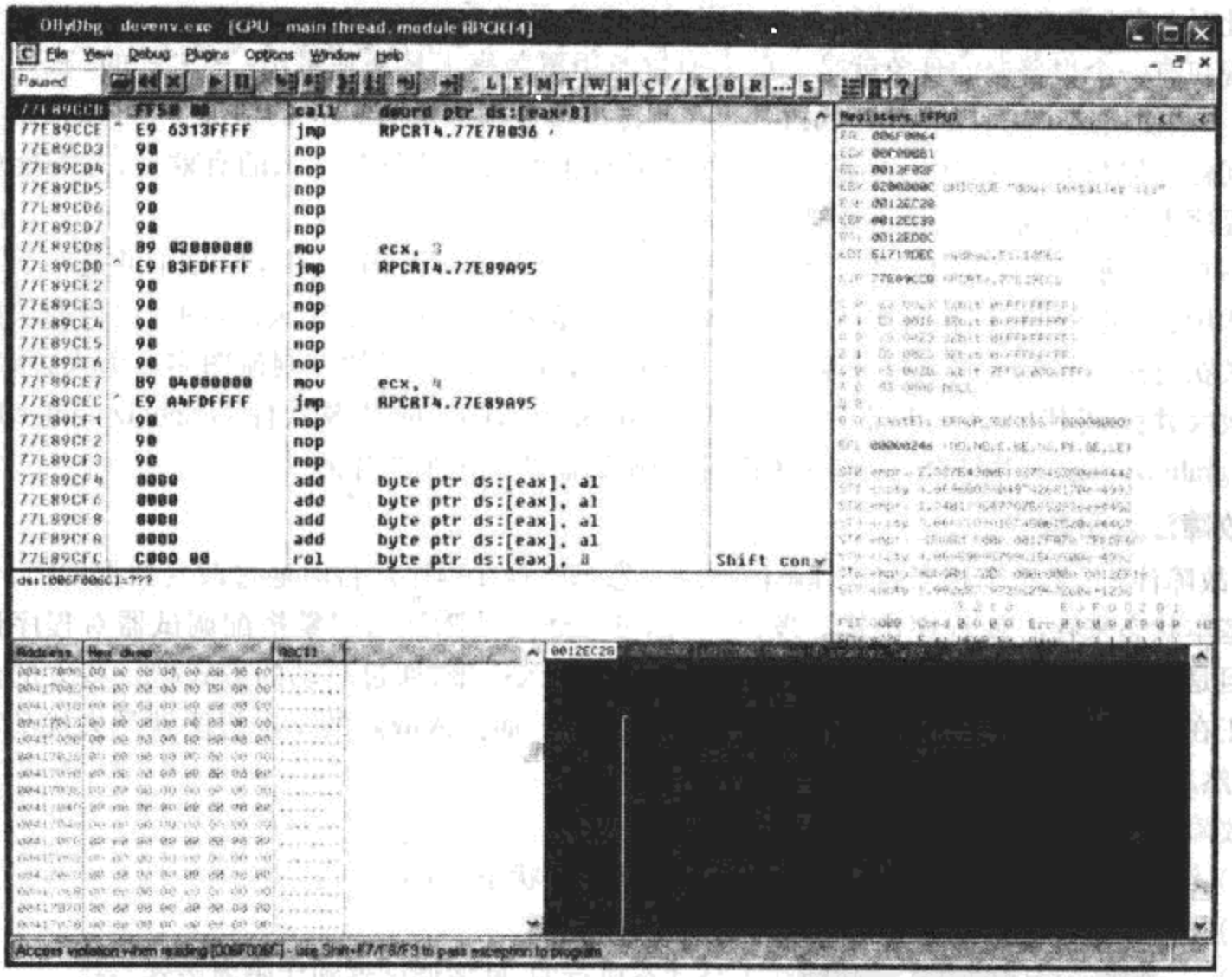


图 6-3　OllyDbg，一个流行且免费的在微软 Windows 平台上的调试器

覆盖率测试工具

覆盖率工具的出现是为了帮助测试人员测出程序在运行中被覆盖到的代码的一个粗略的量。它所做的就是在程序执行过程中跟踪并保存所经过的路径。这就使得测试人员能有效地了解一个被测试的程序在运行过程中，哪些部分的代码执行过，而哪些部分没有被执行。测试包运行后可能得出一个程序的执行覆盖了代码的90%，然而在比较低效的情况下，测试包可能得出只执行了代码的40%。覆盖率分析也分成多种，从功能的覆盖率情况到各种条件判断的覆盖率。

覆盖率工具是通过在测试状态下追踪程序的运行来进行工作的，目的是对部分程序在执行（或运行）时查看所经过的路径。（注意，覆盖率工具有两种，一种是测试源代码，还有一种是测试二进制指令，这两种类型的工具应用都很普遍。）当一组测试在动态执行的时候，当然同时程序也在运行中，被执行过的数据和指令就会被跟踪到。比如，一个覆盖率工具可能测试一条 `if-then-else` 表达式，在这种情况下测试人员可以通过覆盖率工具的测试包了解 `then` 条件和 `else` 条件是否都被执行过。

显然，对于那些有兴趣寻找软件漏洞的人来说，覆盖率工具是相当有用的，因为在寻找软件漏洞过程中，找到问题就等于你已经完成了一半的工作。比如一个攻击者可能发现一个可以攻击的漏洞，而这个漏洞产生在程序中很深的部位（也就是说，是用代码扫描工具发现的），此时攻击者就需要用某种方法使程序执行到这个易受攻击的位置，并且程序停在这个位置时处于可以被攻击状态。此时此刻，覆盖率工具就能很有效的完成这样的工作了。

现在有一个很普遍的现象就是一些人通过滥用覆盖率工具来估量出一个“好的”程序。也就是说，一些使用这个工具的测试人员相信，在测试中覆盖率结果为90%的情况下，可以找出其中的一些问题。但我们认为，代码覆盖率所做的结果真正所表达的只是测试的有效性，并不能就结果说明程序本身的问题，所以请不要被这个结果误导。

图6-4显示了一个例子，是一个很小的程序表示的覆盖率图片。图片所显示的是一个在控制流结构中，一组代码块的执行图片，其中涂成阴影的表示在测试中已经执行过的，白色的块则表示没有被执行过。当然这些只是控制流结构中的一部分，因为实际的控制流图实在太大，所以我们只能放大并强调其中的一小部分。我们是使用来自 AT&T 的开源软件 Graphviz tool < http://www. grahviz. org/ > 并结合自己编写的特制的调试器产生这张图片的。

故障注入引擎

“故障注入”是一个相当容易理解的概念。意思就是在程序运行时通过改变数据的值，消息和其他基本状态来存心使程序出错。理论上，你可以通过故障注入引擎搭配调试器对程序做任何事。但是在通常情况，用一个带有特殊目的的故障注入引擎，通过自动的方式把一些大的数字注入到正在运行的程序中去，就可以产生很好的效果了。而注入故障的基本想法就是去扰乱程序和数据，然后密切地关注什么将会发生。

故障注入引擎是由三个部分组成。

1）注入引擎（这部分是专门用来扰乱程序和数据状态的）。

2）监视系统（这部分主要是警惕将会发生什么）。

3）一个执行程序的方法。故障注入是动态执行的，就像调试器和代码覆盖器一样。

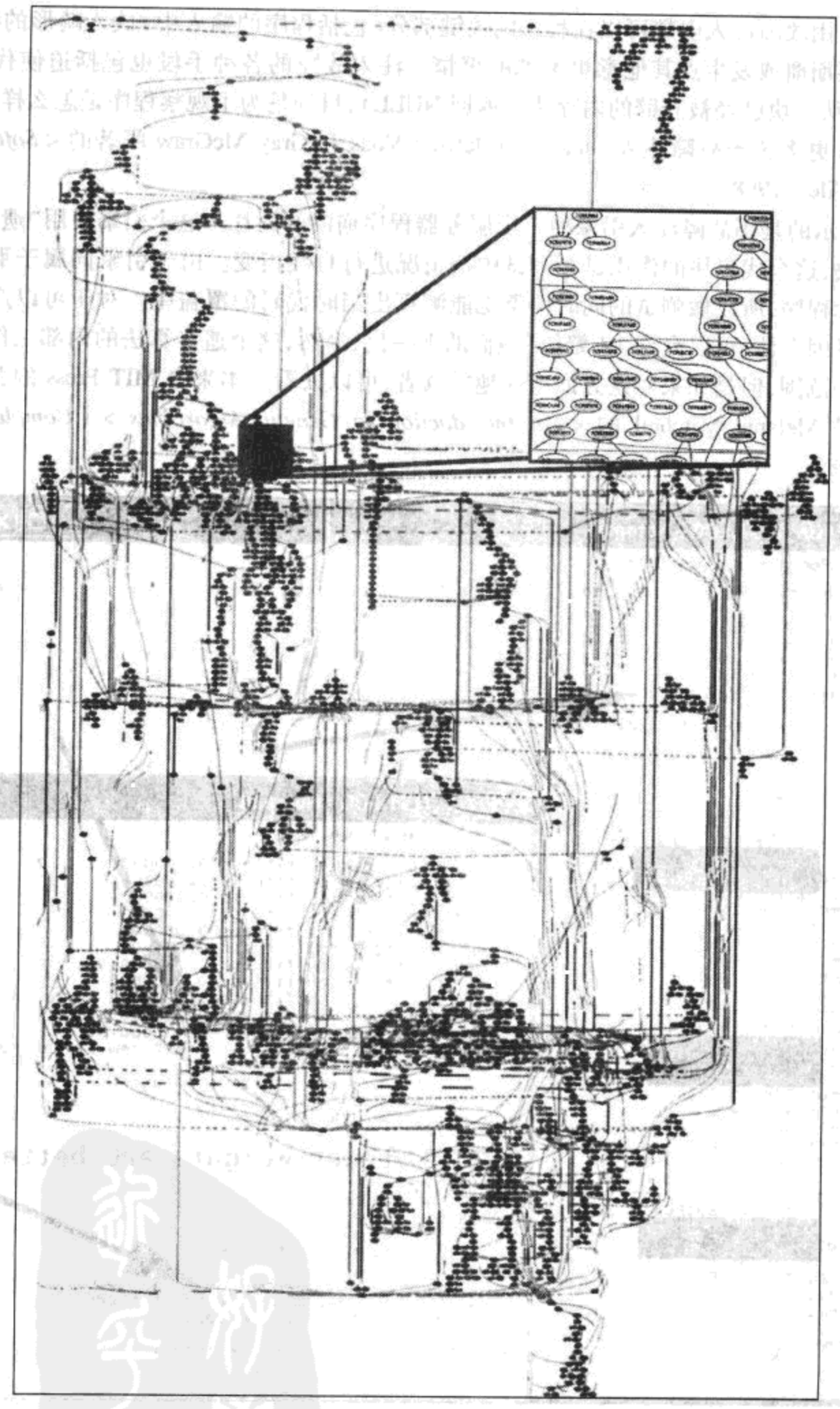


图 6-4　这是一个软件中的一个大型控制流结构图。阴影表示已经执行了动态测试，白色的表示没有执行过。这种分析方法对检查测试结果很有效

测试人员用故障注入引擎可以在程序的关键部分(包括程序的输入中)放入畸形的数据,并试图使程序直接崩溃或发生点其他意想不到的事情。注入故障的各种手段也包括迫使代码进入错误状态(比如从一块已经被开辟的内存块中返回 NULL),目的是为了观察程序是怎么样处理错误。如果想要了解更多关于故障注入,可以参阅 Jeffrey Voas 和 Gray McGraw 所著的 < *Software Fault Injection* > (Wiley,1998)。

图 6-5 显示的是用故障注入引擎使一个服务器程序崩溃的图片。这个引擎使用"遗传算法"产生一个代码块,这个代码块的作用是在测试中按情况进行自我衍变。由于引擎同属于那些像调试器之类的目标程序,所以做测试的同时引擎也能测算出当时代码的覆盖率。对于可以产生更好代码覆盖率的基因个体,可以允许去"繁衍"他们的下一代,当然,这个遗传算法的内部工作原理已经超过了本书的范围,但是如果对这方面有兴趣的读者,可以去看一本来自 MIT Press 的关于这方面初级的书,是 Melanie Mitchell 的 < *An Introduction to Genetic Algoritbms* > (*Complex Adaptive Systems*) (1998)。

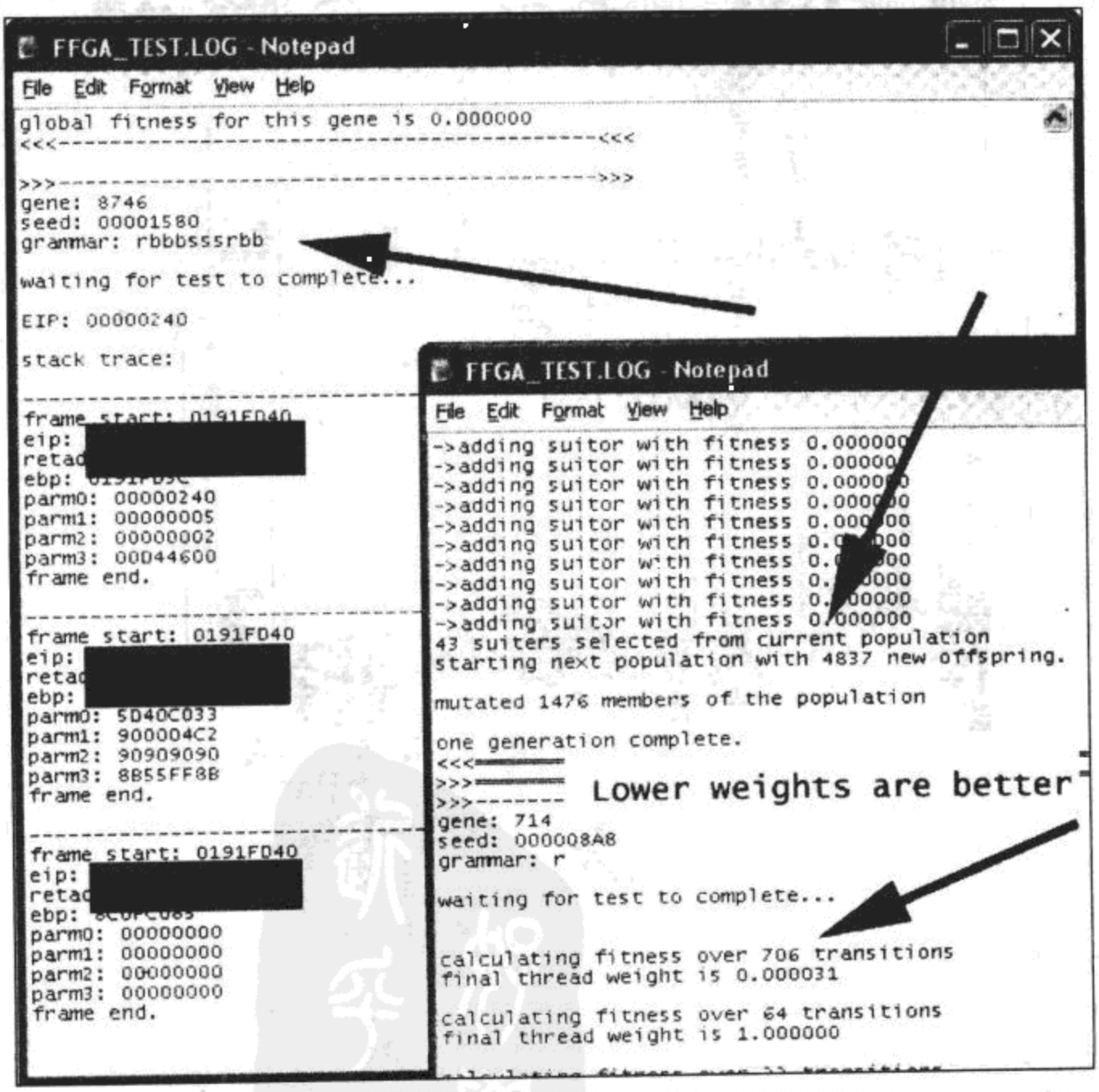


图 6-5 一个使服务器崩溃的基于遗传算法引擎的工作图。根据遗传算法的原理,这个引擎包含了更有效的测试过程

虚拟机模拟器

虚拟机模拟器(VM)实际上是一个来自于计算机科学的,很早之前就形成的概念,具有一段悠久的历史。其实际的想法就是:你用一台机器,不管这台机器是什么型号或什么配置,都可以用来模拟其他的机器的运行环境(或是服务器)。一个现代一点的例子,比如程序 Parallels 可以允许一个苹果机的用户在自己的机器上安装一个 XP 的虚拟机,然后就可在这个专为 Mac 操作系统设计的苹果机上同时运行一个 Windows XP 操作系统 <http://www.parallels.com/>。相当有趣的是,你甚至可以在 XP 操作系统上运行一个 XP 的虚拟机。

实现完美模拟的功劳全在于虚拟机软件。比如在 Mac 机器上运行 Windows 虚拟机的例子中,主要是有一个软件层工作在 XP 操作系统和苹果机的硬件之间,这个层起到了向上和向下解释的作用。对 XP 操作系统而言,它会感觉自己正运行在自己的硬件上……究竟是什么使得软件相信了这一切只有“VMs”明白。

如果你使用虚拟机,就相当于在被监视的状态下执行程序。你可以做一些正常情况下很难用硬件去实现的事,比如状态回转,程序在极限状态下运行或者在崩溃发生后很快重置。虚拟机可以完全控制它内部运行的软件。这就非常像勒奈·笛卡尔的著名思想实验“桶中之脑”中的“恶毒魔鬼”一样。在 17 世纪,笛卡尔做了一个“桶中之脑”著名思想实验,这个实验的主要疑惑在于他怀疑我们每个人都只是一个放在桶中的大脑,(类似黑客帝国描述的思维世界的故事)而人们的大脑在桶中以某种方式接受恶魔给我们的各种感受形成了我们的世界,这个实验主要目的是用于对照生物的自由意志。通过这个实验他也得出了他的那句名言“我思故我在!”。

现在回到软件上来,实验者把软件运行在虚拟机上就像控制一个桶中大脑。这使虚拟机可以完全控制这款软件。同时也表现出这是一个可以用来对付那些调试小把戏的很好的方法。

6.2 对逆向工程的防范对策

游戏开发人员知道黑客是用反汇编器和调试器用逆向工程的方法破解游戏,所以他们也寻找了很多对策对付那些黑客们。这些技术包括:代码混淆,或者在调试器监视的时候错乱的显示运行时检查。相反黑客也能很好的攻破这些小把戏,但首先他们必须要知道开发者要的是什么把戏。这两个对立面就在进行计算机安全中形成所谓的军备竞赛。

6.2.1 代码打包

打包是一种非常简单的防范对策,也是一个最流行的保护罩。打包过的文件可以改变被编译后的二进制文件在磁盘中看上去的样子。如果你试图在反汇编器中导入一个被打包过的二进制文件,它好像并不能为你读出文件。就算换成导入一个打包过的程序代码,也就只能得到一个错误。也许在某些情况下,可以反汇编出一些代码,但大多数只是以原有的形式被保留并显示出来。

可以通过反打包程序获得打包前的代码。有几个常用的反打包工具,可以理解并破解大多数流行的打包格式。然而,这些工具还不能对付那些自定义的打包程序。对于设置成自定义打包的程序,需要直接从内存中读取。这个技术需要在程序执行中,将内存中把执行镜像读取出来。因为大多数情况下,内存拖网技术的工作是使打包程序在运行前反打包。这个反打包过程的描述在软件执行期间一直会留在内存中,所以提取出内存中的内容就可以对其进行分析。

另一种破解打包程序的方法是在打包程序运行时进行单步跟踪，并获取运行时的所有指令。假设软件并不知道它被跟踪的话，这个破解方法就可以让你得到很多程序指令并进行解包。

6.2.2 反调试

程序也可以通过“反调试”的方式来对付那些逆向破解。这个技术是扫描系统并查找系统内是否有调试器进程，如果找到的话，程序会以各种离奇的方式出错并退出，这样调试器只能接受到一个异常，而失去了跟踪程序的机会。对付侦测调试器是否存在这样的防范技术非常简单，只需要在进程环境块(PEB)去除调试器进程存在标志即可以。

在进程环境块(PEB)中改变数据

以下这个代码小片段可以用来为一个正在被调试的程序获得进程环境块的基地址。

```
DWORD GetPEBBase(HANDLE theProcessHandle)
{
        PROCESS_BASIC_INFORMATION pbi;

        DWORD len = sizeof(pbi);
```

使用一个叫做 NtQueryInformationProcess 的本地 API 函数：

```
NtQueryInformationProcess(
                theProcessHandle,
                ProcessBasicInformation,
                &pbi,
                len,
                NULL);

        return (DWORD)(pbi.PebBaseAddress);
}
```

一旦获得这个进程控制块(PEB)，就可以很简单的清除调试器存在标志。记住，如果你开着调试器的话，你需要通过远程处理而不是直接在本地来使内存完成这样的改变。为了做到这一点，可以使用 ReadProcessMemory 和 WriteProcessMemory 函数：

```
void ClearDebuggerPresentFlag(HANDLE theProcessHandle)
{
        DWORD pebBase = GetPEBBase(theProcessHandle);
        PEB aPeb;
        DWORD lpRead;
```

通过远程读取 PEB：

```
ReadProcessMemory(
        theProcessHandle,
        (LPCVOID)pebBase,
        &aPeb,
        sizeof(PEB),
        &lpRead);
```

如果读取到，则对所获得的内容进行修改：

```
aPeb.BeingDebugged = FALSE;
```

然后把修改好的内容通过函数远程读回：

```
        WriteProcessMemory(
                theProcessHandle,
                (LPVOID)pebBase,
                &aPeb,
                sizeof(PEB),
                &lpRead)

}
```

这个例子说明了怎样远程操作内存，我们将在接下来详细讲解更多细节。

发出异常

许多反调试的陷阱都是依靠这样一个方法，就是阻止使用调试器或者在使用调试器时发生一个异常。例如，一个游戏程序可能通过抛出一个“被零除”的异常来迷惑调试器。这个异常本身在程序中是的一个很普通的异常，但在这里它却故意混淆真相。

游戏程序会特别建立了一个异常处理的方法，在一些特别的异常发生时就会去调用这个方法。比如当异常在游戏运行中抛出时，按设计的思路肯定是会去调用异常处理方法。可这时你开着调试器，并先捕捉到这些特意安排的异常，这时你要是把这些当成真的异常去处理，也就是说让调试器去截获处理了这个本该由游戏内部的异常处理方法处理的异常。结果，会导致游戏中的异常处理方法没有得到异常，并且因为调试器提供这样的一个优势功能也使得把程序本身的潜在异常给暴露出来了。

像通过发出异常这样的小陷阱实际上很好对付。你所要记住的就是把全部异常都往目标程序发送。调试器事件发送给程序的代码如下：

```
ContinueDebugEvent(
        thePid,
        dbg_evt.dwThreadId,
        DBG_EXCEPTION_NOT_HANDLED);
```

这个关键词就是 DBG_EXCEPTION_NOT_HANDLED。这个参数告诉调试器可以把异常都发给目标程序。这样，游戏软件就能得到并处理这些自身发出异常，显然这也是异常处理方法所应该做的。我们将在第 7 章更详细地介绍这个 ContinueDebugEvent 方法调用。

计算“单步”（Step）调试的时间

另一种反探测的方式就是用时间差计算。通常软件的探测方式是让游戏程序每隔一段间隙读取系统时间，如果这个时间间隔超过了程序员预先设定的值，系统就会默认自己正在被调试。这个方法的原理是调试器会使程序的运行减慢。要成功地规避这种检查，可以暂停运行中的目标程序，或者避免在调试时严重影响程序运行速度。通常这个间隔时间不会很精确，所以这样的防范策略是很容易攻破的。

比起现在所描述的，实际情况还有许多更麻烦的调试器陷阱，对于刚开始学习的游戏分析人员来说，偶尔遇到也不失为一种挑战。以上现象说明，一些反调试陷阱可以做的相当复杂，接着可以攻破防线的方法也变的越来越复杂。感觉就像一场持久的猫鼠大战。

到此为止，最后一种技术就是使用虚拟机去执行调试器，这要求游戏即可以在虚拟机上运行，同时游戏自身也不会检查它是否同时运行在虚拟机上。最终的技术就是让虚拟的计算机自

己执行除错功能。要做到这一点，首先要有一台能让游戏顺利运行的计算机，其次是游戏不会自己去检测它正在虚拟机上运行。

6.3 数据，数据，无处不在

现在我们能让工具和软件绑定在一起运行了，那我们就转向游戏吧。在游戏中，有两样东西是可以被改变的——代码和那些与代码关联的数值变量（在更复杂的解释下，代码本身也是一种特殊存储形式的数据）。最终，你会发现任何东西在内存中都仅仅只是数据——像一片精心设计的0与1的海洋。对于那些攻击在线游戏的黑客们，能有这样的概念意识，本身在思想方面就等于划开了一个巨大的分水岭。

任何从服务器发送给网络游戏客户端的数据包都可以被截取并修改，一旦数据在客户端中存在，你就可以截取和操作它们——当然客户端也可以“假装”没有收到这些数据。比方说，如果游戏客户端知道某种秘密隐藏的药水所在的位置，那它即使没有在用户界面和场景地图上显示这种药水，但数据肯定已经存放在客户端中了。更有趣的事，很多时候这些数据还可以被得到和修改，举个例子来说，如果这个隐藏的神奇药水离游戏角色太远，没关系，你可以修改这个药水在游戏中所处位置的数据，这样修改后，药水就能轻松而容易的出现你的面前，让你得到。

游戏客户端通过游戏界面（GUI）给玩家提供信息反馈。然而，这个界面只显示了客户端处理的部分信息——这部分是允许玩家看到的内容。显然还有更多的信息被遮掩起来了。图6-6生动地用图片显示了这种关系，假设有这样一种药水，包含的特性有：强度、力量和持续时间。这些数据在私底下被游戏管理着。在例子中有这样的一种情况，就是如果玩家在游戏中所喝药水的强度大于100，那游戏中的角色就可以再得到一些额外的奖励。参考下面的图可以看到一个关于条件判断的语句。当你按下这个 *Drink*（喝下药水）按钮，软件就会去执行这个药水的强度判断，并且给你奖励（当然仅在条件满足的状态下）。

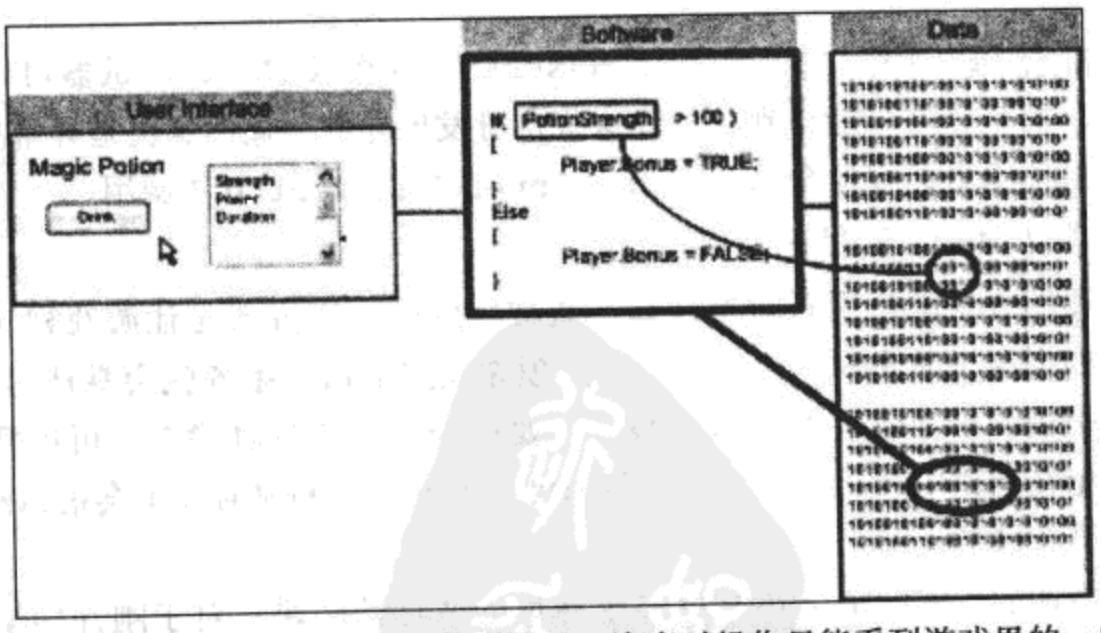


图6-6 这张界面只显示了客户端部分的信息。许多时候你只能看到游戏里的一些信息，这个魔法药水则包含更多，我们可以查看这些数据，了解到更多信息

回到我们刚才所说的，实际上在游戏客户端最底层的任何东西都只是数据。甚至软件自己也是作为数据被存储的。在图6-6中，喝药水这个判断语句作为一种数据存储在内存

某个地址中，药水可以增加多少生命强度的数据存放在另一个地址中。这是非常典型的——软件中的执行代码和它要用到的数据远离的案例。为了从例子中弄明白我们所要说的问题，我们可以做个试验，比如你可以“欺骗”一下程序，使它给你一个奖励，方法就是通过在内存中修改药水强度值的数据，这样就能达到这个目的。如果你在内存中把药水强度值改成大于100，然后当软件执行的时候（在你点击这个 *Drink* 键），显然，你的游戏角色就能得到了一个奖励了。实现这种入侵可以用错误注入引擎或者调试器完成，或者仅仅通过手动直接把修改过的值放入内存中。

这个例子避开了实际操作中的困难，直接找到了药水强度值在内存中存放的地址。这个寻址也许感觉上就像大海捞针一样，因为在一个运行的软件中，存在有上千万个数据。在没有向导帮助的情况下，仅仅通过简单的在数据中徘徊来寻找真的就是大海捞针。所以，你需要通过“向导”来找到你所要类型的数据，说的更具体一点就是，怎样去得到所要的数据。幸运的是，一些工具和技术可以帮你解决这个问题。

首先，也是最重要的，就是先得到已经使用过的数据。这些数据如果不是指令，就是被指令操作的数值（更复杂的情况下两者皆是）。如果这个字节是指令，那么在某个时间点上，它会存进 CPU 中并被执行。因为这是指令所拥有的特性，大多数反汇编器可以在数据中找到指令，接着按功能翻译成汇编语言。图 6-7 显示了反汇编器把二进制字节转换成汇编语言。

```
00010408 mov edi , edi
0001040A push ebp
0001040B mov ebp , esp
0001040D push ecx
0001040E mov [ ebp - 4 ] , ecx
000104E1 push 0
000104E3 push dword ptr [ 0001F574 ]
000104E9 push dword ptr [ 0001F578 ]
000104EF push dword ptr [ ebp - 4 ]
000104F2 push F7
000104F7 call dword ptr [ __imp_ntoskrnl . exe ! KeBugCheckEx ]
000104FD int 3
000104FE int 3
000104FF int 3
00010500 int 3
00010501 int 3
00010502 int 3
00010503 cmp ecx , [ 0001F578 ]
00010509 jne 00010514
0001050B test ecx , FFFF0000
```

图 6-7　这个反汇编器将二进制字节转换成汇编语言。
图中左侧列的数字是内存地址，右侧是对应的汇编语言

但对于分辨普通的数值我们该如何做呢？如果你已经反汇编了使用过的代码，那么它们就会提供一些数值特征的线索。图 6-8 显示了一个指令把一个位于 0001F574 位置的数据压栈。通过在反汇编下代码构建出的样子，我们可以知道在内存地址 0001F574 中存放的肯定是某个数值。这就暗示，通过大多数包含以上这种信息的指令可以帮助我们找到感兴趣的数值。

```
000104D8 mov edi , edi
000104DA   push ebp
000104DB   mov ebp , esp
000104DD   push ecx
000104DE   mov [ ebp - 4 ] , ecx
000104E1   push 0
000104E3   push dword ptr [ 0001F574 ]
000104E9   push dword ptr [ 0001F578 ]
000104EF   push dword ptr [ ebp - 4 ]
000104F2   push F7
000104F7   call dword ptr [ __imp_ntoskrnl . exe ! KeBugCheckEx ]
000104FD   int 3
000104FE   int 3
000104FF   int 3
00010500   int 3
00010501   int 3
00010502   int 3
00010503 cmp ecx , [ 0001F578 ]
00010509 jne 00010514
0001050B test ecx , FFFF0000
```

图 6-8 数据通过汇编语言进行压栈

6.3.1 数据曝光的对策

显然你可以任意地在游戏程序中寻找和修改数据。但是你要记住，游戏程序也会搜索内存。他们在防止攻击上的方法包括：扫描游戏代码和数据并找出是否有数据被修改过（例如，通过代码完整检测）和扫描内存中游戏代码和数据中是否被某个第三方软件注入过数据（可以用恶意软件扫描出）。现在我们所拥有的攻击技术包括：修改代码，改变数据和在游戏执行中注入线程或 DLLs（动态连接库），不过所有这些实时攻击反过来也都会被游戏软件查出。明显的例子就是 Blizzard 的 Warden，这个软件就是专门用以上对策来保护游戏 WoW（见第 2 章）的。当然，黑客们也有方法隐藏自己的修改或应对很多游戏在内存中的自我扫描，可就算应付了其中的一部分，所用到的方式也相当复杂。

6.3.2 动态数据和静态数据

有时数据是静态的，就是说它就存在于你电脑的内存中（在客户端启动时候注入内存的不会变化的信息）。而有的数据是动态的，就是说通过网络连接收到服务器的指令不断变化的数据，它们的数据交互是靠网络在程序间相互通信来完成的。如果你只关注那些存在于内存中的静态数据，很显然你将损失攻击网络游戏中一半的乐趣。对于动态数据，你可以修改游戏中服务器和客户端程序之间相互通信的数据包内容，得到的结果和在本地内存中修改数据一样，对游戏的运行产生影响。事实上，如果你非常了解网络通信协议的工作原理，你甚至可以用把自己编写的程序加入到原来游戏的客户端中来达到你所预期的目的。而这样的程序就需要带有特殊的输入接口和产生特殊输出的接口。

让我们描述一个实际情况：比如有时，只要能在网络中嗅探出正确的数据包，甚至可以不通过调试器就能找到游戏中某种秘密药水所在的位置。图 6-9 就是一个嗅探器的例子，这个叫 WoWSniffer 的嗅探器清楚地展示了它可以在 WoW 游戏中捕获玩家的聊天信息。值得一提的是，

网络中的聊天信息在传输时都被加密过了，所以很明显，这个 WoWSniffer 的作者在截获游戏中聊天信息的数据包时，用自己做的程序把加密功能破解了。

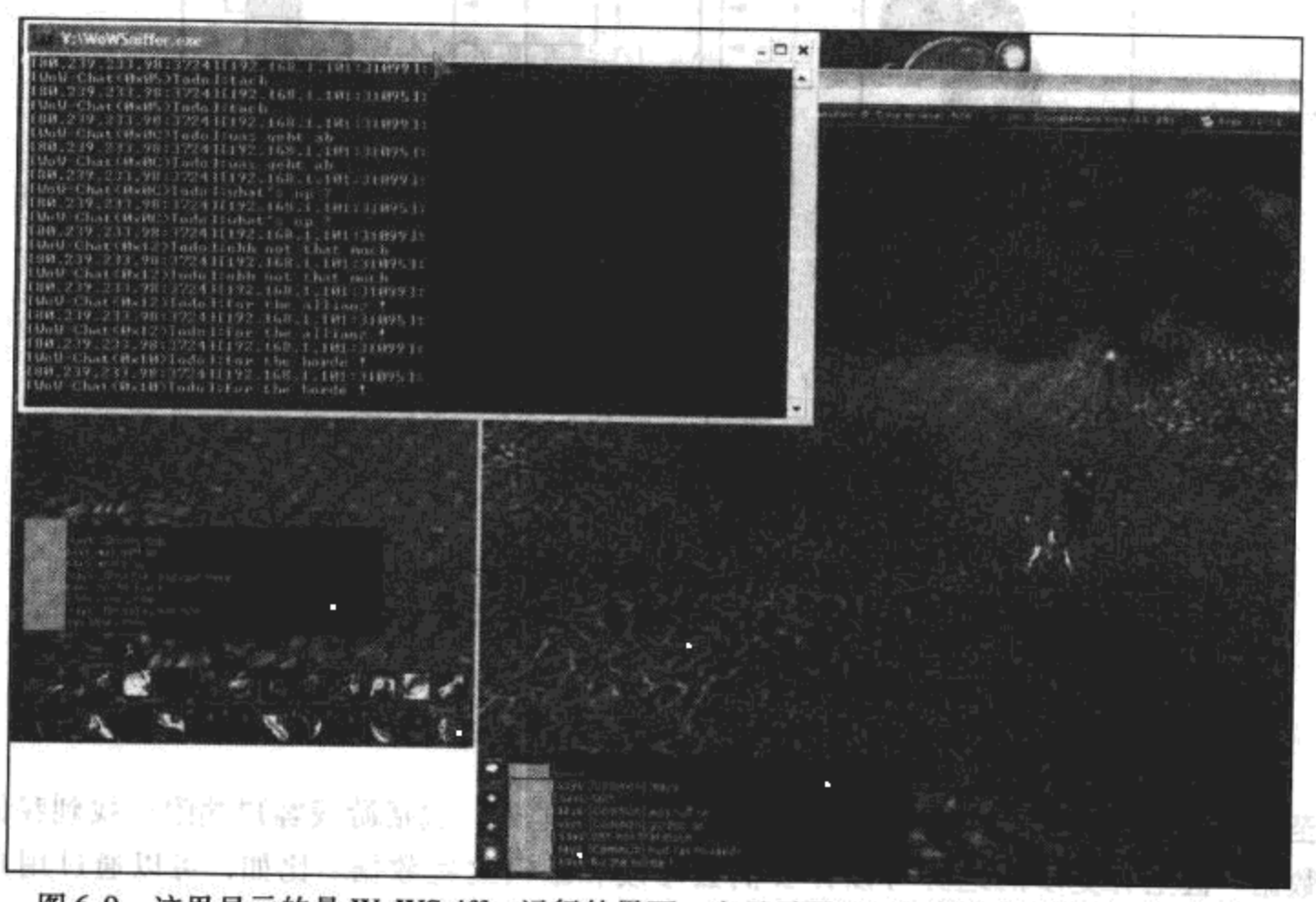


图 6-9　这里显示的是 WoWSniffer 运行的界面，它显示了 WoW 客户端工作的一些信息。
（来自网站 < http：//www. firepacket. net > ；已获得转载许可）

6.3.3　在别处寻找数据

在第 2 章，我们已经介绍过“自动瞄准机器人”的概念了。让我们重新回顾一下，自动瞄准机器人是一种可以帮助玩家在无意识的情况下，像超人一样瞄准敌人的程序。它可以查到游戏中敌人的 3D 坐标并精确的计算出你所用武器该在怎么样的角度下瞄准，并帮你完成最后一击。自动瞄准并向敌人射击之前，首先需要算出那些敌人在游戏中所在位置的 3D 坐标数据。

让我们对自动瞄准机器人程序感兴趣的地方就在于它的运行不依靠程序中的数据，而是利用显卡芯片中保存的内部数据的相互关系来实现的。因为显卡本身就在自己的存储器中携带程序，这些程序专门负责将 3D 物体的描述在屏幕上进行构造（当然，玩家很重视游戏的画面效果，喜欢用最新的图像效果去玩他们游戏）。所以这就表明，自动瞄准器是利用存储在显卡中的 3D 数据为线索去找出在游戏中所要瞄准目标的 3D 坐标。多么有创意！

图 6-10 显示了自动瞄准机器人程序怎样计算并恢复出一张描述所有 3D 对象的内存副本——里面的内容甚至包括那些不会直接在屏幕上显示出来的信息。它的工作原理就是在游戏软件和微软提供的 Direct3D video library 之间拦截数据。

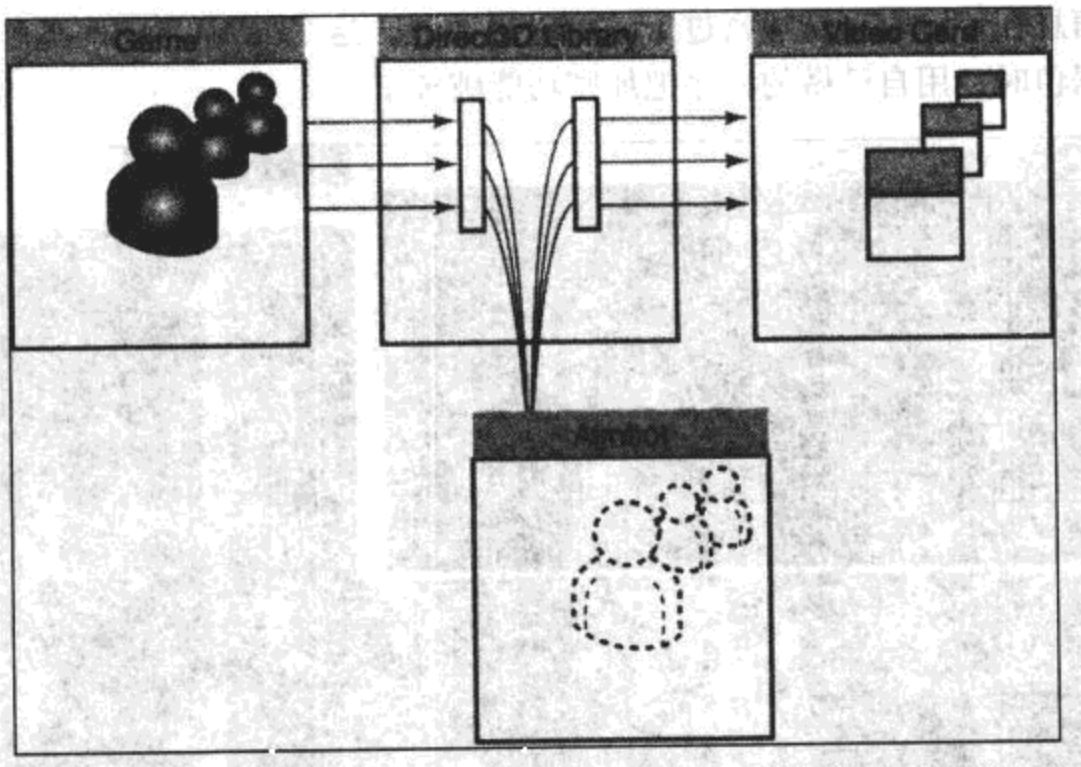


图 6-10 自动瞄准机器人是如何截取发送给显示系统的3D坐标信息的

6.4 在游戏周边得到所有的信息

至今为止，已经很清楚的说明了我们可以在很多地方，包括游戏客户端附近找到我们感兴趣的数据。甚至有更多的地方可以让我们去修改和编辑这些数据。比如，可以通过用户界面（GUI）来发送信息给游戏客户端去破坏数据（这是在游戏软件上层的应用方法），可以通过进入游戏内部并从内存中找到数据来进行破坏（这是内部应用的方法），可以通过在游戏和它所需要的底层设备之间的通信来破坏数据（在游戏软件底层的方法）。最后甚至可以在游戏客户端和服务器之间模拟和破坏数据（在游戏软件外部的方法）。

准备好，我们现在开始了。

6.5 在游戏软件上层：控制用户界面

在第2章，我们已经展现了一个简单的“农场”类游戏机器人。它是通过与WoW游戏客户端的交互来工作。虽然这只是相当普通的技术，不过也很值得再一次粗略的了解一下。如果你需要一个引导性的例子来理顺对接下去的思路，你可以回顾一下第2章。

在游戏上层入手的基本思想是，发送一些让游戏软件感觉是来自于电脑硬件的操作事件。就是说，你可以发送一些游戏客户端感觉只是通常的键盘，鼠标或其他游戏相关事件的信息。有时我们把这个叫做按键精灵。许多按键精灵程序可以有效地向游戏界面（GUI）发送鼠标或键盘事件。我们想，与其使用别人设计的按键精灵程序，不如告诉你怎么去做一个你自己的。以下就从一些简单的例子开始我们的工作。因为自己的软件可以做的扩展是最多的也是最好控制的。

6.5.1 控制键盘敲击

第一个例子所说的是用程序给游戏客户端发送键盘类消息，使客户端以为这个消息和

通过敲击键盘得到的事件一样。这个小功能相当重要，因为它在游戏中可以发展为，自动帮你移动游戏中的人物，帮你为某些行动和事件点击相应的热键，甚至可以帮你在窗口中敲打聊天信息。

这里就开始描述可以发送计算机敲击键盘事件的代码片段。这个操作和发送敲击键盘事件并不考虑当时哪个窗口在最前面或者哪个应用正在运行。这种技术已经普遍用于按键精灵程序中。很多游戏的破解和按键精灵脚本都使用基本的键盘和鼠标事件去和游戏客户端交互。不过使用这种技术也有副作用，就是你在使用自己的机器人程序时就不可以在计算机上做其他事了，因为需要这个技术在运行后直接接管用户界面。

```
DWORD PostKeystroke(
        BYTE theScanCode,
        DWORD theTime)
{
        keybd_event(
                theScanCode,
                MapVirtualKey(theScanCode, 0), 0, 0);
        Sleep(theTime);
        keybd_event(
                theScanCode,
                MapVirtualKey(theScanCode, 0), 0 |
                      KEYEVENTF_KEYUP, 0);
        return 0;
}
```

这个代码片段调用了 API 函数库中的 keybd_event()。调用中把你想要按下的键扫描码作为函数参数。这种编码不同于 ASCII 码。在使用这个函数之前你先要把想要按下的字符对应于它们各自的键扫描码。

6.5.2 神奇的键盘队列

WoW 有一个叫做 Paladin“圣骑士”职业，这个职业需要较多的键盘操作组合——这可能是游戏中最难操作的职业了。在游戏中用这些键来操作，可以在适当组合下给对方最大伤害或者给自己补充法力。本身游戏中就提供热键直接对应到键盘字符上。所以，一个按键精灵可以执行一套 Paladin 的按键组合并在游戏操作中实现最大操作效率。这样的按键序列可能像下面一样：

F1 key ：cast spell Sanctity Aura ——increases Holy Damage

F2 key ：cast spell Seal of Holy Might

F3 key ：cast spell Judgement

F4 key ：cast spell Seal of Command

F5 key：cast spell Hammer of Justice

F6 key：cast spell Crusader Strike

这里总共设置了 6 个按键。现在，在恰当的按键顺序和时机下使用这些键可能产生如下的效果：

开始战斗……

F1 - F2 - F3 - F4

在战斗中等待 15 秒

F5 – F3 – F4

如果目标还活着，继续再战斗 15 秒以后，然后

F5 – F3 – F4

如果定时器允许，重复上述动作。

很显然，这里有很多键盘敲击。如上面所述那样，在连续的按键中，首先使用了 Sanctity Aura，这个技能的作用是提高自身对目标伤害量，然后，使用 Seal of Holy Might 并紧接着掷出 Judged——有效的在对手身上放置一个诅咒并使其在一段时间内遭受额外的伤害。接着，使用 Seal of Command 并继续和对手战斗。这时对手身上的诅咒造成在战斗中更多的伤害，而之后使用的技能 Hammer of Justice 把对手打晕，在对手昏迷状态下再一次使用的 Judgement 可以造成一次暴击。只要战斗还在继续，就一直重复这种使用技能的次序。

这些技能的组合非常多，可以在你设计出最有效的按键精灵之前，给你一些可供分析的想法。

6.5.3 控制鼠标释放

我们可以像控制键盘一样对鼠标产生同样的控制效果，因为鼠标在应用时可以产生像键盘一样的事件。通过发送这些事件消息给游戏客户端，我们可以让客户端相信好像真的有用户在使用鼠标移动和点击。

下面的代码片段展示了调用 mouse_event 事件的 API，并附带有注释。

```
DWORD LMouseClick(DWORD x, DWORD y, bool shift)
{
    int ix,iy;
```

获取显示器或屏幕的宽和高；（以像素为单位）

```
ix=GetSystemMetrics(SM_CXSCREEN);
iy=GetSystemMetrics(SM_CYSCREEN);
```

代码把客户端窗口的坐标转换为显示器坐标。这种坐标转换可以让你随意拖动客户端的窗口，从而避免了必须把游戏客户端窗口左上角和显示器的左上角对齐这个令人讨厌的问题。有些按键精灵程序会有这样的要求，但这里我们向你展示如何避免这样的烦恼。

```
POINT p;
p.x = x;
p.y = y;
ClientToScreen(GetForegroundWindow(), &p);
```

mouse_event 的调用需要你先以 *mikey* 为单位定义坐标——显示器的 x 轴和 y 轴都被分成 65536 个 mikeys（16 比特）——所以左上角的坐标是（0，0），右下角的坐标是（65535，65535）。

```
DWORD mikeysX = p.x * 65535 / ix;
DWORD mikeysY = p.y * 65535 / iy;
```

在屏幕上设置鼠标坐标。

```
mouse_event(
        MOUSEEVENTF_ABSOLUTE | MOUSEEVENTF_MOVE,
        mikeysX,
        mikeysY,
        0, 0);
```

如果你喜欢，可以在鼠标事件发送之前定义 shift 键已被按下。有时这样很有用，比如，在游戏中你需要用 shift 键搭配鼠标右键去搜索一个包等情况。

```
if(shift)
{
        keybd_event(
                VK_LSHIFT,
                MapVirtualKey(VK_LSHIFT, 0), 0, 0);
```

这里的 sleep 函数是可选的，不过这个函数在你的游戏中可以作为代码试验。

```
        //Sleep(20);
}
```

你可以替换函数参数使鼠标右击换成鼠标左击。查询 mouse_event 文档去了解所有可选项。

```
mouse_event(MOUSEEVENTF_LEFTDOWN, 0, 0, 0, 0);
//Sleep(20);
mouse_event(MOUSEEVENTF_LEFTUP, 0, 0, 0, 0);
//Sleep(20);
```

如果之前使用了 shift 键，现在把它复原：

```
        if(shift)
        {
                keybd_event(
                        VK_LSHIFT,
                        MapVirtualKey(VK_LSHIFT, 0), 0 |
                                KEYEVENTF_KEYUP, 0);
                //Sleep(20);
        }
        return 0;
}
```

正如你看到的，控制鼠标事件和控制键盘事件一样简单并且可以自动执行。以上所描述的是一种古老且非常经典的技术。

6.5.4 像素点颜色采样

现在你已经知道如何发送鼠标和键盘事件了。那我们又如何从游戏获得反馈呢？比如，如果想知道一个角色或者目标对手的生命值，那该怎么做呢？一个可行的方法是从屏幕上读取像素的颜色值来获得，因为在通常情况下，生命状态条一般显示为红色，并且总是位于屏幕的一个特殊位置。如果你在该区域进行像素值采样，就可以判断出屏幕上生命值状态现在是多少。当然，像素采样还可以做更多的事情，这里仅仅只是举了一个例子。

以下的程序片段演示了如何判定一个像素的颜色值：

```
COLORREF GetColorOfPixel(DWORD x, DWORD y)
{
        HWND hWnd   = GetForegroundWindow();
        HDC hDC     = GetDC(hWnd);
        COLORREF cr = GetPixel(hDC, x, y);
        ReleaseDC(hWnd, hDC);
        return cr;
}
```

代码中首先获得窗口句柄——这个窗口必须是游戏界面 GUI 并且有输入控制焦点，这是为了使游戏界面可以被键盘或鼠标事件响应。而代码必须在游戏以窗口模式（而非全屏模式）运行时才有效，目前大多数游戏都支持窗口模式。代码中 COLORREF 的数据类型中包含了红、蓝、绿三种颜色的值，而这些基本颜色值可以推算出所有像素的颜色。

图 6-11 显示了一个此技术的示例，用于核算 WoW 界面上状态条显示的生命值、魔法值和其他信息。

图 6-11 对血量条上的像素位置进行标记 X，可以获取生命值、精神值以及其他在 WoW 界面中显示的信息

外挂或者战斗辅助程序可以对屏幕上所指示（标记）的坐标位置进行颜色采样。根据一些样本，外挂程序就能判定角色是满血、中等生命值还是低生命值。

6.5.5 对付按键精灵的措施

“按键精灵”相比其他类型的外挂程序来说，有一个优点——它非常容易理解，制作也很简单。这也是它特别流行的原因。它还有一个优点，那就是它不去操作游戏程序的内存，所以游戏相对也不会探测到自己正在被第三方程序入侵。事实上，像这样的按键精灵很难被认为是黑客程序——它仅仅只是模拟玩家按键和鼠标点击行为。游戏程序如果想要发现玩家正在使用按键精灵，就必须扫描系统中的进程和其他程序的窗口名。这点看上去似乎侵犯了隐私（而且我们坚信这确实侵犯了隐私），不过这是游戏想要在这种层次探测到按键精灵的唯一途径。

类似的按键精灵不仅仅用于 WoW 这类网络游戏，它们也可以用在在线纸牌等游戏中，而纸牌游戏中又会有很多游戏筹码。按键精灵可以自动打牌，就像在魔兽世界操作一个圣骑士那样简单。未经证实的消息说，在人气很旺的在线纸牌站点 PartyPoker. com 上，游戏制作者采取的防范措施是对你的电脑显示器进行全屏截图，即截取你电脑屏幕上所能看到的东西，然后把这些数据回传到他们的计算机上做分析。我们认为这是对隐私的侵犯。

在进程列表中隐藏进程

有的游戏会读取你的计算机进程表，并试图找出已知的按键精灵，诸如 AC Tool 或者其他类似的。这种方法总的来说并不能起到很好的作用，因为给你自己的进程改改名不是什么难事。只要在运行前把 . exe 文件名改一改。

有时候扫描进程列表并不仅仅只是判断进程名——还会进一步通过随后的内存扫描来发现进程。在这种情况下，简单的改名就不能起到作用了。而是需要把整个进程给隐藏起来——可以使用 rootkit 实现这种目的。隐藏一个程序并不困难——只需下载简单易用、非常流行的 FU rootkit 就行，下载地址是 < http：//www. rootkit. com > 。运行 FU rootkit 后就可用其内部功能实现进程隐藏。

更改窗口名

游戏程序寻找外挂的另一种方法是，读取所有以窗口方式打开的文本内容。如果你怀疑你现在玩的游戏也在做这样侵犯隐私的扫描，你可以去下载一种叫 Governor 的工具（这个软件在第 2 章介绍过），是本书的官方网站或者 < http：//www. rootkit. com/vault/hoglund/Governor. zip > 。如果游戏正在试图读取内存中的进程名或者窗口文本，这个软件就会即时通知你。

对付这种方式，只需要把你的窗口标题名都随机改掉，游戏就不能找到任何外挂的迹象了。

用 Rootkits 做一些暗中操作

要使外挂神不知鬼不觉的运行，最有效的办法莫过于使用 rootkits。设计 Rootkits 是为了隐藏计算机上的程序和数据。Rootkits 本身并不坏，因为仅仅只是一个工具。不过跟其他的工具软件一样，可能被好人使用，也可能被坏人利用。如果说要入侵游戏，rootkits 确实是特别有用的工具。更多关于怎样使用 rootkits 来隐藏外挂程序的内容，我们会在后面介绍。

6.5.6 产生窗口消息

当然，游戏总有它专用的消息。如果你能生成这些消息并把它们传到游戏客户端中，从某种意义上来说，你就可以明目张胆的操纵游戏客户端了。

6.6 游戏之内：操纵游戏对象

就先前所描述的，实际上游戏中的任何东西最后都被分解为数据——即客户端数据代码。一些黑客攻击游戏的方式是基于读取那些不可见但是存在于客户端内存中的数据对象（回顾我们之前所举的神奇药水的例子），而这种方式所需要做的就是把游戏内存中所有内容都仔细分析。另一些黑客的攻击方式是手动写入一些数据到内存的进程中去，比如写入玩家在游戏中所在位置之类的例子。这两个我们所描述的攻击方式仅仅是在游戏代码中修改了一些内容。而接下来我们所要介绍的方法并不需要对游戏的代码做任何修改，而是直接操作那些客户端代码需要识别和使用的数据。

我们知道如今的游戏中有很多类似物品，金钱，怪物，玩家这样的对象，全都存放在内存中的某处。而这类对象通常是按照组合或链接的方式存储的。一些游戏使用基类管理系统，在这类系统中的构造链表可以构造游戏中的所有东西。我们知道很多游戏都是使用 C++ 并且以面向对象的方式设计的，其中的链表包含的都是基本类型库（也称作基类库），在游戏中，每个具体对象都是从链表中基类库的继承来实现的。

让我们举个具体的例子，在 WoW 的一些版本中，以下所示的结构类型在游戏中被用来链接各个对象。而这个结构本身是综合了一组特定对象的双向链表。

```
typedef struct
{
        DWORD id;
        DWORD unk1;
        DWORD unk2;
        void *prev;  // previous entry
        void *next;  // next entry
        DWORD unk3;
        DWORD unk4;
        DWORD unk5;
        char *name1;
        char *name2;
        char *name3;
        char *name4;
        DWORD unk6[16];
        DWORD id2;
} gob_list_struct;
```

游戏中对象的存储还有很多其他方式，在这里所示的双向链表只是其中的一种。有时游戏会使用标准的链表，二叉树或者哈希表来存储对象。事实上，游戏可以使用所有数据结构的方式存储对象，这些你都可以在计算机科学方面的知识中学到。

一旦你得知游戏对象是以什么形式表现的和它们在内存中存储的地址，就可以调整内存来满足自己的需要。

6.6.1 动态内存问题

然而通过直接操作内存来调整数据可能导致问题，这个问题是一个需要克服的关键问题。因为内存存储的内容不是静态的等着你来读取，而是会经常切换位置。在有些情况下，你一旦在内存中发现了某些东西（比如之前我们提到过的，那个神奇药水在内存中的实际地址），肯定会记下地址并对这个数据进行存取，如果这个地址保持不变，那肯定非常好，可是如果变化了呢？你可能想读取那个内存地址中的坐标值，也可能想在每次发动攻击的时候向这个地址中写入一个变量。如果内存单元不是静态——实际上的确不是静态的，而且变化很快，那么你在每次想要用这个值的时候都需要花时间再去寻找。这是相当枯燥乏味的，而且更糟的情况是，这样会打乱你的游戏玩法。

不过也有好消息，因为有些存储单元是静态的——也就是说，有些特殊的存储单元存放的是软件中的硬编码，所以它们是不会变动的。在之前图 6-7 所介绍的就是硬编码。在这种情况下的存储单元是静态的，并且可以重复使用。可是，即使在这种情况下也会碰到一些狡猾的问题如：游戏的更新维护。当然，游戏总是要被更新维护的，事实上，大多数 MMO 游戏每周都要进行一次维护。维护会直接改动客户端内存中的数据，当然在如此频繁的情况下移动内存中的硬编码也有一定的副作用。

外挂开发者总是被这个移动内存的问题所困扰。图 6-12 显示了一个 WoW 外挂的源代码，你可以从代码中看到，在现在所用到的硬编码地址之前，开发人员已经注释掉好几行之前的地址。每次进行好游戏维护，外挂开发者也要重新埋头于更新自己外挂的代码。真是一件令人讨厌的事。

```
//#define NPC_BREAKPOINT_LOCATION 0x004851C0
//#define NPC_BREAKPOINT_LOCATION 0x00483EC0
//#define NPC_BREAKPOINT_LOCATION 0x00483FB3
//#define NPC_BREAKPOINT_LOCATION 0x00484403
#define NPC_BREAKPOINT_LOCATION 0x0046F810

// dynamic function pointers - TODO make dynamic
DWORD g_pc_target_id_ptr = 0x94BF58;
DWORD g_pc_player_id_ptr = 0x9413C8;
DWORD g_npc_breakpoint_location = NPC_BREAKPOINT_LOCATION; //todo, make dynamic
DWORD g_pcbase_breakpoint_location = 0x45D492;

#define NPC_LEVEL_OFFSET        0x548
#define NPC_HP_OFFSET           0x354
#define NPC_ID_OFFSET           0x4C0
#define NPC_ID2_OFFSET          0x4C4
#define NPC_FACTION_OFFSET      0x54C
#define NPC_X_OFFSET            0x10
#define NPX_Y_OFFSET            0x14
#define NPX_Z_OFFSET            0x18
#define NPC_FACING_OFFSET       0x1C

#define PC_HP_OFFSET            0x1E98      // was 1EB0
#define PC_MANA_OFFSET          0x1E9C      // was 1EB4
#define PC_MAXMANA_OFFSET       0x18        // PC_MANA_OFFSET + 0x18
#define PC_MAXHP_OFFSET         0x18        // PC_HP_OFFSET + 0x18
#define PC_CURRENT_XP_OFFSET    0x27B0
#define PC_MAX_XP_OFFSET        0x27B4
#define PC_RAGE_OFFSET          0x04        // PC_MANA_OFFSET + 0x04
#define PC_ENERGY_OFFSET        0x08        // PC_MANA_OFFSET + 0x08 or PC_RAGE_OFFSET + 0x08??
//#define PC_MAX_RAGE_OFFSET    0x1D08
#define PC_FACING_OFFSET        0x93C
#define PC_Z_OFFSET             0x938
#define PC_Y_OFFSET             0x934
#define PC_X_OFFSET             0x930
#define PC_BK_Z_OFFSET          0x96C
#define PC_BK_Y_OFFSET          0x968
```

图 6-12　一个 WoW 外挂的代码图示。你可以看到开发此外挂的程序员在多个地方修改了注释，因为游戏更新了

不过我们也有办法来解决这个经常要更新硬编码存储地址的麻烦。这个办法的思路是，在程序设计中，用动态方法实时的自动寻找所要用到的硬编码，这样的设计下，就不用每次都在维护后重新更新自己的源代码了。

6.6.2　围绕一些被怀疑的对象

随着技术的发展，有几个程序对象在任何游戏中都会被用到，这些对象包括：

- 游戏玩家对象：这个对象代表游戏玩家。
- 其他玩家对象：在多用户环境下的其他游戏玩家。
- NPC 和 mobs：游戏中那些商店里的商人和游戏中的妖怪。

在接下来的章节里，我们要在例子中了解下以上的每个对象，并且学习如何找到，并且操纵这些对象。

游戏玩家的数据结构

所有 MMO 类型的游戏都包括游戏角色这个概念。这个角色就是游戏中的“你”。而在内存的某处，有一个地方就是专门存放你这个角色的数据结构，在这个结构中存放的信息有，玩家所在游戏中的 3D 坐标值，生命值和一些能力值（比如力量，法力属性等）。很显然这些值必须存放在服务器中，但是在有些时候客户端程序也可以直接控制这些值。而这样的情况在任何时候，对这个角色的信息都是非常危险的。无理由信赖客户端，在软件安全中本质上就等于破坏了一

个最基本的防范措施。

不过这样做也有理由，比如，在游戏服务器中存放了游戏中的所有状态值，并且实时的为玩家们上亿的数据更新，这将需要太多的带宽。而在现今的网络还不能提供这样的流量。不过为了对游戏的安全更挑剔一点，可以考虑管理所有对象的3D坐标。

如果在游戏中所有物品的3D坐标和成千上万玩家操作的游戏角色的3D坐标都由服务器来管理，会用掉太多的带宽，所以开发者直接把角色的位置坐标和移动更新全部交给客户端来处理。只在偶尔的情况下，客户端所处的新位置需要用服务器来更新。

你看到了这个潜在的漏洞吗？很明显啦！只需要使用调试工具就可以在内存中修改角色的3D坐标。而坐标通常是由3个符点数组成，代表游戏的三维坐标值（X，Y，Z）如果是在2D的游戏中那么就只有（X，Y）2个坐标值。在WoW的一些版本中，服务器并不检查角色移动的合法性。也就是说，客户端提供的任何数据它都会愉快的接受。这也意味着，如果你更新给服务器的新坐标超过了游戏设定的位置区域，你将会瞬间被传送到WoW的虚空之中。

让我们把上面说的搞搞明白吧。以下的列表显示了一个角色的数据结构，是通过逆向工程在早期的WoW版本客户端中得到的，你可以清楚的看到，游戏角色被存储在一个双向链表中（next和prev就是两个用来链接前后的指针）。不过也有一些变量没有被逆向工程探测出来，用unk来表示其未知含义。而在这个结构中最重要的就是48个字节的name数组变量，race_type和class_type变量。

```
typedef struct
{
        void *next;
        void *prev_pc;
        DWORD unk1;
        DWORD id;
        DWORD unk2;
        CHAR name[48];
        DWORD id2;
        DWORD unk3;
        DWORD race_type;
        DWORD unk4[2];
        DWORD class_type;
        DWORD id3;
        DWORD unk5[3];
} pc_list_struct;
```

对于上述的结构，变量race和class被赋予以下整型数所获得的意义已被解释出来了（是通过侦测不同角色得到的）。判定这些值的意义是通过在调试器下观察并对不同角色内部的race和class交叉访问得到的，相当简单吧。

```
// race:
// 3 = dwarf
// 7 = gnome

// class:
// 1 = warrior
// 2 = paladin
// 3 = hunter
// 4 =
// 5 = priest
// 6 =
// 7 =
// 8 = mage
```

判断角色是如何在内存中表示是很容易的，实际上，对于改变这些角色的尝试也是很容易实现的。

当然，在游戏中还有很多不同类型对象的数据结构，游戏黑客的任务就是识别这些结构并在以后使用它们来为自己服务。

6.6.3 读取磁盘文件

对于某些多变的游戏运行环境，比如经常升级和打补丁的游戏，游戏数据对象位置不固定。这种情况下，从磁盘读取文件到内存，在内存中搜索符号或者字节序列特征，用这种方法定位代码或数据地址比较好。使用这种方法，游戏辅助工具以及外挂等可以对抗游戏升级以及打补丁等多变环境。

一般情况下，Windows 注册表可以用于定位游戏客户端。下面的代码片断展示了如何打开及查询 windows 注册表。本例通过检查 WoW 客户端安装时写进去的注册表键值来查找 WoW 客户端在磁盘上的位置。

```
void WowzerEngine::InitWowLocation()
{
        HKEY hKey;
        char szValue[MAX_PATH];
        DWORD dwSize = MAX_PATH;
        HANDLE hFile, hMap;
```

Windows 应用编程接口 RegOpenKey 用于打开注册表键值，该 API 接受的参数主要为注册表的根目录以及要打开的键名，本例中根键为：HKEY_LOCAL_MACHINE。如果 API 使用成功，则键名的句柄保存在第五个参数 hKey 中。

```
        if(RegOpenKeyEx(HKEY_LOCAL_MACHINE,
                "SOFTWARE\\Blizzard Entertainment\\World of
Warcraft",
                0,
                KEY_QUERY_VALUE,
                &hKey) == ERROR_SUCCESS)
        {
```

打开键名以后，可以查询该键下所有的键值。本例中查询 GamePath 来获取 WoW 游戏客户端位置。

```
if(RegQueryValueEx(
        hKey,
        "GamePath",
        NULL,
                NULL,
                (LPBYTE)szValue,
                        &dwSize) == ERROR_SUCCESS)
         {
                strcpy(wow_location, szValue);
         }
         else
         {
                MessageBox(
                        NULL,
                        "Warning. could not find WoW.EXE- cannot init pointer values.",
                        "Cannot continue...",
```

```
                    MB_OK);
  }
RegCloseKey(hKey);
}
```

找到游戏客户端程序的磁盘位置后，可以打开文件读写。示例中读取文件并搜索字节序列和匹配模式。

```
hFile = CreateFile(
        wow_location,
        GENERIC_READ,
        FILE_SHARE_READ,
        NULL,
        OPEN_EXISTING,
        FILE_FLAG_SEQUENTIAL_SCAN,
        NULL);
if(hFile != INVALID_HANDLE_VALUE)
{
```

下面映射文件到内存，方便查找内容。

```
        hMap = CreateFileMapping(
               hFile,
               NULL,
               PAGE_READONLY,
               0,
               0,
               NULL);

        if(hMap)
                pbFile = (PBYTE)MapViewOfFile(
                        hMap,
                        FILE_MAP_READ,
                        0,
                        0,
                        0);
        }

        if(!pbFile)
        {
               MessageBox(
                        NULL,
                        "Could not map WoW.EXE",
                        "Error",
                        MB_OK);
        }
}
```

通过注册表查找游戏客户端，对于基于 windows 平台的游戏非常常用并且也很有效。

6.6.4 解析可执行文件 PE 头格式

游戏客户端的可执行程序本身符合 win32 标准，可移植文件格式（PE）。大多数 windows 下的 .exe 文件符合该文件格式，这种文件格式非常文档化。具体细节可以查看 windows PE 格式标准。下列代码片断列出 PE 头的各个字段，本例基于 Bubba 编写的 WoW 外挂的代码（BWH），该代码可以从网络上搜索得到。

扫描 PE 文件，首先需要定位代码段，从这一点出发，代码示例就很容易理解了。

PE 头还可以定位其他文件信息，比如嵌入资源如图片，导入导出函数，数据段等。这里展示的一些扫描技术还依赖于解析 PE 头的方法。下面展示了 PE 解析器编写的基本技巧，同时给出了例子中的 PE 语法分析程序的帮助函数。

```
DWORD OffsetToRva(PBYTE pbImage, DWORD dwOffset)
{
        //ripped from bwh
        DWORD dwSecBorder = -1;
        PIMAGE_NT_HEADERS pNt =
                PIMAGE_NT_HEADERS(
                pbImage + PIMAGE_DOS_HEADER(pbImage)->e_lfanew);
        PIMAGE_SECTION_HEADER pSec =
               PIMAGE_SECTION_HEADER(PBYTE(pNt) +
        sizeof(IMAGE_NT_HEADERS)); for(DWORD x = 0; x < pNt->
        FileHeader.NumberOfSections; x++)
        {
         if( dwOffset >= pSec[x].PointerToRawData &&
             dwOffset < pSec[x].PointerToRawData  +
             pSec[x].SizeOfRawData)
           return (dwOffset -
                    pSec[x].PointerToRawData +
                    pSec[x].VirtualAddress);
         if( pSec[x].PointerToRawData &&
             pSec[x].PointerToRawData < dwSecBorder)
           dwSecBorder = pSec[x].PointerToRawData;
        }
        if(dwOffset < dwSecBorder)
          return dwOffset;
        else
          return NULL;
}
```

下面的辅助函数将相对虚拟地址（RVA）转化为文件偏移地址，该地址用于读写映射到内存的文件。

```
PBYTE RvaToPointer(PBYTE pbImage, DWORD dwRva)
{
        //ripped from bwh
        DWORD dwSecBorder = -1;
        PIMAGE_NT_HEADERS pNt =
               PIMAGE_NT_HEADERS(
               pbImage +
               PIMAGE_DOS_HEADER(pbImage)->e_lfanew);
        PIMAGE_SECTION_HEADER pSec =
               PIMAGE_SECTION_HEADER(PBYTE(pNt) +
               sizeof(IMAGE_NT_HEADERS));
        for(DWORD x = 0; x < pNt->FileHeader.NumberOfSections; x++)
        {
         if( dwRva >= pSec[x].VirtualAddress &&
             dwRva < pSec[x].VirtualAddress  +
             pSec[x].Misc.VirtualSize)
           return( pbImage +
                   (dwRva - pSec[x].VirtualAddress +
                   pSec[x].PointerToRawData));

         if( pSec[x].PointerToRawData &&
             pSec[x].PointerToRawData < dwSecBorder)
           dwSecBorder = pSec[x].PointerToRawData;
```

```
        }
        if(dwRva < dwSecBorder)
          return pbImage + dwRva;
        else
          return NULL;
}
```

从上面的分析可以知道，解析 PE 文件格式可以获取多种游戏数据。

6.6.5 查找游戏对象

从上面的示例可以看出，很多游戏数据是全局存放的。游戏代码编译以后，指向这些全局数据的指针硬编码在代码中。通过查找硬编码的数据指针，可以进行外挂制作的基础试验。查找游戏代码有两种好的方法：第一种方法是扫描 windows 加载运行以后的内存；第二种方法是直接按照映射文件到内存扫描［译注：详细可以查看段钢编著的加密与解密第三版第 13 章和第 16 章］。第一种模拟加载的方法很精确，直接在内存中就可以定位到相对偏移地址，第二种方法基本和第一种等效，但是如果客户端程序重组或者混淆过［译注：比如程序加壳或者使用其他客户端保护方案］，则第二种方法无法正常工作。这两种方法最后都通过内存指针定位数据，如果是映射文件到内存，直接访问映射的内存地址，如果是加载到内存的程序，直接访问程序起始指针。

下面的示例使用 BWH 的代码。注意：这些技巧目前大多数都能直接被检测，使用这些技巧在游戏时作弊可能导致游戏角色被封。本代码使用打开的文件句柄，解析文件 PE 头信息，定位代码段。然后按顺序扫描一系列操作码，最终发现全局存储的值并读取。

由于很多硬编码的数值，代码本身比较难以读懂，但是代码清晰地指出了游戏数据的相互关联性，即：指针指向一段数据，这段数据中又有指针指向另外的数据等等。一般都是通过这种方法来获取你所想要获取的内存数据。

有些常用数据是外挂所必需的，比如：地表数据，打击数，对象位置，对象状态等。外挂必须知道位置信息来定位自己处于虚拟世界的何处，以及如何与其他对象交互。

所以，一旦我们加载文件以后，我们可以检查感兴趣的物件。下列示例如何搜索 WoW 游戏客户端的对象。

玩家角色尸体定位

在 WoW 游戏中，玩家角色死亡以后，尸体遗留在原地，而角色会出现在墓地。玩家必须返回尸体所在地才能复活并继续游戏。良好的外挂程序应当能够自动返回尸体位置，返回复活点的方法有两种，瞬移到复活点或者驱使玩家角色通过地形手工返回尸体位置。两种方法都需要知道尸体位置的坐标。下列代码片断扫描内存位置，检查尸体坐标保存地点，同样，该代码片断摘录至 BWH。

```
DWORD WowzerEngine::GetLoc_PCorpse()
{
        // ripped from bwh
        DWORD x, y;
        PIMAGE_NT_HEADERS pNtHdr = PIMAGE_NT_HEADERS(pbFile +
              PIMAGE_DOS_HEADER(pbFile)->e_lfanew);
        PIMAGE_SECTION_HEADER pSecHdr =
```

```
            PIMAGE_SECTION_HEADER(pNtHdr + 1);
        PBYTE pbCode =
            RvaToPointer(
                pbFile,
                pSecHdr[0].VirtualAddress );
        DWORD dwCodeSize =
            pSecHdr[0].SizeOfRawData;
```

以下代码在操作码中扫描特定序列，该模式很长，足够保证代码段中只匹配唯一的正确数据。

```
BYTE bCode[] =
        {0xD9, 0xC0, 0xD8, 0xC9, 0xD9,
         0xC3, 0xD8, 0xCC, 0xDE, 0xC1,
         0xD9, 0xC2, 0xD8, 0xCB, 0xDE,
         0xC1, 0xD8, 0x1D};

for(x = 0; x < dwCodeSize; x++)
{
  if(MemoryCompare(&pbCode[x], bCode, sizeof(bCode)))
    {
```

下面的代码很粗糙，其功能是查找一个硬编码的指针。本指针指向一个全局数据，该数据保存了尸体位置。不幸的是，代码比较难看懂，实际上应该可以变得更清晰。但是，可以看出，本段代码回退扫描部分早期匹配的位置。如果一切都符合条件的话，代码返回硬编码的指针。

```
            if(*((PWORD)&pbCode[x-6]) == 0x25D8)
              {
                for(y = 0; y < 30; y++)
                {
                  if(pbCode[x-6-y] == 0xFF)
                    return *((PDWORD)&pbCode[x-4]) - 8;
                }
              }
        }
    }
    return NULL;
}
```

NPC 数据链表处理代码断点

本代码使用查找玩家尸体数据的相同方法，查找处理 NPC 列表的代码区域。在游戏运行过程中，在该位置下断点，这样一来，游戏运行过程中如果要处理 NPC 相关结构，断点就会被触发。断点触发后，外挂可以从某个寄存器获取 NPC 结构指针。由于该结构指针指向链表结构，从第一个 NPC 结构可以遍历到所有的 NPC。所以，仅仅使用断点就可以动态获取 NPC 列表。

```
DWORD WowzerEngine::GetLoc_NPC_Breakpoint()
{
        // ripped from bwh
        DWORD x, y;
        PIMAGE_NT_HEADERS pNtHdr =
            PIMAGE_NT_HEADERS(
            pbFile + PIMAGE_DOS_HEADER(pbFile)->e_lfanew);
        PIMAGE_SECTION_HEADER pSecHdr =
            PIMAGE_SECTION_HEADER(pNtHdr + 1);
        PBYTE pbCode = RvaToPointer(pbFile,
            pSecHdr[0].VirtualAddress);
        DWORD dwCodeSize = pSecHdr[0].SizeOfRawData;
```

上面的代码查找代码段指针以及代码段长度。下面对内存进行匹配，查找关注的位置，查到位置之后，立即返回，下面的步骤会使用该地址进行后续操作。

```
        BYTE bCode[] =
        {0x8B, 0x41, 0x38, 0x8B,
         0x49, 0x3C, 0x83, 0xEC,
         0x40, 0x8B, 0xD0, 0x0B, 0xD1};

        for(x = 0; x < dwCodeSize; x++)
        {
          if(MemoryCompare(&pbCode[x], bCode, sizeof(bCode)))
            {
            return *((PDWORD)&pbCode[x]);
            }
         }
        return NULL;
}
```

定位玩家角色摄像机视角

下面的代码片断和前面介绍的不太一样，该代码不使用操作码序列来扫描定位，而是仅仅比较数值。实际上，该方法和前面介绍的方法本质上都是一个道理。本段代码的功能主要是返回保存摄像机镜头视角的内存位置。

```
DWORD WowzerEngine::GetLoc_PCameraAngle()
{
        // ripped from bwh
        DWORD x;
        PIMAGE_NT_HEADERS pNtHdr =
                PIMAGE_NT_HEADERS(
        pbFile + PIMAGE_DOS_HEADER(pbFile)->e_lfanew);
      PIMAGE_SECTION_HEADER pSecHdr =
                PIMAGE_SECTION_HEADER(pNtHdr + 1);
      PBYTE  pbCode =
                RvaToPointer(pbFile, pSecHdr[0].VirtualAddress);
```

代码同样使用 Windows 提供的 SDK 函数解析 PE 头，这么做可以很快找到代码段的位置，然后得到代码段的长度，并且在整个代码段中查找 0xD8DDD8DD。

```
DWORD dwCodeSize = pSecHdr[0].SizeOfRawData;
for(x = 0; x < dwCodeSize; x++)
{
 if(*((PDWORD)&pbCode[x]) == 0xD8DDD8DD)
   {
```

在代码段中找到目标值后，需要检查附近的几个相关数值。如果所有的都检查无误，则确认找到需要查找的游戏函数。该函数使用了全局数据，外挂代码通过函数中 x +6 处的指针定位并返回所指数据。

```
                if(*((PWORD)&pbCode[x+4]) == 0x05D9)
                  {
                    if(*((PWORD)&pbCode[x+10]) == 0x05D9)
                      {
                        if(*((PWORD)&pbCode[x+16]) == 0xF3D9)
                        {
                          return *((PDWORD)&pbCode[x+6]);
```

```
                    }
                }
            }
        }
    }
    return NULL;
}
```

尽管上面的代码比较丑陋，而且没有办法文档化，但是该技巧仍然比较合理。使用该方法，扫描器可以定位到某个给定的函数，哪怕游戏客户端升级过都是一样（which happens to WoW clients and other game clients with some regularity）。也就是说，游戏升级后，哪怕代码的位置移动了，扫描器仍然可以定位到正确的全局内存地址。

查找玩家角色属性区

下一步扫描定位玩家角色结构。玩家角色结构包含玩家角色当前位置，血，魔法等关键数据。不幸的是，不能通过简单的修改内存结构数值来达到修改游戏的目的，因为服务器也保存了相关数据。但是角色结构仍然对制作外挂有很大的帮助。

下面的代码片断使用了一个很有趣的技巧，WoW 客户端以及其他许多游戏都在二进制文件中保存了部分调试字符串，用于显示错误信息。可以使用这些字符串来定位感兴趣的函数。对于 WoW 来说，部分代码实际上包含了源代码的名字等信息。下面的代码通过扫描第一个已知的调试字符串来扫描定位角色。

```
DWORD WowzerEngine::GetLoc_PCharBlock()
{
        // ripped from bwh
        DWORD x,y;
        PIMAGE_NT_HEADERS pNtHdr =
                PIMAGE_NT_HEADERS(
                pbFile + PIMAGE_DOS_HEADER(pbFile)->e_lfanew);
        PIMAGE_SECTION_HEADER pSecHdr =
                PIMAGE_SECTION_HEADER(pNtHdr + 1);
        PBYTE pbData = RvaToPointer(pbFile,
                pSecHdr[2].VirtualAddress);
        DWORD dwDataSize = pSecHdr[2].SizeOfRawData;
        PBYTE pbCode = RvaToPointer(pbFile,
                pSecHdr[0].VirtualAddress);
        DWORD dwCodeSize = pSecHdr[0].SizeOfRawData;
        PBYTE pbString = NULL;
        BYTE bCode[] = {0xBA, 0, 0, 0, 0};

        // scan for "..\Object/ObjectClient/Player_C.h" in the
        // data section
        for(x = 0; x < dwDataSize - 15; x++)
          {
            if(MemoryCompare(
                  &pbData[x],
                  (PBYTE)"..\\Object/ObjectClient/Player_C.h", 33))
              {
                pbString = &pbData[x];
                break;
              }
          }

        // if we didn't find it then return failure
        if(pbString == NULL)
```

```
        return NULL;

      // build code search pattern
      *((PDWORD)&bCode[1]) =
            OffsetToRva(
            pbFile,
            (DWORD)(pbString - pbFile)) +
                  pNtHdr->OptionalHeader.ImageBase;

      // search for the code
      for(x = 0; x < dwCodeSize - sizeof(bCode); x++)
        {
         if(MemoryCompare(&pbCode[x], bCode, sizeof(bCode)))
           {
            // found the first code, now search for next call
            for(y = x; y < x + 20; y++)
             {
               if(pbCode[y] == 0xE8)
               {
               // found call instruction;
               // decode the instruction for the pointer
                 return OffsetToRva(
                       pbFile,
                       (DWORD)(&pbCode[y] - pbFile)) +
                            pNtHdr->OptionalHeader.ImageBase +
                            5 + *((PDWORD)&pbCode[y+1]);
               }
             }
           }
        }
        return NULL;
}
```

定位玩家角色数据

再展示个与前述代码一样技巧的代码，该代码片断用于查找玩家数据。

```
DWORD WowzerEngine::GetLoc_PCharData()
{
      // ripped from bwh
      DWORD x, y;
      PIMAGE_NT_HEADERS pNtHdr =
            PIMAGE_NT_HEADERS(
            pbFile + PIMAGE_DOS_HEADER(pbFile)->e_lfanew);
      PIMAGE_SECTION_HEADER pSecHdr =
            PIMAGE_SECTION_HEADER(pNtHdr + 1);
      PBYTE pbCode = RvaToPointer(pbFile,
            pSecHdr[0].VirtualAddress);
      DWORD dwCodeSize = pSecHdr[0].SizeOfRawData;

      for(x = 0; x < dwCodeSize; x++)
        {
          // scan for mov ???, [???+D0]
          //          mov ???, [???+D4]
          if(pbCode[x] == 0x8B &&
            *((PDWORD)&pbCode[x+2]) == 0xD0 &&
            pbCode[x+6] == 0x8B &&
            *((PDWORD)&pbCode[x+8]) == 0xD4)
          {
            // found it; now search for the
            // next mov above it to get the pointer
            for(y = x-1; y > x - 20; y-)
              {
```

```
                if(pbCode[y] == 0x8B)
                  return *((PDWORD)&pbCode[y+2]);
                if(pbCode[y] == 0xA1)
                  return *((PDWORD)&pbCode[y+1]);
              }
            }
          }
          return NULL;
}
```

上面我们已经展示了如何查找各种各样的数据指针和代码位置，掌握该技巧后，我们可以构建通用工具，用于扫描游戏程序，这样游戏升级以后，工具仍然可以使用。

6.6.6 构建 WoW 反编译器

游戏公司对游戏的升级和补丁包的发布常常给游戏破解人员带来麻烦，使其疲于奔命。暴雪公司也不例外。为了客户升级带来的重复分析等问题，可以构建 WoW 反编译器，该程序并不是一个真正的反编译器，而是一个数据指针查询程序。

思路比较简单：在分析过一份 WoW. exe 后，分析人员会有反汇编后的代码以及相关引用数据。当游戏升级以后，可以通过原来分析过的数据，通过写工具自动查找新版本的代码及数据位置。代码如下：

```
#include "stdafx.h"
```

程序首先加载 WoW. exe 到内存，扫描已知结构。定义全局指针：

```
DWORD g_binBufSize = 0;
char *g_binBuf = NULL;

bool ReadBinaryBuffer(char *filepath);
DWORD FindOffset( char *theName );

int _tmain(int argc, _TCHAR* argv[])
{
```

函数 ReadBinaryBuffer 读取 WoW. exe 到内存：

```
if(true == ReadBinaryBuffer("wow.exe"))
{
       DWORD offset = 0;
```

读取程序到内存后，可以扫描各种函数。函数 FindOffset 用于方便的查找偏移。首先查找 RenderWorld，本函数在游戏中经常被调用，而且由于该函数在主线程中，是个很好的挂钩点。

```
offset = FindOffset( "RenderWorld");
if(offset != -1)
{
```

获取偏移后，需要添加 wow. exe 在内存中加载的首地址。由于程序读取的是磁盘文件，而不是加载运行好的游戏。这意味着游戏加载后，所有内容的基址为 0x400000，访问游戏进程内

存地址的话，可以将刚刚获取的偏移添加基地址来获取实际的存储地址。

```
    // add base of file in memory
    offset += 0x00400000;
    printf(
          "got offset 0x%08X for RenderWorld\n",
          offset );
}
else
{
  printf("could not find RenderWorld\n");
}
```

下一步查找 ProcessMessage，该代码存在于一个 C++ 类 NetClient 中，通过这个函数可以嗅探所有游戏客户端的网络数据：

```
            offset = FindOffset( "NetClient::ProcessMessage" );
            if(offset != -1)
            {
              // add base of file in memory
              offset += 0x00400000;
              printf(
                    "got offset 0x%08X for
NetClient::ProcessMessage\n", offset );
            }
            else
            {
              printf("could not find ProcessMessage\n");
            }
```

接着查找代码 ClearTarget，该函数在类 CGameUI 中。这是制作外挂的另外一个技巧，直接调用游戏代码销毁当前所有目标。使用这个技巧可以使外挂不再使用前面提到的模拟按键和模拟鼠标操作。

```
            offset = FindOffset( "CGGameUI::ClearTarget" );
            if(offset != -1)
            {
              // add base of file in memory
              offset += 0x00400000;
              printf(
                    "got offset 0x%08X for
CGGameUI::ClearTarget\n", offset );
            }
            else
            {
              printf("could not find ClearTarget\n");
            }
```

再就是很有用的函数 CastSpellByID。调用该函数可以让玩家角色直接施放所有角色能够使用的魔法。从而也避免了模拟按键和模拟鼠标操作。

```
            offset = FindOffset( "Spell_C::CastSpellByID" );
            if(offset != -1)
            {
              // add base of file in memory
              offset += 0x00400000;
              printf(
                    "got offset 0x%08X for
Spell_C::CastSpellByID\n", offset );
```

```
            }
            else
            {
              printf("could not find Spell_C::CastSpellByID\n");
            }

            if(g_binBuf) delete[] g_binBuf, g_binBuf = NULL;
        }
        return 0;
}
```

按照这个思路，还可以获取更多的函数，但是作为一个简单的示例，可以先考虑如何通过内存搜索来获取偏移。

下面的代码读取文件到内存，把读取的内存地址和大小保存在全局变量中：

```
bool ReadBinaryBuffer(char *filepath)
{
        HANDLE hFile;

        hFile = CreateFile(
                filepath,
                GENERIC_READ,
                FILE_SHARE_READ | FILE_SHARE_WRITE,
                NULL,
                OPEN_EXISTING,
                FILE_ATTRIBUTE_NORMAL,
                NULL);

        if(!hFile || hFile == INVALID_HANDLE_VALUE)
          return false;

        g_binBufSize = GetFileSize(hFile, NULL);
        g_binBuf = new char[g_binBufSize];
        DWORD nBytes;
        ReadFile(
                hFile,
                g_binBuf,
                g_binBufSize,
                (LPDWORD)&nBytes,
                NULL);

        CloseHandle(hFile);

        if(nBytes != g_binBufSize)
          return false;

        return true;
}
```

下面的代码按照模式比较内存，'*'为通配符：

```
bool _f_memcmp(const char *in, const char *pat, int len)
{
        for(int i = 0;i<len; i++)
        {
          if(*pat == '*')
            {
              // skip wildcards
            }
          else if( *pat != *in )
            {
```

```
            // the two don't match
            return false;
          }
          pat++;
          in++;
        }
        return true;
}
```

ScanForBytes 函数用于按照给定模式匹配读入的内存：

```
// return -1 if scan fails to find needle, treats * as wildcard
// for now requires at least 4 bytes to search for
DWORD ScanForBytes( const char *haystack,
        DWORD haystack_size, const char *needle,
        DWORD needle_size )
{
        const char *curr = haystack;

        assert(haystack_size >= needle_size);

        while(curr <= (haystack + haystack_size))
        {
          if(*curr == *needle)
            {
              if(true == _f_memcmp(curr, needle, needle_size))
                {
                  // haystack is the beginning of the buffer,
                  // and curr is where string occurs
                  DWORD offset = curr - haystack;
                  return( offset );
                }
              curr++;
          }
          return -1;
}
```

最后，介绍需要在读取文件中查找匹配的各个函数的查找方法：

```
DWORD FindOffset( char *theName )
{
```

注意下文如何注释需要查找的每个函数的反汇编代码。反汇编代码可能对于每个更新版本都有不同，特别是堆栈 cookie，所以，该类代码必须转化为通配符。另外，对于全局变量的引用也常常变化，这类代码也需要使用通配符来匹配。

```
        if(!strcmp(theName, "RenderWorld"))
        {
          // find RenderWorld
          /*
          .text:00479270          push    ebp
          .text:00479271          mov     ebp, esp
          .text:00479273          sub     esp, 80h
          .text:00479279          push    esi
          .text:0047927A          mov     esi, ecx
          .text:0047927C          lea     ecx, [ebp+var_40]
          .text:0047927F          mov     [ebp+var_40], 3F800000h
          .text:00479286          mov     [ebp+var_3C], 0
          .text:0047928D          mov     [ebp+var_38], 0
```

```
        .text:00479294              mov     [ebp+var_34], 0
        .text:0047929B              mov     [ebp+var_30], 0

    55 8B EC 81 EC 80 00 00-00 56 8B F1 8D 4D C0 C7 45 C0 00 00
80 3F
    */
```

上述字符序列所表示的代码可以转化为字符数组用于扫描：

```
            char s[] = {
0x55,0x8B,0xEC,0x81,0xEC,0x80,0x00,0x00,0x00,0x56,0x8B,0xF1,0x8D,
0x4D,0xC0,0xC7,0x45,0xC0,0x00,0x00,0x80,0x3F };
            int offset = ScanForBytes( g_binBuf, g_binBufSize,
s, sizeof(s) );
            if(offset != -1) return offset;
    }
```

注意，代码中将全局变量_aCounter 的地址转化为通配符，因为全局地址很可能随着版本不同有所变化：

```
    if(!strcmp(theName, "NetClient::ProcessMessage"))
    {
    /*
    .text:00514630 arg_0           = dword ptr  8
    .text:00514630 arg_4           = dword ptr  0Ch
    .text:00514630
    .text:00514630                 push    ebp
    .text:00514631                 mov     ebp, esp
    .text:00514633                 mov     edx, _aCounter ****
    .text:00514639                 push    ebx
    .text:0051463A                 mov     ebx, [ebp+arg_4]
    .text:0051463D                 push    esi
    .text:0051463E                 push    edi
    .text:0051463F                 lea     eax, [ebp+arg_4+2]
    .text:00514642                 mov     esi, ecx
    .text:00514644                 inc     edx
    .text:00514645                 push    eax

    55 8B EC 8B 15 * * * * 53 8B 5D 0C 56 57 8D 45 0E 8B F1

    */
            char s[] = { 0x55, 0x8B, 0xEC, 0x8B, 0x15, '*', '*',
'*', '*', 0x53, 0x8B, 0x5D, 0x0C, 0x56, 0x57, 0x8D, 0x45, 0x0E,
0x8B, 0xF1 };
            int offset = ScanForBytes( g_binBuf, g_binBufSize,
s, sizeof(s) );
            if(offset != -1) return offset;
    }
```

下面的代码中，将子函数调用的地方也转化为通配符，因为该类代码也会随着版本更新而变化。

```
    if(!strcmp(theName, "CGGameUI::ClearTarget"))
    {
    /*
    .text:004884C0                 push    ebp
    .text:004884C1                 mov     ebp, esp
    .text:004884C3                 sub     esp, 1Ch
    .text:004884C6                 push    ebx
    .text:004884C7                 push    esi
    .text:004884C8                 push    edi
```

```
        .text:004884C9                  mov     [ebp+var_4], ecx
        .text:004884CC                  call    sub_45DA90
        .text:004884D1                  mov     ecx, dword_B13764
        .text:004884D7                  mov     edi, eax
        55 8B EC 83 EC 1C 53 56 57 89 4D FC E8 * * * * 8B 0D * * * *
8B F8

        */
                char s[] = { 0x55, 0x8B, 0xEC, 0x83, 0xEC, 0x1C,
0x53, 0x56, 0x57, 0x89, 0x4D, 0xFC, 0xE8, '*', '*', '*', '*',
0x8B, 0x0D, '*', '*', '*', '*', 0x8B, 0xF8 };
                int offset = ScanForBytes( g_binBuf, g_binBufSize,
s, sizeof(s) );
                if(offset != -1) return offset;
        }

        if(!strcmp(theName, "Spell_C::CastSpellByID"))
        {
        /*
        .text:006B3270                  push    ebx
        .text:006B3271                  mov     ebx, esp
        .text:006B3273                  sub     esp, 8
        .text:006B3276                  and     esp, 0FFFFFFF8h
        .text:006B3279                  add     esp, 4
        .text:006B327C                  push    ebp
        .text:006B327D                  mov     ebp, [ebx+4]
        .text:006B3280                  mov     [esp+8+var_4], ebp
        .text:006B3284                  mov     ebp, esp
        .text:006B3286                  sub     esp, 28h
        .text:006B3289                  test    ecx, ecx
        .text:006B328B                  push    esi
        .text:006B328C                  push    edi
        .text:006B328D                  mov     [ebp-14h], edx
        .text:006B3290                  mov     [ebp-0Ch], ecx

        */

                char s[] = { 0x53, 0x8B, 0xDC, 0x83, 0xEC, 0x08,
0x83, 0xE4, 0xF8, 0x83, 0xC4, 0x04, 0x55, 0x8B, 0x6B, 0x04, 0x89,
0x6C, 0x24, 0x04, 0x8B, 0xEC, 0x83, 0xEC, 0x20, 0x56, 0x8B, 0xF1,
0x85, 0xF6, 0x57, 0x89, 0x55, 0xF4, 0x89, 0x75, 0xE8, 0x0F, 0x8C,
0x97, 0x04, 0x00, 0x00, 0x3B, '*', '*', '*', '*', 0x00, 0x0F,
0x8F, 0x8B, 0x04, 0x00, 0x00, 0xA1, '*', '*', '*', '*', 0x8B,
0x3C, 0xB0, 0x85 };
                int offset = ScanForBytes( g_binBuf, g_binBufSize,
s, sizeof(s) );
                if(offset != -1) return offset;
        }

        return -1;
}
```

这里展示的技巧是制作不随版本更新而变化外挂的基本技能。外挂可以内嵌扫描功能，或者可以将扫描功能作为开发/构建外挂的一步。这就看外挂制作者想如何使用外挂，或者如何扩展外挂的使用范围了。

6.6.7 读写进程内存

读写计算机内存比较繁琐，精确计算出读取内存中什么内容并不容易。现代计算机的内存容量非常大，从内存中查找你想要的东西就和在一堆稻草中寻找一根绣花针一样困难。

很多情况下，可以根据一个已知的位置开始搜索（比如：一些加载的基本对象）。从这些已知对象，我们可以动态计算并搜索出想要读写的位置。

在微软操作系统读写进程内存，最好的办法是使用如下系统接口：

```
ReadProcessMemory
WriteProcessMemory
```

另外，有些游戏破解者注入 DLL 到游戏进程，从而直接操作游戏进程内存。

6.7 游戏之下：操纵表现信息

读者可能注意到，游戏客户端常常依赖底层库函数来实现游戏功能。一类很经典的游戏破解方法是通过底层图像显示系统破解游戏。

网络游戏的所有对象最终将要在显示器上显示，音响设备中播放等。也就是说，所有三维渲染对象最终需要通过显卡设备进行图像处理。

可能现在读者还不知道如何在游戏之外获取对象位置信息，但是可以肯定的是：所有图形对象最终将在显卡以标准格式显示。和老一代游戏不一样，当代的 3D 游戏都采用工业标准的 API 进行图像处理。现在图像处理的标准有 OpenGL 和 Direct3D。需要注意的是，在标准编程接口以下，有另外一套硬件标准：显卡硬件标准。

一般来说，标准有助于行业的发展和标准化。比如 NVIDIA 显卡，一种流行的游戏显卡程序，使用文档化的格式显示信息。这样，在显示系统的某个层次，可以获取已知格式的数据，从而获取精确的对象坐标信息。

6.7.1 3D = X Y Z

学过计算机图形学课程的人都知道，通过使用简单的几何学（或者更进一步，线性代数），可以计算出和其他对象的相对偏移。事实上，自动计算如何向对象移动，如何瞄准对象，如何逃离对象的范围也是很宝贵的信息。

6.7.2 穿墙技术

几何计算破解技巧首先在第一人称射击游戏中应用。穿墙技巧就是这个技术的应用实例，使用硬件层次或者 API 层次的挂钩技巧实现穿墙射击。直到今天，该类技巧仍然存在于第一人称射击游戏。这类作弊的思路很简单，也很优雅：修改显示系统，使墙体透明。这样作弊玩家可以看到其他墙体后面的玩家，而其他玩家看不到作弊者，从而使作弊者处在绝对的优势下进行游戏，（参见第 2 章）。

从图形学的角度来看，游戏服务端程序本身不可能计算游戏客户端何时该显示墙体和对象。这样会使得计算非常复杂，特别是有对象如敌人和墙体相互有阻隔关系的时候。游

戏服务器一般采用的方法是：发送所有的对象到渲染系统，由显卡系统计算玩家何时该处于哪个视角。这样一来，通过读取显示系统缓存，作弊程序可以感知所有对象的位置。更进一步，从显示系统的缓存中找到墙体对象后，设置墙体结构为透明，这样，作弊者的视角中，墙体消失。

很有意思的是，大多数情况下，游戏客户端本身没有办法控制这些显示缓存。因为客户端已经将显示权交给显卡了。自然而然，显卡本身成为攻击的目标。

6.7.3 DLL 注入

给应用程序（包含游戏客户端）注入新代码，最常用的技巧是 DLL 注入。使用这个方法，可以让远程进程加载到你自己设计的 DLL。DLL 本身的功能可以是挂钩函数，修改目标程序的内存空间。DLL 注入是很隐蔽，很方便的注入手段（实际上，该方法很容易被发现，有不少常规方法可以发现注入的 DLL）。本章后面将介绍更隐蔽的手段。

DLL 注入的技巧是让目标进程中已存在的一个线程跳到某个代码片断，该代码片断调用 LoadLibrary 来加载模块。这样的思路有很多种实现方法。比如：使用 CreateRemoteThread 创建新的远程线程。本小节使用的技巧没有创建新线程，而是劫持现有线程进行加载操作。

首先，需要将加载 DLL 的代码写到远程进程。定义代码段如下：

```
char InjectedCodePage[4096] =
{       0xB8, 00, 00, 00, 00,
        // mov EAX,  0h | Pointer to LoadLibraryA() (DWORD)
        0xBB, 00, 00, 00, 00,
        // mov EBX,  0h | DLLName to inject (DWORD)
        0x53,                                   // push EBX
        0xFF, 0xD0,                             // call EAX
        0x5b,                                   // pop EBX
        0xcc                                    // INT 3h
        // DLL name string will be placed here
};
int length_of_injection_code=15;
```

上述代码有一些占位字节用来填充 LoadLibraryA() 和需要加载的 DLL 名指针地址。对于 LoadLibraryA 的地址，代码简单的假设被注入远程进程的 LoadLibraryA 地址和现在的进程地址相同。作者没有遇到过不相同的情况，所以假设这样做是安全的。[译注：实际上，目前对不少游戏这样操作不能得到正确的地址，原因是 kernel32. dll 位置被移动了。]

```
FARPROC LoadLibProc = GetProcAddress(
        GetModuleHandle("KERNEL32.dll"),
        "LoadLibraryA");
```

对于 DLL 名称，可以将 DLL 名直接写在上述的代码段之后，即代码中偏移为 15 的地方。下面计算所有要用到的偏移：

```
char *DLLName;
DWORD *EAX, *EBX;
//////////////////////////////////////////////////////////////
```

```
// pointers to be used for "stamping" values into the
// injection code (see below)
//////////////////////////////////////////////////////////////////
DLLName = (char*)((DWORD)InjectedCodePage
       + length_of_injection_code);
EAX = (DWORD*)( InjectedCodePage +  1);
EBX = (DWORD*) ( InjectedCodePage +  6);
```

这样就得到了两处占位字节序列的地址和将要填写的 DLL 名地址。需要将代码写到远端进程才能填写这几个地址，因为代码将在远端执行，而不是本地。

内嵌在代码段中的 INT3 指令用于和调试器交互，内嵌断点通知调试器，注入代码执行完毕。现在不管这些，先了解一下如何查找注入代码位置。

下面的代码，假设已经打开调试器并发送初始调试信号，如果该操作为附加操作，将会为初始断点事件创建线程。代码将劫持该线程并利用该线程运行注入的代码片断。在调试事件循环中，使用如下代码获取线程句柄：

```
hThread = fOpenThread(
              THREAD_ALL_ACCESS,
              FALSE,
              dbg_evt.dwThreadId );
```

记得使用句柄后关闭句柄。

DLL 名称为 DLL 文件的全路径，hProcess 为被调试的远端进程。hModuleBase 可以从调试事件循环中获取，或者可以假定 0x400000 为基地址（绝大多数情况都是这样）。从调试事件循环中获取的方法为：

```
if(dbg_evt.dwDebugEventCode==CREATE_PROCESS_DEBUG_EVENT)
{
       hModuleBase = dbg_evt.u.CreateProcessInfo.lpBaseOfImage;
}
```

注入的下一步骤为定位远程进程有效的可执行代码段。这需要解析 PE 文件头，查找 DOS 头：

```
res = ReadProcessMemory(
       hProcess,
       hModuleBase,
       &DOShdr,
       sizeof(DOShdr),
       &read);
if( (!res) || (read!=sizeof(DOShdr)) )
{
  printf("Could not get DOS header\n");
  return FALSE;
}
```

现在通过 MZ 字符串检查文件是否为 DOS 头：

```
if( DOShdr.e_magic != IMAGE_DOS_SIGNATURE ) //Check for 'MZ
{
  printf("Could not find MZ for DOS header\n");
  return FALSE;
}
```

通过 DOS 头信息查找 NT 头信息：

```
//Get NT header
res = ReadProcessMemory(
       hProcess,
       (VOID*)((DWORD)hModuleBase +
       (DWORD)DOShdr.e_lfanew),
       &NThdr,
       sizeof(NThdr),
       &read);

if( (!res) || (read!=sizeof(NThdr)) )
{
  printf("Could not get NT header\n");
  return FALSE;
}
```

检查 PE 字样判断文件是否为 PE 格式文件：

```
//Check for 'PE\0\0
if( NThdr.Signature != IMAGE_NT_SIGNATURE )
{
  printf("Could not find PE in NT header\n");
  return 0;
}
```

现在可以判断内存为有效的 PE 格式，开始检查可用可写可执行代码区：

```
if( (dwD=NThdr.FileHeader.NumberOfSections) < 1 )
{
  printf("No sections to scan!\n");
  return FALSE;//Section table: (after optional header)
}

// nasty ptr arithmetic (yucky PE)
pSecHdr = (IMAGE_SECTION_HEADER*)
       (
       ((DWORD)hModuleBase + (DWORD)DOShdr.e_lfanew) +
       (DWORD)sizeof(NThdr.FileHeader) +
       (DWORD)NThdr.FileHeader.SizeOfOptionalHeader + 4
       );

res=FALSE;
dwD2 = (DWORD)GetModuleHandle(0);

for( dwD2=0 ; dwD2<dwD ; dwD2++ )
{
```

遍历区段，检查可写区段，需要确保区段不为 .idata 段，因为输入段被破坏可能导致文件无法执行。

```
if( !ReadProcessMemory(
       hProcess,
       pSecHdr,
       &SecHdr,
       sizeof(SecHdr),
```

```
        &read) )
{
  printf("ReadProcessMemory failed, error %08X\n",
  GetLastError());
  return FALSE;
}
    if(read != sizeof(SecHdr))
    {
      printf("section size mismatch!\n");
      return FALSE;
    }

  printf("looking at target section, %s\n", SecHdr.Name);

  //writable section? And
  //not .idata (import data)

  if(
   (SecHdr.Characteristics & IMAGE_SCN_MEM_WRITE)&&
   ( strcmpi((const char*)SecHdr.Name, ".idata")!=NULL ))
  {
    printf("FOUND useable code page: %s\n",
          SecHdr.Name );

    res = TRUE;
    break;//OK!!
   }

  pSecHdr++;
}

if(!res)
{
  printf("couldn't find usable code page!\n");
  return FALSE;
}
```

得到可写代码段后，先保存原始代码，将开始构造的代码片断写入得到的区段。下面代码获取区段指针：

```
gRemoteSectionPtr = (VOID*)(SecHdr.VirtualAddress +
                    (DWORD)hModuleBase);
printf("Using section pointer 0x%08X\n", gRemoteSectionPtr);
```

现在可以知道实际的远端代码地址，填充 LoadLibrary 和 DLL 名称地址等信息：

```
strcpy( DLLName, DllName );
*EAX = (DWORD)LoadLibProc;
*EBX = length_of_injection_code + (DWORD)gRemoteSectionPtr;
```

保存远程进程代码：

```
if(!ReadProcessMemory(
       hProcess,
       gRemoteSectionPtr,
       gOriginalCodePage,
       gSizeOfRemoteCodePage,
       &read) )
{
  printf("ReadProcessMemory failed, error %08X\n",
         GetLastError());
```

```
  return FALSE;
}

if(read != gSizeOfRemoteCodePage)
{
  printf("Could not write the correct number of bytes\n");
  return FALSE;
}

printf("writing new code to remote address 0x%08X\n",
       gRemoteSectionPtr);
```

在远程进程填写要注入的代码：

```
res = WriteProcessMemory(
       hProcess,
       gRemoteSectionPtr,
       InjectedCodePage,
       gSizeOfRemoteCodePage,
       &written);

if( (written!=0) && (written!=gSizeOfRemoteCodePage) )
{
  printf("Error writing injection code. Remote process MAY
  CRASH\n");

// try to save face and put the old code back...
WriteProcessMemory(
       hProcess,
       gRemoteSectionPtr,
       gOriginalCodePage,
       gSizeOfRemoteCodePage, &written);
return FALSE;
}

if((!res) || (written!=gSizeOfRemoteCodePage))
{
  printf("Error injecting code\n");
  return FALSE;
}
```

现在已经将代码注入远程进程内存，准备劫持主线程（the primary thread in the remote process），强制修改指令执行指针（EIP）指向刚刚修改的代码位置：

```
printf("setting thread EIP to 0x%08X\n", gRemoteSectionPtr);

Context = gRemoteThreadOriginalContext;
Context.Eip = (DWORD)gRemoteSectionPtr;
res = SetThreadContext(hThread, &Context);
if(!res)
{
  printf("Error, could not set remote thread EIP, process
    MAY CRASH\n");
  return FALSE;
}
```

注入代码执行后，加载 DLL，然后软件断点消息发送出来，调试消息循环获取断点消息后，再恢复原始代码和原始线程信息：

```
if(EXCEPTION_BREAKPOINT ==
        dbg_evt.u.Exception.ExceptionRecord.ExceptionCode)
        &&
        g_codepage_breakpoint_has_fired == FALSE )
{
        ////////////////////////////////////////////////////
        // the breakpoint in our injected code has fired
        // this means the "injected" DLL has been loaded
        // and it's now time to fix the remote exe
        // back to normal and let the DLL do the rest
        // of the work.
        ////////////////////////////////////////////////////
        CONTEXT ctx;

        ctx.ContextFlags = CONTEXT_FULL;
        if(!GetThreadContext(hThread, &ctx))
        {
          printf("[!] GetThreadContext failed ! \n");
          return 0;
        }

        printf("Hit codepage breakpoint\n");
        printf("EAX: 0x%08X\n", ctx.Eax);
        printf("EIP: 0x%08X\n", ctx.Eip);

        g_codepage_breakpoint_has_fired = TRUE;
```

RestoreOriginalCodePage 仅恢复我们保存的原始代码。

```
        RestoreOriginalCodePage( hTargetProc, hThread, 0);

        // we are done using the debugging loop
        bDoneDebugging = TRUE;
}
```

远程进程代码段恢复后，退出调试器。实际上，也可以保持调试器运行，让注入的 DLL 与调试器交互，这个时候有很多可供选择的实现方法，取决于你想要到达的目的。

RestoreOriginalCodePage 代码很简单，如下：

```
DWORD RestoreOriginalCodePage(
        HANDLE hProcess,
        HANDLE hThread,
        DWORD *outSize )
{
        BOOL res;
        DWORD written;
        CONTEXT Context;
        if(outSize) *outSize = gSizeOfRemoteCodePage;

        Context.ContextFlags = CONTEXT_FULL;
        GetThreadContext( hThread, &Context);

        printf("Restoring original code to remote section ptr
0x%08X, len %d\n",
                gRemoteSectionPtr,
                gSizeOfRemoteCodePage);

        res = WriteProcessMemory(
                hProcess,
                gRemoteSectionPtr,
                gOriginalCodePage,
```

```
            gSizeOfRemoteCodePage,
            &written );

        if(!res)
        {
          printf("WriteProcessMemory error %08X\n",
               GetLastError());
          return -1;
        }

        if(written!=gSizeOfRemoteCodePage)
        {
          return written+1;
        }

        res=SetThreadContext(
            hThread,
            (CONST CONTEXT*)&gRemoteThreadOriginalContext);
        if(!res)
        {
          printf("SetThreadContext error %08X\n", GetLastError());
          return -1;
        }

        return 0;
}
```

本小节介绍了注入运行进程的高级技术，代码劫持。该技术目前在游戏黑客领域内仍然很有效。

DLL 劫持技术

另一种注入 DLL 的方法为：替换程序所用的 DLL 为自己编写的 DLL。替换的 DLL 必须容易转发到原始 DLL 函数（the easiest way to mimic functionality）。如果被替换的 DLL 包含很多导出函数，重写 DLL 的工作将会很繁琐［译注：yonsm. net 上有个工具叫 aheadlib 可以自动生成转发函数代码］。特别是 DLL 持续更新的话，工作量更大。总的来说，DLL 替换技术目前使用并不广泛。这个技巧曾经在第一人称射击游戏中替换 OpenGL 库，用于穿墙功能。一个很好的例子就是“使命召唤 2”中的穿墙技术。

替换 DLL 技术中调用规范很重要。必须遵从正确的函数参数，类型和调用规范才能正确的写出转发函数。如果出错的话，链接器将不能正确地解析调用，DLL 不能加载。这样，游戏无法正常运行，会抛出运行错误：无法找到所需 DLL 文件。

一些程序可能会在使用前检查 DLL 版本。DLL 包含资源，指明厂商，版本号，DLL 名等信息。有通用工具可以修改这些字符串，很多 PE 编辑器有这些功能。

Direct3D，DirectX 相关

回到图像问题，常用截取 Direct3D 和 OpenGL 渲染数据的方法是将原始 DLL 替换为木马 DLL。木马 DLL 用于传递所有系统调用，查看或者修改关注的数据。进入游戏客户端和视频系统后，木马 DLL 可以劫持，修改任何渲染相关数据。

比如，穿墙技巧可能有选择性的修改特定对象的渲染设置，可以透过阻碍物观察到。或者也可以修改颜色，使敌人的显示特别显眼。[⊖]

⊖ 由于 Direct3D 是微软公司提供的标准接口，因此网上充斥着大量的第三方的截获软件。——译者注

6.7.4 隐藏注入 DLL 文件

注入 DLL 是游戏分析者的常用技巧，所以很多游戏已经开始扫描注入的 DLL。DLL 可以从加载模块列表加载或者卸载，从而躲过加载 DLL 列表检测。这样，DLL 的内存并不安全。

从加载模块列表卸载 DLL，可以采用 NtIllusion rootkit 采用的方案，http：//www. rootkit. com 上可以下载。下面介绍 NtIllusion rootkit 如何从加载模块链上摘除 DLL。

PEB 中有 3 个双向队列与这个技巧有关。前面介绍过远进程获取 PEB 指针的方法，但是注入 DLL 不需要这么麻烦，可以直接引用本地内存获取 PEB。[译注：PEB 结构中 Ldr 字段指向 PPEB_ LDR_ DATA 结构，该结构为本进程维护 3 个进程模块（PE 结构文件）列表，使用 windbg 的！peb 命令可以看到这 3 个列表为 InInitializationOrderModuleList，InLoadOrderModuleList 和 InMemoryOrderModuleList，这三个队列中比较特别的是第一个 InInitializationOrderModuleList，该队列不包含 exe 本身的映像，其他两个基本相同，主要用于不同的搜索定位功能。]

下面的函数用于检查 DLL 的不同双向列表。出于完整性的考虑，应当调用这个函数 3 次，每次使用不同的列表标志符：LOAD_ORDER_TYPE、MEM_ORDER_TYPE、INIT_ORDER_TYPE，这 3 个列表查找和删除 DLL 模块的操作方法相同。

```
#define LOWCASE(a) ((a>='A' && a<='Z')?a-('Z'-'z'):a)
#define LOAD_ORDER_TYPE 0
#define MEM_ORDER_TYPE  1
#define INIT_ORDER_TYPE 2

int WalkModuleList(char ModuleListType, char *szDllToStrip)
{
        PLIST_ENTRY pUserModuleListHead, pUserModuleListPtr;
        PLIST_ENTRY *pHiddenModuleListPtr;

        int i;
        DWORD PebBaseAddr, dwOffset=0;
        DWORD ImageBase, ImageSize;
        PPEB_LDR_DATA pLdrData;
        PUNICODE_STRING pImageName;
        char szImageName[BUFMAXLEN]; // Non-unicode string
```

下面的代码获取本进程 PEB：

```
        PebBaseAddr = GetPEB(0);
        if(PebBaseAddr == FUNC_ERROR)
          return 0:
        // PEB.ProcessModuleInfo = PEB + 0x0C
        pLdrData = (PPEB_LDR_DATA)(DWORD *)(*(DWORD *)
                (PebBaseAddr + PEB_LDR_DATA_OFFSET));
        if(!pLdrData->Initialized)
          return 0;

        // Init chained list head and offset
        if(ModuleListType == LOAD_ORDER_TYPE)
        {
          // LOAD_ORDER_TYPE
          pUserModuleListHead =
          pUserModuleListPtr =
```

```
  (PLIST_ENTRY)(&(pLdrData->ModuleListLoadOrder));
  pHiddenModuleListPtr = &pUserModuleListLoadOrder;
  dwOffset = 0x0;
} else if(ModuleListType == MEM_ORDER_TYPE)
{
  // MEM_ORDER_TYPE
  pUserModuleListHead =
  pUserModuleListPtr =
  (PLIST_ENTRY)(&(pLdrData->ModuleListMemoryOrder));
  pHiddenModuleListPtr = &pUserModuleListMemoryOrder;
  dwOffset = 0x08;
} else if(ModuleListType == INIT_ORDER_TYPE)
{
  // INIT_ORDER_TYPE
  pUserModuleListHead =
  pUserModuleListPtr =
  (PLIST_ENTRY)(&(pLdrData->ModuleListInitOrder));
  pHiddenModuleListPtr = &pUserModuleListInitOrder;
  dwOffset = 0x10;
}
else return 0;

do
{
  // Jump to next MODULE_ITEM structure
  pUserModuleListPtr = pUserModuleListPtr->Flink;
  pImageName = (PUNICODE_STRING)(
               ((DWORD)(pUserModuleListPtr)) +
               (LDR_DATA_PATHFILENAME_OFFSET-dwOffset));

  ImageBase = *(DWORD *)(((DWORD)
               (pUserModuleListPtr)) +
                        (LDR_DATA_IMAGE_BASE-dwOffset));

ImageSize = *(DWORD *)(((DWORD)(pUserModuleListPtr)) +
            (LDR_DATA_IMAGE_SIZE-dwOffset));

//Convert string from unicode and to lower case
for(i=0; i < (pImageName->Length)/2 && i<BUFMAXLEN;i++)
  szImageName[i] =
        LOWCASE(*( (pImageName->Buffer)+(i) ));
  szImageName[i] = '\0';

  if( strstr((char*)szImageName, szDllToStrip) != 0 )
  {
```

文件名为要隐藏的 DLL 名，从双向链表摘除节点。

```
          (pUserModuleListPtr->Blink)->Flink =
                (pUserModuleListPtr->Flink);

          (pUserModuleListPtr->Flink)->Blink =
                (pUserModuleListPtr->Blink);

           whImageBase = ImageBase;
           whImageSize = ImageSize;

          *pHiddenModuleListPtr = pUserModuleListPtr;
          }
      }
```

```
        while(pUserModuleListPtr->Flink != pUserModuleListHead);

        return 1;
}
```

上面代码执行后，如果模块列表存在该指定 DLL，该 DLL 将从链表摘除。这不影响 DLL 的正常工作，因为 DLL 仍然在内存中，只不过进程的加载模块列表显示不出该 DLL 而已。但是，该 DLL 的内存仍然能被访问到，从而还是能够被检测。

获取本进程 PEB 的方法很简单，如下：

```
DWORD dwPebBase
__asm
{
        push eax
        mov eax, FS:[0x30]
        mov [dwPebBase], eax
        pop eax
}
```

可以看到，FS 寄存器选择子偏移 0x30 的内容指向 PEB 结构。

6.8 游戏之路：操纵网络数据包

上面已经讨论了游戏之上，游戏之下，游戏之内的各种分析破解技巧。现在讲解游戏客户端分析中非常有效的手段：网络数据包。很明显，需要和服务器交互的或者需要长期保存的客户端状态最终将要通过网络发送给游戏服务器。基于这一点，很多游戏破解者仅关注网络数据。

关注网络数据的最大好处是，所有网络数据包发送接受只有一个单一的接口，所有有用的数据最终都要经过这里。这样一来，游戏破解者不需要关注进程空间的各种数据，代码，截获数据和各种各样的代码保护，要跟踪查看还得调试，下断点，反汇编等等。

数据加密

游戏开发商也不是傻瓜，开发商基本都意识到网络传输的风险，开始采用算法对传输的数据进行加密。也就是说，游戏加密所有客户端与服务器交互的数据，从而没有办法查看明文数据。

这样一来，如果在网络中途截获数据包，不管是查看还是需要修改，都需要理解游戏加密算法。这属于密码学范畴的问题了，但是，由于通信的需求，终端需要知道使用何种加密算法和加密因子。换句话说，游戏客户端也知道如何解密加密的数据。这样，通过逆向工程，可以获取正确的加解密算法，从而理解游戏协议。

图 6-13 显示了 WoW 的数据包解密过程。

```
BYTE *DecryptPacket(BYTE *Packet, WORD PacketLen, bool type)
{
    int i = 0, cryptedBytes;
    BYTE curKey, tmp, index, lastByte;
    WoWKeyIndex *KeyIndex;

    if(!(IsBadWritePtr(CryptInfo, [redacted]) && CryptInfo->IsCrypted)
    {
        if(type == SENT_PACKET)
        {
            KeyIndex     = &(CryptInfo->SendKey);
            cryptedBytes = [redacted]

            //Received packets don't need the key state returned because they haven't been touched by
            for(i = cryptedBytes; i != 0; i--)      //Return our key index to the state it was in when
            {
                KeyIndex->Index--;
                if(KeyIndex->Index == [redacted]          //-1, BYTES are unsigned..
                    KeyIndex->Index = (CryptInfo->KeyLen - 1);
            }

            KeyIndex->LastByte = htons(PacketLen) & [redacted] //the length field is stored in network byte
            KeyIndex->LastByte = (KeyIndex->LastByte ^ *(CryptInfo->Key + KeyIndex->Index));
            KeyIndex->LastByte = *Packet - KeyIndex->LastByte;
            lastByte = KeyIndex->LastByte;
            index = KeyIndex->Index;

        }
        else
        {
            cryptedBytes = [redacted]
            KeyIndex     = &(CryptInfo->RecvKey);

            index    = KeyIndex->Index;
            lastByte = KeyIndex->LastByte;
        }

        for(i = 0; i < cryptedBytes; i++)
        {
            curKey   = *(CryptInfo->Key + index);
            index++;
            index = index % CryptInfo->KeyLen;

            tmp              = *(Packet + i);
            *(Packet + i)    = (*(Packet + i) - lastByte) ^ curKey;
```

图 6-13 代码摘录自一个通过逆向游戏客户端得到的网络探嗅器的报文解密代码，这个解密的目的正是企图获取 WoW 数据包的内容

6.9 最终之路：从内核操纵客户端

纵观本章，已经讲解了很多操纵游戏客户端的技巧：内存扫描，使用断点，程序挂钩，代码注入。但是这些操作都可以被游戏客户端检测到。如何对抗这种应用层检测呢？所以就有了这一小节：很多游戏都不包含内核模块，游戏分析人员可以从内核出发分析游戏。

前面章节假设可以通过 rootkits 隐藏游戏分析工具，但是如果把外挂本身全都放在内核来做会怎样呢？

首先，如果外挂程序完全存在于内核，外挂不作为进程存在，应用层无法通过扫描进程内存感知外挂，外挂检测程序需要通过扫描内核模块来发现外挂程序。一般情况下，这是不被允许的，除非游戏使用高级别的权限。其次，在内核中，外挂和游戏本质上的权限一样，从这个角度

上来看，外挂程序也就相当于 rootkits。

从内核来实现外挂看起来比较美好，但是实现还是有很多困难：

1. 内核模块软件和应用层软件差别很大。

2. 不少游戏已经采用内核模式保护，比如 nProtect 和 PunkBuster。

对于大多数人来说，上面的第一个困难应该不是问题。没有内核编程经验的人不熟悉 Windows 操作系统内核结构，不敢从内核编写外挂程序。实际上，内核编程不像想象中那样困难。只要了解部分操作系统知识，大多数应用编程人员可以很方便的编写内核模式驱动程序。

上面的第二个困难确实会难倒很多人。游戏保护公司开始采用 rootkit 类似的技巧来保护游戏。如果内核模式外挂被公开，内核模式游戏保护程序很快可以对付该外挂。对于私有外挂来说，也存在被通用扫描发现的风险（比如检查 IDT 和 SSDT）。另外最大的问题是：存在内核保护会带来协同工作的问题，内核同时存在保护系统和内核外挂会带来潜在的不稳定，甚至会蓝屏死机。[译注：np 等保护系统存在以后，系统会相对不稳定，而且很多操作不可完成，不少程序也无法打开就是这个问题。]

上述两个困难，第二个比较难解决。但是，像 WoW 这样的游戏，并没有内核模式保护［译注：不知道作者提这个的目的，写技术书籍的容易想到什么写什么，可能作者是想说游戏策略重要］。更进一步来说，对于任何特定游戏的保护技术，都可以有特定的突破方法。从上面的介绍可以看出，掌握内核技术可以扩展外挂的工作范围。

内存伪造

现代大多数操作系统都是多任务的，CPU 的保护模式是操作系统实现多任务的基础，多任务隔离主要通过虚拟内存映射实现，即在保护模式下，每个任务都有自己的虚拟内存空间，任务之间相互隔离，由操作系统对每个任务的虚拟内存进行管理，映射到物理内存。从而，尽管不同的任务可以访问相同的地址，比如 0x00400000，但是不同任务所读取的实际物理地址是完全不同的，从而每个进程的视图相同，但是内容不同。

在第 7 章将具体讨论细节，这里简单介绍一下，虚拟地址由操作系统通过页表根目录［译注：CR3 指向的页表］转换为实际的物理地址。所以，修改进程页表，可以操纵进程内存，可以修改那些内存为可见，或者重新映射进程内存到其他地方。

图 6-14 中注入的代码页实际是存在与察看的位置，但是如果调试器试图读取该内存，将读取到一页伪造的内存，内存内容全为字母'A'。内核驱动通过操纵页表来切换内存转换，是很强悍的方法。

在第 7 章中，将会简要介绍内核模式技巧，包含中断挂钩技巧。

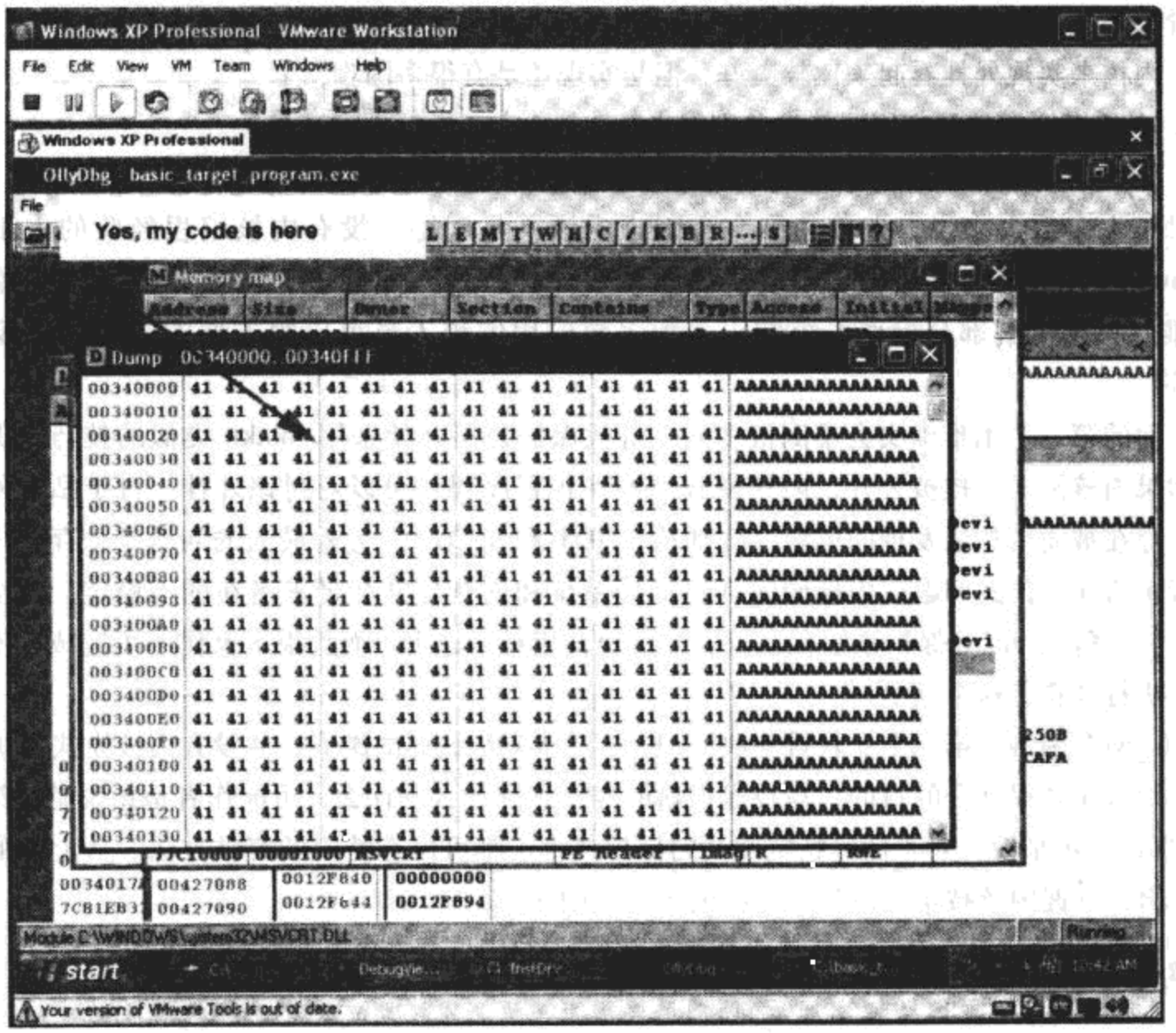


图 6-14 伪造内存可以用来隐藏本进程的实际内容，
使其他进程查看的内容为假。本例仅替换内存为字母'A'

6.10 结论

本章主要分析和理解游戏客户端的一系列技巧。展示了如何分析代码，采用常用技巧去影响和入侵游戏客户端。最后介绍了内核技巧。

游戏客户端破解的最终目标是脱机类的外挂程序，完全模拟客户端运作。要做到这一点，关键是要对游戏进行调试，理解游戏协议，这一点将在下一章详细说明。

第7章 “外挂”软件技术点

在第6章中介绍了很多拆解分析游戏客户端的方法。这些方法本身都很有意思，有通过介入游戏软件和视频显示系统修改游戏表现效果的，有注入DLL或者通过写游戏进程内存来修改游戏流程和数据的，还有监控修改网络封包的。然而，只有将第6章介绍的技巧综合使用才能得到强大的游戏辅助功能。现在，读者已经掌握了很多游戏操纵技巧，本章将综合使用这些技巧构建外挂。

外挂是游戏分析的最终成果。可以使用很多方法制作外挂。从外挂本身来看，外挂是结合了第6章各种技巧实现的一个带有特定逻辑功能的程序。前面介绍的分析游戏之内，游戏之下，游戏之上等各种技巧是相对独立的，可以单独进行，但是编写外挂则需要理解游戏逻辑相关信息。比如：玩家角色血量信息对于不同游戏来说，保存方式会很不一样。同样，玩家角色执行的一系列操作及其顺序也和游戏紧密相关。

外挂需要知道游戏运行逻辑，比如玩家角色装备的武器，玩家角色属于何种类型，玩家角色有什么技能，魔法和能力等。从这点来考虑，本章专门针对WoW分析，尽量描述清楚真实的游戏分析是如何进行的，如果想要做出一个外挂，则必须要清楚这些特性。

真实的游戏外挂有很多种类，有第一人称射击游戏的辅助工具（自动瞄准工具，如Unreal游戏的ZelliusBot），也有无人值守（AFK）战斗外挂（如魔兽中的WOWGlider和WoWSharp）。基本上，服务器是禁止外挂存在的，根据游戏的不同，如果游戏检测到玩家使用了外挂，玩家角色很可能会被封号，暴雪公司素以封号著名。如果账号被封，玩家在游戏上所有的投资将付之东流。使用本章介绍的技巧作弊，很可能会被查封，如果你被查封的话，请不要向作者抱怨你是因为使用本书介绍的技巧作弊被封。

本章介绍了很多资料，帮助读者理解外挂如何帮助玩家自动玩游戏。在外挂制作基础小节中，讲述了状态机和事件驱动模型：使用该方法，展示了如何实现自动移动玩家角色，自动查找怪物列表，自动战斗，自动捡物和自动选择需要攻击的怪物等功能。在作为调试器存在的外挂小节中，介绍了调试器基础，调试事件，调试循环；介绍了调试进程退出和调试权限；介绍了如何设置，使用和清除断点；最终介绍了如何从运行进程中抓取有用信息。然后介绍了一个可以用于多个外挂开发的平台：Wowzer外挂引擎。最后一节描述了外挂制作高级技巧，主要介绍了外挂技术中的三个技巧：抗检测的内核外挂，宏外挂和外挂用户界面。

7.1 外挂制作基础

现在在第6章的工具和技术基础上，进行外挂制作讲解。需要注意的是，外挂制作是很复杂

的事情，需要不少技术基础。

7.1.1 事件驱动设计

外挂制作的第一个关键概念涉及一个常用设计模式，事件循环。回顾下第2章介绍的简单外挂。程序逻辑结构很清晰：外挂循环等待事件发生，对出现的事件分别进行处理。所有外挂程序结构都差不多如此：等待事件发生（游戏客户端内部事件，网络事件，或者某个内存位置的值符合预定条件等），然后响应产生的事件。这就称为事件驱动设计。

7.1.2 状态机

本书其他地方也提到状态机的概念，这里需要指出的是：外挂程序不仅时刻关注游戏状态，也需要关注自身的状态。外挂的主要任务是监视游戏客户端状态，特别是监视与游戏角色相关的状态信息，从玩家的角度来考虑问题。前面也提到，外挂需要了解玩家状态如血量（health status）、位置（position）等等。监视血量可以通过第2章介绍的屏幕取色的方法，或者可以通过内存搜索监控等来获取血量。无论采用什么方法，外挂与游戏客户端之间通信应当使用某种状态机模型表示。

不管是有意还是无意，外挂多数设计为状态机形式。外挂代替玩家控制游戏角色，而游戏角色永远处于某些状态如：跑动、打坐、战斗和治疗等。状态之间转换的规则由外挂制定，比如：外挂可以制定状态转化规则如下，“战斗中”如果“血量小于正常值的25%”则“逃离战斗场景”。每个状态有特定的转换规则，这些转换规则分别由事件和游戏状态驱动。熟悉人工智能的读者应该会联想到人工智能学中的产生式系统（production system）。很显然，要驱动该系统运行，需要使用第6章中介绍的技巧，首先获取玩家的血量、魔法值（spell-casting capability 或者mana）、战斗对象坐标等。获取玩家及游戏状态后，外挂控制的玩家角色可以根据第6章介绍的技巧和状态机转换规则来自动完成移动、战斗、释放技能等操作。

图7-1是一个外挂的状态机示意图。这个状态机的功能是在WoW中实现玩家的自动挂机。如图所示，该外挂开始的时候会去从客户端内存中读取玩家结构信息。然后初始化玩家结构指针，通过设置断点的方式（见第6章）获取怪物列表。然后，计算出玩家角色血量剩余白分比，来决定是否进行战斗。WoW有个设定，会设置一个临界点（其实是一点很小的血量，比如为1）让角色的血量到达此点的时候便进入“虚无状态”。达到这个状态后，角色会被传送到复活点（被杀后，灵魂被传送到特定地方）。在该状态机的更底层是查找怪物列表，按照怪物列表来杀怪。从图7-2可以看到，如果发现目标怪物，而且玩家有足够的血量继续战斗，外挂会试图拉近怪物和玩家角色的距离。此处可以应用多种策略，比如始终维持一个固定的距离，在该距离下，玩家角色可以释放技能伤害怪物，但是该距离又足够长，使得怪物不能近身战斗。魔兽世界中一个很有名的利用坐标实现外挂功能的技巧叫做Z-hacking，该技巧直接修改玩家Z轴坐标，使玩家漂浮在怪物上方死角，从而使怪物始终无法攻击到玩家角色。

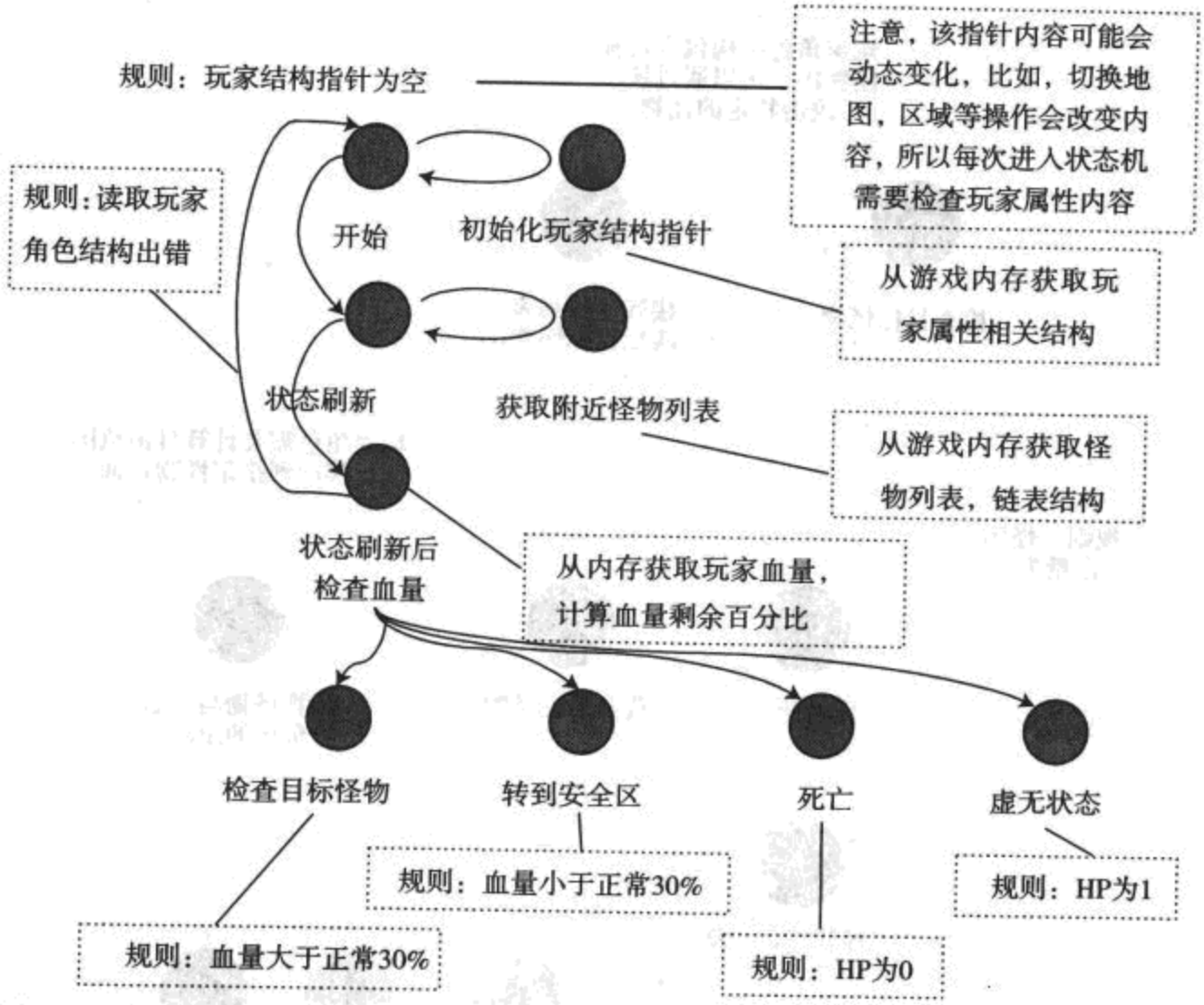


图 7-1　外挂状态机可以被视为连通图。灰色的圈代表状态，箭头显示可能的状态转换

图 7-2 中，外挂移动玩家角色，调整玩家角色和怪物的距离，打怪策略为 30 码内使用技能打击怪物，5 码内与怪物肉搏。如果玩家角色与怪物的距离太远，则向怪物方向移动玩家角色。大多数游戏中，一旦怪物受到攻击，怪物将向玩家移动，所以玩家角色不需要修改位置就可以继续战斗（非常方便）。当然，如果外挂在玩家角色附近区域中无法发现怪物，则外挂需要随机移动玩家角色来发现怪物。

7.1.3　移动玩家角色

如上所述，玩家角色和怪物距离较远时，需要移动玩家角色靠近怪物。有如下方法可以移动玩家角色：

- 模拟按键移动玩家角色
- 挂钩主线程中处理玩家移动的过程，然后调用该过程来移动玩家角色
- 模拟网络封包，告诉服务器玩家移动的位置
- 直接在内存设置玩家角色的位置坐标（如果移动的距离较远，这种方法也叫瞬移）

最后一种方法是最直观的方法。通过这种方法设置坐标，如果玩家角色的坐标位置移动不是很远，根本察觉不到玩家作弊。

瞬移指用设置坐标的方法，直接移动玩家角色到相对于当前坐标很远的位置的方法。这个概念区别于使用设置坐标的方法移动很小的范围的移动。

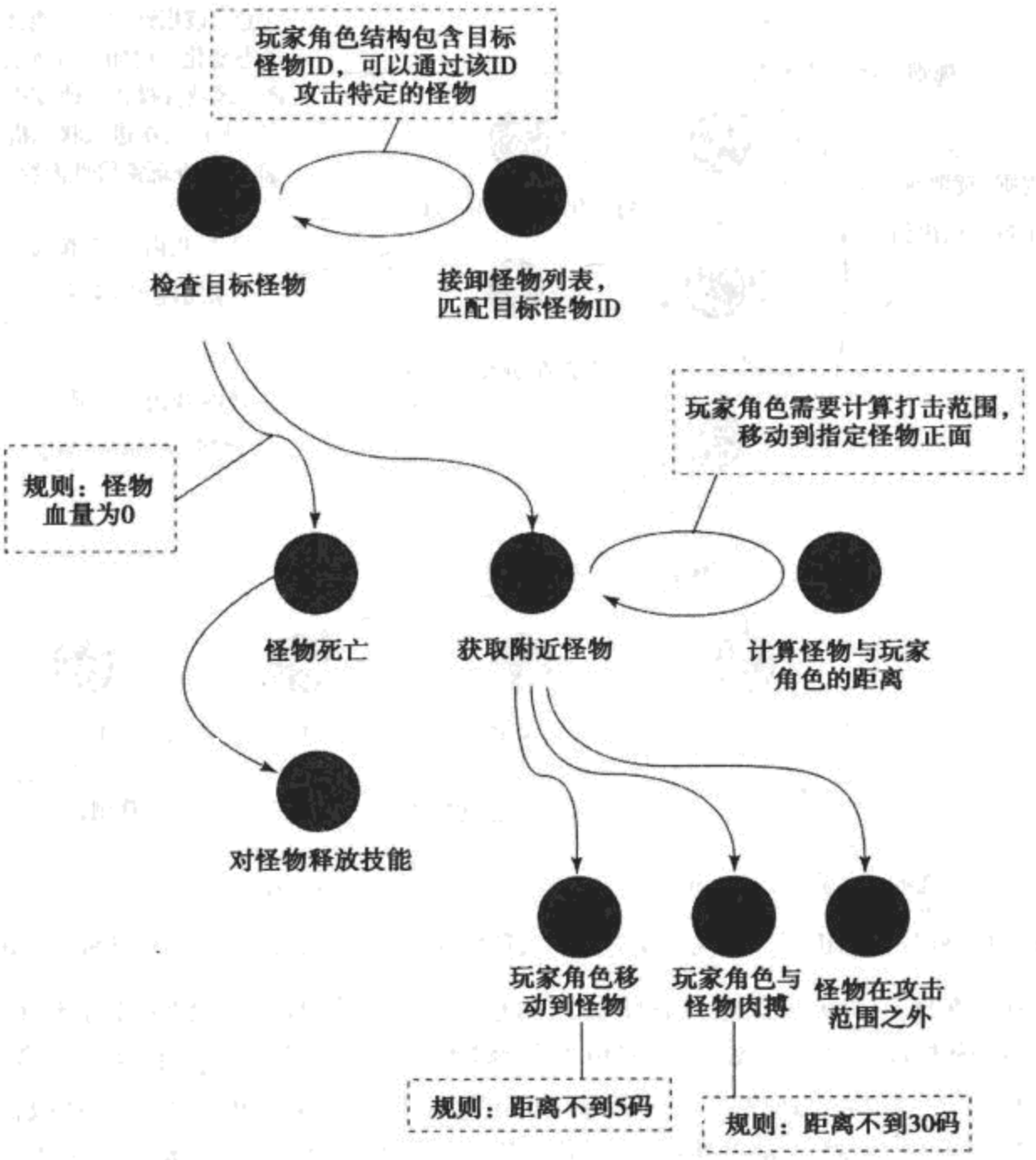


图 7-2 移动到目标怪物的状态机

下面具体描述本概念：以下代码片段检查玩家角色到怪物的距离，模拟按键一段时间，使游戏中的玩家角色移动。

首先：通过从游戏客户端读取当前怪物数据，刷新怪物列表结构。

```
assert(m_target_mob);
m_target_mob->Refresh();
```

调用 SetSelfToFace()设置玩家角色位置为后面的 m_x,m_y,m_z 坐标，同时方向直接面对怪物。(请查看计算方向小节查看更多细节。)

```
SetSelfToFace(m_target_mob->m_x,
              m_target_mob->m_y,
              m_target_mob->m_z);
```

计算玩家角色和目标怪物的距离。(请查看下一个小节的详细介绍。)

```
float range = CalculateRangeFromSelf(
            m_target_mob->m_x,
            m_target_mob->m_y,
            m_target_mob->m_z);

logprintf("mob range is %f units from self", range);
```

最后，模拟按键，发送向上的方向键，持续时间为根据距离计算的时间。PostKeystroke()方法在第6章定义。

```
// calc time for key press based on distance
// ---------------------------------------
// try to move the PC only close enough to begin
// ranged attacks

// TODO: make the calculation account for user-supplied
// min range value
DWORD time_to_press = range * 50; //TODO, make configurable

PostKeystroke(VK_UP, time_to_press);
```

距离计算

玩家角色和目标怪物的距离对战斗非常重要。距离不仅影响图7-1和图7-2的状态图中的转换规则，而且被用来计算模拟按键的时间长短。可以通过两点之间的x，y，z坐标来计算两个对象之间的距离。

下面的函数计算玩家角色和目标怪物坐标之间的距离：

```
float CalculateRangeFromSelf(float mobX, float mobY, float mobZ)
{
```

aCharcterMob对象代表玩家角色数据结构。本例中，看起来术语 *mob* 被重载了，玩家和怪物用的一个名字，不过从实现角度来看，玩家角色和其他怪物确实基本一样。

```
Mob *aCharacterMob = GetSelfMob();
if(!aCharacterMob)
{
  logprintf("warning, could not get self mob");
  return 0;
}
```

此处的Refresh()函数从游戏客户端进程读取当前游戏中玩家角色数据到外挂内存中。这样就可以获取最新的玩家角色坐标。

```
aCharacterMob->Refresh();
```

距离计算返回值为浮点数。Sqrt()函数计算平方根，pow()函数计算平方，这里用的公式为几何计算中最简单的两点间的距离公式。

```
//calculate range
float range = sqrt( (float)(
      pow(aCharacterMob->m_x - mobX, 2)
      +
      pow(aCharacterMob->m_y - mobY, 2)
      +
```

```
            pow(aCharacterMob->m_z - mobZ, 2)
        ));

        return range;
}
```

方向计算

很多游戏中，玩家角色攻击怪物时，需要面对目标怪物攻击才行。魔兽世界中，对象所朝向的方向保存在客户端内存中。通过读写该变量，可以感知或者修改对象面对的方向。与瞬移技巧修改 x，y，z 坐标一样，方向也是可以直接修改，并且在屏幕显示出修改效果。（详见第 6 章）

下列代码片段计算所需的方向，并修正玩家角色的方向为该方向。

```
void SetSelfToFace( float mobX, float mobY, float mobZ )
{
```

下面的调用获取玩家角色数据结构。

```
Mob *aMob = GetSelfMob();
if(!aMob)
{
  logprintf("warning, could not get self mob");
  return;
}
```

本地玩家信息结构更新：从 WoW. exe 内存空间中读取数据信息，并进行本地化拷贝。

```
aMob->Refresh();
```

获取玩家角色的 x，y 坐标后，根据怪物 x，y 坐标计算面对怪物的方向。它能使角色方向改变为朝向目标方向。

```
float g = 0;
float RealX = aMob->m_x - mobX;
float RealY = aMob->m_y - mobY;

if(RealX < 0)
{
  RealX = RealX * -1;
}
if(RealY < 0)
{
  RealY = RealY * -1;
}
```

下一部分使用 RTOD 宏转换刚刚计算的方向为度数单位。

```
#define M_PI 3.14159265358979
#define RTOD(r) ((r) * 180 / M_PI)
```

值 g 的单位为度数，代表了玩家角色需要面对怪物的方向。

```
if((mobX > aMob->m_x) && (mobY > aMob->m_y))
{
  g = RTOD( atan(RealY/RealX) );
}
```

```
if((mobX > aMob->m_x) && (mobY < aMob->m_y))
{
  g = (( RTOD(atan( RealY / RealX )) * -1 ) + 90) + 270;
}
if((mobX < aMob->m_x) && (mobY < aMob->m_y))
{
  g = RTOD(atan( RealY/RealX )) + 180;
}
if((mobX < aMob->m_x) && (mobY > aMob->m_y))
{
  g = (( RTOD(atan( RealY/RealX )) * -1 ) + 90) + 90;
}
```

再把度数转化为弧度，保存到玩家角色方向字段。

```
        float to_rotate = ((2*3.14159265358979)/360)*g;

        //logprintf("to_rotate: %f\n", to_rotate);

        SetSelfRotation(to_rotate);
}

void SetSelfRotation(float to_face)
{
        Mob *aMob = GetSelfMob();
        if(aMob)
        {
          aMob->SetFacing(to_face);
        }
        else
        {
          logprintf("warning, could not get self mob");
        }
}
```

最后，一段小程序用于修改游戏客户端进程内存，修改方向变量。该函数为 Mob 类的成员函数：

```
void Mob::SetFacing(float to_face)
{
        SIZE_T bread;
```

m_baseaddress 为魔兽世界角色结构在客户端进程中的地址。PC_FACING_OFFSET 为方向变量的偏移，该变量为一个 32 位的浮点数。

```
        //TODO read from config file
        if(WriteProcessMemory( m_parent_engine->m_prochandle,
                (void *)(m_baseaddress + PC_FACING_OFFSET),
                &to_face,
                4,
                &bread) == FALSE)
        {
          logprintf("error writing to facing");
          return;
        }
}
```

本节描述了最基本的移动操作。设置玩家方向的操作很重要，比如，可以使用该技巧瞬移到对手后面进行攻击，特别是玩家之间（player versus player，PvP）PK 时，瞬移到对手后面需要

转身才能攻击到对手。由于PvP战斗受方向的影响很严重，理论上，可以开发一个PK助手，可以把你转移到对手背后，再移动方向面对对手，这样就可以实现背后攻击了。

瞬移

现在，读者已经了解到如何计算玩家角色和怪物之间的方向，并通过手工模拟按键移动玩家角色走向怪物。魔兽世界以及很多其他游戏架构信任客户端上报的坐标位置，并不计算移动是否合法。也就是说，服务器并没有检测玩家是不是通过按键来达到移动效果的。所以，一旦修改玩家角色位置信息，服务器认为玩家是正常移动到该位置的。

前面已经讲到，该技巧如果移动玩家角色到一个很远的位置，就被称作瞬移。所以，如果玩家不喜欢慢慢移动到怪物或者对手面前，可以通过这种方法瞬移到怪物或对手面前进行攻击。有了这个概念后，可以实现很多很多技巧性的应用，比如，穿墙、移动到不可到达的位置、抢怪、抢物品等。

下面的代码从内存里修改游戏中角色的位置信息，使角色瞬移到目标地点：

```
void Mob::SetXYZ(float x, float y, float z)
{
        SetX(x);
        SetY(y);
        SetZ(z);
}
```

一切都很简单，使用Mob类中的成员函数写远端进程内存即可：

```
void  Mob::SetX(float x)
{
        SIZE_T bread;

        // Write new X
        if(WriteProcessMemory( m_parent_engine->m_prochandle,
                (void *)(m_baseaddress + PC_X_OFFSET),
                &x,
                4,
                &bread) == FALSE)
        {
          logprintf("error writing to X");
          return;
        }
```

下面的代码片段是外挂中注释掉的部分，外挂制作者发现游戏客户端中有两份玩家坐标信息，有时，这种信息可能会带来校验的麻烦，所以，作者同时修改了备份的坐标信息。但是看起来，备份的坐标信息没有用到，所以被注释掉了，但是为了完整性，本处还保留了该信息。

```
#if 0
        // Write new X to backup
        if(WriteProcessMemory( m_parent_engine->m_prochandle,
                (void *)(m_baseaddress + PC_BK_X_OFFSET),
                &x,
                4,
                &bread) == FALSE)
        {
```

```
            logprintf("error writing to backup X");
            return;
        }
#endif

}
```

下面的代码修改游戏客户端中的 Y 坐标。

```
void  Mob::SetY(float y)
{
        SIZE_T bread;

        // Write new Y
        if(WriteProcessMemory( m_parent_engine->m_prochandle,
                (void *)(m_baseaddress + PC_Y_OFFSET),
                &y,
                4,
                &bread) == FALSE)
        {
          logprintf("error writing to Y");
          return;
        }
```

修改备份信息：

```
#if 0
        // Write new Y to backup
        if(WriteProcessMemory( m_parent_engine->m_prochandle,
                (void *)(m_baseaddress + PC_BK_Y_OFFSET),
                &y,
                4,
                &bread) == FALSE)
        {
          logprintf("error writing to backup Y");
          return;
        }
#endif
}

void  Mob::SetZ(float z)
{
        SIZE_T bread;
        // Write new Z
        if(WriteProcessMemory( m_parent_engine->m_prochandle,
                (void *)(m_baseaddress + PC_Z_OFFSET),
                &z,
                4,
                &bread) == FALSE)
        {
          logprintf("error writing to z");
          return;
        }
#if 0
        // Write new Z to backup
        if(WriteProcessMemory( m_parent_engine->m_prochandle,
                (void *)(m_baseaddress + PC_BK_Z_OFFSET),
                &z,
                4,
                &bread) == FALSE)
        {
          logprintf("error writing to backup Z");
          return;
        }
#endif
}
```

瞬移是非常强大的技巧，可以使用该技巧实现非常多的应用，比如可以查看下面介绍的快速位置切换技巧。一个比较有趣的应用是如影随行技巧，玩家角色始终和怪物或对手保持一个固定的距离。这个技巧特别在 PvP 模式中有效，可以始终在对手的背后，对手永远攻击不了使用这种技巧的角色，只有被攻击。很显然，魔兽中，位置在战斗中很重要，如果玩家角色永远不能被攻击，肯定位于不败之地。

快速位置切换（Ping-Ponging）

对于具有远程攻击技能的玩家角色（回忆一下前面讲到的几种类型的玩家角色，各有各的特色），位置快速切换技巧非常有用，玩家角色远程攻击怪物后，使用位置定位，变换到另外一个怪物攻击不到的地方，继续使用远程攻击打击怪物。这种技巧严重依赖瞬移技巧。

7.1.4 控制玩家角色战斗

将玩家角色移动到目标对象的某个范围内，外挂根据不同的策略和状态机模型判断下一步该如何操作，比如使用技能战斗等。本节介绍外挂原型代码中战斗相关部分，展示了如何控制玩家角色自动进行战斗。

首先刷新目标对象信息：

```
assert(m_target_mob);
m_target_mob->Refresh();
bool slapin_extra_range_attack = FALSE;
```

然后根据状态信息判断是否需要贴身战斗还是使用远程攻击技能。根据这些信息就可判断是否需要使用瞬移等技巧。

```
if( (FALSE==m_melee_class) && (TRUE==m_telestick) )
{
```

如果要使用远程攻击技能，可以使用位置快速切换技巧保持玩家角色和怪物的距离，持续不断的进行打击，这种情况下，没有远程攻击技能的怪物将永远处于无还手之力的状态。

```
if( m_pingpong_target &&
       (m_pingpong_target == m_target_mob->m_id))
{
  if(m_pingyes)
  {
    //goto pong
    Mob *aSelfMob = GetSelfMob();
    aSelfMob->SetXYZ( m_pong_x, m_pong_y, m_pong_z );
    m_pingyes = FALSE;
    slapin_extra_range_attack = TRUE;
  }
  else
  {
    //goto ping
    Mob *aSelfMob = GetSelfMob();
          aSelfMob->SetXYZ( m_ping_x, m_ping_y, m_ping_z );
          m_pingyes = TRUE;
          slapin_extra_range_attack = TRUE;
        }
      }
}
```

移动到新的位置后，设置玩家角色面对目标对象：

```
SetSelfToFace( m_target_mob->m_x,
               m_target_mob->m_y,
               m_target_mob->m_z);
```

仅设置位置信息是不够的，实际上，需要稍微走几步，通知服务器玩家角色的新位置，因为通过写内存设置的坐标通常都不能马上获得实际效果，需要稍微移动位置才能让游戏客户端感知到玩家移动的新位置。可以通过键盘模拟来达到效果：

```
// back up a tap
PostKeystroke(VK_DOWN, 120);

float range = CalculateRangeFromSelf(
               m_target_mob->m_x,
               m_target_mob->m_y,
               m_target_mob->m_z);

logprintf("mob range is %f units from self", range);
```

对怪物使用如影随行技巧时，有时会碰到这样的情况：移动位置和怪物靠的太近，移动后怪物在玩家角色背后，但是转身此时无效，因为后面的位置被挡住了。这种情况下，可以模拟按键稍微移动一点位置：

```
// this tick back is needed so we don't move past the target
if( abs(range) < 3.2)
{
      PostKeystroke(VK_DOWN, 120); //TODO, make configurable?
}
```

对目标对象使用位置快速切换技巧的话，最好使用游戏定义的快捷键释放远程攻击技能。

```
if(slapin_extra_range_attack)
{
      PostKeystroke(0x32, 120);
      PostPause(m_cast_time);
}
else
{
```

上面的代码都假定玩家角色处于战斗状态，实际玩家战斗状态为可以切换的状态，需要战斗时必须保证玩家处于战斗状态，下面的代码检查玩家是否处于战斗状态：

```
      // the melee attack is a toggle,
      // so don't apply it unless it's not already on
      if(FALSE == inAttackMode)
      {
        //perform melee attack, slot 1
        PostKeystroke(0x31, 10);
      }

      // tap for a hero strike in slot 3
      PostKeystroke(0x33, 10);
}
```

除了通过逆向工程获取战斗模式信息，还可以通过对屏幕取色来达到该效果。第 2 章和第 6 章均介绍了该技巧。

```
// return true if we are in attack mode
// TODO come up with an in-memory way to do this...

BOOL WowzerEngine::GetAttackMode()
{
```

坐标为某个图标的边框。

```
        COLORREF cr = GetColorOfPixel(26,731);
        //logprintf(" got %d,%d,%d",
                GetRValue(cr),
                GetGValue(cr),
                GetBValue(cr));
        //DWORD difference = abs(GetGValue(cr) - 138);
        if(GetGValue(cr)>100)
        {
          // we are in agro mode
          return true;
        }

        return false;
}
```

本节包含了许多外挂制作的关键概念：计算玩家角色和怪物的方向和距离、构建实施位置快速变换、通过模拟按键控制玩家角色战斗。这些代码可以扩展到很多其他操作上。

7.1.5 自动拾取

大多数游戏中，怪物死亡后，会随机掉落物品。正常游戏情况下，可以通过鼠标点击等操作捡物。在外挂中，可以通过全屏点击可能的掉落位置，模拟人物鼠标操作来捡物，这样就可以实现自动捡物功能。

```
void HunterBot::TeleportAndLootMob(Mob *theMob)
{
        TeleportToTarget(theMob->m_id);
        LootTheDead();
}

void HunterBot::LootTheDead()
{
        logprintf("LOOT THE DEAD");

        RMouseClick(440,390, TRUE, 120);
        RMouseClick(440,443, TRUE, 120);
        RMouseClick(440,500, TRUE, 120);
        RMouseClick(440,550, TRUE, 120);
        RMouseClick(440,600, TRUE, 120);
        RMouseClick(440,680, TRUE, 120);
        RMouseClick(513,390, TRUE, 120);
        RMouseClick(513,443, TRUE, 120);
        RMouseClick(513,500, TRUE, 120);
        RMouseClick(513,550, TRUE, 120);
        RMouseClick(513,600, TRUE, 120);
        RMouseClick(513,680, TRUE, 120);
        RMouseClick(580,390, TRUE, 120);
        RMouseClick(580,443, TRUE, 120);
        RMouseClick(580,500, TRUE, 120);
```

```
        RMouseClick(580,550, TRUE, 120);
        RMouseClick(580,600, TRUE, 120);
        RMouseClick(580,680, TRUE, 120);
}
```

这种方法实现很简陋，但不可否认的是，该方法很有效，至少对魔兽世界很有效。⊖

7.1.6 怪物选择及过滤

在玩家角色战斗过程中，常常会碰到不能被杀死的怪物［译者注：太强大，或者是任务怪物等］，或者是一些未知数据结构。在这种情况下，可以把这类怪物放到黑名单中，不攻击该类怪物。

在杀怪状态机中，可以判断是否玩家角色持续攻击同一个怪物，如果经过100次状态机，还没有杀死怪物，即将怪物ID放到黑名单，下次再碰到该怪物就不攻击。［译者注：此处策略很多，比如升级后就解除黑名单等］。

```
if(m_last_target_mob_id == m_target_mob->m_id)
{
        logprintf("grind counter at %d",
              m_number_of_kill_attempts);
        if(++m_number_of_kill_attempts > 100)
        {
          logprintf(" we have been grinding on this mob for a
while...");

          logprintf("blacklisting mob id %d", m_target_mob->m_id);

          AddMobToBlacklist(m_target_mob->m_id);
          m_target_mob = 0;
          m_last_target_mob_id = 0;
          m_number_of_kill_attempts=0;

          // nothing, find one in the database
          SetCurrentState(SH_SELECT_Mob);
          break;
        }
}
```

黑名单中为怪物ID列表，在怪物选择阶段，自动过滤该列表中的怪物。下列代码自动选择最近的怪物并开始攻击。

```
Mob *aSelfMob = GetSelfMob();
if(aSelfMob)
{
  aSelfMob->Refresh();
```

GetNearestMob()函数从WoW.exe内存中获取怪物列表，再从怪物列表数据库选择距离最近的怪物：

```
// get the nearest, living mob
Mob *aTargetMob = GetNearestMob(MobTYPE_NPC);
if(aTargetMob)
{
  m_target_mob = aTargetMob;
```

⊖ 译者注：可以通过判断掉落物品的位置，直接模拟点击；或者可以发送数据包告诉服务器物品捡起了，不过后者可能会使屏幕显示效果不同步或者无显示；也可以找到捡物的代码，修改判断流程实现捡物。

如果怪物和上次攻击的不一样，则重置杀怪计数器，否则，保持该计数器，用于判断是否该怪是杀不动的怪物。

```
    if(m_target_mob->m_id != m_last_target_mob_id)
    {
      logprintf("detected a new target mob, id %d",
            m_target_mob->m_id);
      m_last_target_mob_id = m_target_mob->m_id;
      m_number_of_kill_attempts=0;
    }
    else
    {
      logprintf("target mob remains the same");
    }

  // TODO, now attempt to TAB set the
  // given mob

  SetCurrentState(SH_MOVETO_Mob);
  }
  else
  {
    m_target_mob = 0;
    SetCurrentState(SH_NO_Mob);
  }
}
```

怪物列表本身为链接结构，结构首地址可以通过 WoW. exe 的全局指针获取。下面代码显示了如何获取怪物列表。注意此处最开始的地址为一个硬编码的地址，会随着版本的不同而不同。

```
DWORD WowzerEngine::ReloadMobList()
{
      // empty database (todo: maybe not empty, so we
      // can track ppl who move in and out of area)
      //m_pc_database.clear();
```

该地址为一个全局指针，可以使用第 8 章介绍的技巧（比如反编译器）动态获取该地址。

```
// contains a ptr to mobTable
DWORD mobTablePtrAddr = 0x00A41b0C;
DWORD mobTableFirst = 0;
DWORD mobTablePtr = 0;
DWORD bread = 0;

CHAR name [48];

if( ReadProcessMemory(
      m_prochandle,
      (void *)mobTablePtrAddr,
      &mobTablePtr,
      4,
      &bread) == FALSE)
{
  logprintf("failed to read address mobTablePtrAddr");
  return 0;
}
logprintf("mobTablePtrAddr: 0x%08X\n", mobTablePtrAddr);

mobTableFirst = mobTablePtr;
mob_list_struct mob;
```

代码循环遍历链表结构，直到返回链表头。

```
        do {

              if(ReadProcessMemory(
                     m_prochandle,
                     (LPCVOID)mobTablePtr,
                     &mob,
                     sizeof(mob_list_struct),
                     &bread) == FALSE)
              {
                logprintf("failed to read address mobTablePtr:
%08.08x", mobTablePtr);
                break;
              }

              memset (name, 0x00, sizeof(name));

              if(ReadProcessMemory(
                     m_prochandle,
                     (LPCVOID)mob.name1,
                     &name, sizeof(name), &bread) == FALSE)
              {
                logprintf("failed to read address mob name:
%08.08x", mob.name1);
              }

              logprintf( "%08.08x %08.08x unk1:%08.08x
unk2:%08.08x unk4[0:%d 1:%d 2:%d 3:%d 4:%d 5:%d] %08.08x:%s",
                           mobTablePtr, mob.next,
                           mob.unk1, mob.unk2,
                           mob.unk4[0],mob.unk4[1],mob.unk4[2],
                           mob.unk4[3],mob.unk4[4],mob.unk4[5],
                           mob.id, name);

              mobTablePtr = (DWORD) mob.next;

        } while ((mobTablePtr != 0)
                 &&
                 (mobTablePtr != 0x00a41b09)
                 &&
                 (mobTablePtr != mobTableFirst));


        return 0;
}
```

怪物列表的刷新时间比较固定，通过如上方法可以稳定获取到最新的怪物类型及位置等信息。注意，怪物列表不包括玩家角色对手。

7.1.7 “引怪”的模式管理

在打怪过程中，其他怪物可能会攻击玩家角色。所以，在整个战斗过程中，需要关注玩家是否受到攻击，何时受到攻击等信息。另外即使外挂没有自动攻击怪物，也需要感知玩家角色是否受到攻击。下面代码检查是否战斗中出现其他怪物攻击。

```
if(FALSE == amUnderAgro)
{
  if(TRUE == m_loot)
  {
    SetCurrentState(SH_LOOT_Mob);
  }
  else
  {
    SetCurrentState(SH_HEALUP);
  }
}
else
{
  // we are still under agro, try to target the bugger
  PostKeystroke(VK_TAB, 40);
  Sleep(400);
  SetCurrentState(SH_CHECK_TARGET_Mob);
}
```

如果玩家角色处于引怪模式（吸引怪物攻击），则直接攻击怪物，否则，先捡物品。为了检查玩家是否处于引怪模式，可以通过屏幕取色来检查。

```
// return true if we are being agro'd
// TODO come up with an in-memory way to do this...
BOOL WowzerEngine::AmUnderAgro()
{
        COLORREF cr = GetColorOfPixel(30,65); //the sword's X

        //logprintf(" got %d,%d,%d",
        //      GetRValue(cr),
        //      GetGValue(cr),
        //      GetBValue(cr));
        DWORD difference = abs(147 - GetRValue(cr));
        if(difference<20)
        {
          // we are being agro'd
          return true;
        }

        return false;
}
```

如同第2章解释的那样，很多状态可以通过屏幕取色获取。开发外挂过程中，屏幕取色的方法是最容易，消耗时间最短的方法。该方法不需要逆向工程，不需要定位游戏内存信息。

但是使用屏幕取色的方法，常把用户界面限制在某个分辨率和颜色质量下，而且窗口位置常不能移动，用户界面很不友好。

使用进程内部状态变量直接控制状态信息是更优雅的实现方法，但是外挂制作本身就是在和游戏更新赛跑。使用屏幕取色完成一个外挂成品比一个永远无法执行的半成品外挂要好很多。所有软件开发中都有类似的折中方案。如果开发者希望把外挂给其他人使用，用内部状态变量可以使其他人在需要更少手工配置情况下使用外挂。比如，解压出来就可以使用等。

7.2 外挂作为调试器

细心的读者，特别是对第6章读得很仔细的读者可能已经发现了外挂和调试器的关系。如果将外挂作为调试器编写将会带来很多好处。本节中介绍如何将外挂附加到运行进程，利用调试

事件实现外挂功能。

外挂作为调试器存在将大大加强外挂操纵目标程序的能力。另外，好的调试器程序对理解和操纵游戏客户端很有帮助。非常幸运的是：Windows 本身提供的调试器功能都共享相同的基本框架。本节介绍基本的调试技巧，并使用 Wowzer 程序来实践这些游戏破解技巧。

请注意，不少游戏客户端会检测客户端自己是不是被调试。对本节介绍技巧使用不当的话可能会导致玩家账号被封停。如果账号被封的话，请不要向作者抱怨。

7.2.1 调试循环

下面代码为标准调试器的调试主循环。代码为控制台程序，没有用户界面，调试数据输出到标准输出 stdout。

```
int _tmain(int argc, _TCHAR* argv[])
{
        HANDLE hProcess;
        DEBUG_EVENT dbg_evt;
        int aPID;

        if(argc != 2)
        {
          printf("wrong number of parameters\nusage %s
<pid>\n",argv[0]);
          return 0;
        }
```

上述代码中检查程序参数。本例中，调试器将连接到（附加到）一个已经运行的进程。游戏分析中这是比较好的常用方法。在游戏客户端启动过程中，客户端常常会检查自己是否是从调试器启动的，如果是从调试器启动，则执行退出自身等操作。附加到一个已经运行的进程可以避免启动检查的步骤（注意，检查自身是否被调试的代码可能在很多地方都存在，不一定仅在启动期检测）。采用附加的方法还可以让游戏先登录再附加。

```
        // load the ptr to fDebugSetProcessKillOnExit
        fDebugSetProcessKillOnExit = (DEBUGSETPROCESSKILLONEXIT)
              GetProcAddress( GetModuleHandle("kernel32.dll"),
              "DebugSetProcessKillOnExit" );
        if(!fDebugSetProcessKillOnExit)
        {
          printf("[!] failed to get fDebugSetProcessKillOnExit
function!\n");
        }

        aPID = atoi(argv[1]);
```

上面代码动态获取函数 DebugSetProcessKillOnExit 的地址，该函数设置调试线程离开进程时的行为，比如可以在不关闭被调试进程的情况下离开被调试进程，XP 以后的系统支持该函数，下面小节专门介绍该函数。

```
hProcess = OpenProcess(PROCESS_ALL_ACCESS |
        PROCESS_VM_OPERATION,
        0,
        aPID);
```

通过 OpenProcess 打开目标进程，本情况下，目标进程需要已经加载运行。可以在游戏登录完成后附加到进程，从而实现分析和挂机。

```
if(hProcess == NULL)
{
  printf("[!] OpenProcess Failed !\n");
  return 0;
}

SetDebugPrivilege(hProcess);

// Alright -- time to start debugging
if(!DebugActiveProcess(aPID))
{
  printf("[!] DebugActiveProcess failed !\n");
  return 0;
}
```

上述调用后，将开始调试进程，如果该进程不是你创建的，需要给调试器本身赋予调试权限。调试事件处理循环如下：

```
// don't kill the process on thread exit, XP and above
if(fDebugSetProcessKillOnExit)
      fDebugSetProcessKillOnExit(FALSE);

while(1)
{
```

下一步等待调试事件。很多情况可以触发调试事件，如线程创建销毁，新模块加载等。另外有不少调试器相关事件，如执行到断点，单步操作等。函数 WaitForDebugEvent 可以用于设置调试事件超时时间，本代码设置超时时间为 INFINITE，必须等待某个调试事件出现，直到调试事件出现才执行接下来的代码。

```
if(WaitForDebugEvent(&dbg_evt, INFINITE))
{
  printf("debug event detected...\n");
```

现在已经到了检查调试事件的代码。如果事件代码为异常，检查是否为断点或者单步。断点常常是 Int 3，单步为 Int 1。需要注意的是，某些断点类型会发出 int 1，从而发出单步消息（即使它们是断点）。另外访问违规也是需要检查的异常类型，比如读写无效内存会发送该类型异常。

```
            if(EXIT_THREAD_DEBUG_EVENT == dbg_evt.dwDebugEventCode)
            {
              printf("[!] Target thread id %d
has finished executing. \n",
                    dbg_evt.dwThreadId);
            }
              if( dbg_evt.dwDebugEventCode == EXCEPTION_DEBUG_EVENT)
              {
                switch (dbg_evt.u.Exception.ExceptionRecord.
                        ExceptionCode)
                {
                  case EXCEPTION_ACCESS_VIOLATION:
                  break;
```

```
        case EXCEPTION_BREAKPOINT:
        printf("breakpoint hit\n");
        break;
        case EXCEPTION_SINGLE_STEP:
        break;
        default:
        break;
      }
    }
```

此处展示了一个有意思的技术，检查调试字符串。打印调试跟踪信息时，将会发送一个特别的事件码。调试信息本应当仅在开发过程中出现，但产品构建后开发人员常常忘记去除调试信息。这些字符串常包含游戏相关函数名及功能相关提示信息，对于逆向分析非常有用。

```
if(dbg_evt.dwDebugEventCode ==
        OUTPUT_DEBUG_STRING_EVENT)
{
  OUTPUT_DEBUG_STRING_INFO *inf =
         &(dbg_evt.u.DebugString);
  LPSTR remote_address =
         inf->lpDebugStringData;
  char _local[4096];
  printf("got debug string len %d\n",
         inf->nDebugStringLength);

  memset(_local, NULL, sizeof(_local));

  if(inf->nDebugStringLength <
         sizeof(_local)-1)
    {
      unsigned long num_read;
      ReadProcessMemory(
             hProcess,
             (LPCVOID)remote_address,
             _local,
             inf->nDebugStringLength, &num_read);
      printf("string: %s\n", _local);
    }
  }

  printf("continuing...\n");
  if(!ContinueDebugEvent(
         aPID,
         dbg_evt.dwThreadId,
         DBG_CONTINUE))
  {
    return 0;
  }
 }
}

  return 0;
}
```

现在了解了基本调试循环，可以使用该技巧附加到目标进程实现外挂功能。这是基于断点分析的基础，可以用来获取或设置线程上下文，劫持线程等。下面讲解部分重要的细节。

7.2.2 SetProcessKillOnExit

在 Windows XP 之前，附加到一个程序调试后，如果调试器退出，被调试进程也必须退出。Windows XP 之后，微软公司添加了一个新的功能，允许调试器与被调试进程分离时，被调试进程无需退出。

现在的问题是，该函数没有输出到开发环境，不能直接在函数中调用，如果要使用该函数，只有通过 GetProcAddress 函数动态获取。

下面代码展示了动态获取未导出函数的方法，该方法可以获取任何模块中导出的函数指针（需要猜测函数参数个数及类型信息）。

```
typedef
BOOL(__stdcall *DEBUGSETPROCESSKILLONEXIT)
(
  BOOL KillOnExit
);
DEBUGSETPROCESSKILLONEXIT fDebugSetProcessKillOnExit;
```

7.2.3 SetDebugPriviledge

微软操作系统的安全特性可以管理每个登录用户拥有的权限。正常进程本身没有调试其他进程的权限，调试器本身需要被赋予进程调试权限才能调试其他进程。对于游戏分析来说，可以通过 CreateProcess 创建进程，这样就无需申请调试权限。但是一个好的调试器需要调试系统进程或者其他已经运行进程，需要为自身设置调试权限。

下面代码给调试器赋予调试权限。

```
bool SetDebugPrivilege( HANDLE hProcess )
{
        LUID luid ;
        TOKEN_PRIVILEGES privs ;
        HANDLE hToken = NULL ;
        DWORD dwBufLen = 0 ;
        char buf[1024] ;

        ZeroMemory( &luid,sizeof(luid) ) ;

        if(! LookupPrivilegeValue( NULL, SE_DEBUG_NAME, &luid ))
          return false ;

        privs.PrivilegeCount = 1 ;
        privs.Privileges[0].Attributes = SE_PRIVILEGE_ENABLED ;
        memcpy( &privs.Privileges[0].Luid,
                &luid,
                sizeof(privs.Privileges[0].Luid ) ) ;


        if( ! OpenProcessToken( hProcess,
              TOKEN_ALL_ACCESS,&hToken))
          return false ;

        if( !AdjustTokenPrivileges(
              hToken,
              FALSE,
```

```
                &privs,
                sizeof(buf),
                (PTOKEN_PRIVILEGES)buf,
                &dwBufLen ) )
            return false ;

        return true ;
}
```

赋予调试权限后，调试器可以附加到服务进程等在系统启动完毕之前加载的组件，有些游戏可能有类似组件，在系统启动完成前加载部分模块。

7.2.4 断点

断点是基本调试技巧，通过断点可以让程序运行到预设好的位置中断执行。不管程序在执行什么操作，只要执行到设置的断点，就会停止执行，从而调试器程序获取被调试程序控制权，然后可以重定向控制流程，获取数据或者修改数据。断点功能相当于修理工的扳手，特别常用，也特别重要。

断点有两种类型，软件断点和硬件断点。软件断点需要修改代码段，内存断点或者说硬件断点使用 Intel 芯片的调试寄存器实现断点。下面详细讲解该技术。

软件断点需要覆盖进程代码段中需要设置断点的地方为断点指令。为便于使用，定义该指令长度为 1 字节，内容为 0xCC。如果该代码执行后，将会发出一个 Int 3 中断，WinWdows 内核将会接管该中断，发出调试事件。当然，也可以对 Int3 挂钩，这需要内核驱动来实现该功能（see our book Exploiting Software for more on that technique）。

```
void CreateBreakpoint()
{
        char a_bpx = '\xCC';
```

定义断点指令为字节 0xCC。然后检索目标进程内存，确认内存地址有效。因为需要写入 0xCC 字节到目标进程内存，如果内存无效将出现访问异常。

```
MEMORY_BASIC_INFORMATION mbi;
VirtualQueryEx(
        hProcess,
        (void *)(g_start_breakpoint),
        &mbi,
        sizeof(MEMORY_BASIC_INFORMATION));
if(VirtualProtectEx(
        hProcess,
        mbi.BaseAddress,
        mbi.RegionSize,
        PAGE_EXECUTE_READWRITE,
        &mbi.Protect ))
{
```

对目标进程代码内存写入一个字节，需要保存原始字节，从而可在去除断点时恢复该字节。

```
// now read the original byte
if(!ReadProcessMemory(
        hProcess,
        (void *)(g_start_breakpoint),
        &(g_start_orig_byte),
```

```
        1,
        NULL))
{
  MessageBox(
          NULL,
          "[!] Failed to read process memory ! \n",
          "oops",
          MB_OK);
  return;
}
```

写入断点：

```
          if(!WriteProcessMemory(
                hProcess,
                (void *)(g_start_breakpoint),
                &a_bpx,
                1,
                NULL))
          {
            char _c[255];
            sprintf(_c, "[!] Failed to write process memory, error
%d ! \n", GetLastError());
            MessageBox(NULL, _c, "oops", MB_OK);

            return;
          }
        }
}
```

现在，断点已经设置，不管代码如何运行，只要执行到该断点所在位置就会引发软件断点事件。去除断点仅需恢复原始代码就可以了。

```
void RemoveBreakpoint()
{
        MEMORY_BASIC_INFORMATION mbi;
        VirtualQueryEx(
                hProcess,
                (void *)(g_start_breakpoint),
                &mbi,
                sizeof(MEMORY_BASIC_INFORMATION));

        if(VirtualProtectEx(
                hProcess,
                mbi.BaseAddress,
                mbi.RegionSize,
                PAGE_EXECUTE_READWRITE,
                &mbi.Protect))
        {
          if(!WriteProcessMemory(
                  hProcess,
                  (void *)(g_start_breakpoint),
                  &g_start_orig_byte,
                  1,
                  NULL))
          {
            char _c[255];
            sprintf( _c,
                    "[!] Failed to write process memory, error
%d ! \n",
                    GetLastError());
```

```
            MessageBox(NULL, _c, "oops", MB_OK);
            return;
          }
        }
}
```

断点命中以后，可以修改控制流程，执行其他代码，可以抓取内存信息，也可以修改内存内容，比如函数参数。不管如何操作，记得在断点命中后，去除断点，恢复执行，否则该断点将会一直命中，程序无法正常执行。

```
if(dbg_evt.dwDebugEventCode == EXCEPTION_DEBUG_EVENT)
{
```

执行异常码分发：

```
switch (dbg_evt.u.Exception.ExceptionRecord.ExceptionCode)
{
case EXCEPTION_ACCESS_VIOLATION:
       break;
```

下面的异常码用于处理 CC 软件断点：

```
case EXCEPTION_BREAKPOINT:
       if(g_initial_break)
       {
         g_initial_break = FALSE;
         printf( "initial breakpoint, debugging started...\n");
         if(g_start_breakpoint) CreateBreakpoint();
       }
       else if(g_start_breakpoint)
       {
```

软件断点命中后，需要去除字节 0xCC，替换回原始字节。另外需要将指令指针寄存器减一，从而恢复正常执行流程。

```
printf("user supplied breakpoint hit, removing...\n");
// breakpoint fired, now remove it
// roll back the EIP and correct the opcode

CONTEXT ctx;

HANDLE hThread =
       fOpenThread(
       THREAD_ALL_ACCESS,
       FALSE,
       dbg_evt.dwThreadId );

if(hThread == NULL)
{
  printf("[!] OpenThread failed ! \n");
  return 0; }
```

使用线程上下文来对指令指针寄存器（EIP）操作。

```
       // rewind one instruction
       ctx.ContextFlags = CONTEXT_FULL;
       if(!GetThreadContext(hThread, &ctx))
       {
         printf("[!] GetThreadContext failed ! \n");
```

```
            return 0;
          }

          ctx.Eip--;
          ctx.ContextFlags = CONTEXT_FULL;
          if(!SetThreadContext(hThread, &ctx))
          {
            printf("[!] SetThreadContext failed ! \n");
            return 0;
          }

          RemoveBreakpoint();
          g_start_breakpoint=0;

        }
break;
case EXCEPTION_SINGLE_STEP:
        break;
default:
        break;
}
```

上面的代码中实现了设置软件断点以及断点命中后去除断点的操作。

7.2.5 从上下文获取信息

下面代码展示了如何从上下文获取寄存器 EAX 的值。当然，也可以获取任何其他寄存器的值，使用某个值作为指针，可从内存获取更多信息，比如动态分配的结构，或者可以运行时实时修改某个值。

```
void take_sample(CONTEXT ctx)
{
        DWORD reg;
        struct hit *h = new struct hit;
        SYSTEMTIME thetime;
        GetSystemTime( &thetime);
        char addr[32];

        // build report item
        reg = ctx.Eax;
        h->mReport += "EAX: ";
        _snprintf(addr, 30, "v:0x%lx/%d", ctx.Eax, ctx.Eax);
        h->mReport += addr;
        if(can_read( (void *)reg ))
        {
          SIZE_T lpRead;
          char string[32];
          string[31]=NULL;

          // read the target memory
          if(ReadProcessMemory(
                hProcess,
                (void *)reg,
                string,
                30,
                &lpRead))
          {
            h->mReport += " -> ";
```

```
            h->mReport += string;
            h->mReport += "\r\n";
          }
          else
            h->mReport += "\r\n";
        }
        else
          h->mReport += "\r\n";
```

获取的数据保存在列表中，可以根据目的不同按照不同方式组织数据。

```
gHitList.push_back(h);

char _c[255];
_snprintf(
        _c,
        250,
        "Time: %d:%d:%d:%d",
        (thetime.wHour - 7),
        thetime.wMinute,
        thetime.wSecond,
        thetime.wMilliseconds);
LV_ITEM lvi;
lvi.mask = LVIF_TEXT | LVIF_PARAM;
lvi.iSubItem = 0;
lvi.iItem = 0;
lvi.pszText = _c;
lvi.lParam = (LPARAM)h;
```

本代码将获取的信息填充到 GUI 列表组件。

```
        ListView_InsertItem(ghWndSplitter, &lvi);
}
```

下面展示如何设置断点并处理单步事件。

```
if(dbg_evt.dwDebugEventCode == EXCEPTION_DEBUG_EVENT)
{
        switch (dbg_evt.u.Exception.ExceptionRecord.ExceptionCode)
        {
          case EXCEPTION_ACCESS_VIOLATION:
          MessageBox(
                NULL,
                "[!] Target experienced an ACCESS_VIOLATION ! \n",
                "hehehe",
                MB_OK);
          break;
          case EXCEPTION_BREAKPOINT:
          {
            CONTEXT ctx;

            HANDLE hThread =
            fOpenThread(
                  THREAD_ALL_ACCESS,
                  FALSE,
                  dbg_evt.dwThreadId);

            if(hThread == NULL)
            {
              MessageBox(
                    NULL,
                    "[!] OpenThread failed ! \n",
```

```
            "oops",
            MB_OK);
    return;
}
    // rewind one instruction
    ctx.ContextFlags = CONTEXT_FULL;
    if(!GetThreadContext(hThread, &ctx))
    {
      MessageBox(
             NULL,
             "[!] GetThreadContext failed ! \n",
             "oops",
             MB_OK);
      return;
    }

    if(ctx.Eip == g_bp_address+1)
    {
      ctx.Eip--;
      ctx.ContextFlags = CONTEXT_FULL;
      if(!SetThreadContext(hThread, &ctx))
      {
        MessageBox(
               NULL,
               "[!] SetThreadContext failed ! \n",
               "oops",
               MB_OK);
        return;
      }

                RemoveBreakpoint();

                take_sample(ctx);
```

断点事件处理结束后，需要重新写断点，这样下次执行到该处时，就可以继续处理。通过先单步再修改原来指令的方式来实现该功能。

```
                // we are going to run the instruction,
                // and then put the breakpoint back.
                SetSingleStep(dbg_evt.dwThreadId);
              }
              CloseHandle(hThread);
            }
            break;
            case EXCEPTION_SINGLE_STEP:
            {
              // put the breakpoint back, single step
             // is no longer active
             SetBreakpoint();
           }
           break;
           default:
           break;
        }// end switch
}
```

现在了解了如何使用断点获取某关键地址的信息。使用这个技巧可在游戏的某个函数上设置断点，该函数的某个参数是分析者关注的数据。函数有可能用于处理 NPC 列表，从而处理整个列表中的 NPC 数据。

7.2.6 用断点拉取链接信息

设置断点可以用来从运行进程中获取指针链接信息。从而根据结构信息抓取怪物列表，人物属性等。本例从魔兽客户端抓取所有重要玩家角色信息。

```
DWORD g_npc_breakpoint_location = NPC_BREAKPOINT_LOCATION;
DWORD g_pcbase_breakpoint_location = 0x45D492;
```

设置断点：

```
// this is a one-shot sampler that grabs the PC_BASE structure
m_pcbase_sampler = CreateBreakpoint(g_pcbase_breakpoint_location);
AddBreakpoint(m_pcbase_sampler);
```

相关处理过程：

```
DWORD WowzerEngine::OnBreakpoint(Breakpoint *theBreakpoint)
{
        if(theBreakpoint == m_pcbase_sampler)
        {
          // read EAX for the PC-BASE
          DWORD pc_base = theBreakpoint->m_context.Eax - 0x08;
          logprintf("SNAGGED PC BASE!  0x%08X", pc_base);
          m_player_character->m_baseaddress = pc_base;
        }
```

从代码中可以看到，断点发生以后，EAX 保存玩家角色信息。

上面讲解了基本的调试器技巧，可以应用该技巧附加外挂到游戏客户端。掌握该技巧对分析游戏非常有帮助。

7.3 Wowzer 外挂引擎

通过上面的学习，读者已经掌握很多外挂制作技巧，现在将这些技巧组合到一个实际的外挂程序中。比如构建一个通用外挂引擎，该引擎可以处理外挂所做的大部分工作。本部分介绍一个未公开的魔兽引擎 Wowzer。

从最顶层来看，Wowzer 引擎为类集合，类之间关系按照前面所述的状态机实现。图 7-3 展示了 Wowzer 的主要类结构。

. Wowzer 的基类实现了底层调试功能，可以用于任何游戏。这种抽象比较好，容易适用于很多游戏。从该类派生出 WowzerEngine 类，专门针对 WoW 调试。WowzerEngine 的子类用于计算小任务，比如查找 NPC 列表，计算操纵玩家角色。一般来说，和 WoW 相关的内容都在该类进行处理。最终，特定的游戏辅助功能也作为 WowzerEngine 的子类存在。可以查看图 7-4 中的 HunterBot，一个 AFK 战斗游戏辅助工具。

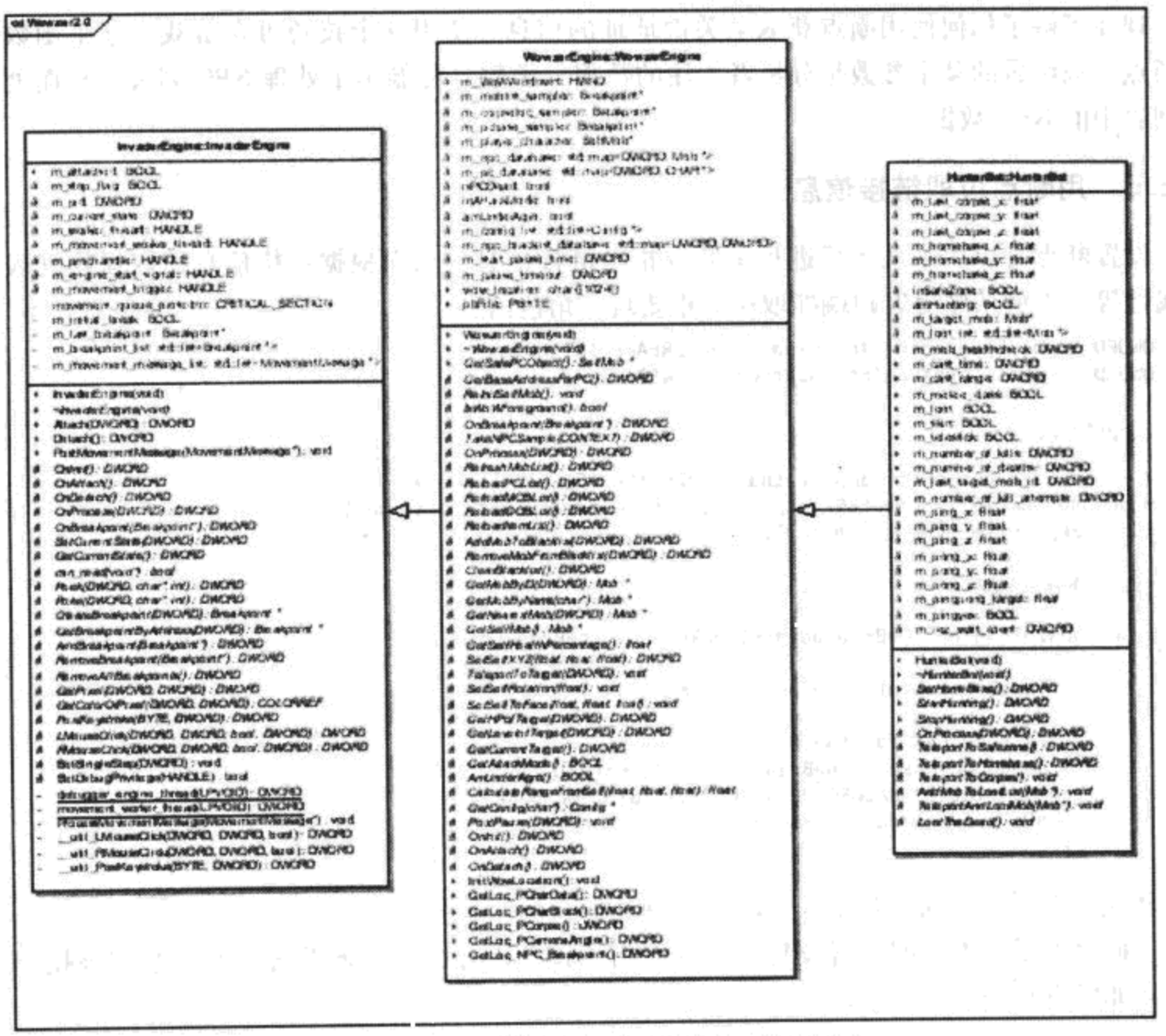


图 7-3 Wowzer 辅助平台在三个模块中使用的类

图 7-4 中还可以看到很多其他类型的辅助功能（大多在开发中），他们也作为 WowzerEngine 的子类存在，该设计允许基础代码尽量复用。

每个辅助功能对应一个状态机。图 7-5 中展示了 AFK 战斗辅助功能的 HunterBot 的状态机。

同样，可以创建如图 7-6 所示的状态机实现辅助功能 ZHackBot。

ZHackBot 使用的技巧很有意思，该功能将玩家角色移动到一个怪物永远无法攻击到的地方，但是玩家角色可以对怪物使用技能，杀死大量怪物。但是很不幸的是，当暴雪公司意识到这个问题后，暴雪公司对游戏作了修改，该功能的效果已经不明显。

WowzerEngine 工程非常大，开发周期长达数月，详细介绍该引擎超出了本书的范围。作者在介绍外挂制作基础时对各种基础技巧讲的非常详细，相信对读者构建自己的辅助工具平台很有帮助。

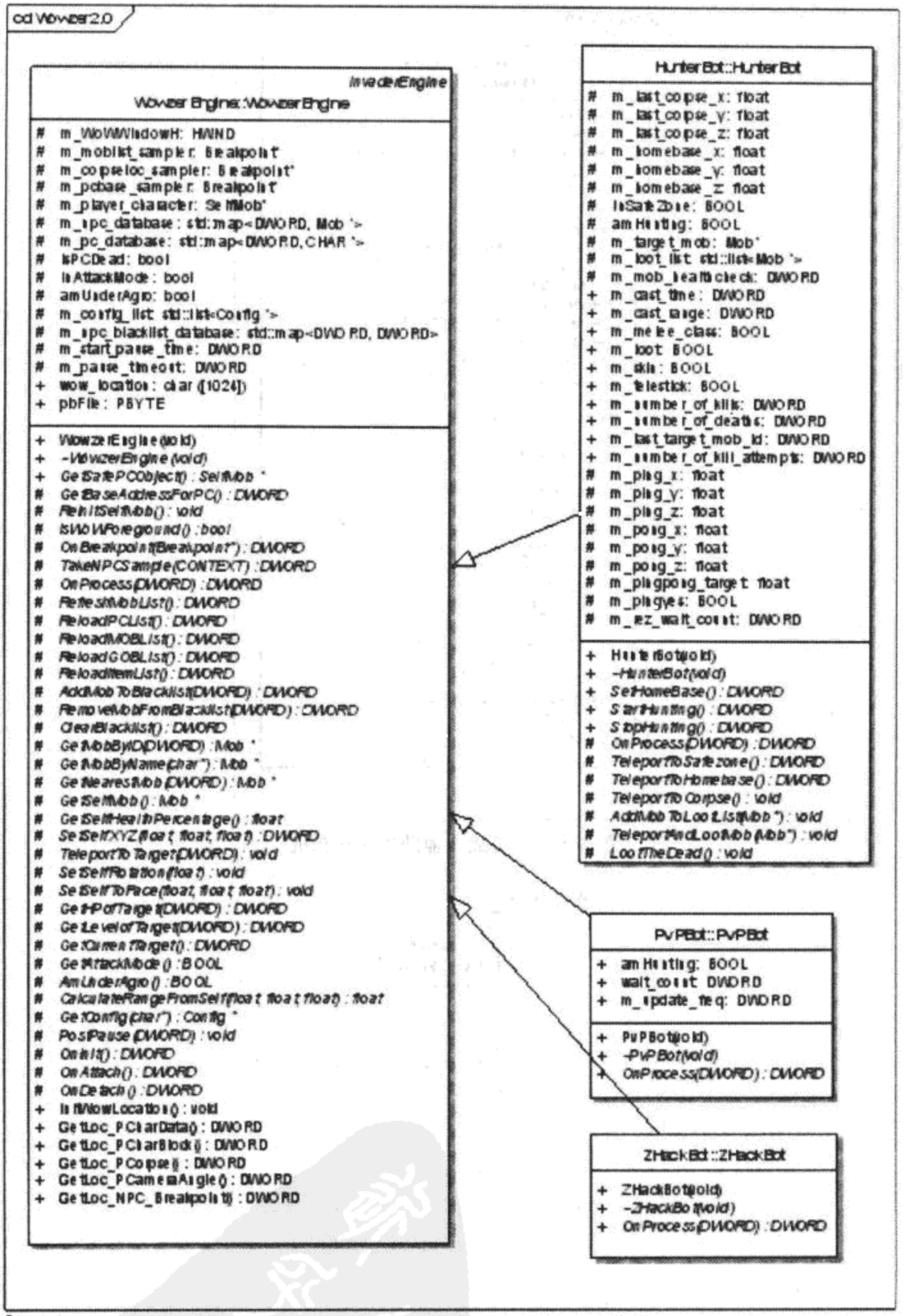


图 7-4 WowzerEngine 类中派生多个外挂辅助功能

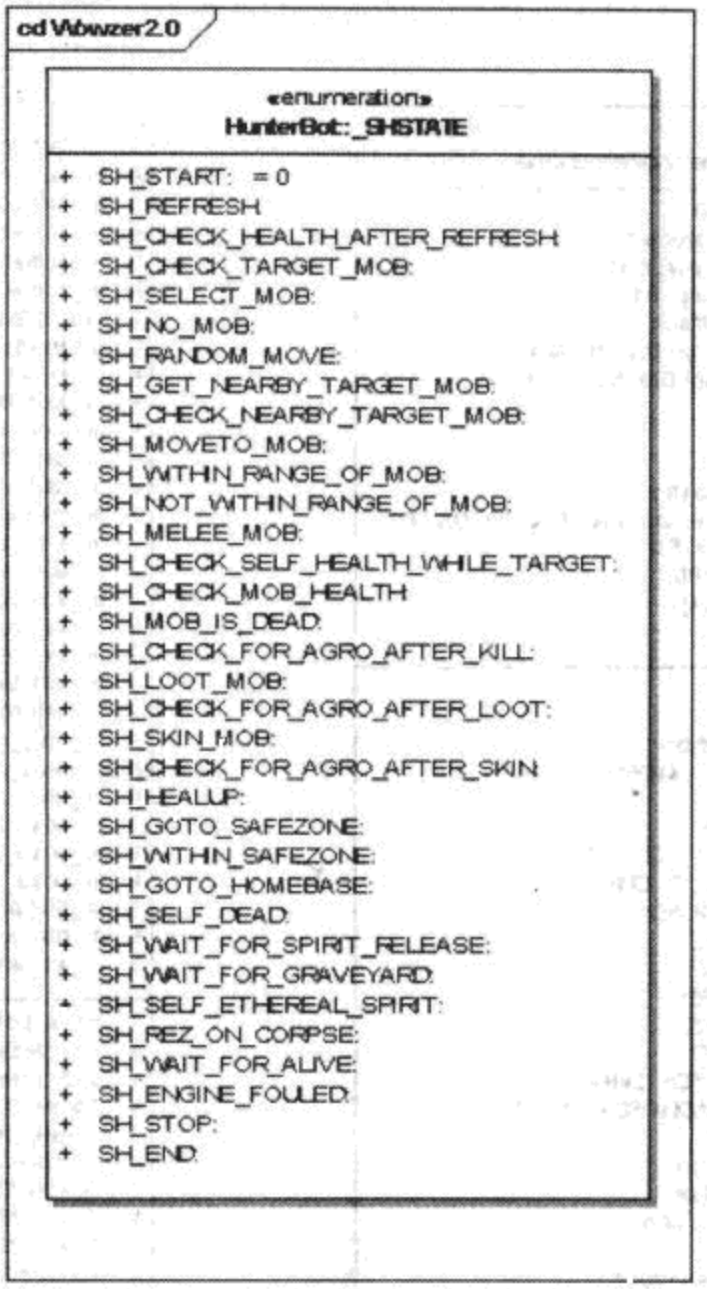


图 7-5 AFK 战斗辅助功能状态机

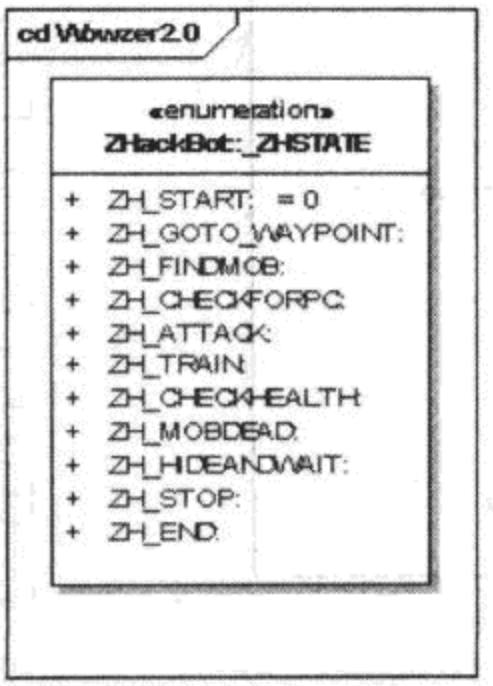


图 7-6 ZHackbot 辅助功能状态机

7.4 外挂制作高级技巧

本章接下来的部分介绍其他可以用于外挂制作的方法。首先介绍内核支持的外挂制作，采用内核驱动隐藏外挂。其次，介绍了构建外挂辅助工具宏。最后介绍如何开发高级外挂程序接口。

7.4.1 外挂和内核

在外挂制作者和游戏公司之间一直存在技术竞赛，一方试图躲过检测，另一方则试图检测所有作弊工具。就和象棋比赛一样，这场比赛也涉及你来我往的战争，但是和这些简单比赛不一样的是，目前还看不到这场比赛的最终结果会怎样。

在这场侦测和反侦测的比赛过程中，外挂制作者终于走到了系统内核，使用与 rootkits 类似的技术来隐藏自己。而大多数游戏本身不包含内核模式保护。现在随着nProtect GameGuard <http://eng.nprotect.com/nprotect_gameguard.htm> 和 PunkBuster <http://www.punkbuster.com/index.php> 的流行，游戏也开始包含内核保护模块。

本节介绍内核模式辅助工具（即前述外挂）通用结构。

内核辅助工具通用结构

工具首要设计目标为，游戏机器上所有辅助相关代码都在内核执行。这样做的理由是如果不扫描内核的话，游戏客户端无法扫描外挂代码，所有代码从进程空间无法获取，这和通常的外挂制作方法完全不同。

为了达到上述目的，可以将所有应用层代码移到另外一台机器，叫做控制机；运行游戏的机器叫被控机，两台机器通过通信链路通信。见图 7-7。

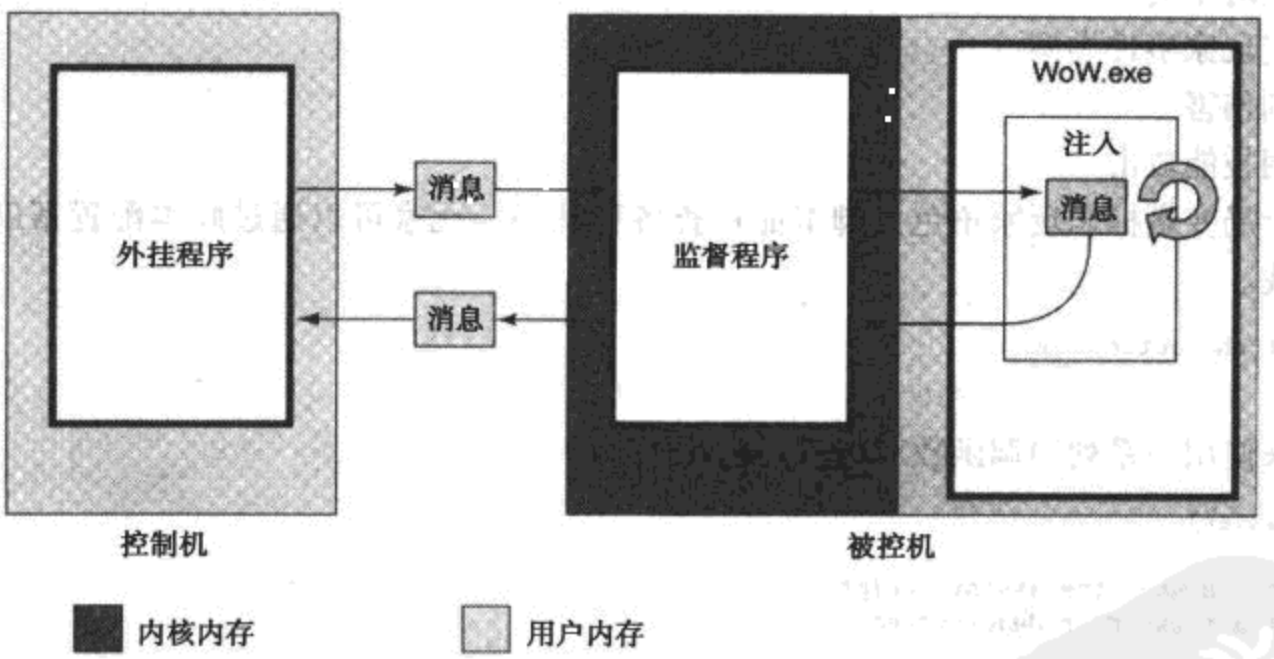


图 7-7 内核模式外挂结构，内核模式外挂是目前最复杂的在线游戏作弊手段

控制机运行外挂相关逻辑和脚本，驱动外挂执行。被控机运行游戏和相关内核驱动，内核驱动作为监督程序。两台机器之间使用 TCP/IP 通信。被控机中，游戏客户端进程被注入代码，该代码不通过注入 DLL 或线程实现，功能也非常简单，从而难以检测。另外，可以通过内核隐藏

注入在游戏空间的代码。

现在可以看到，关键的部分就是注入游戏客户端的代码。在 WoW. exe 中，注入的代码执行特别的字节码序列［译者注：虚拟机机器码］。所以，注入代码为虚拟机，就和 Java 虚拟机很类似。注入的代码执行通用指令，相当于一个虚拟的 CPU。

开发外挂监控端花费的时间长达数月，而且会用到很多非常底层的工具。比如，开发一套能在内核运行的调试框架非常困难，原因是代码不能自己调试自己。Hoglund 在 2006 年黑帽子会议上讲解了该技术。

7.4.2 战斗辅助工具

外挂的类型很多，最流行的外挂是无人值守类战斗辅助工具。第 2 章介绍了 AFK 战斗容易获取战斗经验，避免了重复打怪操作。

AFK 辅助工具涉及到几个难点：比如寻路和处理人工智能等。本节介绍了一个新的游戏辅助方法，适用于如 WoW 和其他类型的回合制大型多人在线游戏。

在大型多人在线游戏流行之前，第一人称射击游戏（FPS）的外挂很多。第一人称射击类的外挂的主要功能为自动瞄准，帮助玩家在射击过程中自动瞄准对手。同样的理念也可以应用到回合制的游戏中，与自动瞄准不同的是，新辅助工具处理复杂操作，主要是保证了打击敌人的时候使对手受到的伤害最大化，玩家角色自身伤害最小化。这里描述的外挂可以应用到 WoW 以及其他类型的游戏。

战斗辅助工具维护了一个对手列表，该列表产生方法与本章前述的怪物列表产生类似。使用脚本管理玩家角色战斗。这个方法很容易扩展到玩家和 NPC 的战斗。

脚本组织的结构为：脚本保存了一系列的函数，每个函数完成固定的功能。

- 判定对手类型
- 判定玩家事件类型
- 受到伤害
- 使用技能攻击

使用全局变量标志玩家角色。脚本能检查各种状态，玩家可以通过脚本配置帮助和辅助角色自动游戏。

```
ASSIST( character name )
{
```

接下来使用一系列回调函数对应不同的事件。

```
OnActivate()
{
  // called when the assist script
  // is activated or deactivated
}
```

然后使用定时器处理通用状态，主要用于处理玩家角色状态。

```
OnTimer()
{
```

变量 ME 代表玩家角色。

```
// called once per second, to manage all buffs
if( ME.Buffs.Has( "Concentration Aura" ) )
{
}
else
{
```

Cast 命令用于释放技能。

```
    Cast( "Concentration Aura" );
  }
}
```

使用伤害检测函数检查当前伤害状态是否容易导致死亡，玩家是否需要对自身进行防护等。

```
OnDamage()
{
  // called when taking damage
  if(RATE > 50%)
  {
    // RATE is a % of damage, measured against
    // total hitpoints, taken in the last second

    Cast( "Divine Shield" ); // cast a spell
    RETURN;
  }

  if(ME.Health < 20%)
  {
    TargetSelf();
    Cast( "Holy Light" );
    TargetLastTarget();
  }
}
```

下面的代码仅作为示例，回调函数根据每个类型的对手而不同。如果有 3 个对手在攻击范围内。回调函数对 3 个对手分别调用，每个一次，最终来看，每个对手类型对应一个回调函数。

```
ForHunter()
{
  // called for any opponent who is of the Hunter class
```

在回调函数中，如果玩家瞄准了一个对手，接下来将发生一系列攻击步骤。

```
if(ME.Target == HUNTER)
{
  if(RANGE < 30)
  {
```

下列伪代码描述了圣光战士如何操作可以给对手最大化伤害。

```
if(ME.Buffs.Has("Seal of Crusader"))
{
  Cast("Judgement");
  Cast("Seal of Command");
}

EnsureMeleeMode();

if(ME.Buffs.Has("Seal of Command"))
{
```

下列代码检查是否有足够的魔法值，以及命令圣印是否小于10秒，如果条件都满足的话则使用审判技能。

```
        if(ME.Mana > 10%)
        {
          if(ME.Buffs.TimeLeft(
                 "Seal of Command") < 10)
          {
            Cast("Judgement");
            Cast("Seal of Command");
          }
        }
      }
      else
      {
      if(ME.Mana > 60%)
        CastIfNotBuffed( "Seal of Crusader" );
      }
    }
}
```

有些对手有宠物，也可以在回调函数中处理宠物攻击。本例中，脚本会自动攻击宠物，并通知玩家角色，该对手需要优先处理。

```
ForHunterPet()
{
  // called for any Hunter's pet

  // RANGE in meters between character and the pet
  if(RANGE < 10)
  {
    // pet is close
    if( PET.Target == ME OR PET.Owner.Target == ME )
    {
      // this opponent is going for us
      // stun the pet and go for the opponent
      Target(PET);
      Cast( "Hammer of Justice" );

      // set target with priority 5
      PriorityTarget(PET.Owner, 5);
      SelfMessage(" GO FOR THE HUNTER ");
    }
  }
}
```

可以为其他对手类型设置类似的回调函数。当然，这只是一种可能的外挂实现方式，本文采用这种方法的原因是每种类型的对手需要使用不同的战斗策略。

```
ForMage()
{
}

ForWarlock()
{
}

//etc...
```

7.4.3 外挂用户界面

前面所述的调试器类外挂应用了很多技巧，但是该外挂用户界面很不友好。如果想使外挂更加好用，需要为外挂添加友好的用户界面。本节使用基于对话框的应用实现外挂控制功能。

采用基于对话框的 MFC 应用创建外挂程序，可以使用 Visual Studio 应用向导自动生成应用框架。(图 7-8)

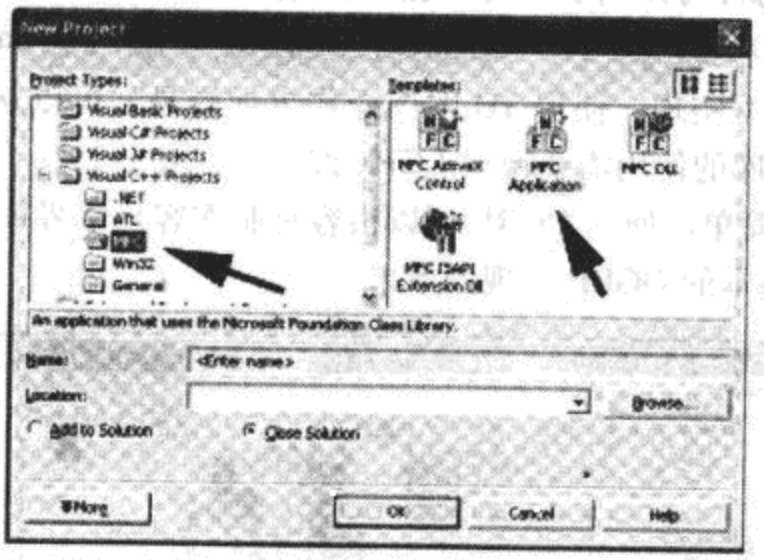


图 7-8 使用 Visual Studio 应用向导自动生成基于对话框的应用框架

生成应用框架后，可以在对话框界面添加各种各样的控件，比如列表框控件可以用于列出游戏世界对象，按钮用于控制行为等。图 7-9 展示了如何生成对话框程序。使用 Visual Studio 的向导功能生成外挂界面非常方便。

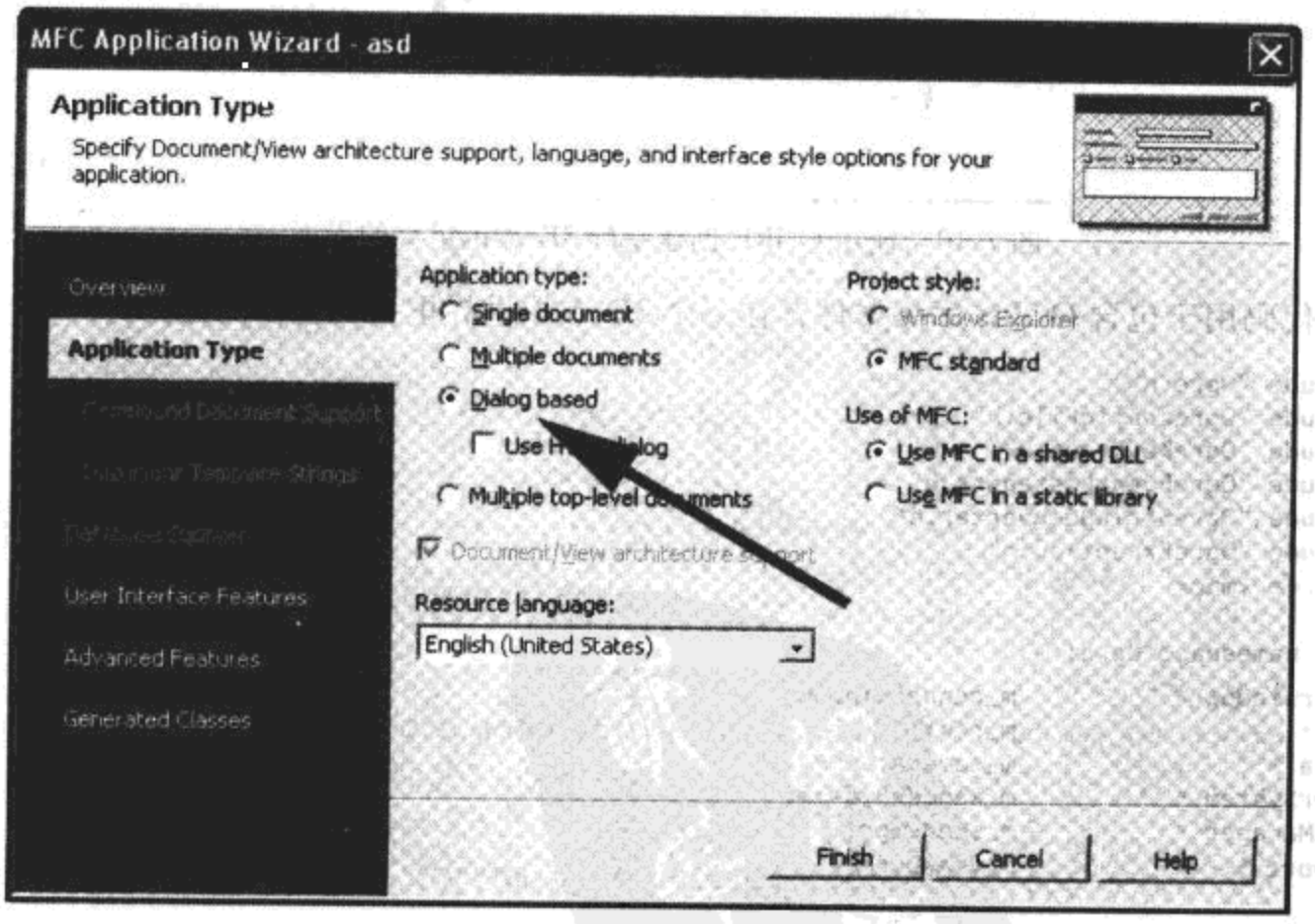


图 7-9 通过程序向导，可以在对话框中加入各种功能，这可以轻松地使外挂具备一个丰富的界面

集成3D渲染效果

大多数在线游戏使用绚丽的3D渲染效果构建游戏世界。如果要想使外挂看上去和游戏很相配的话，外挂界面也应当有3D渲染能力。有了3D渲染能力以后，外挂可以漂亮地显示游戏中的对象，甚至可以单独创建游戏界面（请查看第9章）。

OGRE 3D渲染库

幸运的是，有很多图像库可以用于3D游戏渲染。这意味着开发人员不需要一切重头开始开发。这当中最好的图像库就是OGRE（Object-Oriented，Graphics Rendering Engine面向对象图像渲染引擎），该库由一系列C++类实现，使得3D图像操作更加容易。可以从http：//www. ogre3d. org下载OGRE库，注意，该库的使用需要遵循LGPL许可。

OGRE渲染库使用很简单。使用OGRE可以很容易制作客户端界面，以及描述3D场景中对象位置。图7-10显示了基本的OGRE实现的效果。

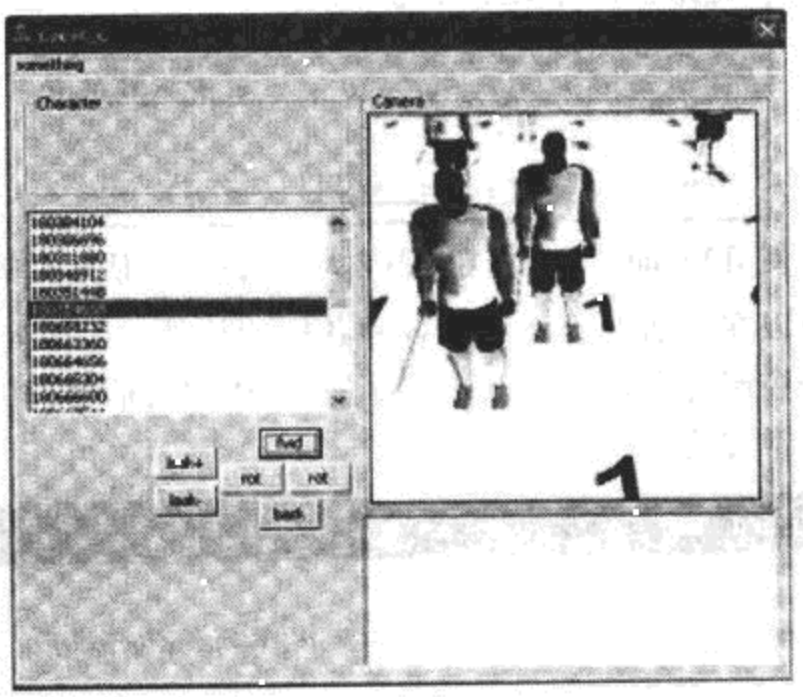


图7-10 使用OGRE创建基础3D虚拟世界的界面

下面代码用于包含OGRE库头文件，在一个3D虚拟世界里实现基本的渲染：

```
#include "Ogre.h"
#include "OgreConfigFile.h"
#include "OgreKeyEvent.h"
#include "OgreEventListeners.h"
#include "OgreStringConverter.h"
#include "OgreException.h"
#include <map>

using namespace Ogre;
RenderWindow *          m_renderwindow;
Root *                  m_root;
Camera *                m_camera;
RenderSystem *          m_RenderSystem;
SceneManager *          m_sceneMgr;
Viewport *              m_viewport;

Entity *                m_miner;
Entity *                m_bldg;
```

```
void Init(HWND hWindow);
void Update();
void Kill();
void UpdateObjectPosition( DWORD id, POINT p);

void MoveCamera( float distance );
void RotateCamera( float degrees );
void PitchCamera( float degrees );
void LookAt( int id );

BOOL selectRenderingPlugin( char *theName );
void loadResources();
void createDebugObjectsInScene();
void createPlane();

std::map<DWORD, SceneNode *> g_points;
```

下面的代码需要调用者传递父窗口句柄。如果使用 MFC 框架创建应用，可以通过对话框对象的成员 m_hWnd 来获取窗口句柄。下面的代码创建 OGRE 3D 渲染窗口，放置该窗口在父窗口上。图 7-10 就是通过这种方法创建的。

```
void Init(HWND hWindow)
{
```

OGRE requires a Root object to be created first

```
m_root = new Root("","); //root without config files
```

现在加载两个图像子系统，使用 OpenGL。

```
m_root->loadPlugin("RenderSystem_Direct3D9");
m_root->loadPlugin("RenderSystem_GL");

// use the openGL renderer
if(FALSE == selectRenderingPlugin("OpenGL")) return;
```

调用函数 initialise 后（译者注：作者在此处开玩笑的怀疑 OGRE 的作者不是欧洲人:)），加载资源。资源包含场景中的纹理结构等素材。可以从网络下载到很多工具，用于将对象转换为 OGRE 资源文件。

```
m_root->initialise(false);
loadResources();
```

接下来创建渲染窗口。OGRE 负责创建渲染窗口，开发人员只需要使用 getCustomAttribute 获取创建的窗口句柄即可。获取窗口句柄后，使用 SetWindowLong()修改窗口风格为 WS_CHILD。

```
m_renderwindow =
        m_root->createRenderWindow(
                "theCamera",
                280,  //width
                290,  //height
                false,  //fullscreen or not
                0 );  //optional
// dirty tricks to make it a child window
HWND aHandle;
```

```
m_renderwindow->getCustomAttribute(
        "HWND",
        &aHandle);

SetParent(aHandle, hWindow);
SetWindowLong(
        aHandle,
        GWL_STYLE,
        WS_CHILD | WS_BORDER | WS_VISIBLE );
```

现在获取 SceneManager,SceneManager 为 OGRE 中非常重要的对象，该对象直接管理所有 3D 对象。

```
m_sceneMgr = m_root->getSceneManager( ST_EXTERIOR_CLOSE );
```

现在创建照相机，用于移动场景位置，修改视角和观察点。可以使用照相机查看不同对象。

```
m_camera = m_sceneMgr->createCamera("MainCam");
m_camera->setNearClipDistance(1.0f);
m_camera->setFarClipDistance(50000.0f);
```

同时需要建立照相机的视图通道：

```
m_viewport = m_renderwindow->addViewport(m_camera);
m_camera->setAspectRatio(
        Real( m_viewport->getActualWidth()) /
        Real(m_viewport->getActualHeight()));
```

最后，加载资源和对象到创建的世界中。

```
        // load the default objects
        m_miner = m_sceneMgr->createEntity("miner", "knot.mesh");
        m_miner->setMaterialName("MinerMaterial");

        m_bldg = m_sceneMgr->createEntity("bldg", "knot.mesh");
        createPlane();
        createDebugObjectsInScene();
}
void Kill()
{
        HWND aHandle;
        m_renderwindow->getCustomAttribute("HWND", &aHandle);
        DestroyWindow(aHandle);
}

void Update()
{
        m_renderwindow->reposition( 260, 27 );
        m_renderwindow->resize( 280, 290 );
        m_renderwindow->update();
}

// for testing, create a debug object
void createDebugObjectsInScene()
{
        Entity* myKnot =
                m_sceneMgr->createEntity("knot", "knot.mesh");
        myKnot->setCastShadows(true);

        SceneNode* myNode1 =
                m_sceneMgr->getRootSceneNode()->
```

```
            createChildSceneNode( "node_1" );
        myNode1->attachObject(myKnot);

        // set the nodes position
        myNode1->setPosition( Vector3(0, 0, 0) );
        m_camera->setPosition( Vector3(0,300,0) );
        m_camera->lookAt( Vector3(0, 0, 0) );

        // Set ambient light
        m_sceneMgr->setAmbientLight(ColourValue(0.4, 0.4, 0.1));

        // Create a light
        Light* l = m_sceneMgr->createLight("MainLight");
        l->setType(Light::LT_POINT);
        l->setDiffuseColour( 200, 200, 200 );
        l->setPosition(30,30,30);
}
BOOL selectRenderingPlugin( char *theName )
{
        assert(m_root != NULL);

        // list them
        RenderSystemList *rList = m_root->getAvailableRenderers();
        RenderSystemList::iterator it = rList->begin();

        // example of iterating them
        while( it != rList->end() )
        {
          RenderSystem *rSys = *it;
          it++;
          if(rSys->getName().find(theName))
            {
              m_root->setRenderSystem(rSys);
              m_RenderSystem = rSys;
              break;
            }
        }

        // how to end if we can't find one
        if(m_root->getRenderSystem() == NULL)
        {
          assert(m_RenderSystem == NULL);
          return FALSE;
        }

        return TRUE;
}

void UpdateObjectPosition( DWORD id, POINT p)
{
        if( g_points.find(id) == g_points.end() )
        {
          char node_name[64];
          _snprintf(node_name, 62, "node_%d", id);

          // it was not found, create a new object
          SceneNode* a_node =
                m_sceneMgr->getRootSceneNode()->
                createChildSceneNode( node_name );
          Entity *e =
                m_sceneMgr->createEntity(node_name, "ninja.mesh");
          e->setMaterialName("MinerMaterial");
```

```
            a_node->attachObject(e);
            a_node->setPosition( Vector3(p.x, 0, p.y) );
            g_points[id] = a_node;
        }
        else
        {
            // it exists, so update its position
            SceneNode *a_node = (SceneNode *)g_points[id];
            a_node->setPosition( Vector3(p.x, 0, p.y) );
        }
}

void loadResources()
{
        ResourceGroupManager::getSingleton().addResourceLocation(
                "./media/models",
                "FileSystem",
                "General");

        ResourceGroupManager::getSingleton().addResourceLocation(
                "./media/scripts",
                "FileSystem",
                "General");

        ResourceGroupManager::getSingleton().addResourceLocation(
                "./media/textures",
                "FileSystem",
                "General");

        ResourceGroupManager::getSingleton().
                initialiseAllResourceGroups();
}

void MoveCamera( float distance )
{
        m_camera->moveRelative( Vector3( 0, 0, distance) );

        char _t[255];
        Vector3 v = m_camera->getPosition();

        _snprintf(_t, 252, "camera is %f %f %f", v.x, v.y, v.z );
        OutputDebugString(_t);
}

void RotateCamera( float degrees )
{
        m_camera->yaw( Radian(degrees) );

        char _t[255];
        Vector3 v =m_camera->getDirection();
        _snprintf(
                _t,
                252,
                "camera direction at %f %f %f",
                v.x, v.y, v.z );
        OutputDebugString(_t);
}

void PitchCamera( float degrees )
{
        m_camera->pitch( Radian(degrees) );
```

```
        char _t[255];
        Vector3 v =m_camera->getDirection();
        _snprintf(
                _t,
                252,
                "camera direction at %f %f %f",
                v.x, v.y, v.z );
        OutputDebugString(_t);
}

void createPlane()
{
        Plane plane;
        plane.normal = Vector3::UNIT_Y;
        plane.d = 0;
        MeshManager::getSingleton().createPlane(
                "plane_1",
                ResourceGroupManager::DEFAULT_RESOURCE_GROUP_NAME,
                plane,
                50000,
                50000,
                10,
                10,
                true,
                1,
                50,
                50,
                Vector3::UNIT_Z);

        Entity *aPlaneEntity_1 =
                m_sceneMgr->createEntity(
                        "c_plane_1",
                        "plane_1" );
        aPlaneEntity_1->setCastShadows(false);
        SceneNode *aSceneNode_1 =
                m_sceneMgr->getRootSceneNode() ->
                        createChildSceneNode("n_plane_1");
        aSceneNode_1->attachObject(aPlaneEntity_1);
        aPlaneEntity_1->setMaterialName("PlaneMaterial");
        aSceneNode_1->setPosition( 0,-1000,0 );
}

void LookAt( int id )
{
        SceneNode *a_node = (SceneNode *)g_points[id];
        Vector3 v = a_node->getPosition();
        m_camera->setPosition( Vector3( v.x, v.y+500, v.z-200 ) );
        m_camera->lookAt( v );
}
```

可以将用户界面编写的非常复杂，甚至达到重新创建客户端的程度。实际上，一些外挂确实可以做到完全替换客户端。

本节介绍了三个高级外挂制作主题：内核模式外挂、宏驱动外挂和外挂用户界面，这仅仅展示了很多技术发展过程正在研究的三种方法而已。可能外挂制作本身需要一本书才能介绍清楚。

7.5 外挂制作结论

从本章内容的长度可以看出，外挂软件制作是游戏逆向分析的中心议题。玩游戏的过程中要是有程序能自动帮助玩家打怪并赢得游戏是件很舒服的事情，特别是这种自动程序还可以给玩家带来金钱利益的话就更再好不过了。

本章讲解了很多基础外挂制作知识，这些知识并不针对某一款游戏。虽然本章中以 WoW 客户端为目标讲解外挂制作，但实际上，本章所讲的知识适用于任何在线游戏。

第8章　软件逆向工程

“逆向工程”主要通过拆解分析程序的二进制代码，从而理解代码工作情况，这是一门技术性很强的艺术。游戏破解人员的大多数精力都花在逆向工程上，特别是游戏客户端本身以及游戏客户端和游戏服务器通信上。理解游戏客户端逻辑和游戏客户端数据结构是理解游戏的好方法。通过逆向工程，理解游戏客户端是如何工作的，这是游戏破解的基石。

本章介绍的是分析任何新游戏的预备知识。新的 MMO 大规模多人在线游戏无时无刻不在出现，每个游戏都可能会有本书中介绍的各种安全问题。也就是说，本书介绍的技巧和游戏弱点并不针对某几款游戏，而是普遍存在的。不少技术在功能上实际是差不多的。逆向工程，绝对是模拟构造服务端（见第9章）所必需的技能。完全破解游戏客户端和服务器交互的协议以后，就可能构建第三方的游戏服务器。一旦掌握了游戏逆向能力，可以做的事情就很多，比如编写探嗅器、中间攻击、制作外挂、客户端状态模拟等等。

要做到这些，最基本的技能就是：通过逆向工程，理解游戏客户端逻辑结构，理解对象是如何组织和相互交互的，理解每个部分代码都分别有什么作用。

8.1　游戏破解

软件有各种各样的结构。游戏破解者常常全力以赴分析、检测游戏一小块区域的代码，关注特定功能的代码片断，主要是因为破解者只需要关注一部分功能代码。然而，从整体来理解代码可能是更好的方式。

从整体来理解代码是专业逆向人员的能力之一，专业逆向人员也常常训练自己能从大范围考虑问题。逆向工程也常被称为黑色艺术，事实上，该技术已经越来越规律化。逆向工程艺术正在慢慢成为一门科学。

计算机如何工作和软件如何构建的知识对逆向工程的帮助很大。例如：大多数游戏都采用面向对象编程的方法写成。熟练的逆向人员懂得如何使用面向对象的方法编程，更进一步会去理解编译器如何将高级语言如 C++ 语言转化为机器可以理解的语言。对软件和操作系统架构有深刻理解的逆向工程师知道对象在游戏中如何组织。游戏是不是多线程的？在多个线程间是不是有消息队列？对象是不是使用继承？是不是抽象对象，或者是不是使用了模板和遍历器？是不是使用了单向或者双向队列？对象销毁是显式的还是使用引用计数的垃圾回收机制？被逆向程序中这些关键点大大地影响了游戏操作，这些关键点的理解也大大地影响了逆向人员的逆向速度和效果。

8.1.1 代码逆向过程

逆向软件的时候，逆向人员基本都比较了解软件功能，软件如何操作等诸如此类的信息。但实际上，如果想寻找软件漏洞的话，逆向人员最好对软件信息一无所知。这是因为，假如逆向人员对目标软件一无所知的话，就很可能会在软件操作中触发软件开发人员从来不曾想到过的情况。另外一方面，如果逆向人员比较熟悉游戏的某一方面（比如，知道在物品栏移动物品有可能会导致游戏 bug 等），逆向人员等于有了该类游戏漏洞的外挂半成品（本例中，比如物品复制等功能有可能就可以实现）。在这两种情况下，都是利用游戏开发人员的假设，使程序员未曾考虑到的极端情况成为潜在漏洞。比如：假定玩家在游戏战斗中不会修改其 XYZ 坐标位置有可能成为潜在的漏洞。

“假定事物情形”这个安全漏洞是开发人员常常忽略，而安全人士却常常视为重点的问题。软件开发人员常常对软件交互做一系列复杂的假定；在很差的情况下，这些假定将有可能会成为安全漏洞。攻击者和安全分析人员会尝试各种边际情况来发现这些假定情况。从软件安全性考虑，软件开发不应当假设任何前提条件。

用不假定任何条件的态度来实施逆向工程，将会比纠缠在二进制代码的细节之中（比如为什么这样或者那样的攻击无效）更为有效。比如，一般来说，找到服务器上另一个玩家的 IP 地址看起来是个不可能完成的任务。但是，如果不仔细的检查，逆向者就不应该作出这样的假定。服务器在某些情况下会不小心复制未初始化的数据到聊天数据，在这种情况下，有可能在聊天的数据中找到和你聊天的玩家的 IP 地址。获取其他玩家的 IP 地址之后，可以使用网络黑客工具来攻击其 IP 地址，从而迫使其玩家角色离开游戏。

任何情况下，开始逆向工程一个软件时，需要对该软件的整体框架有直观的感受。常用的好办法是加载运行游戏后，使用如 Process Explorer NT < http：//www. sysinternals. com > 来列出所有加载的 DLL 和打开的文件等（图 8-1）。

很多软件逆向工作者为了对软件系统整体框架有了解，常常用画图的方法来理解系统架构。建模语言如 UML（Unified Modeling Language）非常适合于描述软件架构。软件系统中模块如 DLL，在 UML 用套件（package）表示。套件用于描述任意组合在一起的事物。几乎所有被关注的套件都包含一组关系紧密地代码和数据集合。所以，通过如图 8-1 中检查出的加载 DLL，可以构造出软件的套件图及各个套件间的相互关系。将被分析的软件按照这种方法分割为各个组件后，就可以感知整个软件系统的软件框架。

分析软件系统，首先获取 EXE 文件和 DLL 文件列表，将每个可执行模块视为一个套件。图 8-1 中显示了一个大型多人在线游戏《天行者：英雄诗歌（Vanguard：Saga of Heroes）》的 DLL 文件。从该 DLL 列表中一眼就可以看到游戏使用了 `crypt32. dll` 文件，这个加密函数代码库由微软公司提供，另外还可以看到该游戏使用了许多由操作系统提供的 DLL 文件。实际上，游戏中只有很少几个 DLL 是由游戏开发者编写，大多数代码库为第三方开发的库文件，用于显示渲染、声音播放和加密等。从发现的 EXE 和 DLL 列表中，可以开始着手构建软件的架构图了。

图 8-2 显示了逆向工程时初步获取的套件图。

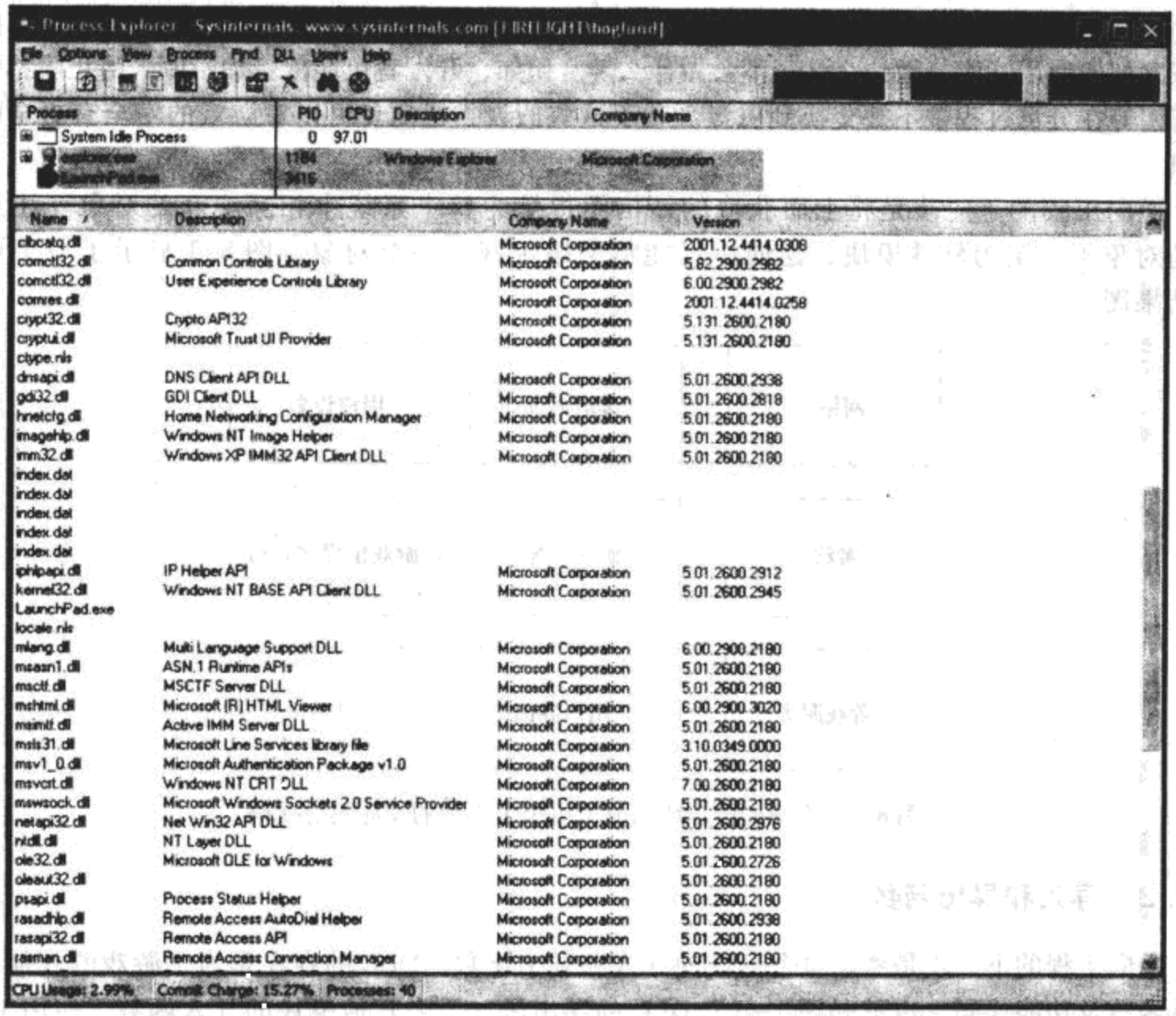


图 8-1　即将上市的游戏（作者写作时）《天行者：英雄诗歌（Vanguard：Saga of Heroes）》加载的 DLL 文件。这些 DLL 为渲染效果，操作游戏的软件组件

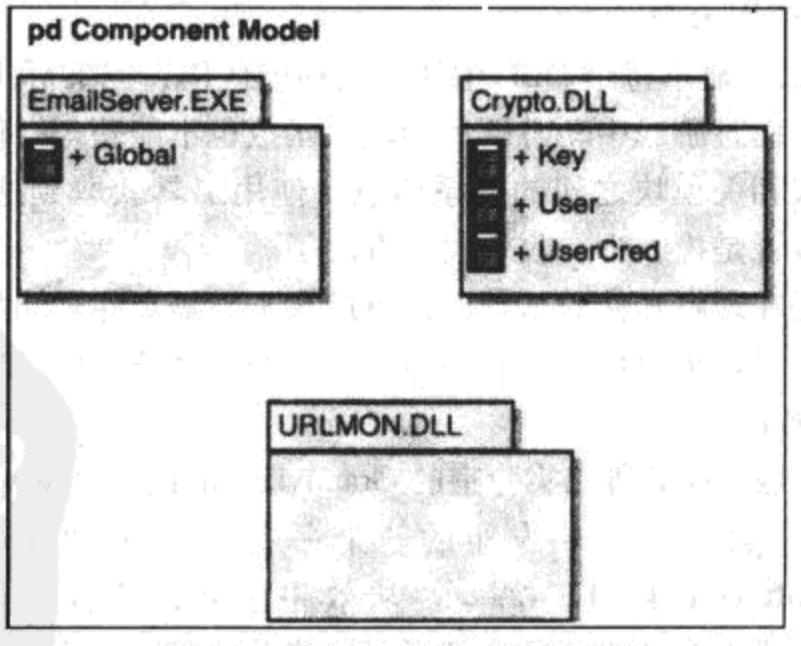


图 8-2　逆向初步获取的套件图

通常情况下，游戏开发者对第三方库（比如：crypt32. dll）假定过多了，开发者认为他们知道该库如何使用，库能做什么，库不会做什么等。比如，开发者可能依赖加密库来防止隐私信息如密钥等在分页过程中不被交换到磁盘。基于这些假设，逆向者可以对游戏的安全有更全面的认识。

逆向工程的下一步是将上面获取的结构图提炼为整个系统中相互交互的对象。注意此处的对象不一定为软件模块，逆向中可能将网络等视为一个对象。图 8-3 显示了一个可能的结果图。

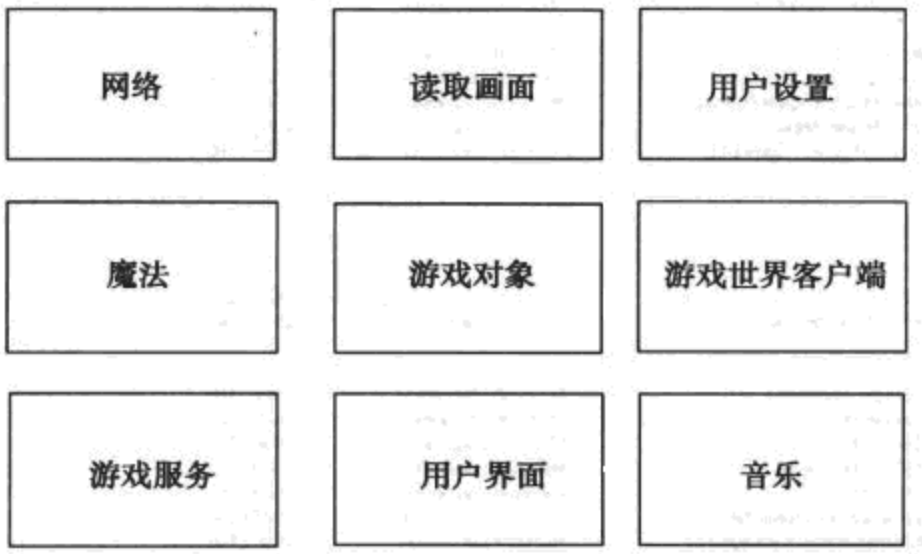


图 8-3 大规模游戏代码分类组织为小型的功能集合代码

8.1.2 导入和导出函数

逆向工程的下一步是考虑每个套件的导入和导出函数。DLL 的存在是为了游戏的其他部分提供函数及功能代码。也就是说，一个 DLL 的导出函数，为其他模块的导入函数。使用工具，比如微软公司的 Dependency Walker < http：//www. dependencywalker. com > ，可以很容易检查 DLL 之间的依赖关系，而且该工具可以显示模块之间调用所使用的 DLL 功能函数。图 8-4 显示了 Dependency Walker 的使用示例。

Dependency Walker 试图显示每个 DLL 中使用的功能代码，但实际上，该工具不能显示所有游戏中动态加载的代码，而且游戏中动态调用的输入函数也不能获知，但是不可否认的是，通过该工具可以获取很多游戏相关模块之间的关系信息。使用工具获取到这些相关信息所用时间大概只需要一个小时，可以说是投入少，收效大的方法了。

逆向过程的下一步是在理解套件的基础上，更进一步理解套件之间如何互动。通过图 8-5 中的简单示例，读者应该可以理解套件互动的具体含义。图 8-5 显示了 Windows 常用程序记事本 NOTEPAD. EXE 的输入函数图。

用来画图 8-5 的工具是 AT&T 研究实验室的 Graphviz < http：//www. graphviz. org > ，以及一个简单的 Perl 脚本，该工具调用微软发布的一个命令行工具 DUMPBIN，具体请查看网页 < http：//support. microsoft. com/kb/177429 > 。从效果上看，这样获取的数据和 Dependency Walker 获取的数据一样。下面是脚本文件，读者可以修改该脚本，用于其他用途。

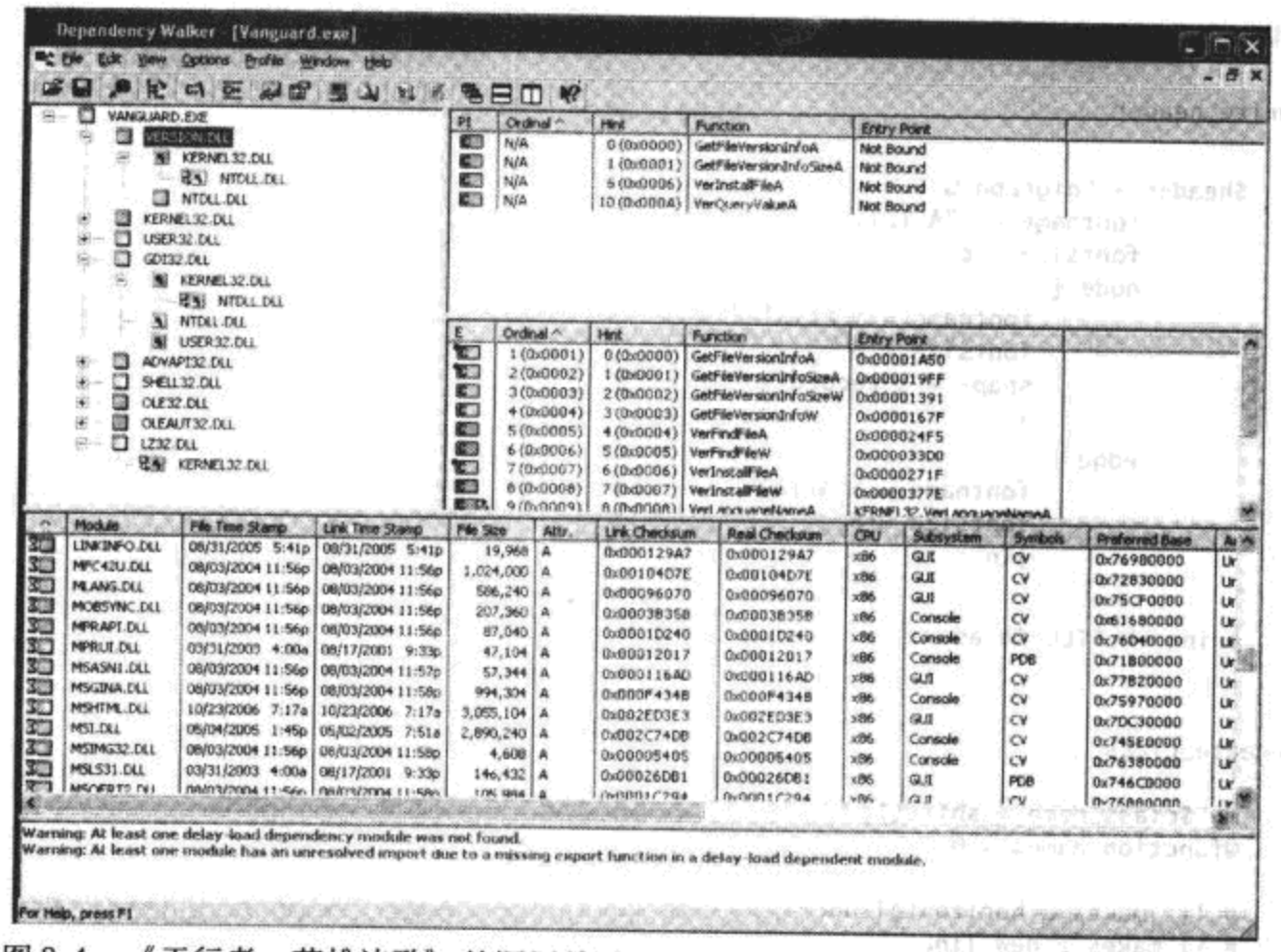


图 8-4 《天行者：英雄诗歌》的调用关系。从图中，我们可以看到 VANGUARD. EXE 使用了 VERSION. DLL，而 VERSION. DLL 使用了 KERNEL32. DLL，KERNEL32. DLL 使用了 NTDLL. DLL。这种链接结构有时会变得很长。换句话说，上层的功能实际是由很多底层的代码一起实现的，游戏依赖这些代码来完成功能

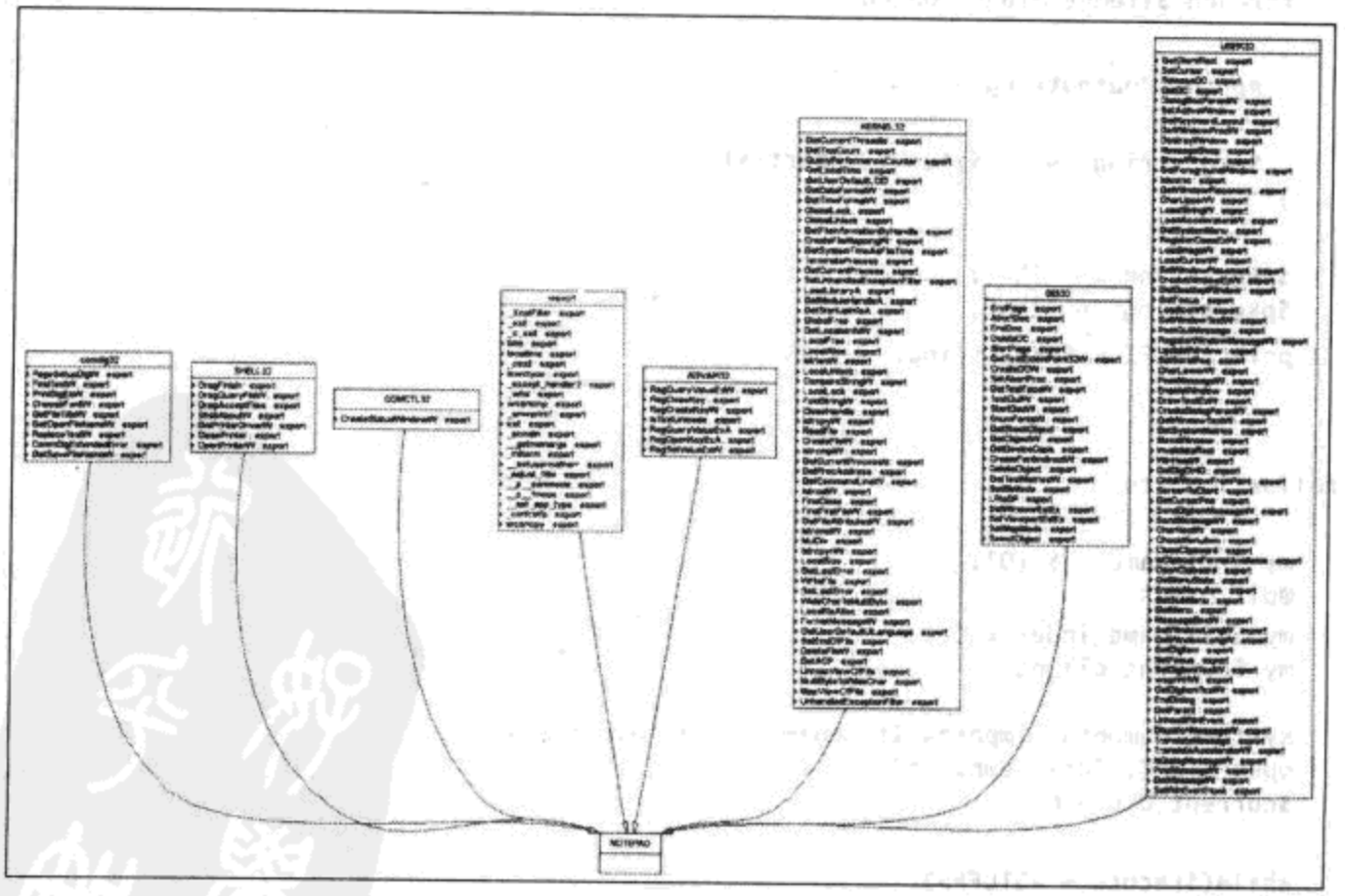


图 8-5 逆向分析 NOTEPAD. EXE 的套件调用关系

```
open(UMLFILE, ">c:\\umlfile.dot");

sub write_header
{
        $header = "digraph G {
                fontname = \"Arial\"
                fontsize = 8
                node [
                        fontname = \"Arial\"
                        fontsize = 8
                        shape = \"record\"
                        ]
                edge [
                        fontname = \"Arial\"
                        fontsize = 8
                        ]\n";

        print UMLFILE $header;
}

sub create_class
{
        my $class_name = shift();
        @function_names = @_;

        # '|' makes a horizontal bar
        # \l makes a new line

        $pkg_string = "
                $class_name [
                        label = \"{ $class_name|";

        foreach $item ( @function_names )
        {
          #print "outputting $item \n";

          $pkg_string .= "+ $item : export\\l";
        }

        $pkg_string .= "}\"";
        $pkg_string .= "]\n";
        print UMLFILE $pkg_string;
}

sub follow_imports
{
        my $filename = $_[0];
        @dll_names;
        my $dll_name_index = 0;
        my $current_dll_ptr;

        system("dumpbin /imports $filename > c:\\temp.txt");
        open(STUFF, "c:\\temp.txt");
        $current_DLL = 0;

        while($record = <STUFF>)
```

```
        {
          print $record;
          if($current_DLL)
          {
            if($record =~ m/[ ]+[0-9A-Fa-f]+[ ]+[A-Fa-f0-9]+
([A-Za-z_0-9]+)$/)
            {
              print "pushing $1 \n";
              # get reference to the anonymous array created below
              $arrayptr = $dll_names[$current_dll_ptr + 1];
              #push import into anonymous array
              push( @$arrayptr, $1);
            }
          }
          if ($record =~ m/[ ]+([0-9A-Za-z]+\.dll)$/i )
          {
            print "got DLL: $1\n";
            $current_dll_ptr = $dll_name_index;
            $dll_names[$current_dll_ptr] = $1;      #name of DLL
            #new anonymous array
            $dll_names[$current_dll_ptr + 1] = [];
            $dll_name_index += 2;

            $current_DLL = $1;
          }
        }

        return( @dll_names );
}
sub write_tail
{
        print UMLFILE "\n}";
}

write_header();

$target_exe = "c:\\windows\\system32\\notepad.exe";
$target_exe_name = "NOTEPAD";

# create base object
print UMLFILE "$target_exe_name [ label = \"{ $target_exe_name| }\"
]\n";

# the return value is a nasty perlism, every other member of the
# array is a reference to another array containing the list of
# functions imported from the given DLL
@import_list = follow_imports($target_exe);


# parse the alien-doubled array
my $count = @import_list;
print "got $count entries\n";
$i = 0;
while($i < $count)
{
```

```
        $dll_name = $import_list[$i];
        $arrayptr = $import_list[$i + 1];
        my @myarray = @$arrayptr;
        #foreach $item (@myarray)
        #{
        #  print "dll: $dll_name : got import" . $item . "\n";
        #}

        # get rid of the .dll at the end
        my $safe_name;
        if( $dll_name =~ m/(.+)\./gi )
        {
          print "fixing $1 \n";
          $safe_name = $1;
        }
        else
        {
          $safe_name = $dll_name;
        }

        # now create a class to represent the imported module
        create_class($safe_name, @myarray);


        #now create a link
        print UMLFILE "edge [ arrowhead = \"empty\" ]\n $safe_name
-> $target_exe_name \n\n";

        $i += 2;
    }

    write_tail();

    close UMLFILE;
    close STUFF;

    system(".\\graph_bin\\dot -T png -o .\\class.png c:\\umlfile.dot");
```

8.1.3 字符串

绝大多数游戏包含了丰富的 ASCII 码字符串信息，这些信息包括错误消息，提示信息，有时甚至包含调试数据，如编译所使用的源代码文件名等。这些信息有助于逆向人员理解具体的函数功能。

图 8-6 显示了使用简单串函数获取的有用信息。这些信息显示了 WoW 客户端编译所用的源代码的名字以及其目录结构，查看这些信息的方法很简单，只要从游戏客户端的二进制文件中就可以获取该信息。

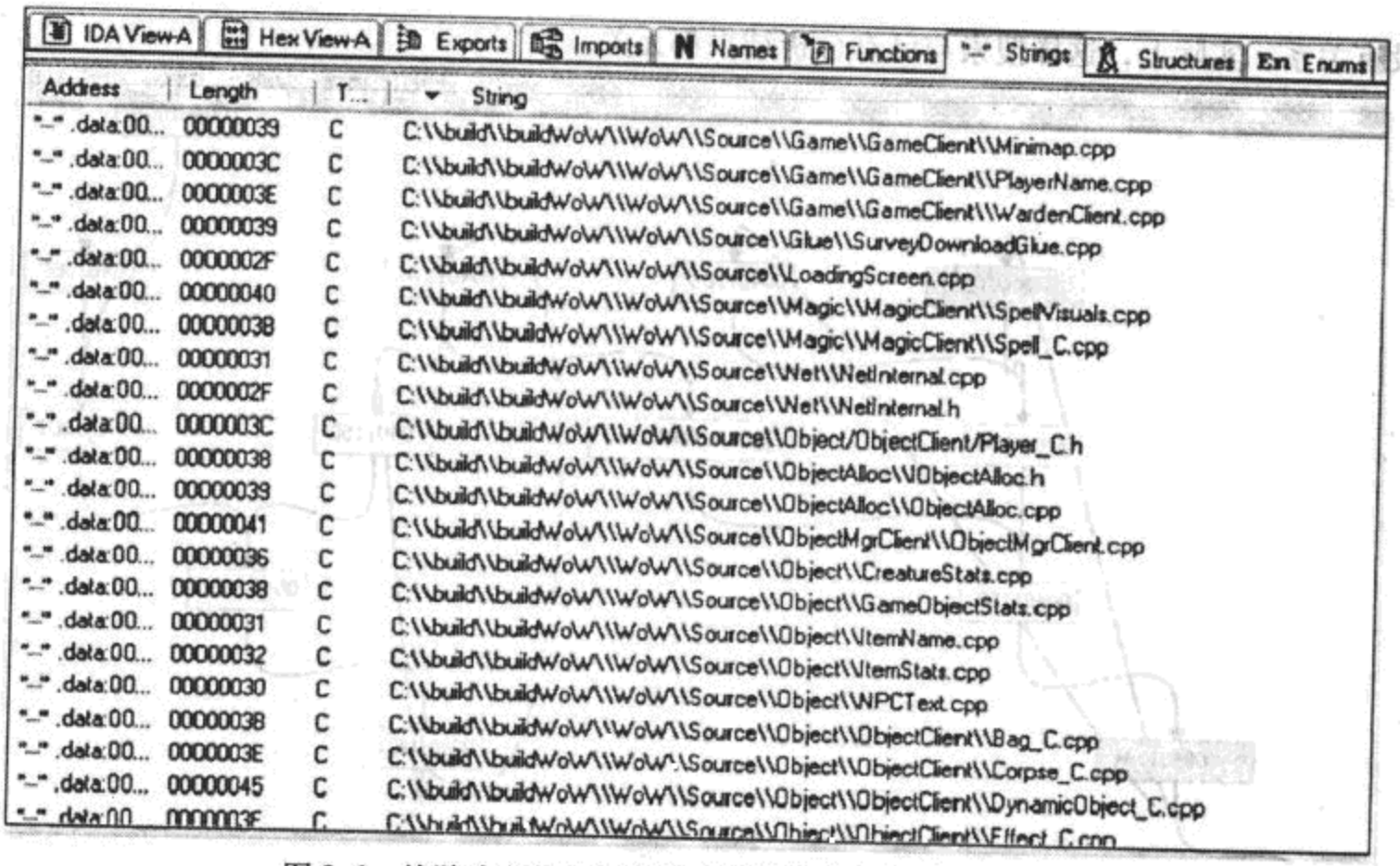


图 8-6　从游戏文件中可以查找到很多非常有用的字符串，
在这个截屏中，可以看到 WoW 客户端的源文件名

8.1.4　静态分析

上面已经简单介绍了对象，函数，套件的基本概念，现在讲解静态分析。静态分析是不运行文件，而通过静态查看二进制文件来理解程序的代码做什么[⊖]。使用静态分析技巧，可以查看程序控制流程（见图 8-7）。

另一种有用的静态分析技巧是在代码中搜索匹配固定模式的代码。模式可以为任何特征：比如内存分配、使用特定指针或者特定函数等。

图 8-8 显示了模式搜索网络相关操作 recv 获取的结果。用于搜索的模式为，任何输出信息到屏幕的代码。搜索标准为任何包含%s 或者%d 的字符串，串相关函数使用这些字符串来输出信息，图中 3 个暗色区域为输出信息到屏幕的代码。这样的搜索技巧可以很快定位到有用的信息，缩短分析时间，也可用于判断哪部分代码更加值得分析。找到字符串后，可以检查相关代码区域（图 8-9）。

搜索解析器和内存分配代码也可以查看很多有用信息，因为内存解析相关代码常常有很多安全问题。

另外一个搜索检查代码的方法为检查所有的二进制代码，并反汇编，从而得到感兴趣的代码区域。反汇编中对任何匹配特定模式的代码打上标记，记录相关信息。比如，图 8-10 显示了扫描二进制代码后发现的所有单字节操作相关的代码片断。这种搜索和前述查找 recv 函数的方

⊖ 请查看 McGraw 的《Software Security：Building Security In》（Addison-Wesley，2006）的第 4 章来了解安全分析所使用的静态分析技巧及其工具。

法没有关系，实际上，这样搜索与已有分析结果都无关。

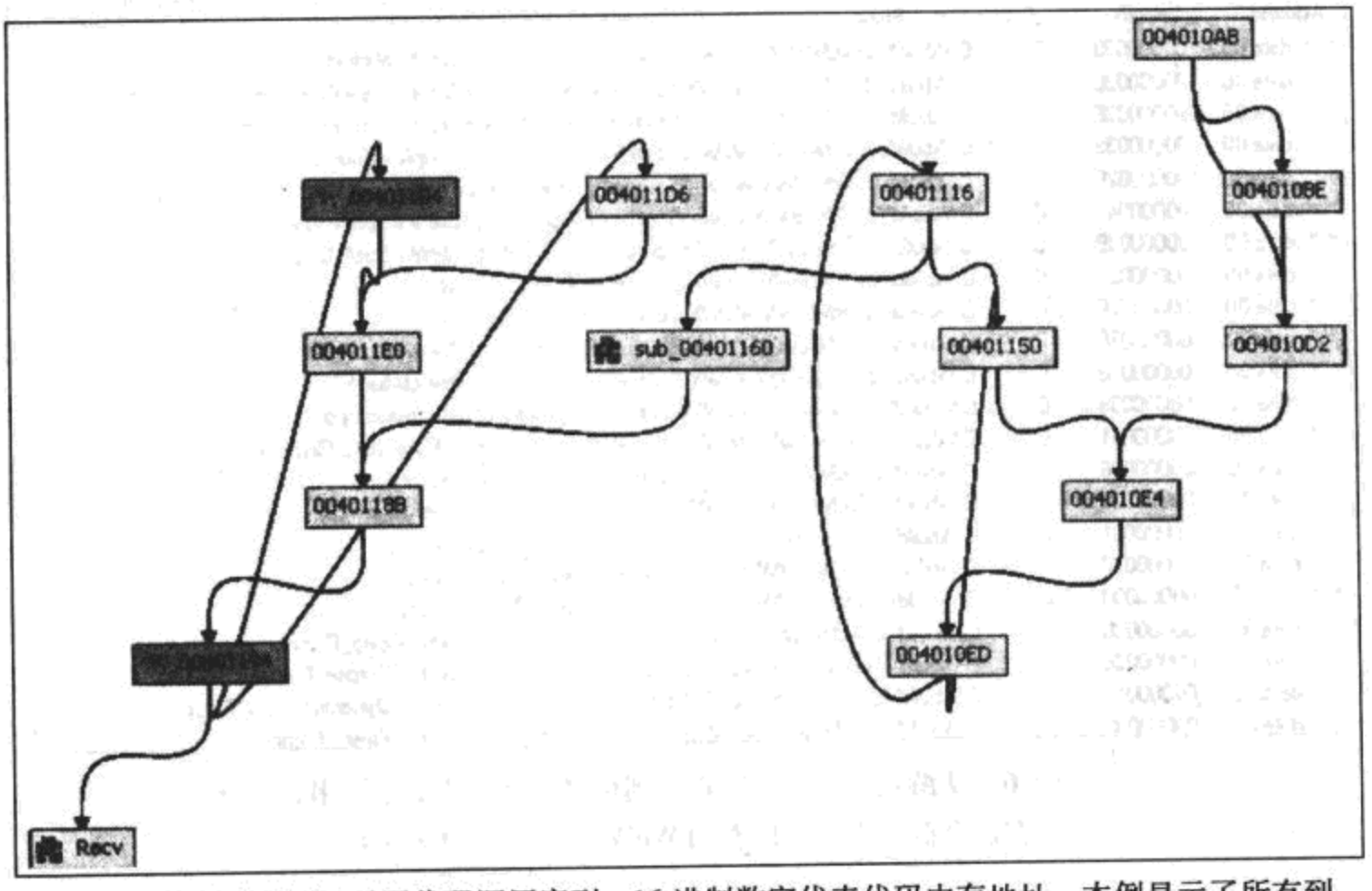


图 8-7 控制流程图显示了代码调用序列。16 进制数字代表代码内存地址。本例显示了所有到 Recv 函数调用的路径，recv 用于接受 tcp/ip 网络数据包。生成该类图形的方法有：分析目标程序或者通过反编译器生成。作者使用的 HBGary 的逆向工具 Inspector 生成控制流图。（http：//www. hbgary. com/technology. html）

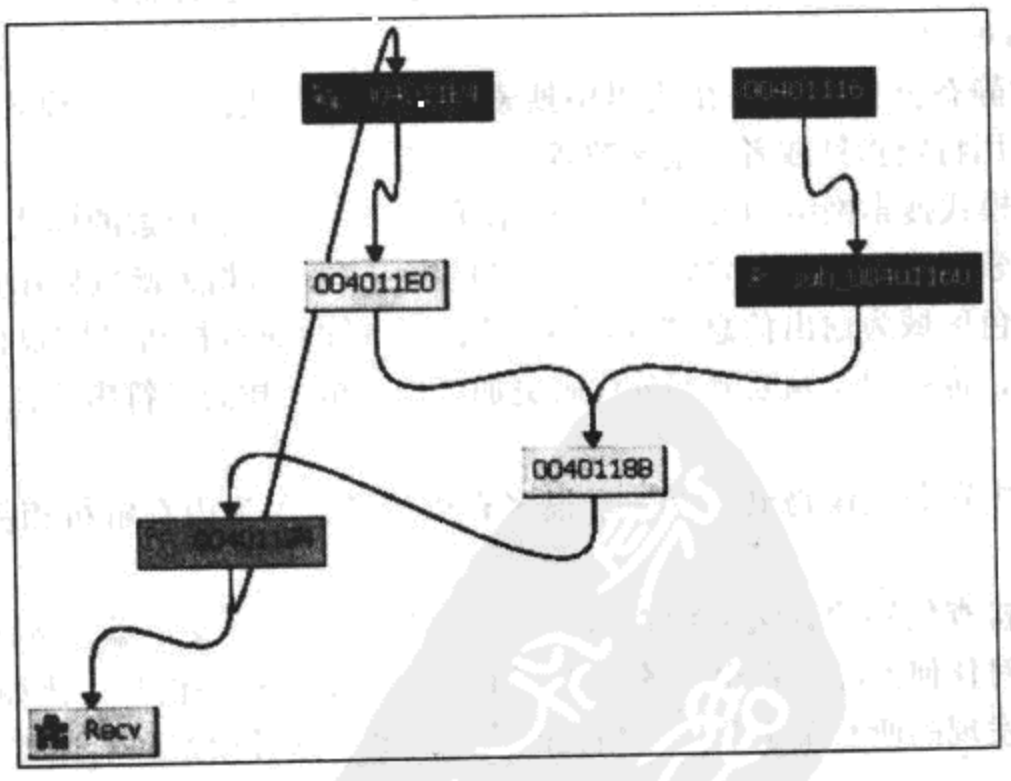


图 8-8 模式匹配搜索获取 recv 函数调用相关代码。本例中，暗色区域代表屏幕输出信息代码段，代码从图 8-9 的代码列表获取

```
00401107    mov [ebp-000001C0], eax
0040110D    cmp dword ptr [ebp-000001C0], FF
00401114    je 00401150
            mov ecx, [ebp-000001AE]
            push ecx
            call 15
            movzx edx, ax
            push edx
            mov eax, [ebp-000001AC]
            push eax
            call 12
            push eax
            push 40718C //[*] Client accepted from: %s:%d
            call sub_0040126E
            add esp, C
            mov ecx, [ebp-000001C0]
            push ecx
            call sub_00401160
            add esp, 4
00401150    jmp 004010E4
00401152    xor eax, eax
00401154    mov esp, ebp
00401156    pop ebp
00401157    ret
```

图 8-9　搜索 recv 相关操作时发现的一个代码块，本代码明显输出字符串信息

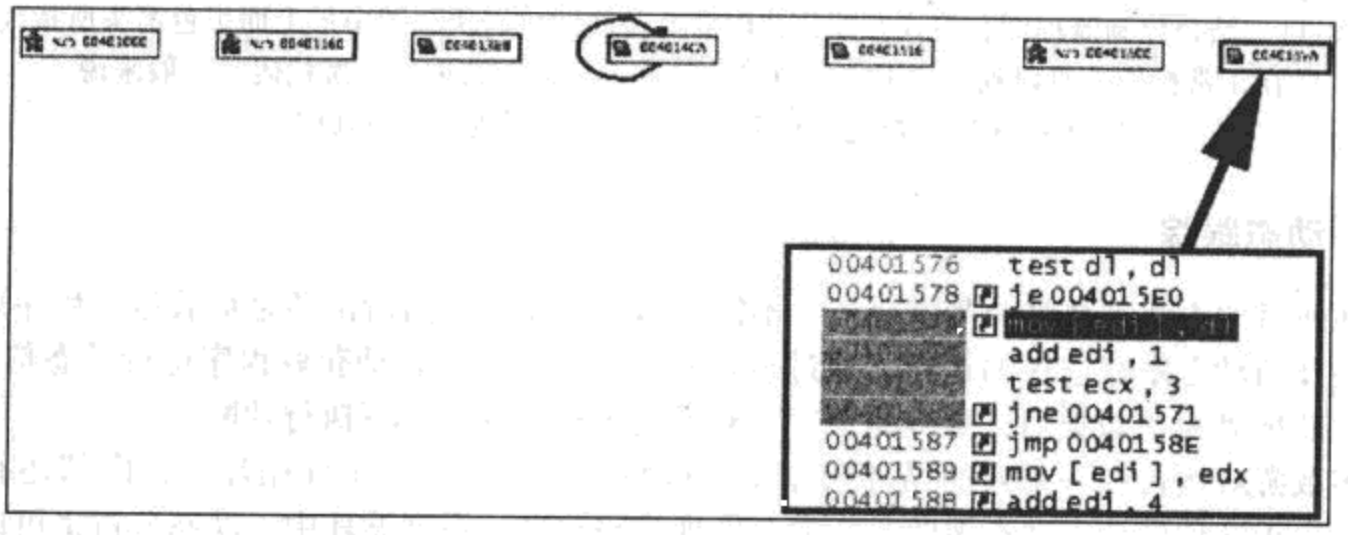


图 8-10　从整个二进制文件中搜索出所有单字节操作代码。方法很通用，但是可以产生有效结果

对二进制文件全面扫描包含的信息非常丰富，搜索到的代码有可能有用，也可能和需要分析的内容毫无关系。举例说明一下，假设逆向人员想要搜索处理聊天信息文字颜色的相关代码。图 8-10 的方法将会查找到包括颜色处理外的很多代码，定位具体的代码就好比是在一堆稻草中寻找绣花针一样困难。但是可以肯定的是，相对于搜索之前，稻草已经少了很多。

从上面介绍的方法中，逆向者可以获取到控制流图（图 8-7）和模式匹配获取的结果（图 8-8），这样就可以定位需要分析哪些代码，哪些代码能够被程序运行。也就是说，检查出操作和具体的代码相关联的地方。运气好的话，可以找出更多的程序关联。最终可以假定出如何到达具体的代码区域。图 8-11 显示了关联视图。

逆向过程中，现在的情况是安全分析和黑客大大不一样了。安全分析师需要注意整个程序的安全性及各方面的复杂度，而黑客只需要关注有价值的目标，而放弃其他内容。从这个角度来说，软件安全更加对黑客有利，这看起来很不公平，但是事实就是如此。可以看到的情况是，游戏开发人员目前处在失败的一方，游戏黑客仅需要找到任意一种方法来破解游戏，而游戏开发人员则需要防止所有的破解方法，也要关注游戏的各个方面。

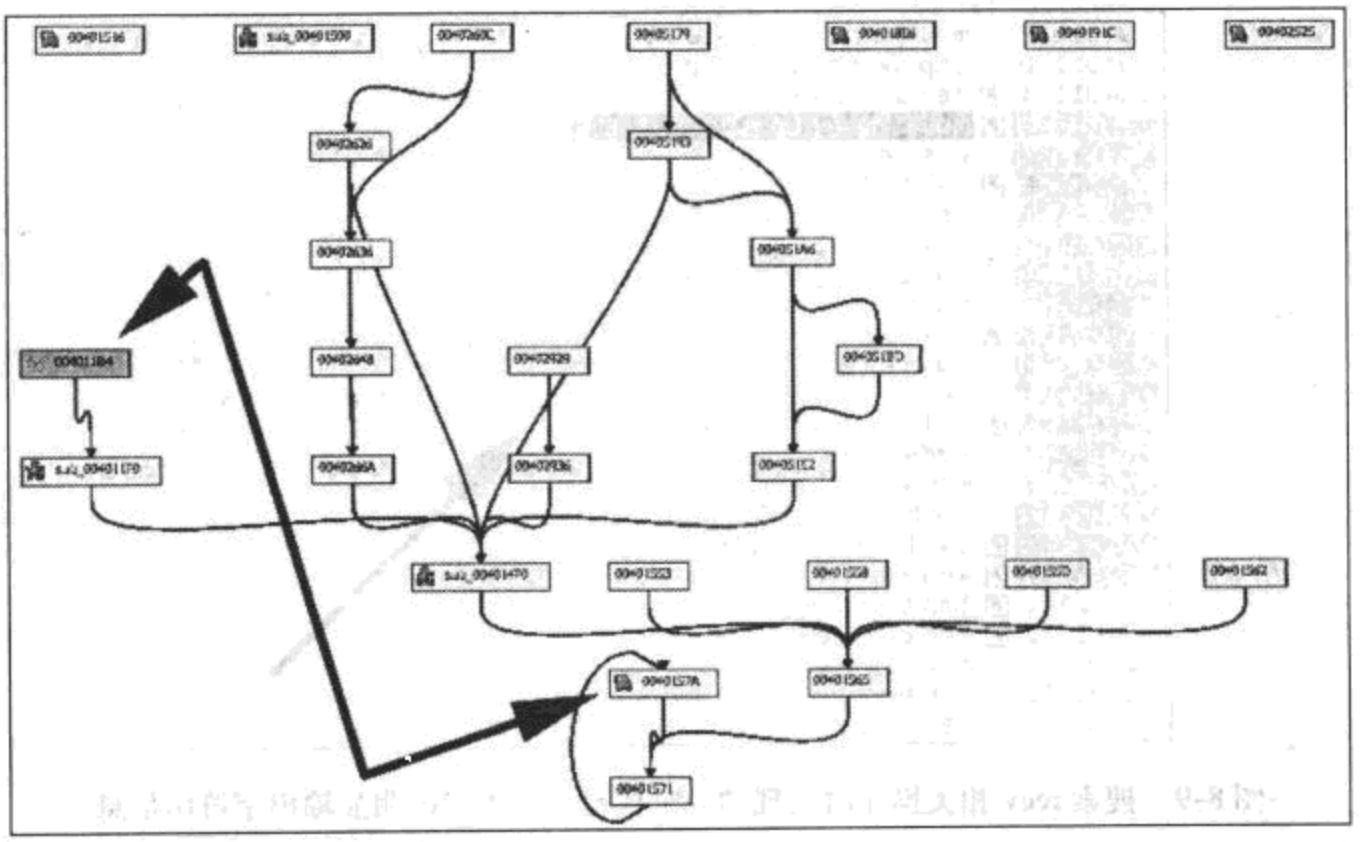


图 8-11 结合控制流程图和模式匹配，获取需要关注的代码块。图中最上面黑色箭头所指代码在正常操作下可以到达。下方黑色箭头指向模式匹配获取的可疑代码，一般来说，可以通过检查代码知道如何通过该路径到达下方代码处

8.1.5 动态跟踪

现在所讲的方法主要涉及静态分析，不需要运行程序。下面所讲的是加载运行程序后，利用静态分析技巧动态跟踪程序的运行。通过覆盖测试分析，可以自动获取程序运行动态信息。（见第6章）获取到覆盖数据后，检查是否存在有用的各种不同情况下执行片断。

大多数游戏运行在一系列的循环操作中。这些循环大体上为事件循环，在内部还有事件循环嵌套。由于这种设计，动态跟踪时常常会发现代码跑在一系列循环中。动态分析也可以根据特定测试下动态运行的所有代码来构建消息流程图。下一步可以使用该流程图作为分析的起点，扩展到静态分析的结果中。图 8-12 给出了可以构建的图表实例。

由于现在软件都比较大，特别是游戏软件，如果没有框架性的地图文件（map），在分析过程中容易迷失在大堆代码中无法自拔。下面介绍如何定位代码段并从那里开始分析，而不是将整个代码作为一整块来分析。

可以将动态分析获取的流程图和静态分析的控制流程图结合，从而得到有用的结果。图 8-13 中显示了这种结果图，后面文中将这种图称为软件地图。示例中可以看到，一部分子程序在程序运行中没有执行到。这些路径可能在其他条件下才会执行到，具体的情况需要逆向分析才能知道。

通过上述方法构建软件地图，可以将整个软件视为一个巨大的城市。如果要在城市游玩，需要定位到具体的街道，在图中做标记等，从感兴趣的地点出发，又可以感知更多的地点。笔者分析过程就是这样的：首先设定好目标区域列表，一个一个分析相关区域，分析过程中可以将这些点连接起来，从而得到大的视图，对软件有更好的了解。

现在，逆向工作才刚刚开始，困难的部分现在随之呈现在读者前面：找到有用的代码之后，如何读懂代码区域。

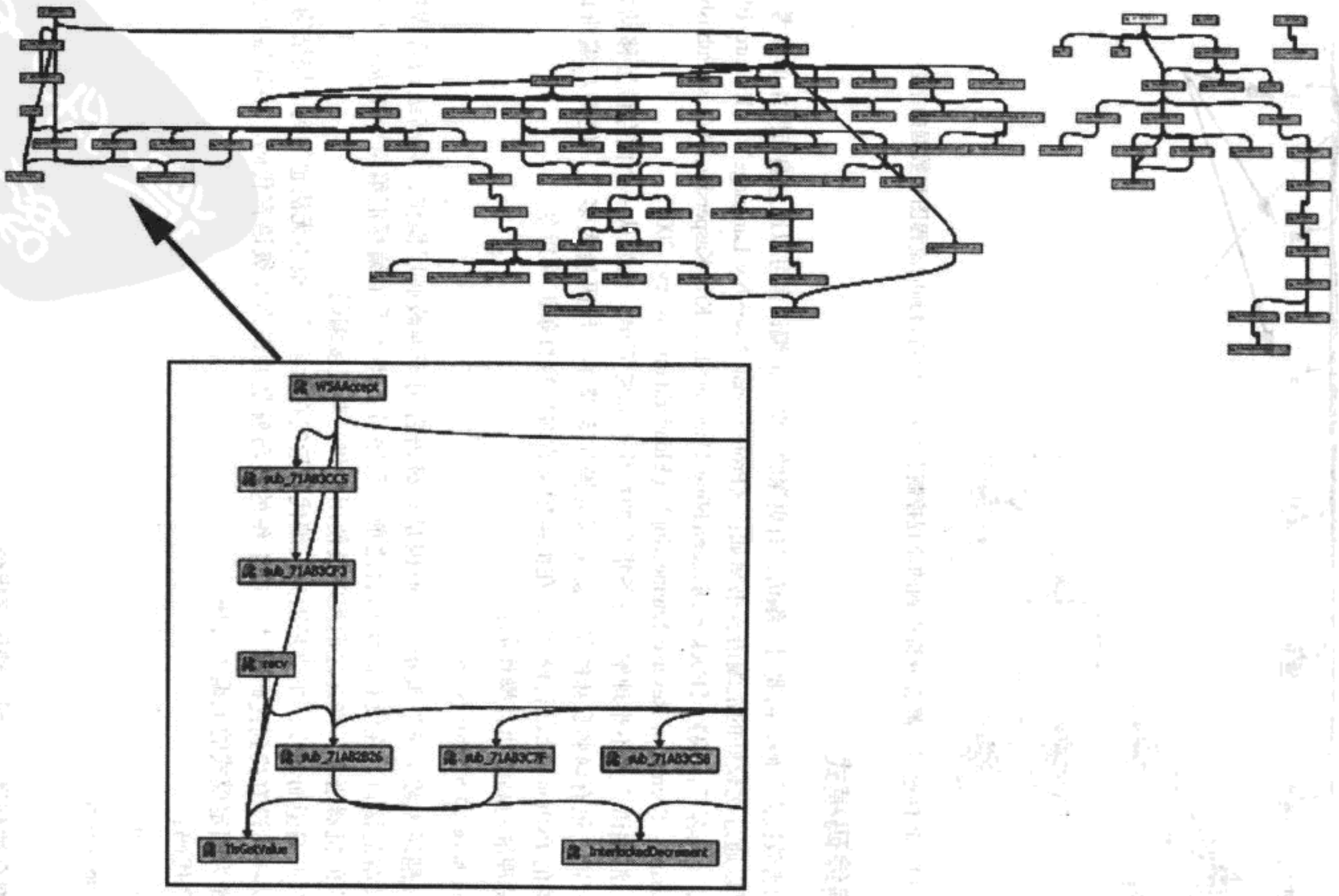


图8-12 动态分析游戏正常运行时的执行路径，主要是从起点分析执行过的代码。从初始执行点，可以更进一步分析其他区域

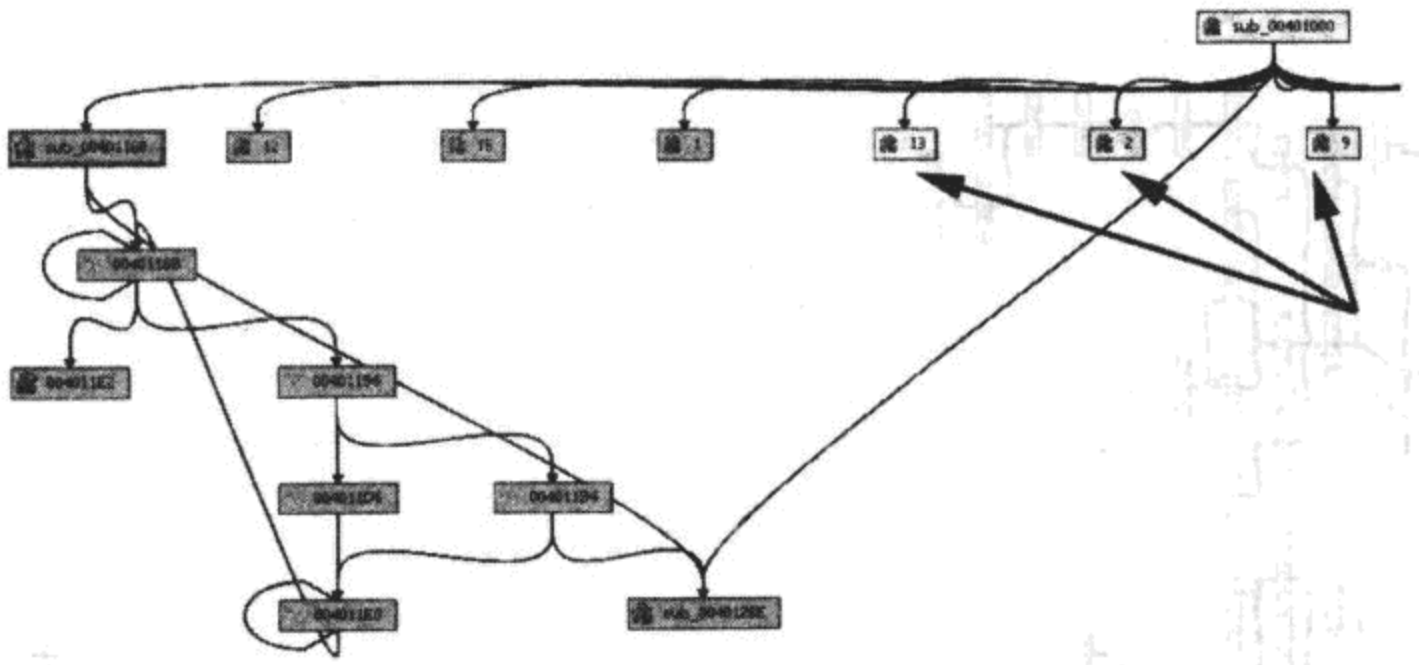


图 8-13 软件地图：静态分析得到的流程控制图与动态分析得到的流程图和数据的结合

8.2 汇编编码模式

为了更加有效读懂代码，需要对汇编语言有更深的了解。汇编语言可以通过遍布网络的参考手册来学习，也可以通过比较好的汇编语言书籍如：《Professional Assembly Language》（Richard Blum, Wrox, 2005），《黑客反汇编》《Hacker Disassembling Uncovered》，Kris Kaspersky, A-List Publishing, 2003），《Reversing：Secrets of Reverse Engineering》（Eldad Eilam, Wiley, 2005）。

事实上，汇编语言并不难理解。每条指令的操作都非常简单，比如从一个内存位置移动数据到另外一个位置。很有讽刺意味的是，就是因为操作太简单，代码量太多了，使得汇编难以分析。一条简单的 Python 高级语言操作，在汇编中常常对应数百条汇编指令。

一个容易理解的高级语言操作如：

```
PRINT "You have 30 hitpoints left"
```

有可能被翻译为数千条汇编指令，原因是高级语言用于屏蔽底层操作，给高级语言程序员提供更容易理解的接口。在没有高级语言之前，程序员不得不手工编写汇编指令来完成所有工作，或者对应的机器语言，在这之前程序员使用0和1来直接编码。

很多人对汇编望而却步，主要是因为汇编代码量实在太多了，根本无法感知到上层结构。这主要是因为这些人没有找到代码模式，反汇编的技巧就是要识别高级语言代码。编译器将高级语言声明编译为固定模式的汇编语言块。

高级语言代码：

```
if(somevar == 0)
{
        //do something
}
// after
```

基本上都会被翻译为下列汇编语言指令：

```
mov  eax, somevar
test eax, eax
jnz  after
// do something
...

// after
```

使用简单的模式识别技巧，可以将汇编语言转化为高级伪 C 语言。

本小节讲解了 C 和 C++ 常用的重要转化方法。文中将只针对 X86 汇编指令来讲解，因为 X86 是目前最流行的平台。另外本篇介绍的是以 C 为中心的方法，忽略了无数种其他高级语言，（注意：其他语言也会和 C 生成的差不多的汇编指令）。但是，大多数游戏都采用 C 或者 C++ 写成，所以学习这个技巧是一个好的开始。值得一提的是，EVE Online 大型多人在线游戏开始使用 Python 编译。

8.2.1 数据传送操作基础

计算机的两个基础操作为：算术运算，数据移动。数据移动很直观，基本上都是使用 MOV 指令来从一个地方读取数据，保存到另外一个地方。如果 MOV 指令存在于循环中，就可以移动数组数据了。文中 MOV 指令的语法为：

MOV [目标位置],[源位置]

这是 *Intel* 风格的汇编格式。第一操作数为目标数据存放地址，第二操作数为数据源。MOV 指令有多种格式。

- MOV EAX,EBX

寄存器操作，等价于 EAX 等于 EBX。

- MOV EAX,[EBX]（间接寻址）

从内存读取数据到寄存器，本例中，EBX 为内存指针，指令将 EBX 所指的内容放到 EAX 中。EBX 本身不受影响。等价于 EAX 等于 EBX 所指内容。

- MOV EAX, [EBP + 10]（函数参数）

本例中，EBP 为指针，操作把该指针加 10 处数据移动到 EAX，EBP 本身不变，操作等价于 EAX 等于 EBP + 10 处的内容。一般来说，在函数中，EBP 加上某个值为函数参数。

- MOV EAX,[EBP - 10]（局部变量）

和上面类似，只是，这次的操作为 EBP 减 10，在函数中，EBP 减去某个值为函数的局部变量。

- LEA EAX, [EBP + 10]（获取指针）

该指令将目标操作数地址保存到 EAX 中。尽管指令中使用了大括号，但是该大括号被忽略了。本例中，操作等价于 EAX 等于 EBP 加 10，等价于下面的汇编指令：

```
MOV EAX, EBP
ADD EAX, 10
```

这意味着 EAX 指向栈中的变量，变量指针是 C 语言中传递参数到子函数的方法，等价于下面的 C/C++ 代码：

```
int *pInt;
int some_int = 0;
pInt = &some_int;
```

本例中，LEA 指令用于初始化 PINT。

全局数据

全局数据和代码分开存放，一般都保存在 .data 区段中。这些数据可以在程序运行前就存在，也可以在程序运行过程中修改。不管怎样，都比较容易识别全局数据。下列字符串存在于 WoW. exe 中：

```
.data:0083B7D8 'SPELL_FAILED_ALREADY_AT_FULL_POWER',0
```

下面的代码片断中引用字符串：

```
.text:006D4D92    mov     eax, 83B7D8h
.text:006D4D97    retn
```

注意，全局数据通过地址引用，检查该地址可以知道引用 ASCII 字符串为：SPELL_FAILED_ALREADY_AT_FULL_POWER。

在字符串解析一节中，我们将更多地介绍数据移动操作。

8.2.2 比较操作基础

基础逻辑涉及值与值之间的比较，取决于值的大小，以及分支走向。再结合算术，这是所有标准算法的基本。

比较操作

典型比较操作中，两个数值相互比较。数值常常保存在寄存器中，有时有一个数值保存在内存中，比较结果决定了条件分支的走向。所以，比较操作常用于 if 和 while 等语句中。下面为比较的简单实例：

- CMP eax, ebx
- CMP [eax],ebx
- CMP al, 3Ch
- TEST eax,eax

比较指令比较有意思，该指令和 SUB 指令一样操作，仅仅不保存结果到第一操作数。指令对两个操作数作减法操作，如果结果为0，则比较结果为真。TEST 指令专门用于判断0，常常用于返回值判断。

Test 操作

与 CMP 和 SUB 的关系一样，TEST 和 AND 的操作一样，但是没有保存结果到第一操作数。TEST 也用于控制条件分支。

TEST 常用于判断函数返回值是否为0，下面为一个示例：

```
CALL some_function
TEST EAX, EAX
JZ somewhere
```

等价于：

```
result = some_function();
if(result)
{
  // do something
}
```

Wintel 平台（Intel 平台上跑 Windows）上返回值常保存在 EAX 寄存器中，这就是为什么函数调用返回后，CMP 和 TEST 指令需要比较 EAX。

相信读者都了解了 CMD 和 TEST 指令用于控制条件分支。该指令通过影响标志寄存器来影响代码执行流程。标志寄存器用于指示运算结果是否有溢出，结果是否为 0，要不要进位等。条件分支相关指令通过检查标志寄存器来决定分支走向。

- TEST 影响标志寄存器的下列标志：
  - 符号标志（SF）
  - 零标志（ZF）
  - 奇偶校验标志（PF）
- CMP 指令影响下列标志寄存器位：
  - 零标志（ZF）
  - 溢出标志（OF）
  - 符号标志（SF）
  - 进位标志（CF）

如果 ZF 为 1，表示两个操作数相等，否则说明两个操作数不等。操作数不等的话，溢出标志、符号位和进位标志用于指出到底哪个数大。

函数调用的真假判断

程序常常需要判断布尔值（保存真假），有时函数也会返回布尔值。比如，使用 Ellipse 画椭圆，MSDN 中函数文档如下 < http：//msdn2. microsoft. com/en-us/default. aspx > :

供您参考　Ellipse 函数用于画椭圆。椭圆中心为指定矩形的中心。椭圆使用当前的画笔和刷子生成。

```
BOOL Ellipse(
  HDC hdc ,          // DC 句柄
  int nLeftRect ,    // 矩形左上角横坐标
  int nTopRect ,     // 矩形左上角纵坐标
  int nRightRect ,   // 矩形右下角横坐标
  int nBottomRect    //矩形右下角纵坐标
);
```

返回值：

函数成功，返回真，否则返回假。

可以看到，函数返回值为布尔值。程序员有可能需要检查返回值才知道函数是否执行成功。

```
if( FALSE == Ellipse( hdc, r1, r2, r3, r4 ))
{
      // error
}
```

汇编语言中，实现该功能的代码为：

```
mov     edx, [ebp-r4]
push    edx
mov     edx, [ebp-r3]
push    edx
mov     edx, [ebp-r2]
push    edx
mov     edx, [ebp-r1]
push    edx
mov     edx, [ebp-hdc]
push    edx
call    Ellipse
add     esp, 4
movzx   eax, al
test    eax, eax
jz      short loc_401128
mov     ecx, [ebp-var_8]
```

该代码段中有几处有意思的代码。首先，所有函数参数压栈，注意，所有参数是以参数相反的顺序入栈的，可以看到第一个参数是最后压入栈中的。压栈后，hdc 为栈顶元素。这里，可以了解到参数是如何通过栈传递给函数的。当分析代码时发现 push 操作后跟有 call 操作，基本就可以认定 push 操作的为函数参数。

注意，传递到栈中的值首先放到寄存器中。使用 EBP 操作数为常用规范，EBP 指向栈的基地址，该片区域内存被当前的函数临时使用。这样一来，观察到 EBP，就差不多可以肯定变量在栈上，有可能为局部变量，也可能为函数的参数。本例中，矩形数据和显示上下文保存在局部变量中，逆向后代码可能为如下：

```
SomeReturnType SomeFunction( SomeNumberOfArguments )
{
       int r1, r2, r3, r4;
       HDC hdc;
       ...
       if( FALSE == Ellipse( hdc, r1, r2, r3, r4 ))
       {
         // error
       }
       ...
}
```

通过检查少数的汇编指令，可以理解当前函数的许多操作了，下一步压入参数，调用 Ellipse，该函数将返回值保存在 EAX 中。接着的操作为把 AL 扩展到 EAX：

```
movzx   eax, al
```

该操作常常发生在当 EAX 为布尔值的时候，通过这个特征，就可以知道 Ellipse 返回布尔值。当然，从文档中读者已经知道该函数返回值为布尔值。但是在分析未知函数（如游戏开发中所使用的内建代码）时，该特征有助于定位函数返回值的类型。

分支操作：if

上面已经讲解了比较操作，现在分析高级语言中分支结构如何编译为汇编语言的比较和跳转指令。if 为常见的高级语言分支语句，常常用于检查值的合法性等。

```
if(somevar == 0)
{
  //do something
}
// after
```

该声明被转化为下面的汇编语言指令：

```
mov eax, somevar
test eax, eax
jnz after
// something
...

// after
```

上面的代码中可以看到需要比较的数据首先放到寄存器中。前面也曾经看到过 PUSH 操作也有类似行为。不少指令需要指令操作数在寄存器中，而不是在内存中。本例中，somevar 首先移动到寄存器，然后使用 TEST 指令检查寄存器的值是否为 0。jNZ 指令的含义为如果不为零则跳转，如果寄存器的值不等于零，则跳转到块 after，如果寄存器的值为零，则跳到块 someting。

注意 jmp 指令和 if 语句的含义相反，if 语句表示如果相等则执行，jNZ 指令的含义为如果不等于零则跳转。跳转指令和 if 语句效果相反，但是，可以这样看待跳转指令，JNZ 等价于 if (something = = something2)，JZ 等价于 if(someting ! = something2)。

分支操作：if else

另一个常见的分支操作为 if-else，该语句的语法为：

```
if(somevar == 1234)
{
  //do something 1
}
else
{
  //do something 2
}
// after
```

代码对应的汇编指令为：

```
mov eax, somevar
cmp eax, 1234
jnz something_2
// something_1
...
jmp after
// something_2
```

```
...
// after
```

和上面 if 的例子不一样，这段代码根据比较结果，会分别执行两个不同的区块。这种类型的代码得关键匹配模式为 something_1 代码块中的无条件跳转指令。JMP after 指令必须存在，否则，两段代码执行有重叠，只有使用无条件跳转，才将两个区块隔开。逆向过程中发现类似的代码，就可以联想到 if 语句后面可能跟了 else 语句。

有时 else 代码块中还包含了比较，如：

```
if(somevar == 1234)
{
  //do something 1
}
else if(somevar == 5555)
{
  //do something 2
}
// after
```

这种情况下，汇编代码如下：

```
mov eax, somevar
cmp eax, 1234
jnz something_2
// something_1
...
jmp after
// something_2
cmp eax, 5555
jnz after
...
// after
```

从上面可以看到，多了条件判断的情况下，唯一的区别就是 something_2 代码块多了一个比较操作，如果 somevar 不等于 5555，则代码将不会执行。这里的模式是 something_2 后面跟的比较指令。

逻辑运算

与或等逻辑运算符常常用于在程序中控制分支走向。比如，下面为一个使用与操作的示例代码：

```
if(somevar == 7 && somevar == 42)
{
  //do something 1
}
// after
```

该代码和前面分析的 if 语句很像，但是现在判断条件有两个。分支逻辑几乎还是一样的，只不过前面多了一个比较操作。

```
mov eax, somevar
cmp eax, 7
```

```
jnz after
cmp eax, 42
jnz after
// something_1
...
// after (aka "bail out")
```

可以看到，所有的比较操作都在开始完成，如果有条件不满足则跳转到 after 区块执行，所有的跳转指令都为 JNZ。

对于或操作，代码变化也不大：

```
if(somevar == 7 || somevar == 42)
{
  //do something 1
}
// after
```

对应的汇编代码为：

```
mov eax, somevar
cmp eax, 7
jz something_1
cmp eax, 42
jz something_1
jmp after
// something_1
...
// after (aka "bail out")
```

与操作一样，所有的比较操作都在代码块开始的地方完成，条件不满足则直接跳到代码尾部执行。本例中使用的跳转指令为 JZ。

对于任意的比较操作，比如如下代码：

```
if( something && something || something )
{
  // do something
}
// bail out to here
```

在 if 代码块后面为出口，所有跳转到该点的操作对应一个条件判断。

- JNZ 到出口对应 if(A == B)
- JZ 到出口的操作对应 if(A! = B)

如前所述，跳转类型和高级语言操作中的表示正好相反。

8.2.3 字符串操作

循环和递增指针

指针可以指向任意对象，结构体、函数或者字符串等。只要有数组对象，都可以使用指针遍历数据。比如字符串，实际上就是字节数组，在字符串中查找特定字符，汇编代码可能如下所示：

```
mov     edx, [ebp+var_4]
add     edx, 1
mov     [ebp+var_4], edx
```

代码的操作为把一个值存放在局部变量中（注意 EBP），上面的代码首先将局部变量读到寄存器中，加 1，然后再放回内存。该局部变量的值可以表示很多意思：计数器、数组索引或者一个实际的字符串指针。

如果在内存中复制字符串，常常用循环操作来完成字符串复制。内嵌字符串复制代码在逆向代码中很常见，循环操作也可以用来在字符串中搜索特定的字符。逆向中常常可以发现小的循环操作，如果其中包含算术指针，基本上可以肯定代码在扫描或者复制字符串。请看下例：

```
loop:
        mov     al, [ecx]
        mov     [edx], al
        inc     ecx
        inc     edx
        test    al, al
        jnz     short loop
```

该代码复制以 0 结尾的字符串到另外的内存空间，类似的代码也可用来扫描任何字符。比如：下面的代码复制内存直到碰到“:”字符。

```
loop:
        mov     al, [ecx]
        mov     [edx], al
        inc     ecx
        inc     edx
        cmp     al, ':'
        jnz     short loop
```

可以看到，上例中 ECX 和 EDX 保存的值为内存中字符串指针，代码主要将 ECX 所指字符串复制到 EDX 所指的字符串中。复制过程中 EDX 和 ECX 同步增长，直到循环结束。

串拷贝

在内存缓冲区中复制字符串是很常见的操作。下面的代码为内嵌的 strcpy 函数，编译器编译过程中将所有 strcpy 调用直接替换为下面的代码。这种内联函数主要是出于性能的考虑，不需要对堆栈进行操作，也不需要函数调用，同时，该类代码的模式也比较容易识别：

```
shr     ecx, 2
mov     [ebx+10h], edi
rep movsd
mov     ecx, eax
and     ecx, 3
rep movsb
```

字符串长度在 ECX 中，该长度除以 4（shr ecx, 4），然后按照 DWORD 大小复制数据（一次 32 位，这个大小也是通用寄存器本身的大小）。这样做的好处是比单字节操作大大提高了性能，一般来说，源字符串由 ESI 寄存器指定，目标地址有 EDI 指定。

因为字符串不一定是 32 位对齐的，最后的几个字节用单字节复制方法复制到目标内存。

对于字符串复制，下面是常用指令：

- REP MOVS D（每次移动32位，长度保存在ECX中）
- REP MOVS B（每次移动8位，长度保存在ECX中）
- 源操作数：ESI，目标地址：EDI所指内存
- SHR 2：对32位操作时，字节长度除以4获取到双字长度
- ECX：保存字符串长度

串比较

计算机常常需要进行字符串比较操作，字符串比较可以用于很多方面，如：查找命令或者解析基于字符的协议等。下面的代码调用strncmp比较Web地址信息。

```
push    7               ; size_t
push    offset aHttp    ; "http://"
mov     eax, [ebp+arg_0]
push    eax             ; char *
call    _strncmp
add     esp, 0Ch
test    eax, eax
jz      short loc_401076
push    8               ; size_t
push    offset aHttps   ; "https://"
mov     ecx, [ebp+arg_0]
push    ecx             ; char *
call    _strncmp
```

解析－扫描元字符（metacharacter）

很多字符串由元字符串分割表示，这些字符有特殊含义，如":" 或者TAB键等。假设服务器发送过来的聊天消息包含特殊字符，用于表示字体颜色。该编码可能用特殊字符序列表示。比如[183可能用来修改字体颜色为红色。游戏通过“[”解析消息，再分别设置颜色信息。

以下代码在字符串中搜索0x3A，表示“:”字符，同时检查是否到达字符串结束：

```
mov     edx, [ebp+var_4]
add     edx, 1
mov     eax, [ebp+var_4]
movsx   ecx, byte ptr [eax]
cmp     ecx, 3Ah
jz      short loc_4010F7
mov     edx, [ebp+var_4]
movsx   eax, byte ptr [edx]
test    eax, eax
jz      short loc_4010F7
jmp     short loc_4010D7
```

可以看到，这里有两个检查，一个检查0x3A，另一个通过TEST指令检查是否到了字符串末尾。如果任一条件成立的话，代码跳到loc_4010F7执行。

MOVSX指令常用于检查空字节。下面代码片断检查是否到达字符串末尾：

```
mov     edx, [ebp+var_4]
movsx   eax, byte ptr [edx]
test    eax, eax
jz      short loc_40109C
```

查找字符串解析位置，留意如下类型的操作：

- MOV AL, BYTE PTR[EBX]

- AL, BL
- CMP AL, 00H
- TEST AL, AL
- CMP AL, 3Ch（另外有 ASCII 码如“.”，“/”，“\”，“:”，“-”也需要注意）

将 AI 替换为其他寄存器也属于该模式。

有很多字符串解析的变种，比如循环可以用于获取字符串长度，也可以用于获取特定字符的指针。下面代码循环扫描，检查 NULL 字符，最终 ESI 遍历到字符串末尾：

```
mov     cl, [esi]
inc     esi
test    cl, cl
jnz     short loc_
```

这类代码在字符串粘贴操作中常常使用。

也可以反向扫描。下面代码从字符串结尾开始扫描（很可能是上例初始化的指针），直到发现空格字符：

```
cmp     eax, edx
jbe     short loc_
lea     ecx, [eax-1]
cmp     byte ptr [ecx], 20h
jnz     short loc_
mov     eax, ecx
jmp     short loc_
```

代码也可以通过标准函数获取字符串长度或者其他操作，这随着编译器的不同而不同。下面代码使用 strlen 获取字符串结尾指针：

```
lea     eax, [ebp+var_48]
push    eax                     ; char *
call    _strlen
add     esp, 4
lea     ecx, [ebp+eax+var_48]
mov     [ebp+var_4C], ecx
```

本例中，var_48 为字符串，计算结果是返回字符串结尾的位置，ECX 指向字符串结尾。注意 LEA 用于计算新指针位置。

字符解析也可以用于跳过部分字符块，下面解析器如果发现特定字符则跳过几个字节：

```
mov     edx, [ebp+var_4]
add     edx, 1
mov     [ebp+var_4], edx
mov     eax, [ebp+var_4]
movsx   ecx, byte ptr [eax]
cmp     ecx, 3Ah
jz      short loc_4010F7
...
mov     ecx, [ebp+var_4]
add     ecx, 3
mov     [ebp+var_4], ecx
mov     edx, [ebp+var_4]
push    edx
```

如果在字符串中找到“:”字节，则指针加3，本代码检查“http：//”中的“：//”。代码是从一个实际程序中抽取出来的，实际上，假定“:”后必定为“//”是不正确的。该程序中，如果提供字符串只有“:”，而不包含“//”会导致严重的问题。

下面列出常见的分隔符：

- 5C = "\"
- 3A = ":"
- 3F = "?"
- 3C = "<"

请查看 ASCII 表获取更多的分隔符信息。（可以很容易从网上获取。）

8.2.4 函数

函数为指令集合，函数为基本指令容器，也是构成程序的基础。函数在代码中比较容易定位，因为函数有固定的入口模式和结束模式。这些结构划分了函数的起点和终点，而且一般来说，它们都是相同的。

入口模式

函数入口设置堆栈，为局部变量分配空间，创建堆栈指针的基指针：

```
push    ebp
mov     ebp, esp
sub     esp, 176
```

原始 EBP 保存在栈上，新的 EBP 指向当前的 EBP，修改 EBP，为局部变量分配空间。根据编译器选项，有时分配堆栈不修改 EBP，如果编译时指定不使用堆栈指针，EBP 的使用可能不像如上代码所示。相反，这种情况下 EBP 被看作一个通用寄存器。不使用这种固定模式的代码通常更加难以理解。

在 Windows XP SP2 系统中，可能发现函数入口点为如下格式：

```
mov     edi, edi
push    ebp
mov     ebp, esp
sub     esp, 176
```

添加的“mov edi,edi”用于支持补丁，这类指令类似于用 nop 分配代码空间。本例中，这两个字节可以被替换为短跳转，用于支持快速补丁，从而修改了调用行为。

结束模式

函数结尾也是一个可以预测的模式。很明显，调用必须用 RET 返回。代码可能看起来如下面的格式：

```
mov esp, ebp
pop ebp
ret
```

这里可以看到，EBP 和 ESP 的值被恢复，函数使用 RET 函数返回到调用点。有时可以看到返回值操作也在函数结尾：

```
mov eax, -1
mov esp, ebp
pop ebp
ret
```

本例中，函数返回 -1。有时，函数使用 LEAVE 指令返回：

```
leave
retn    3Ch
```

LEAVE 操作等价于“mov esp, ebp”和“pop ebp”。另外可以看到 RET 指令的等价体 RETN 指令。RETN 指令指定堆栈恢复的数值，将 ESP 加上一个值，等价于弹出堆栈。

调用约定

从上面的函数入口和出口等讲解，可以知道，函数调用前先把参数压栈，再调用目标函数。这个方法比较常见，但是调用不一定就非得按照这个模式来做，也可以把函数参数传入寄存器。另外有些被调用函数在调用返回前自己恢复堆栈，而有些则需要调用者恢复堆栈。（比如当需要可变参数时）

快速调用（*fast call*）常传递参数到寄存器中。比如，下面的代码中 5 被传递给函数：

```
mov ecx, 5
call ds:some_function
```

标准调用（*stdcall*）格式的函数自己恢复堆栈，从 DLL 导出的函数通常就是这种格式。该类函数通过堆栈传递参数：

```
push 5
call ds:some_function
```

C 调用约定（*cdecl*）格式的函数自己不恢复堆栈，调用者在函数返回后恢复堆栈。这样的格式可以传递可变数目参数给函数，C 库函数如 printf 之类需要可变数目的参数。这类函数比较容易定位，在调用函数后有 ESP 恢复操作：

```
push 88
call ds:some_function
add esp, 4
```

内建函数

内建函数不使用 DLL，而是直接添加到二进制代码。部分函数如 sprintf，可以作为 DLL 库调用，但是为了性能和便利性的考虑，大多数编译器将该类代码直接编译连接到二进制可执行文件中，这样，额外的 DLL 就不需要了。

示例：

```
.text:00630345                push    edx             ; va_list
.text:00630346                push    eax             ; char *
.text:00630347                push    esi             ; size_t
.text:00630348                push    edi             ; char *
.text:00630349                call    ds:_vsnprintf
.text:0063034F                add     esp, 10h
```

注意 vsnprintf 也为 C 调用格式，可以看到，代码后面有堆栈恢复操作。原因就是该函数的参数数目不固定。

内联函数

为了性能考虑，编译器会内联部分函数，如 strlen，strcpy 之类。开发人员也可以在开发中定义内联函数。内联函数在编译时直接嵌入到调用的地方，这样，就不需要 CALL，RET 之类比较费时的指令，整个代码整合到一块了。注意内联和内建不一样：内联是嵌入代码中，内建是复制到同一个可执行文件的其他地方；内联没有 CALL，RET，内建有 CALL，RET 指令。

函数片断

现代编译器常常对代码进行优化，从而得到更好的性能。Windows 编译器有优化选项如：函数分块，页面优化等。这些方法均是将函数分割为小块代码，散布在二进制代码中。对于逆向人员来说，这很恼人，因为这样以后，代码不能连续全部看到。优化以后，代码可能这里有部分，那里有部分，对于习惯于用 IDA-Pro（基于线性文件界面的反汇编器）分析代码的人员来说，这样看起来就比较麻烦，不过，还是可以通过图形格式来解决该问题。

代码分割的原因主要是考虑内存访问，试图使得最近访问的代码都放在一起，这样就减少了内存使用，也减少了 CPU 的负担。操作系统每次读入一页内存，都要消耗 CPU 资源。编译器优化以后，就可以消除不少类似的负担。这类优化技术非常高级，因为编译器必须提前预测代码使用的频率和位置等信息。

FPO 数据

某些函数不使用 EBP 来引用栈空间，而使用 ESP 来引用栈数据。这使得代码非常难读，因为 ESP 一直在变化。想象一下，采用 FPO 技术后，[ESP + 5] 和 [ESP + 200] 可能指向同一个位置（译者注：由于 PUSH 等操作会修改 ESP），带来的代码阅读难度可想而知。

变量重用

到这里为止，上面已经讲解了很多函数使用变量的方法。一种比较少见的优化方法是变量重用。如果编译器知道某个指针从某处开始不再被使用，编译器可能会重用该内存区域，用于存放处理不相干的变量。在逆向过程中，遇到这样的情况让人迷糊，因为逆向人员常常假设该内存为一个特定的操作所用的变量。例如：

```
mov [ebp+var_4], edx
push edx
call ds:function
mov [ebp+var_4], eax
cmp eax, 5
```

8.2.5 C++对象

C++是面向对象的设计的语言，而且 C++的效率比较高，所以 C++在游戏开发中是最常用的开发语言。从而，理解 C++对于逆向游戏非常关键。

ECX 作为指针

C++类包含重载函数，这些函数存在于一个指针表中，重载函数就是替换指针表格中的指

针，使之指向重载的函数。由于这种特性，C++类包含了 this 指针，用于引用当前实例。大多数游戏程序使用微软编译器编译，而微软编译器常常使用 ECX 寄存器来存贮 this 指针。下面的例子代码显示了 this 指针的使用方法：

```
.text:00403D10 sub_403D10      proc near               ; CODE XREF: sub_403AE0+74p
.text:00403D10
.text:00403D10 var_114         = dword ptr -114h
.text:00403D10 var_10          = dword ptr -10h
.text:00403D10 var_4           = dword ptr -4
.text:00403D10 arg_0           = dword ptr  8
.text:00403D10
```

下面为函数典型的入口模式。EBP 指针用于定位栈变量和参数，ESP 用于申请栈空间给局部变量。

```
.text:00403D10                 push    ebp
.text:00403D11                 mov     ebp, esp
.text:00403D13                 sub     esp, 114h
.text:00403D19                 push    ebx
.text:00403D1A                 push    esi
.text:00403D1B                 push    edi
```

注意 ECX 的使用方法，ECX 开始没有初始化，但是其中已经包含了 this 指针的内容。从这点出发，可以确定该函数为某个类的成员函数，而 ECX 包含指向类虚函数表的指针。注意 ECX 是用于加载函数表指针到 EAX：

```
.text:00403D1C                 mov     edi, ecx
.text:00403D1E                 mov     eax, [edi]
.text:00403D20                 lea     ecx, [ebp+var_10]
.text:00403D23                 push    ecx
.text:00403D24                 mov     ecx, edi
.text:00403D26                 mov     esi, edx
```

现在可以看到如何从函数表指针中获取成员函数指针，并调用函数。类一般有很多成员函数，代码调用第 6 个函数（函数表中第 0x14 的位置）。

```
.text:00403D28                 call    dword ptr [eax+14h]
.text:00403D2B                 mov     ebx, [ebp+arg_0]
```

C++虚表

C++对象的虚函数表通过指针获取，this 指针所指的内存区域的开始 4 个字节为该对象的虚表（见图 8-14）。

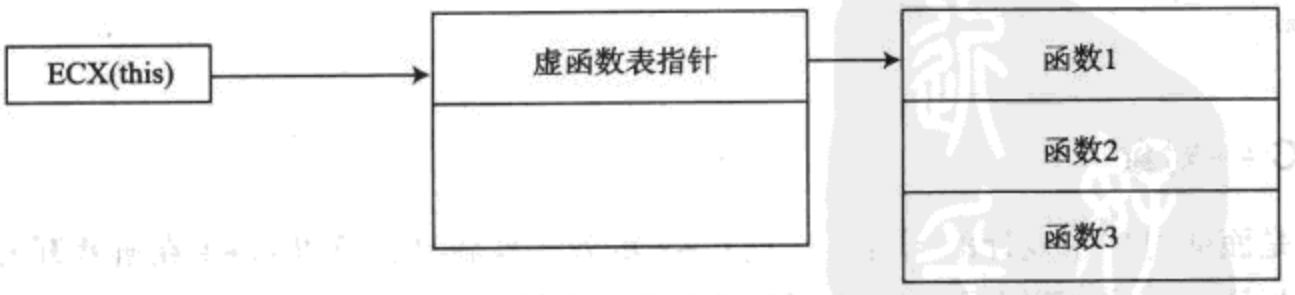


图 8-14　本图显示了如何通过 ecx 查找虚函数表和成员函数指针

8.2.6 异常处理

注册异常处理函数用于处理非正常情况。实际上，该技术不仅用于处理错误，但是本技术的设计意图是用于处理异常。异常处理设计为函数链表，按照链表调用函数，直到链表中有函数处理了该异常，有函数处理异常后，就不再继续遍历链表。

异常帧处理

异常处理通常通过异常帧（函数指针列表）实现。每个异常帧指向下一异常帧，并且包含指针指向异常处理函数。第一个异常帧可以通过 FS 寄存器获取，通过 FS 遍历到 NULL 指针，则异常处理链结束。

下面的代码中，注册了 loc_750A16DB 的异常处理函数。注意如何通过 FS[0]添加新的处理函数和保存原有的处理函数链，这个步骤生成了异常帧。然后一个指向新异常帧的指针被存储在 fs[0]中：

```
push    offset loc_750A16DB
mov     eax, large fs:0
push    eax
mov     eax, [esp+8+arg_4]
mov     [esp+8+arg_4], ebp
lea     ebp, [esp+8+arg_4]
sub     esp, eax
push    ebx
push    esi
push    edi
mov     eax, [ebp-8]
mov     [ebp-18h], esp
push    eax
mov     eax, [ebp-4]
mov     dword ptr [ebp-4], 0FFFFFFFFh
mov     [ebp-8], eax
lea     eax, [ebp-10h]
mov     large fs:0, eax
retn
```

在该汇编代码中，插入新的异常处理帧到 SEH 链。

8.2.7 switch 语句

switch 语句是一种特殊的分支结构，通过该语句，程序可以在多个条件的情况下直接跳转到合适的代码执行。switch 语句常常用于处理程序的核心逻辑，所以理解 switch 语句对于分析游戏程序很重要。

switch 表

当 switch 语句中包含多个顺序切换的值时，编译器会创建指针表，用该表来切换跳转地址。如果切换的值很无序，编译器可能也会这样创建函数指针表，因为这样性能会得到很大的提高。

会创建 switch 表的代码格式如下：

```
switch(n)
{
  case 1:
  ...
  break;
  case 2:
  ...
  break;
  case 3:
  ...
  break;
  case 4:
  ...
  break;
  case 5:
  ...
  break;
  default:
  break;
}
```

上例中 switch 表编译后的使用格式如下：

```
cmp ecx, 5
ja  default_handler
jmp DWORD PTR [dword_40112233 + ecx * 4]
```

本例中，肯定有 5 个条件，数字 1 到 5。跳转表地址为 40112233 ，表中包含了每个条件的跳转地址。注意其中的 “ * 4”，并留意这个地址对每个指针来说，大小是 32bits (4bytes) 的。

switch 树

当条件不为顺序的数，编译器采用另一种方法来实现 switch，将条件分割为小的区块，每个区块包含一小块值，每个区块一个小表格，或者小的块中有可能还会嵌套其他表格，这样，就形成了 switch 树。这样每个值不需要都被检查就可以生成高性能的代码：

会形成 switch 树的代码示例如下：

```
switch(n)
{
  case 104:
  ...
  break;
  case 220:
  ...
  break;
  case 3:
  ...
  break;
  case 44:
  ...
  break;
  case 501:
  ...
  break;
  default:
  break;
}
```

编译后，汇编代码如下：

```
cmp eax, 104
jge LOC_GROUP_2
cmp eax, 3
jz  LOC_case_3
cmp eax, 44
jz  LOC_case_44
jmp DEFAULT
LOC_GROUP_2:
cmp eax, 104
je  LOC_case_104
cmp eax, 501
je  LOC_case_501
DEFAULT:
```

本例中，可以看到，条件逐渐分割到小的区域中，直到最后包含具体数值的分支。

8.3 自修改代码及“加壳”

有时游戏开发人员会对编译出来的游戏客户端进行加壳（译者注），对二进制文件进行加密压缩等操作。这样进一步的操作使得游戏逆向人员更难对游戏客户端进行分析。很多情况下，可以通过寻找合适的脱壳机还原得到原来的文件。如果找不到合适的脱壳机，也可以通过导出进程二进制映像内存来得到解压后的二进制文件。

有时加壳软件会包含反调试手段，这使得抓取进程映像比较困难。但是，一些脱壳辅助工具也有自己的对策。总的来说，加壳过的游戏客户端分析处理起来比较麻烦，但是这并不能阻止逆向人员的脚步。加壳从某个角度来说，确实提高了逆向工程的门槛，使得不少初级逆向人员望而却步，对游戏也起到了基本的保护作用。

8.4 逆向工程总结

不管是不是针对游戏，还是其他软件的逆向技术都是非常重要的技能。逆向技术的掌握程度将初级破解者和资深逆向人员区分开来。如果读者真的对破解软件（游戏或者其他软件）感兴趣，学习逆向工程技巧绝对是值得的。

第9章　高级黑客技术

至此，您已经深入阅读了此书，了解了客户端修改、基于网络的嗅探威胁、中间人攻击以及一些常用的入侵网络游戏的黑客技术。在这一章，我们将讨论一种被称为美术模型替换改装的技术，这是一种有趣的入侵网络游戏的方式。

9.1　资源替换和改装

替换和改装（*Conversions and Modding*）这种方式常常被黑客用来实现对游戏进行定制化的修改。改装有很多类型，从创建简单的裸体模型补丁（通过改变材质和表面贴图使每个角色成为裸体，如图9-1），到完全替换所有游戏中的物件（称之为完全替换（*Total Conversions*）），方法有很多。

有时候一些修改后的效果会很不合常理或很搞笑。比如图9-2，一个无聊的游戏黑客将鱼饵的模型替换成了一尊大炮！这用一根线怎么可能撑得住……

大多数情况下，改装技术是应用于可见的图形物件上（比如模型、材质、线条）以及类似于此的物体。但是改装也不局限于仅用在可见和可渲染的物体上。事实上，从更广的范围来看，它甚至可以直接作用在一个游戏软件的底层。

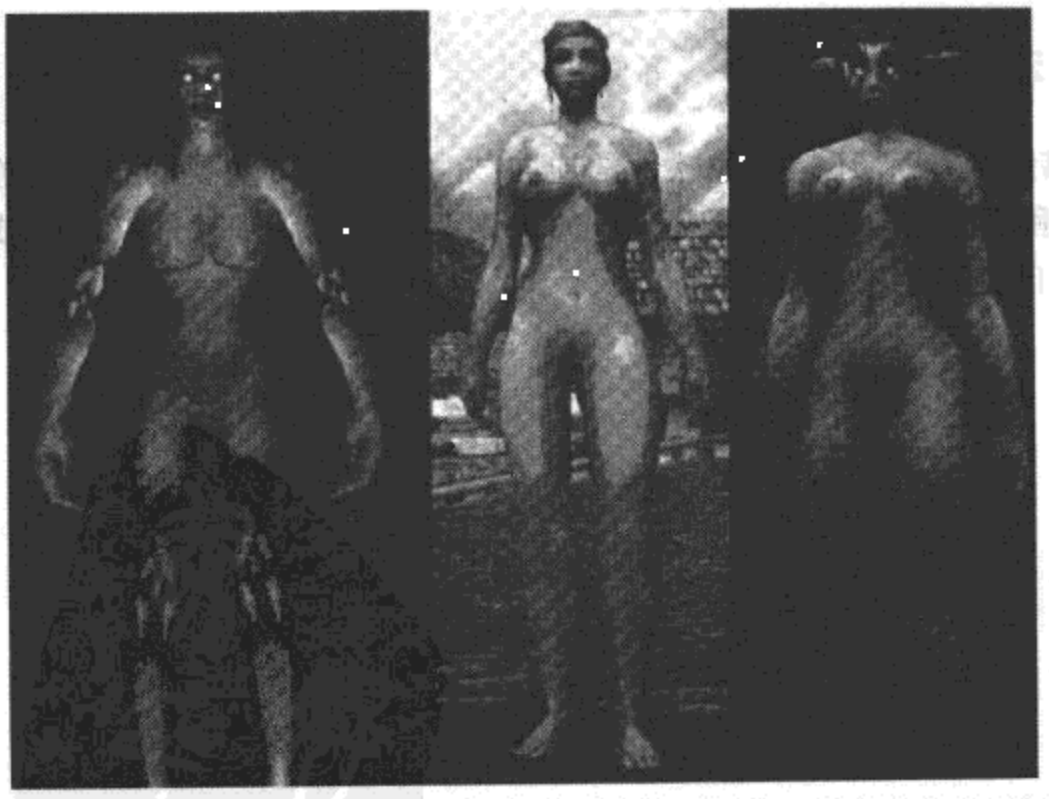


图9-1　一些黑客替换了角色模型的贴图，使模型成为了裸体。显然他们还可能会做更多（来自 http：//www. edgeofnowhere. cc/viewtopic. php? t = 314283，已授权）

改装可以改变游戏的逻辑和规则，甚至有的黑客可以完全地替换掉客户端中的物体为各种

自定义的东西。这样意味着彻底重写了渲染引擎和接口逻辑。当然黑客们有很多方式来进行这种替换，无论是简单的物件替换还是完全的客户端层面的替换。比如，一个黑客可能重修了一些图片，保留了上面一些原来的图案，同时又对渲染引擎或者客户端代码做了较多修改。

大多数改装是将图片和音乐这样的媒体文件进行了替换。它仅仅是将游戏的表面内容进行了调整，但在内部依然是原来的样子（因为没有涉及修改游戏的规则）。如果是一个完全的替换，则可以将游戏变成完全另外一幅摸样（比如，将一个有德鲁伊和绝地武士的星球大战（*Star Wars*）那样的游戏，替换成一个充满骑士与飞龙的中世纪风格游戏）。

关于改装，有些游戏公司反对，有些公司赞成。作为游戏公司，显然是希望能保持自己游戏的本来面貌和最初的体验感。假想在一个笼罩着黑暗氛围，充斥着食人魔、飞龙和剑士的世界里，如果用水仙花来替换食人魔，用巨大的电视显像管替换飞龙，剑士也成为了吹肥皂泡泡的棒棒，那这结果实在是太傻了。不过从另一个角度来想，游戏公司可以提供给喜欢改装的玩家群体这样一个机会，那么，玩家们热衷于这种改装的过程，相当于提高了游戏的乐趣，游戏的生命周期也将会被延长，最终也会给游戏公司带来更多的收入。

游戏改装是很有趣的，因为它可以让玩家的行为与游戏的结构和情节之间实现某种交互，却不需要玩家从底层开始重新编写一个游戏程序。正因如此，改装是一种了解游戏如何制作以及游戏内部各部分是如何运作的特殊方式。

图 9-2　当游戏被替换了模型后的效果往往很有意思。这是一个黑客用一尊大炮替换了鱼饵后的效果。客户端程序轻松而无错地渲染出了这个结果

9.1.1 完全替换

如果制作者想要出售游戏改装的功能包，集成完全替换（以下简称 TC）的功能是十分必要的。因为一个完全的替换将会替换所有具有版权的内容。一些人认为，这完全可以做成一个插件的形式来出售。事实证明，最成功的完全替换的例子就是反恐精英（Counter – Strike）（其实它就是 Valve Software 出的一个 Half – Life 的 TC 版），而它是被正式商业销售的。反恐精英已经被认为是一款历史上最成功的 TC，同时也是覆盖面最广的 FPS 游戏。

我们普遍认为，商业化游戏中的媒体文件是有版权的，所以替换它们（包括客户端引擎）是侵权的。为了避免这种违法行为，一个 TC 必须与本机上已经安装好的正版客户端共存。于是当 TC 运行的时候，它会替换相应的媒体文件，这就像是在已有的游戏系统上进行附着般的成长。

举个例子，一个非常普遍的 TC 是将 FPS 游戏改变成星球大战的主题。当然，如果是这样，具有版权的内容将会被替换成各种类型，但依然是授权的。这种情况下，版权不会根据新替换的内容而改变，除非你自己就是原来内容的版权所有者。另一方面，一些版权所有者并不在意这种行为，也不会去追究法律责任，所以我们看到被替换成星球大战主题的游戏有很多。

9.1.2 重写客户端

如果你想要玩一款魔幻题材的游戏，并且想要 WoW 那样的美术风格和模型设计，你可以替换客户端的可执行文件生成一个独立的游戏，来达到改装的效果。如果你也擅长写服务端程序，那么可以用你所做的客户端——服务端补丁可以架设起属于你自己的在线世界，使它不再是一个由暴雪娱乐（Blizzard Entertainment）才能运营的商业游戏。当然，想要这种方式不构成侵权，必须建立在已经安装原来客户端的前提下，因为重新布局原来游戏的任何部分都将会是非法的。

设想下面这个改装的例子：生成一个全新客户端，它用 WoW 中的角色来替换激烈对抗的赛车游戏中的车子。这样的结果，我们可以称这个游戏是“兽人大竞速”，或者“WoW 地图赛道大比拼”。这是个很好的例子，因为它需要一个全新的客户端包含物理引擎和新地图、赛车等附加内容。这样的一种修改方式，它的结构如图 9-3 所示。

WowMapView 最初是用来离线读取 WoW 中的地形文件（从硬盘或者缓存中），它允许用户在离线状态下对 WoW 的世界进行访问。WowModelView 也一样，它是负责游戏的渲染机制的，也是可以让用户在离线情况下去访问游戏中的模型。这两个工具都依赖于 OpenGL 底层库（第 7 章已简要描述过）中的 3D 渲染文件。

我们提过，在拥有游戏引擎的基础上并且拥有 WoW 客户端提供的数据库，我们就可以生成一个独立的程序，它可以还原 WoW 世界的部分内容。图 9-4 显示了一个将 WoW 中两个场景结合起来的效果：贫瘠荒地上矗立着兽人居住的房屋（它们原本是不住在这片区域的）。

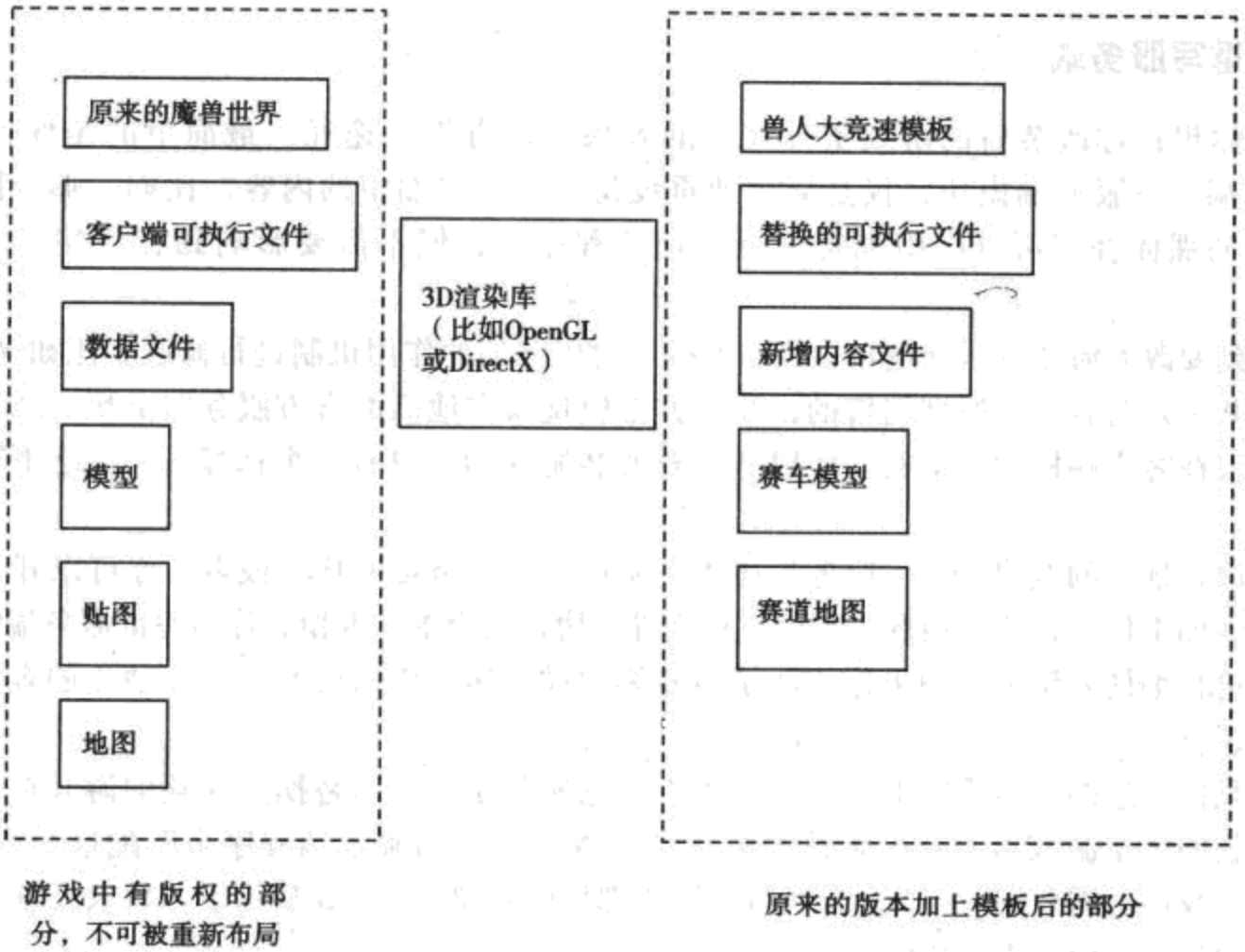


图 9-3　被称作“兽人大竞速”的 WoW 的 TC 版。注意有些部分被替换了，有些依然保留

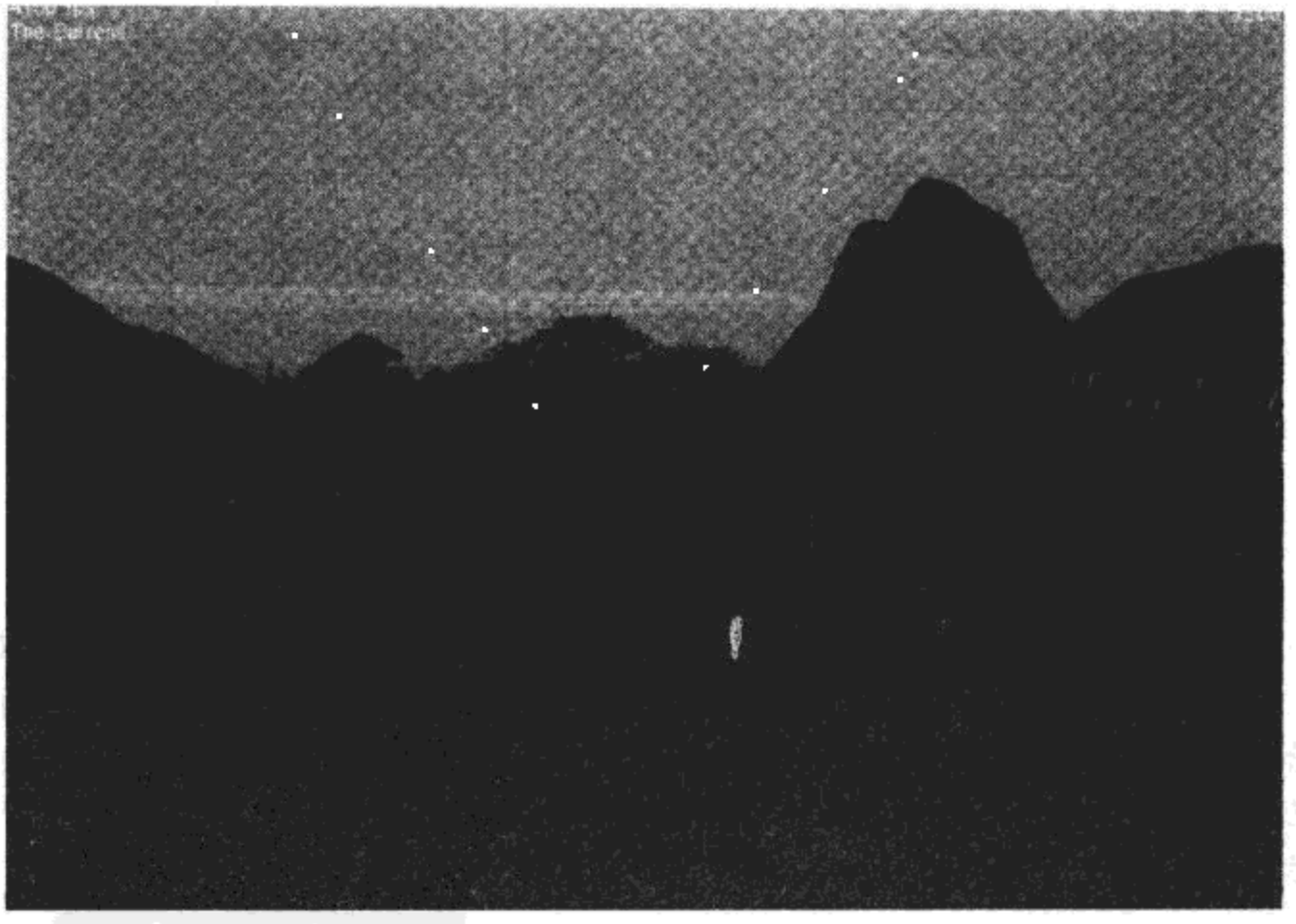


图 9-4　采用 WoW 客户端的一些数据和模型进行改装的效果（离线状态下）。贫瘠的地面上建立着兽人的房屋。（其实兽人是不会居住在这里的）

9.1.3 重写服务端

如果你想让你改装后的游戏实现在线的效果，这有很多途径。最简单的是保持游戏既有的客户端——服务端模块，仅是简单地重设每一个客户端中的内容。比如，那个图9-1中声名狼藉的裸体补丁可以应用到每一个客户端程序中，但不需要影响到客户端——服务端模块。

注意到要改变游戏玩法和逻辑，往往不得不对服务端的作用机制进行修改。比如WoW，它的服务端并不是与客户端捆绑销售的，开发方希望玩家在他们的官方服务器上玩。因为网络游戏不能仅仅在客户端模式下运行，所以这些要改装游戏的人只有一个选择——自己来写服务端程序。

你可以在现有的客户端——服务端协议的基础上写服务端程序，或者，你可以开发一个全新的协议。如果你不打算替换客户端可执行文件，协议将会继续保留，你所写的服务端需要用原来的协议进行编译和调试。如果你修改了游戏客户端逻辑，你需要开发一个独立的客户端——服务端协议。

回到我们畅想的改装话题上来。在“兽人大竞速”中，我们替换了完整的游戏客户端，做出了一个新的赛车游戏客户端。最终我们必须开发--个新的服务端程序来匹配这个赛车游戏。不过注意，我们不需要用一个类似于WoW原来那样的协议。图9-5显示了“兽人大竞速”中这种新的客户端——服务端模块。

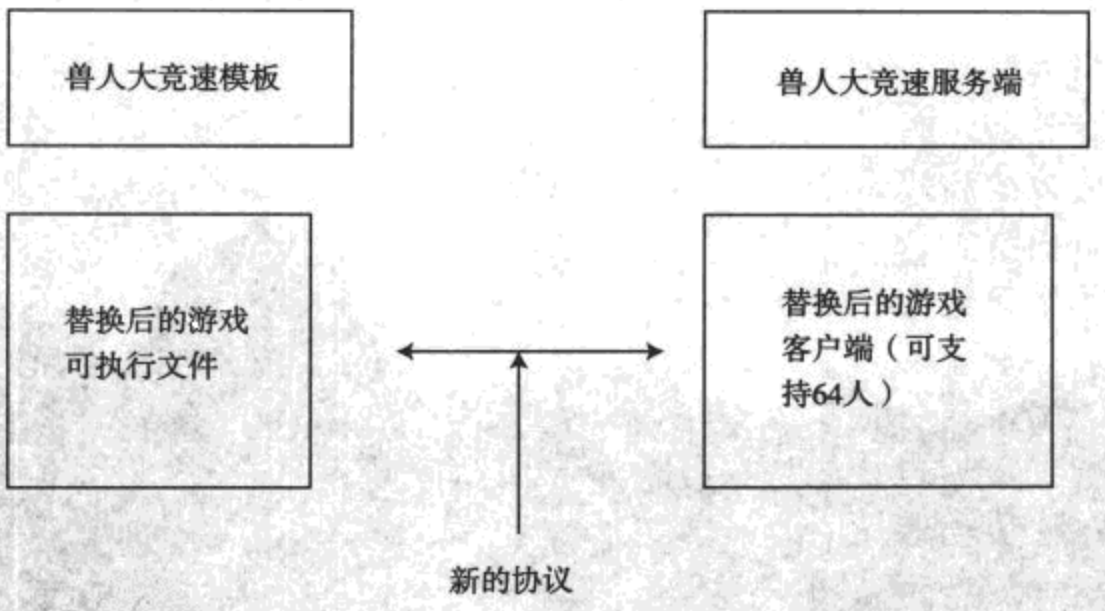


图9-5 兽人大竞速的客户端——服务端架构

图9-6是一个关于附加更新包的模型。在本章，我们假定附加更新包的提供者对WoW世界加入了一个全新的内容。在这个新的世界里，这些新增的东西（新的模型、动画、声效、主线剧情等），我可以感觉到它们是较为独立的，而且是相对应的，额外所提供的服务端功能仅是为这部分提供服务。因此，新增的这些内容是全新且独立的（就像一个新大陆或新岛屿）。

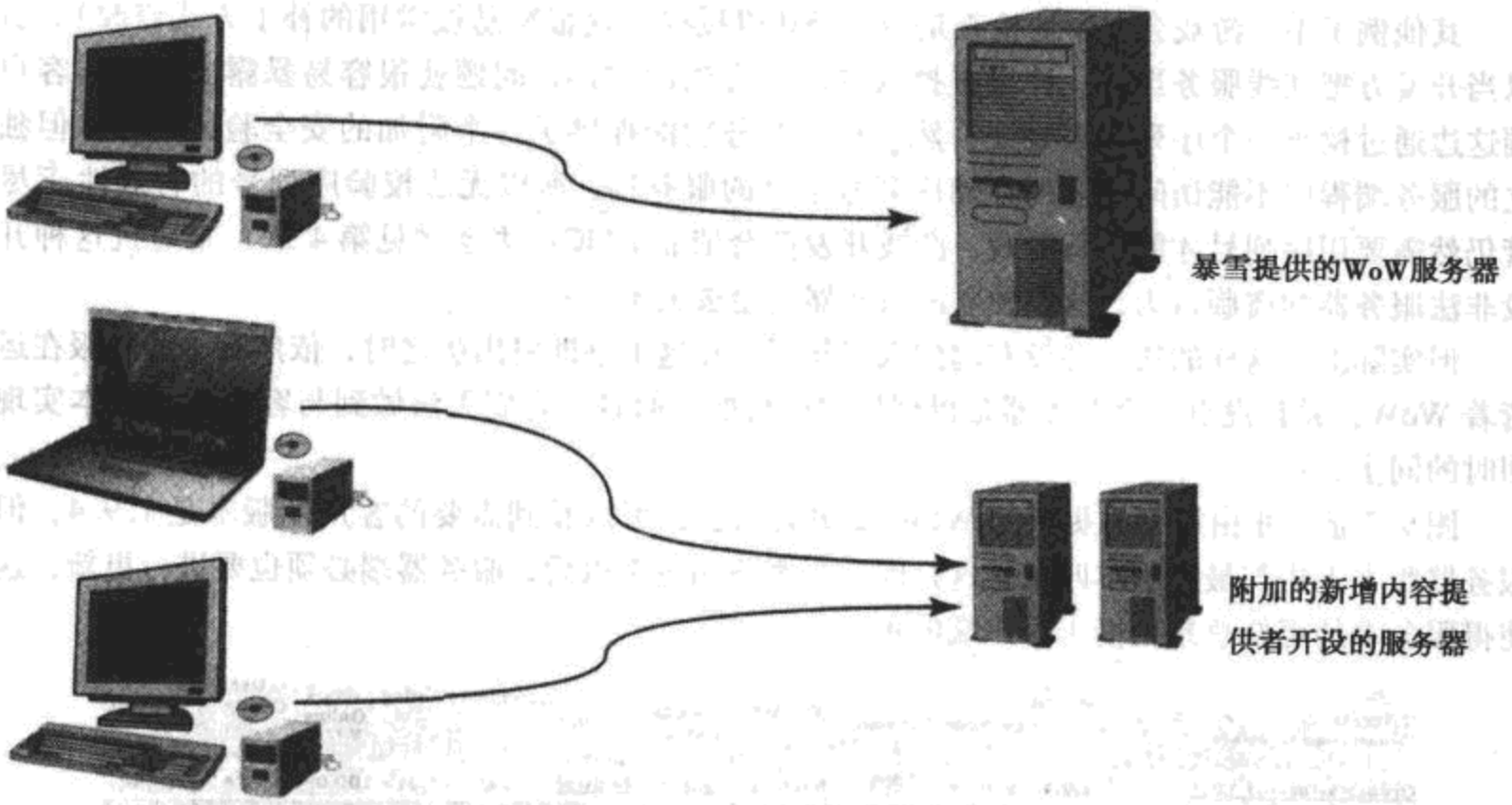


图 9-6　一个具备附加内容的服务模型，提供者同样会给玩家出售相应的账号

用户可以从这些新增内容的提供者那里获得一个账号，然后登录进入这个新的游戏大陆，享受这个附加更新包所提供的新的游戏内容。理论上，这套方式可以让做这种替换的公司以此为一项可以盈利的业务，也不会涉及侵权，并且通过不停地输入新鲜血液来使游戏延长生命周期。但事实上，许多游戏开发公司并不喜欢这种模式，因为他们显然既不愿意看到自己的线上服务出现内部竞争，也不愿意看到这样一个系统会使用户去使用非法的客户端拷贝版本。[⊖]

在 WoW 的例子中，第三方服务提供商也许需要通过 WoW 主客户端的授权验证，因为在登入进程中，需要执行部分的游戏版权保护程序。

WoW 服务端程序的开发是有着很长一段历史的。有些入侵的外挂，通常被我们称为模拟服务端（Emulation Servers）。暴雪娱乐是非常反对这些外挂的，事实上，一旦有人在网络上发布一个模仿型服务端，暴雪就会威胁他要把他们告上法庭。（还记得第 4 章里我们提到的可怜的 BnetD 开发工程师们的命运吗？）暴雪之所以如此，因为他们认为这些模仿型服务端允许用户玩私服版本的游戏（因为一些安全认证需要服务端的响应）。游戏会有很多方式来认证是否获得授权，比如用序列号或者 CD-Key。而 WoW 基本也是这样，并且它采用了一些小技巧可以使服务端的认证与序列号的检测产生依赖关系，因此提供了一个技术平台来支持这种行为验证，防止用户去修改服务端授权的验证过程。简单来说，这是一种很聪明的做法。

⊖ 记得第 2 章里我们提过，防盗版机制通常包括游戏服务端对客户端的一个认证。

其他例子中，游戏会在本地检查是否已经获得授权（这很容易被常用的补丁方式修改）。所以当开发方把在线服务当成一种检查授权的附加方法的时候，问题就很容易暴露出来了。客户端这边通过检查一个序列号来验证，然后在线服务功能提供了一个附加的安全验证过程，但独立的服务端程序不能访问相应的存储序列号信息的服务端，所以无法校验序列号的有效性。[⊖]尽管仍然需要用序列号才能进行游戏，游戏开发商会依靠 DMCA 法令（见第 4 章）来对抗这种开设非法服务器的盗版行为，这些独立的服务器也会被关闭。

但实际上，这样的法律并没有起到太多作用，在这本书即将出版之时，依然有一些私服在运营着 WoW，并且没有一个服务器是得到暴雪授权的。而且，它们无法做到与客户端的版本实现即时的同步。

图 9-7 是一张由第三方提供的 WoW 私服的列表，可以看到需要的客户端版本是 1.9.4，但服务器版本无法与最新版本匹配。这是因为当客户端更新以后，服务器端必须也要进行更新，这使得服务端的更新总是稍慢于客户端的更新。

:: Servers :: Click on name for details	Version	Location	Language	XPRate	Click to Connect*	Account	Comments	Rating	% Online	User Online**	Stats***
Orange Virus² - 1.12.2	1.9.4	Europe		blizzlike	Connect	Register	12	4.6\5	100.0%	N/A	
[illegible]	[illegible]	Europe		blizzlike	Connect	Register	[illegible]	[illegible]	[illegible]	[illegible]	
WoWGeR V 3.0 || 1.12.1 / 1.12.2	1.9.4	Europe		blizzlike	Connect	Register	1	5.0\5	100.0%	N/A	
[illegible]	[illegible]	Europe		blizzlike	Connect	Register	[illegible]	[illegible]	[illegible]	[illegible]	
NordWoW PvP&PvE Server Patch 1.12.2...	1.9.4	Europe		blizzlike	Connect	Register	70	4.3\5	100.0%	N/A	
[illegible]	[illegible]	Europe		blizzlike	Connect	Register	[illegible]	[illegible]	[illegible]	[illegible]	
New Generation 2000MB 24/7 NOW 1.12.1	1.9.4	South-America		blizzlike	Connect	Register	36	3.4\5	98.0%	N/A	
[illegible]	[illegible]	Europe		blizzlike	Connect	Register	[illegible]	[illegible]	[illegible]	[illegible]	
Ovamva Germany	1.9.4	Europe		blizzlike	Connect	Register	13	5.0\5	96.0%	92	
[illegible]	[illegible]	Europe		high	Connect	Register	[illegible]	[illegible]	[illegible]	[illegible]	
DEVASTATOR 1.12.1 / 1.12.2 all-in!	1.9.4	Europe		blizzlike	Connect	Register	6	4.2\5	96.0%	N/A	
[illegible]	[illegible]	Europe		high	Connect	Register	[illegible]	[illegible]	[illegible]	[illegible]	
SGserver	1.9.4	Africa		very high	Connect	Register	9	4.6\5	95.0%	8	
[illegible]	[illegible]	Europe		high	Connect	Register	[illegible]	[illegible]	[illegible]	[illegible]	
Nemáš!	1.9.4	Europe		blizzlike	Connect	Register	1	5.0\5	96.0%	1	
[illegible]	[illegible]	[illegible]		high	Connect	Register	[illegible]	[illegible]	[illegible]	[illegible]	
DIABLOS LAIR SERVER (CZ/SK) 1.12.1 pat...	1.9.4	Europe	?	lovv	Connect	Register	3	3.7\5	100.0%	N/A	

图 9-7 目前 WoW 的一些成功的私服列表（来自 http：//www. wowstatus. net/serverlist. php）

在这本书撰写之际，MaNGOS 和 WoWEmu 是目前最好的私服。[⊜]MaNGOS 是开源的，并且开发团队的实力很强。[⊜]

⊖ 如果游戏公司比如暴雪，能为第三方服务提供商一个授权认证，那么这些第三方服务内容开发商们会非常乐意。当然，对暴雪来说最大的问题是可能造成收入上的损失。如果能保证向暴雪支付每个月的费用，倒是可以通过很多方式来解决这个问题。

⊜ 排名可参考 < http：//forum. ragezone. com/world-warcraft/ > 。

⊜ 更多地了解 MaNGOS，请参考 < http：//www. mangosproject. org/forum/ > 。

9.1.4 客户端渲染选项

你可以在一些开源程序中找到很多有用的代码来创建一个3D游戏世界，进行画面渲染的时候，会面临很多选项。如果你想在渲染功能上做出一个专业的游戏，建议看看GarageGames.com提供的Torque引擎。Torque最初是用来做一款叫做“部落”的游戏。有一个很好的开源库叫ORGE 3D（第7章里描述过）。有了这些强大的开源资源，我们就有很多方法对客户端里的模型或其他媒体文件进行再次渲染和使用。

在下面的例子里，我们成功地提取了WoW中的媒体文件，比如模型、动画、材质、地形等，然后对它们进行了再次处理。因为这些文件通常以某种格式存储在游戏安装目录下的某个地方，很容易找到并且解压。

9.1.5 模型建构

一个模型需要许多的物件来构成，然后进行渲染，才能构成一个游戏世界。3D模型需要的部件更多，这是一个WoW中的小单子：

- 模型
- 动画
- 骨骼
- 贴图动画
- 模型顶点
- 模型视野
- 模型几何设置
- 模型贴图单元
- 模型渲染标识
- 模型色彩
- 模型透明度
- 模型光源
- 模型视觉
- 模型粒子系统
- 模型粒子特效
- ribbon功能区特效

为了进一步说明问题，我们先看看WoW中角色发型的例子。一个角色模型的不同发型包含着许多几何面。玩家选择了一种发型，相应的这种发型的网格将会被渲染。图9-8中，我们将WoW中一个角色模型的网格分为若干个子块，可以看到5件不同的披风，1条肩带，1条腰带，1件外衣，以及许多装甲碎片。当角色穿上衣服的时候，系统将会选择对应的部位进行渲染。

现在的大多数游戏都采用相似的方法和物件来进行建模。游戏制作者觉得这些物件是构成他们游戏模样和风格的重要部分，所以当有人去尝试分离并且使用它们的时候，制作者总会感到不爽。但那些热衷于修改游戏的家伙是不会停手的。所以我们在这里讨论的这些技术，对于熟练的黑客来说，早都不是什么新闻了。

为了更好地说明这些物件，我们继续以WoW中的物件来举例。用一些常用的破解软件，我们就可以从客户端安装好的文件里提取这些资源文件，比如我们提过的WowModelView。表9-1是其他几个游戏的一些破解软件。

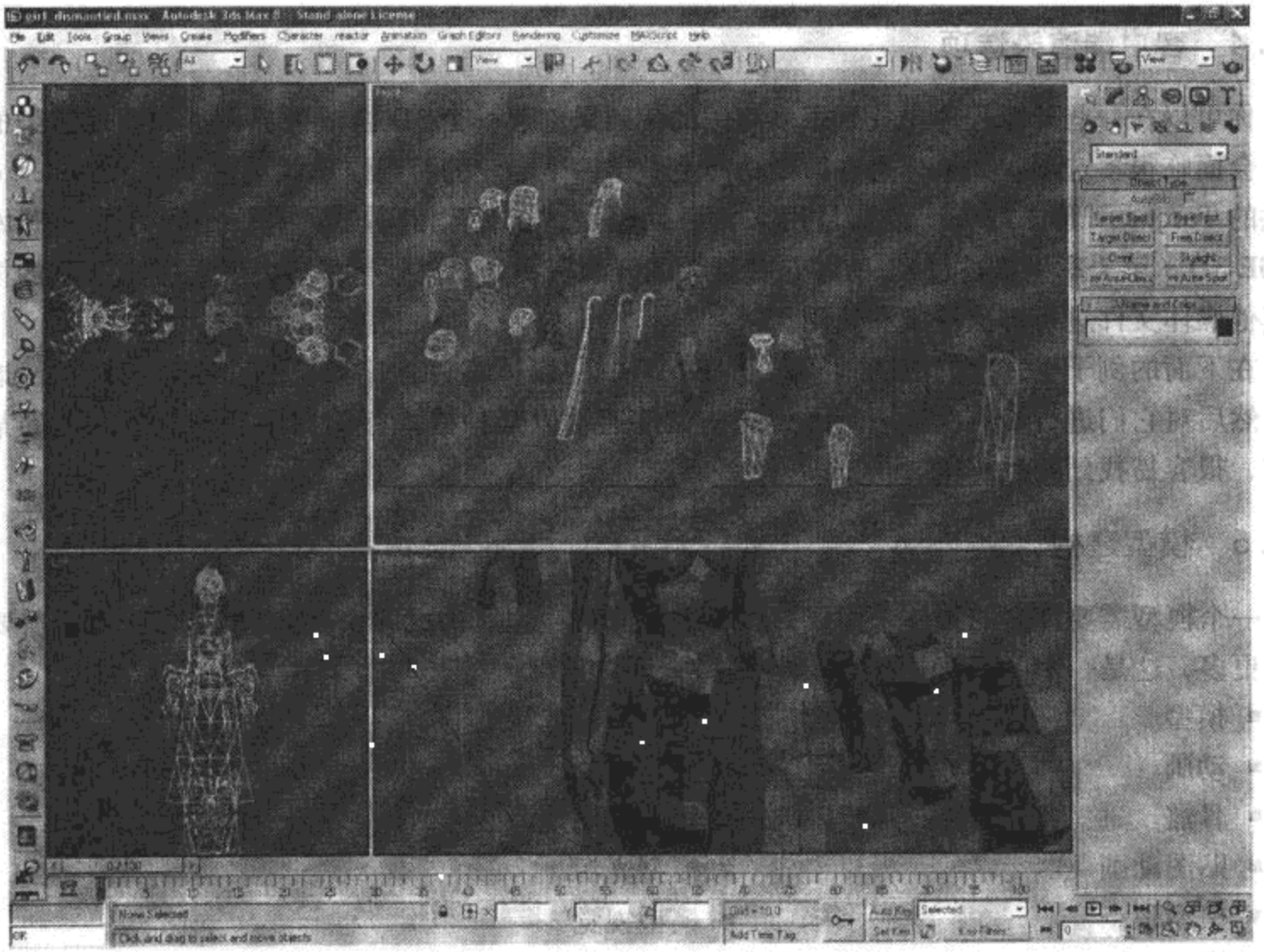


图 9-8 角色的模型可以拆分为若干个 mesh。这里是 WoW 客户端文件中角色模型的一些子 mesh（已由 www. wowstatus. net 授权）

表 9-1 替换游戏模型的工具

工 具	游 戏
WowMapView	WoW
WowModelView	WoW
MyWarCraftStudio	WoW
Radiant	使命召唤 2（Call of Duty 2）
F. E. A. R. Modification Tools	先头部队（F. E. A. R.）
HL2 SDK	半条命（Half-Life）
DS2TK	地牢围攻 2（Dungeon Siege 2）
TES Construction Set	上古卷轴（The Elder Scrolls）

Wow 的 ModelView 界面功能十分强大。图 9-9 是使用 WowModelView 的一个截图，图 9-10 是用相同引擎对 WoW 进行渲染的效果。

有些工具软件是专门为了游戏改装而定制的，并且具有很强的鲁棒性。比如 Elder Scrolls，

一个不错的游戏，它具备完整的支持创造和修改内容的开发环境。图 9-11 显示了 TES Construction Set 的应用截图。

继续以 WoW 为例子，WoW 中的媒体文件可以被提取，然后应用在另外一个环境中，比如第 7 章介绍过的 OGRE 环境。需要注意的是，这些文件是具有版权的，所以如果用这些素材来制作游戏，是无论如何都不可以对外发布的。

转换模型的格式，比如从 WoW 中转换到 3D Max，需要做一些处理。有经验的人会用一系列工具混合起来按序使用（那些做图形制作的人会明白这是什么）。在做模型的格式转换时，有时候会需要到某一个工具，受制于它。我们称这个过程为“工具链”。比如将 WoW 模型导入到 3D Max，先用 MyWarCraftStudio 对模型进行输出（图 9-12），然后用 Milkshape 进行加载，再输出一遍，通过一个插件导入到 3D Max 中。

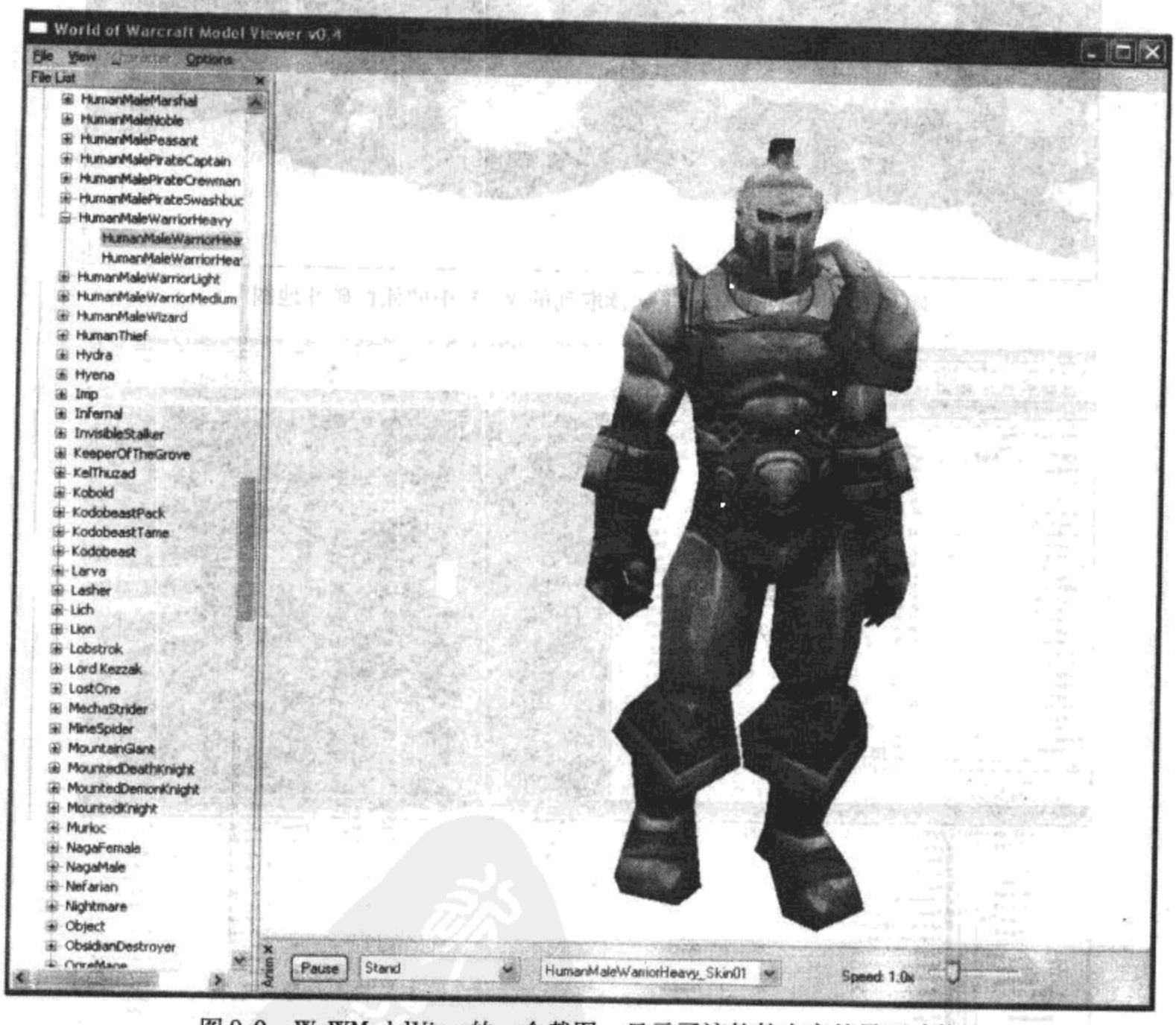


图 9-9　WoWModelView 的一个截图，显示了该软件丰富的界面功能

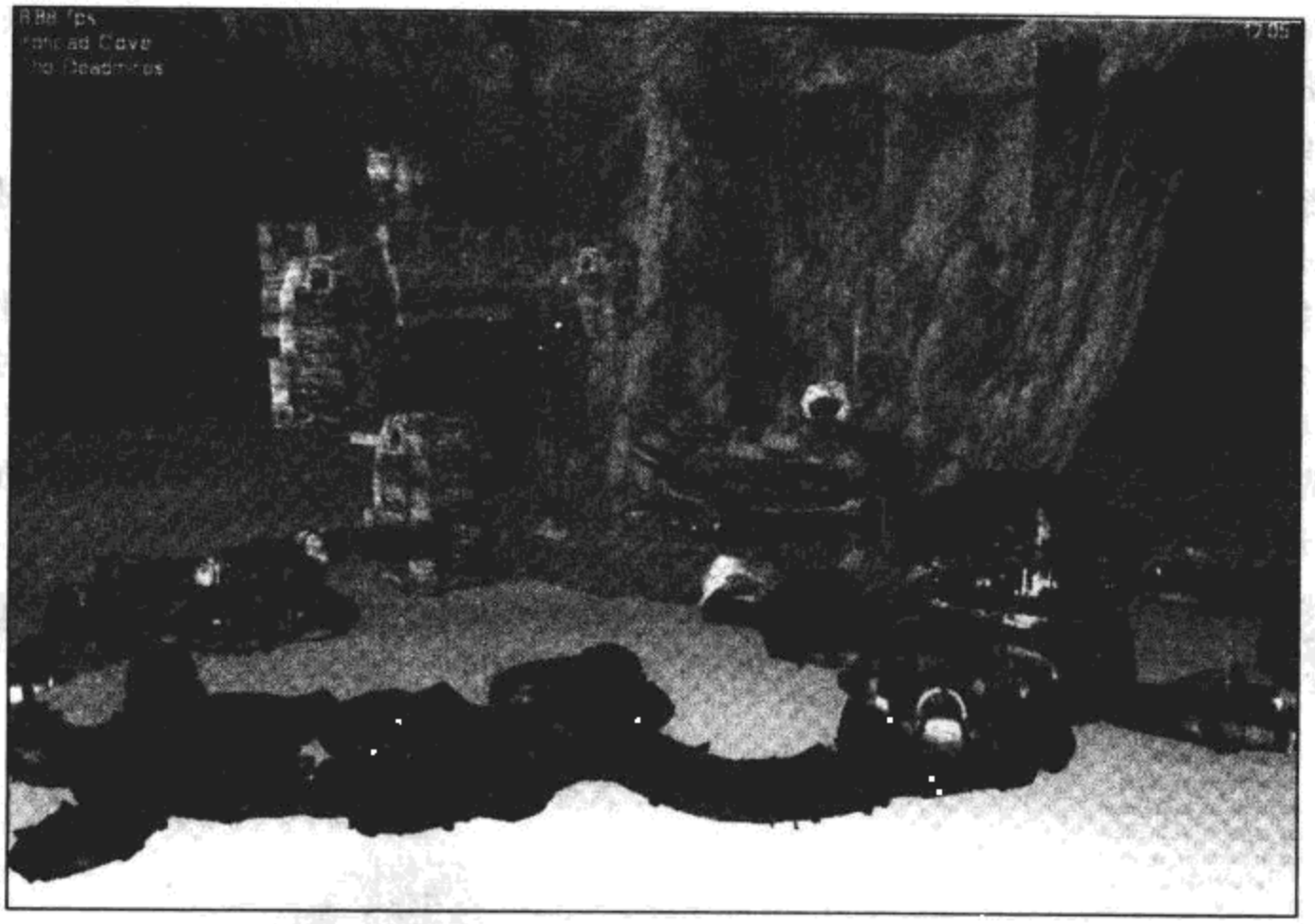


图9-10 用 WoWMapView 读取到的 WoW 中的死亡矿井地图

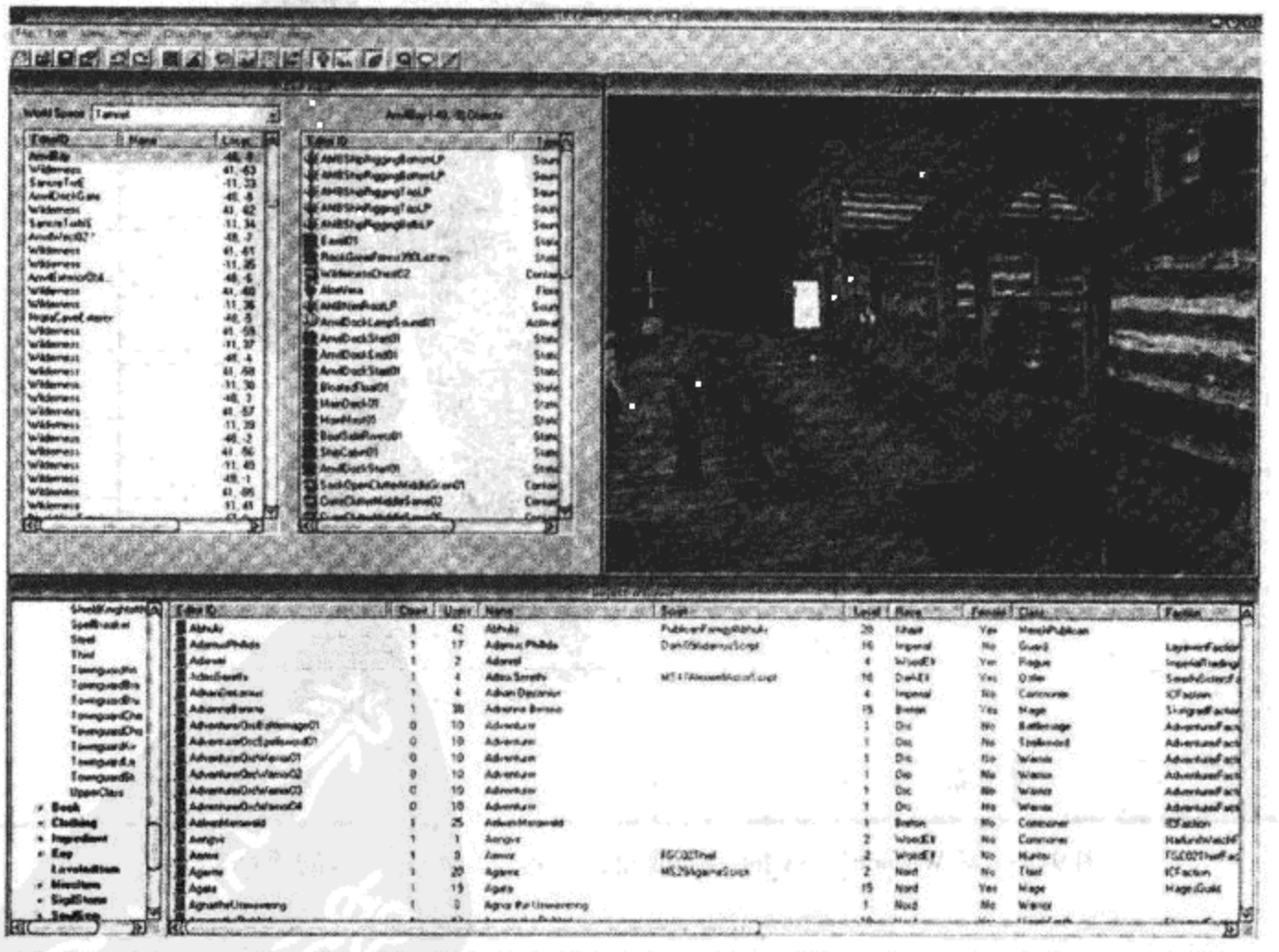


图9-11 TES Construction Set 的应用截图（来自 http：//en. wikipedia. org /wiki/Image：TES4. png）

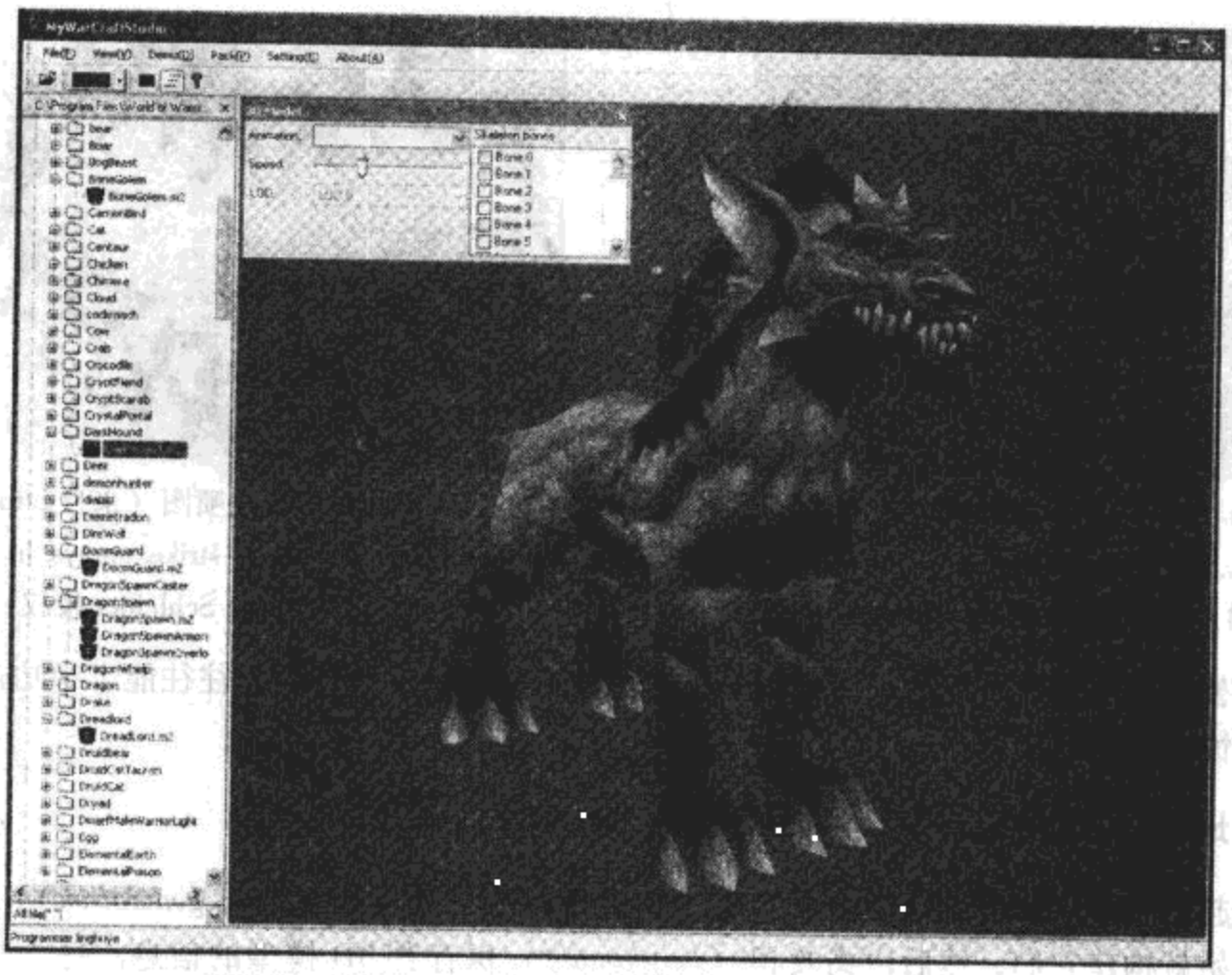


图 9-12 MyWarCraftStudio 可以输出 WoW 中的物体模型

有时候工具链会很长，需要 4 个或者更多的工具才能制作出最后的成品。当模型包含贴图数据和动画时，这个过程就会变得复杂。同时有些工具总会有出错的时候，比如不能正确地转换动画序列和骨骼。

Stand-ins

Hobby 游戏黑客很热衷于去玩一个 MMO 结构的游戏，然后他们可以把其中的模型和物件提取出来，然后用于自己制作的游戏之中。有时候我们称这些用于替换的物件为 Stand-ins，因为制作者会用自己绘制的物件来替代原来的媒体文件。开发一个完整游戏具备的物件和模型，是一个浩大而昂贵的工程。

9.1.6 贴图

最简单的游戏改装就是改变贴图了。本章的开始，我们已经展示过著名的裸体补丁的效果。这么做之所以简单，其中一个原因是因为这些图片都存在一个文件或文件夹里，而绘图软件又可以实现对这些图片的轻松修改。

最酷的莫过于这种贴图替换后的效果可以最直接地显示在游戏中，替换者立刻可以获得满足感和成就感。

个人风格定制

贴图替换往往会展示出艺术家的个人风格。比如，在反恐精英这样的 FPS 游戏中，反战人士会制作一些表示他们意图的涂鸦喷绘，见图 9-13 和图 9-14。

图 9-13 CS 的反战喷图（来自 http://www.opensorcery.net/velvet-strike/sprays.html，已由 Anne Marie Schleiner 授权）

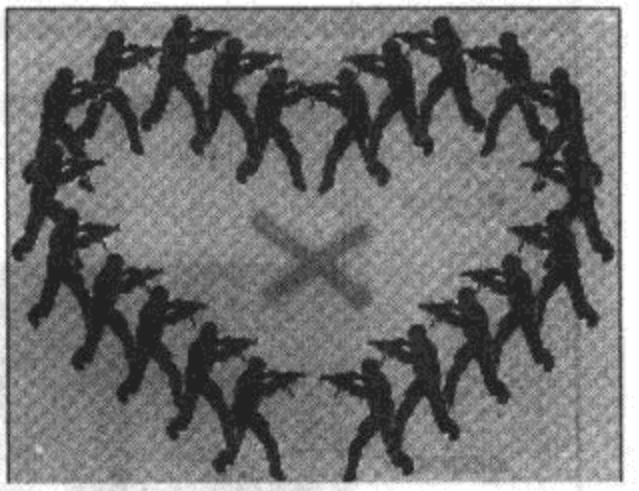


图 9-14 CS 里的更多反战喷图（来自 http://www.opensorcery.net/velvet-strike/sprays.html，已由 Anne Marie Schleiner 授权）

贴图替换是简单而有趣的。一个新手黑客比较适合尝试这种行为，往往能事半功倍，出现意想不到的惊奇效果。

9.1.7 地形

地形起伏和地高都可以自动生成实现（被称作程序生成（*procedural generation*）），或者你可以在最开始就构建它们，然后以高度图（*heightmaps*）保存其 3D 模型的信息。每一个游戏有它自己保存地表信息的文件格式。WoW 中的地表信息是以 ADT 格式保存的（图 9-15）。在使用高度图的情况下，可以用一个位图去定义地面的高度，然后 3D 模型生成的时候会读取这个文件的信息。图 9-16 是一个使用高度图的例子。

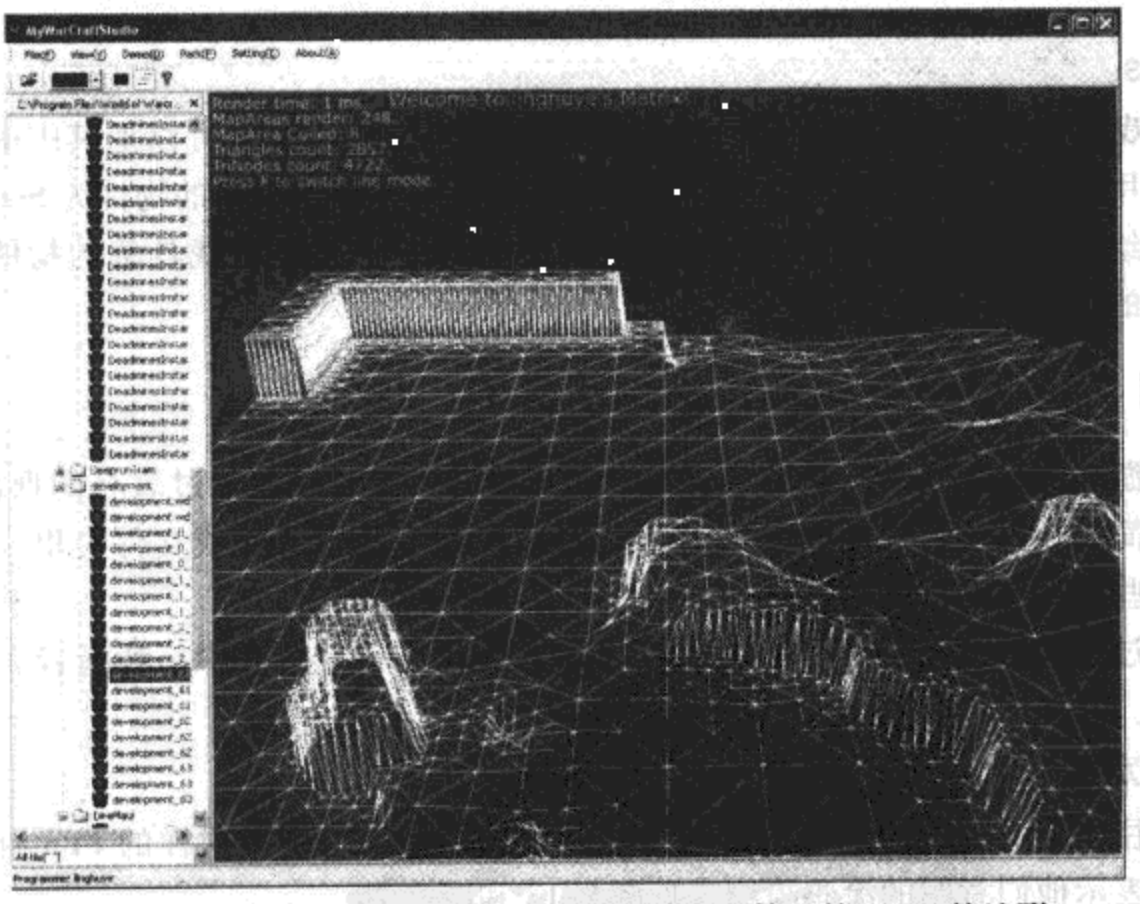


图 9-15 用 MyWarCraftStudio 得到的 ADT 文件里的 WoW 的地形

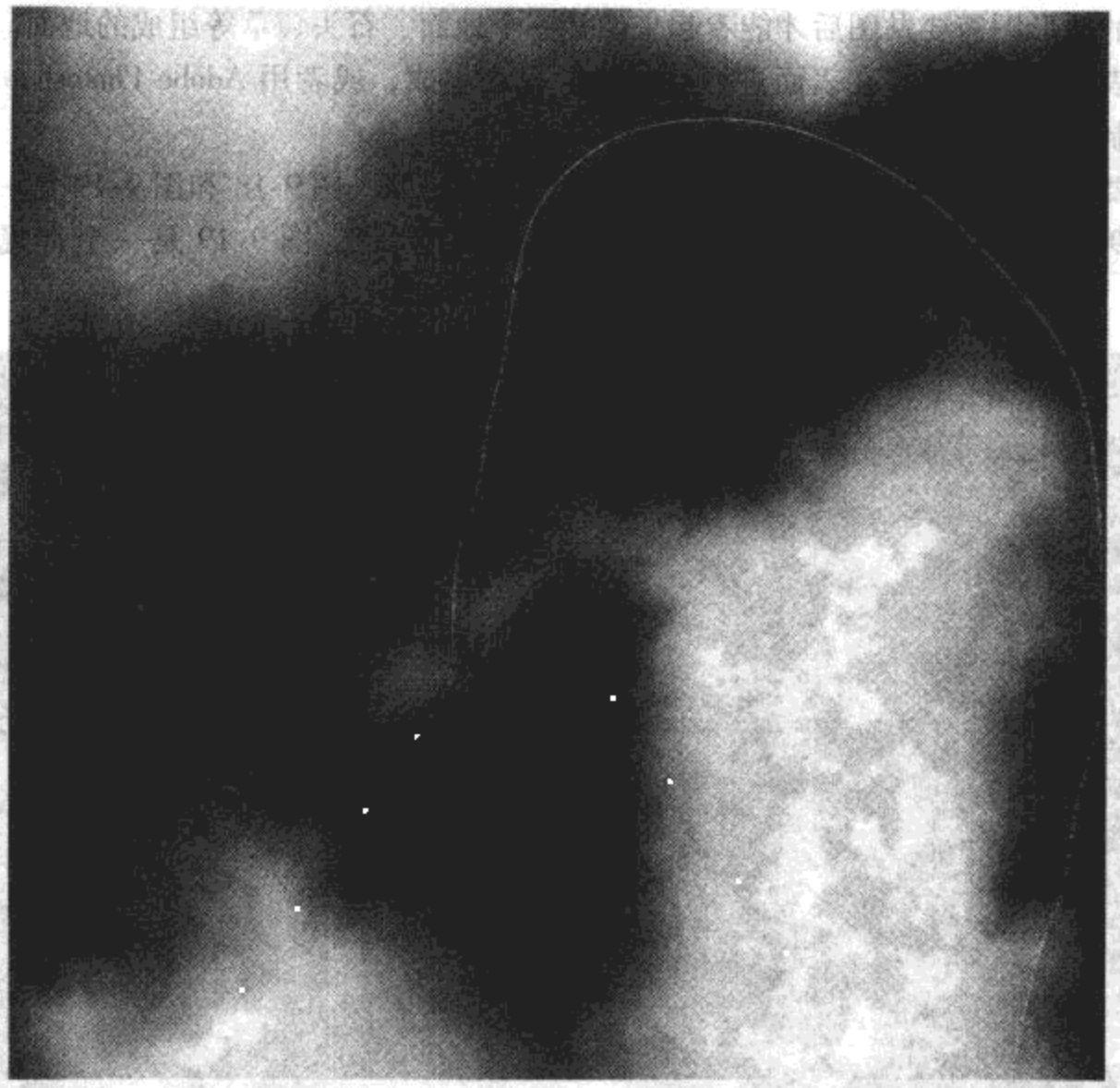


图 9-16　一个正面俯视看到的高度图，不同的颜色块表示不同的地形海拔高度

渲染引擎用图 9-16 中的高度图，生成了图 9-17 的地表信息。

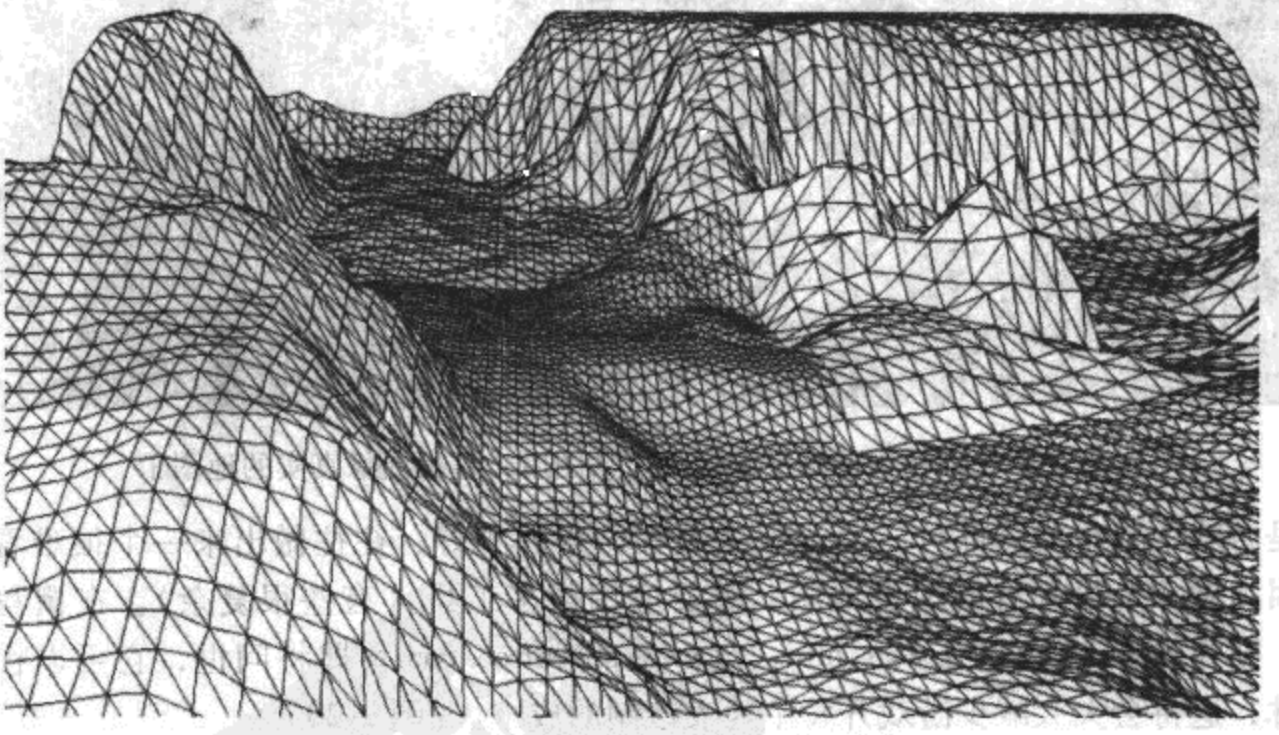


图 9-17　用高度图生成的地表网格图

网格状的地形图需要贴图后才能看起来像是一个地面、石头、草等组成的地面。根据所使用的工具，地表贴图可以直接在表面进行绘制然后自动生成，或者用 Adobe Photoshop 这样的绘图软件进行绘制。

我们需要使用一系列工具的组合才能生成贴图后的地表。图 9-18 和图 9-19 是一个最后效果的例子。图 9-18 是从正面俯视看下去的图（有点像高度图）。图 9-19 是一个常规视角下的地形图。

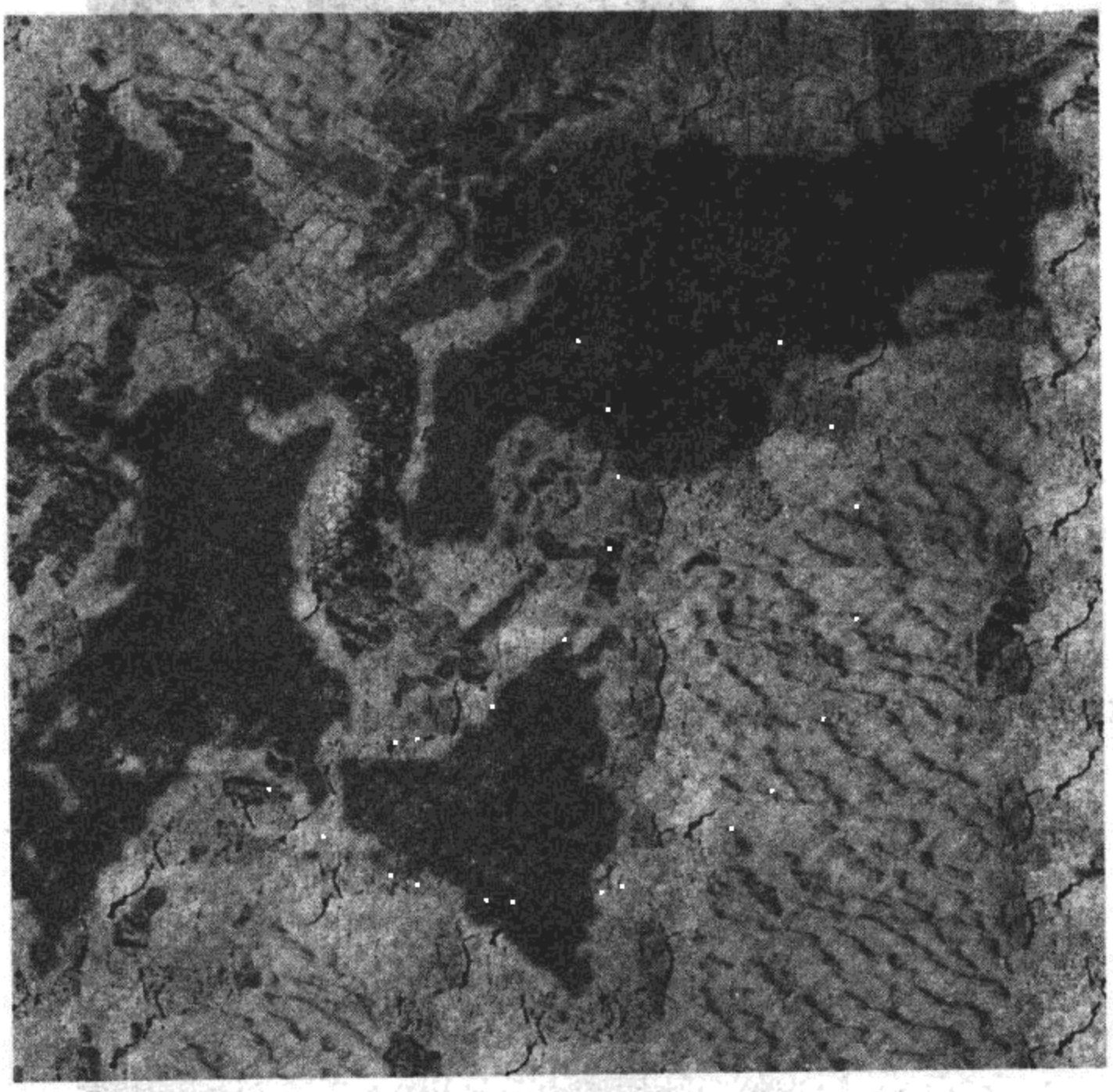


图 9-18　正面俯视的高度图加上了贴图后，显得更像真实的地图了

地表是游戏中很重要的部分，不过很多黑客做得并不好，往往做出来一个很难看的世界地图。这是因为做一个很好看的地表地图，需要应用很多工具（有些很复杂）。在撰写此书的同时，有一个称为 SpeedTree 的工具 < http：//www. speedtree. com > 即将问世，它可以程序化生成青葱的树林，这将会大大减少游戏开发中地表绘制的成本。（图 9-20）

图 9-19　从地面位置开始渲染的地形效果

图 9-20　一个 SpeedTree 的效果图。这是用了 SpeedTree 的一个游戏：整装待发（Fatal Inertia）
（来自 http：//www. speedtree. com/gallery/index. php? Page = 70&Sort = 0# ，
已由 www. speedtree. com 授权）

另一个不是很昂贵的地表绘制工具叫做 Terragen < http：//www. planetside. co. uk/terragen/ > ，它可以渲染出实景的地形图（图 9-21）。同时，Terragen 可以输出高度图形式的地形信息，方便于再导入到另一个游戏中。

图 9-21 一个用 Terragen 渲染后的场景图，具备了写实的真实表现力。（来自 http：//www. planetside. co. uk/gallery/v/tg09gallery/GalleryImage457296. jpg. html；已由 www. planetside. co. uk 授权）

当然，还有很多工具软件可以生成全新的地形和地表图，介于篇幅在这里不一一列出。事实上，全书都在讨论如何改装游戏。这里再举个很好的例子，就是 Epic 公司的 Unreal 引擎 < http：//www. epicgames. com > 。Unreal 引擎很著名，而且 Epic 公司很支持进行模型改装的行为。有一本关于探索 Unreal 引擎的好书叫做《掌握 *Unreal* 引擎：关卡设计的艺术》（*Mastering Unreal Technology*：*The Art of Level Design*，*Jason Busby*，*Zak Parrish*，*and joel Van Eenwyk*（*sams*，2004））

9.2 资源文件格式

刚才展示的改装、模型替换需要用户对资源文件的格式有个基本的了解。暴雪游戏（WoW 和 Diablo）的资源文件封装在一个叫做 MPQ 的加密文件里面。MPQ 这个名字的由来是它的设计者 Mike O'Brien 的名字缩写，比另一个缩写 MoPaQ 要短，后者是 Mike O'Brien PaCK 的缩写[⊖]。所有游戏的资源文件都有其格式，无论是公开的还是封装的。

想要对 WoW 进行改装替换，熟悉 MPQ 格式的文件是基本中的基本。一些程序员已经破解了这种格式并制作出了破解工具，有个叫做 libmpq 的开源库已经可以简单地读取和破解 MPQ 文件。这个开源库被 GNU license 授权，由 Maik Broemme 编写。你可以从很多网站上找到这个被称作 mpq 工具的库文件，比如 < https：//babelize. org/mpq-tools. php > .

⊖ mpg 工具的主页 http：//babelize. org/mpg-tools. php，介绍了 MPQ 名字的由来。

网上有很多致力于 MPQ 文件编写、破解、打包的例子。这些工具实现了对 MPQ 文件内部信息的访问和替换。

9.3 模拟型服务端（私服）

在前面，我们提到了自己架设服务器，并且以 WoW 为例子，在这个领域已经有了很多次尝试。同时我们也明确地表示过，这种做法是被暴雪公司和 WoW 的制作者极力反对的。但另一方面，也不是所有的游戏制作者都像暴雪那样强硬，他们有的还支持用户进行改装和模型替换的行为，对此类用户交流表示出积极的态度。写出一个服务端程序是一件非常有成就感的事情。这好比你是一个上帝，亲手创造出了世界，有谁会不乐意呢？

在这里，我们必须声明一点：尽管我们谈论到了关于 WoW 私服的话题，请注意如果你想使用我们提到的技术手段去尝试写一个改装的服务端程序，你很有可能会遭到法律的制裁。请慎重考虑！ <http://www.gotwow.ic.cz/> 上有很多 WoW 私服的例子。

其实，关于这部分内容，在编写此书的时候，我们也经过了一番争论，是否要把这些写入此书。最终我们认为，我们仅仅是要分享这些内容，而不是要助长任何侵犯版权的行为。我们也是想让用户在没有暴雪认证的账号的情况下就可以玩到游戏。

我们相信这么做没有什么问题，因为暴雪对用户的保护机制是公平的。如果你在店里买了一套正版游戏软件，那你就可以尽情享用它，包括但不局限于在你自己的私服上玩这个游戏，没有网络连接，你将享受在自己的世界里。当然，有些人不这么认为。

好了，关于这个话题，我们就谈到这里吧。

9.3.1 通信协议模仿

要架设一个服务端，首先必须要有直接解读客户端程序的能力，同时还需要具有对网络协议、消息、任何加密和反加密技术进行反推的能力。

下面是我们从一个 WoW 的私服服务端中选取的一段源代码，是有关于消息类型的。将这些协议反推出来是非常大的工程，但这对于从事破解的程序员来说十分重要。

```
public enum OP
    {
      MSG_NULL_ACTION,                    // = 0x0
      CMSG_BOOTME,                        // = 0x1
      CMSG_DBLOOKUP,                      // = 0x2
      SMSG_DBLOOKUP,                      // = 0x3
      CMSG_QUERY_OBJECT_POSITION,         // = 0x4
      SMSG_QUERY_OBJECT_POSITION,         // = 0x5
      CMSG_QUERY_OBJECT_ROTATION,         // = 0x6
      SMSG_QUERY_OBJECT_ROTATION,         // = 0x7
```

为了节省空间，我们删减了很大一部分信息……

```
      SMSG_SET_FORCED_REACTIONS,          //  =  0x029D
      SMSG_SPELL_FAILED_OTHER,            //  =  0x029E
      SMSG_GAMEOBJECT_RESET_STATE,        //  =  0x029F
      CMSG_REPAIR_ITEM,                   //  =  0x02A0
      SMSG_CHAT_PLAYER_NOT_FOUND,         //  =  0x02A1
      MSG_TALENT_WIPE_CONFIRM,            //  =  0x02A2
```

```
        SMSG_SUMMON_REQUEST,                    //  =  0x02A3
        CMSG_SUMMON_RESPONSE,                   //  =  0x02A4
        MSG_MOVE_TOGGLE_GRAVITY_CHEAT,          //  =  0x02A5
        SMSG_MONSTER_MOVE_TRANSPORT,            //  =  0x02A6
        SMSG_PET_BROKEN,                        //  =  0x02A7
        MSG_MOVE_FEATHER_FALL,                  //  =  0x02A8
        MSG_MOVE_WATER_WALK,                    //  =  0x02A9
        CMSG_SERVER_BROADCAST,                  //  =  0x02AA
        CMSG_SELF_RES,                          //  =  0x02AB
        SMSG_FEIGN_DEATH_RESISTED,              //  =  0x02AC
        CMSG_RUN_SCRIPT,                        //  =  0x02AD
        SMSG_SCRIPT_MESSAGE,                    //  =  0x02AE
        NUM_MSG_TYPES                           //  =  0x02AF
};
```

当然上述代码没有包含所有可能的消息类型，事实上另一张表会包含响应代码，包括道具、物件的 ID 等等。所以想要写出一个完整的私服程序是需要大量的数据的，因此私服一般来说不容易做出来。

练习：拦截数据包信息（Hooking the Packet Engine）

WoW. exe 客户端中有个函数用来处理所有的数据包（packet），函数的原型如下：

```
NetClient__ProcessMessage(
        int NetClient,
        int dummy,
        unsigned long 1,
        DWORD *CDataStore)
```

我们可以采用第 6 章里提到的技术来动态锁定这个函数。用调入的 DLL 或者一个外部调试工具（见第 6 章）对这个函数设置跳转点或者断点。当运行到这个点时，你可以拦截发包，然后获取你所要的信息，这是写私服所需要的。当然，这个 WoW 中的经典做法早就被黑客们尝试过很多次了，你也可以继续尝试一下。当面对一个新的游戏时，这种做法依然可以采用。

破解魔兽发包

注意 在此书出版之前，这部分内容已经被删除。如果读者对这部分内容感兴趣，可以去下载 *MaNGOS* 并且阅读源码，可以了解到如何破解魔兽的包。

游戏一般都有一套加密系统，加密是一个令黑客很头疼的事，有时候我们甚至可以称为“迷惑”系统。因为客户端程序必须解密发送的包，所以你在破解的时候，就把自己当成客户端程序那样去解读。

无论一个游戏如何进行加密，必须要进行破解，这是编写一个私服的必需过程。

想要开始编写或者自定义私服程序，可以去下载和编译 MaNGOS 或 WoWEmu 的源码，我们将在相关文档里叙述如何下载 MaNGOS。到目前为此，MaNGOS 已经匹配上了 WoW 的最新版本。这可以开始学着如何建立一个新发布的游戏的私服。

供您参考 下载 MaNGOS

下载 MaNGOS 首先需要一个类似 Subversion 的源码控制工具。建议使用 TortoiseSVN，它支持 Subversion <http://tortoisesvn. tigris. org>。你现在可以去以下网址下载一个最新版本的 MaNGOS：<http://mangos. svn. sourceforge. net/svnroot/mangos/trunk>。MaNGOS 可以在 Microsoft Visual Studio 中进行编译。

MaNGOS 必须被编译和设置好，首先需要机子上装有 MySQL 5.0；为了再现游戏世界，还需要初始化一个数据库，这个数据库存在于另一个文件中。比如 Azzum 的安装包有 200MB 左右的压缩数据。其他一些工具也是必需的，包括 Navicat，XAMPP，MaNGOS DB Handler 以及 nnCron。听起来这个工程很庞大，事实上也是这样。尽管如此，大多数黑客们都认为这很值得。在 <http://forum.ragezone.com/world-warcraft> 上有详细的安装向导。

9.3.2 进入游戏世界必需的几个步骤

想要架设一个私人的服务器，创建自己的角色，还有许多步骤需要去做。当然，第一步就是生成服务端。

私服架设好以后，第二步就是关于如何鉴别用户账号的登入。客户端程序在渲染游戏世界之前，需要登录一个有效的账号。客户端程序会建立起一个登录界面，需要输入用户名和密码。你可以输入一个正确的账号信息，也可以用数据库或者平台文件那样格式的账号信息输入。

验证账号信息后，下来就是服务器的选择界面（至少 WoW 是这样的）。因为一个游戏不可能让百万用户同时登录同一个服务器，所以游戏世界必须分为很多服务器。WoW 中的服务器之间是相互独立的，玩家的一个角色只能在一个服务器中。服务器选择的信息将会传递给客户端，玩家会在界面上选择一个服务器进入。

下一步就是创建角色。玩家在对应的服务器内会有对应的角色。这点上，私服可能更需要给予玩家一个创建或者删除角色的功能。客户端程序会处理角色创建过程中的大部分信息，而服务端可能会调用去做类似掷骰子、状态传输等功能。比如当玩家输入一个角色的名字，服务端将会校验这个角色名字是否已经存在。

最后，当角色创建并被激活后，玩家可以进入游戏世界了。角色会有一个位置信息记录，告诉客户端从哪里开始渲染游戏世界。客户端负责所有的渲染功能，服务端只需要记录每一个物体的位置信息。

在接触到真正的游戏世界之前，还有很多琐碎的步骤。但这些步骤是探索游戏世界之前必须要经历的。一旦掌握了这些步骤，你将不需要再重复这些工作，除非游戏开发商对游戏架构进行了较大的修改。

9.4 法律纠纷

我们讨论的这些技术，在法律上一直是个有争议的问题。一些游戏公司支持改装和替换，也支持第三方服务器，这是为了提高用户的粘性，保证游戏生命周期的长久。但一些公司极力反对任何人进行任何程度的游戏修改。我们不是律师，所以我们不能给予读者任何法律问题上的建议。我们能建议的是，当你进行游戏改装和替换时，你要清楚地认识到其中可能涉及的法律纠纷。

综上所述，本章所提到的所有技术都是现在的黑客们早就知道了的。如今功能越来越强大的改装工具，使得这种行为变得越来越容易。同时，很重要一个原因是，更多的人想通过了解游戏改装来更多地理解游戏是如何制作和运行的。

第 10 章　安全是游戏成功的基础

这本书即将结束之时，我们再来回顾一下全书内容。我们的首要目的是去理解一个具有百万级用户的大规模分布式软件系统的安全机制。我们认为 MMORPGs——一种大规模分布式系统——是软件世界中典型的一个例子。也可以认为，网游是面向未来的软件系统，如今它已经被无数人同时使用着，它的变革和更新就像海滩上的浪花，后浪推前浪。许多从事网络安全的工程师都知道，保护一个分布式系统不受攻击是件很困难的事……而开发一个安全的大型分布式系统是更难的事。很多时候，MMORPG 推动了这种安全保护的进步。通过了解 MMORPG 安全保护中的运行机制和缺陷，我们可以为未来做更多的准备。

网游公司对他们取得的成功感到惊奇，谁又能想到超过 800 万的用户在玩 WoW 呢？但是巨大的成功背后掩藏着沉重的担子——为百万级的用户进行安全服务，而且其中有些用户是不怀好意的。

就像其他一些软件的用户那样，大多数 WoW 或者其他 MMORPG 的用户还是没有意识到安全问题是个隐患，也许他们并不知道游戏开发商正在以维护安全的名义监控他们的电脑。只有很少的用户意识到有人在游戏里作弊（有很多方式），其他人都以为这个游戏世界是公平的。

为什么这么多人对于网游的安全问题视而不见呢？其实答案很简单，因为大多数人都是持乐观态度的，他们没有过多思考便相信他们玩的游戏是安全的。事实上游戏公司没有发表过任何言论或声明来说明它们的游戏是绝对安全的（在这个问题上，如果可能，它们一般都会保持沉默）。所以这一切，只是人们单方面的自我感觉而已。而当人们突然意识到软件安全会有问题的时候，对于软件开发者来说，已经有些措手不及了。看看微软的例子，他们已经花了几亿美元去弥补他们软件的安全漏洞所造成的声誉下降（这么做是值得的）！

就像本书里提到的那样，网络游戏目前最大的安全问题就是对于客户端状态的修改。无论何时，当客户端可以被任意修改或控制之时，问题就大了。这时，就会有人利用这些漏洞，通过一些中间机构的帮助，转化为商机。

这一章所要谈论的，就是如何解决这个问题。首先，给游戏开发者们提出一些建议，关于如何在制作游戏的时候提高游戏的安全性。然后，我们换个角度，来看看对于玩家，他们是如何去发现这些安全漏洞的。

10.1　游戏开发中的安全机制建立

如今的计算机安全领域中最大的问题是，大多数系统在构建的时候，并没有过多地考虑到如何建立起安全机制。一些常规的网络技术，比如防火墙，可以减缓一些简单的对服务器的攻击，但是这不能从根本上确保安全问题，——因为软件本身存在漏洞。如果我们想要真正解决安

全问题，那首先要做一个安全的软件。

软件的安全，其实就是编写一个安全的软件，使它能够承受住恶意的攻击。网络游戏在这方面尤其重要。

现在有三座大山摆在眼前——连通性、复杂性、扩展性（connectivity，complexity，extensibility），它们极大地影响了软件安全问题的发展和变革，很显然它们对于计算机游戏是有直接影响的。起初，我们普遍认为网络的连通性对于计算机安全来说是很有利的，因为可以通过服务器校验版权信息。但如今，由于网络游戏变得很庞大，而且分布得很广泛，连通性反而成了一个安全上的风险。各种各样的用户被连接了起来，有可以信任的，也有入侵者。复杂性，以及它的发展一直是软件工程中的焦点问题，网络游戏也不例外。网游的分布式结构更加广泛，并且需要记录那么多用户各自的状态，所以它也许是目前最复杂的分布式软件系统。这就意味着它的安全问题更让人头疼。最后，由于网络游戏本质上也是一个软件，所以它的扩展性也是重要的一个因素。有些游戏的扩展性在游戏制作之时便已明确（比如第二人生），有些游戏真正的扩展性是被那些黑客们挖掘出来的（比如第9章中提到的改装）。

为了能够在游戏制作之时就在根本上做到这一点，我们必须首先明确一个观点：软件安全不等于安全的软件。这点很容易被开发者们忽视，因为他们更重视开发软件的功能点。现在的大多数网游都是有安全机制功能的，但诸如增加一个SSL证书来保证安全性不是一个彻底的解决办法（第6章里我们提到过，一个CD－Key是很容易找到的）。软件安全是一个系统工程，不仅仅要考虑安全认证机制（比如权限校验），还要做到设计上的规避（比如鲁棒性设计可以使抗攻击能力提高）。有时候这两块内容是交迭的，但大多数情况下要分开考虑。

从另一个角度来看，安全问题在软件系统中往往是突然出现的，网络游戏也是。一个安全问题比起那些固有的软件问题，更有可能突发。因为安全问题往往涉及系统的核心部分（比如控制数据库模块的人机界面，服务器之间的通信）。所以软件的安全问题必须是贯穿在软件的整个生命周期内的，无法通过测试局部功能来确保整个软件的安全，也无法粗略地测试一下就得到是否安全的结论。没有什么魔法书或者圣灵药可以万能地确保软件的安全，所以我们必须在软件构建之初就关注到安全性问题。对于网络游戏，同样如此。

作为最初意识到软件安全重要性的那些先行者们，他们一直在努力地接受和尝试着各种实践，为了在这个问题上走的更远。微软在“Trustworthy Computing Initiative”口号下，做出了一系列令人瞩目的努力。Cigital的很多用户也享受到了该企业软件的安全措施。如今，这方面的行动很多，比如为开发人员、测试人员、架构师的安全培训；假冒软件的审核；安全工程学科的研究。为了能使软件行业更好地发展，审视软件自身的问题（比如设计上的诟病）比给软件吃一片阿斯匹林更有效果。一定要在软件开发阶段就重视到安全问题，并且多参考行业里失败的经验教训，多向那些成功者学习。

10.1.1 软件安全Touch-points

本书的其中一位作者（McGraw）在他的《*software Security*：*Building Security In*》（巩固软件安全）（Addison-Wesley，2006）一书中介绍过称为Touch-Points的7个最好的要点。无论你是不是一个游戏开发者，在实践中要注意到安全问题，需要你改变一下制作软件的方式。这并不是要

你改变一些根本做法和习惯，而是遵循一些最直接、最简单的基本原则，这样，软件的安全问题在开发过程中便可以被考虑到。

将这些时刻保障软件安全的做事方式融入软件开发的周期中，是三个确保软件安全的良药之一。这种做事方式在软件工程中发挥到了基石的作用，并且贯穿整个软件生命周期中的安全情况。这意味着：认识到常规的风险（包括本书提到的那些），设计上规避隐患，彻底消除所有看似安全却留隐患的软件设计，客观地分析和测试。

图 10-1 给出了软件安全的 touchpoints 以及如何在软件研发过程中应用它们。这意味着进行安全机制的工作时，需要从多方面考虑：需求（我们很明白现在的用户就是我们的上帝），架构（大规模分布式软件系统更容易出问题）、设计、写代码（包括调试）、测试、认证、评估、以及维护。

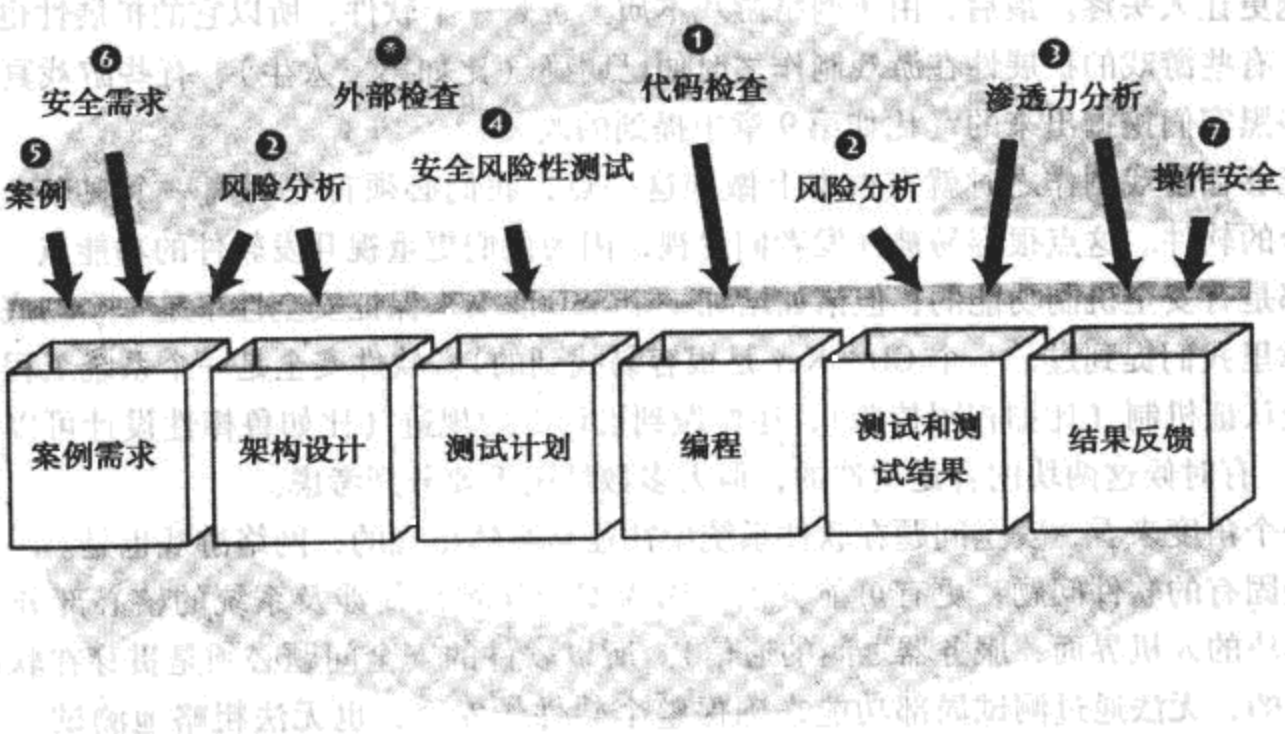


图 10-1 McGraw 的《巩固软件安全》（*Software Security*：*Building Security In*）中介绍的软件安全的 7 个 Touch-Points

尽管图 10-1 很像一个传统的瀑布型模型，但现在大多数开发者会周而复始地运用这个模型，根据软件本身的需要，会不止使用一次。任何情况下，这么做可以避免一些边际问题的出现（包括先前出现的一些问题）。

软件安全的 touchpoints 是一个较为通用的准则，你可以在任何一个软件开发过程中运用它们。甚至在你开发一些很小的软件时（最起码会有编写代码!），也可以运用 touchpoints。

有些 touchpoints 的功能比其他的更强大，所以首先得用它们。将这些 touchpoints 按照效果排列如下：

1. 用统计分析工具来核查代码
2. 架构风险分析
3. 渗透力测试
4. 安全风险性测试
5. 压力测试

6. 安全需求

7. 操作安全

更多的关于 touchpoints 的内容，参见《软件安全》（*Softwave Security*），里面有专门的章节介绍。

当然这个排序也不是放之四海皆准的道理。这个排序只是一个多年开发经验的总结，所以重读代码排在了架构风险分析之前。实际上，这两个都十分重要，如果你只是做到了第 1 个，但跳过了第 2 个，你依然无法做好软件安全机制问题。

本书已经接近尾声，我们已经知道软件被攻击的目标是两类：缺陷和漏洞。重新阅读代码就是为了找到代码编写时候的错误之处，也就是缺陷。架构风险分析是为了找到漏洞。如果这两步缺了其中一个，你最终可能只解决得了问题的一半（缺陷和漏洞各占一半）。同时，这两步 touchpoints 可以互相交换，相对来说不会有什么太多影响。

其他的 touchpoints，这样的排序结果也是基于不同的工程师根据多年的经验得来，这些人员来自于很广的领域，从大型独立软件供应商到巨型投资银行。当然这样的排序也不是绝对的，如果只是想要去尝试着改变这些步骤的顺序，比如先做渗透测试，再去重读代码，很有可能是无法达到我们原来的效果的。

10.1.2 黑帽子和白帽子

黑帽子和白帽子交互影响着软件的安全问题。这个观点，有点类似于太极中阴阳二极的意味。太极图是东方文化中的一种标志，和西方的辩证法有一些共通的特点。东方哲学强调整体论，所以有阴阳调和之说，而我们这里提到的黑帽子和白帽子，也是一个混合体。

*破坏性活动*是指攻击、入侵以及毁灭软件。本书直接介绍了网络游戏中这种破坏性活动。如果不在现实中直接清晰地面对这种攻击，我们也不可能谈到防护这一层面。所以，这方面就代表了黑帽子，也就是我们通常认为的“坏人”做的事。*建设性活动*是关于设计、防护和功能性开发，这方面代表了白帽子。

也许从另一种更轻松的角度来看待这种二分法就是攻击和防守的形式。本质上来说，攻击和防守都没有绝对的对与错，在体育运动中，两者都是必不可少的。

如果你是一个游戏开发者，你很关心本书中提到的这些安全问题，那你一定要让你的工作室的伙伴们采纳这些 touchpoints。

10.2 作为玩家的安全问题

作为一个玩家，你应该要知道你现在玩的这款网游的实际安全状况，这是非常重要的。你可以在读完本书后，问自己一些简单问题如下：

- 游戏运营商为你所玩的游戏做了哪些安全保障措施？
- 玩家们知道这个游戏是否可以作弊吗？
- 这个游戏里有人在做虚拟物品交易的第三方平台吗？
- 打币工作室和机器人外挂多吗？
- 安全认证机制是否侵犯了玩家的个人隐私？

你需要了解这些安全机制，这是很重要的。因为这对于你的游戏体验和你的个人隐私来说，都会产生直接的影响。

我们列出了一个单子，来衡量游戏安全与否，帮助你了解更多你所玩的网游的安全问题。你可以尝试填写一下，看看答案是否适合你。

供您参考　玩家电脑的安全情况核查清单

如果你玩的游戏是一款网络游戏，它会通过 Internet 连接到一个中央服务器，这将会比一个单机游戏具备更高的安全风险。通过以下调查单，你可以了解到你玩的游戏的安全情况。

- 按照下列常规项来检查游戏的安全性：
  - 安全跟踪 <http：//www. securitytracker. com >
  - Bug 检验 <http：//www. securityfocus. com/archive/1 >
  - RISKS 论坛 <http：//catless. ncl. ac. uk/risks >
- 确保游戏具备 SSL 认证或者其他可以连接到服务器进行认证的方式。你可以咨询游戏公司，或者自己用一个嗅探器来检测一下 <http：//www. ethereal. com > 尤其注意 443 端口。
- 使用 Hamachi <http：//www. hamachi. cc > 或者其他软件来保护网络传输，尤其是你处于一个虚拟局域网中。
- 确保游戏在传输信息的时候，不会泄露你的个人信息（无论在你的电脑还是在服务器）。建议使用 boron tagging 或者 Ethereal 作为验证。[⊖]
- 确保自己不会对游戏中某些功能实施的计算机监控感到不适。
- 阅读并且理解游戏的版权政策（如果它有）。
- 阅读游戏附带的 EULA 和 TOU，明确你的权利。阅读 EULA 是枯燥的，但良药苦口利于病。
- 尝试使用本书所介绍的一些方法，去网上寻找该游戏是否存在作弊器和外挂。如果发现介绍作弊的网站很普遍，你应该意识到游戏里很多人正在作弊。
- 确定游戏内是否有虚拟道具交易的组织。如果很容易就可以买到虚拟道具，说明这个游戏正在有作弊或外挂。
- 不要依靠玩网游来谋生，即使这个游戏内部建立起了清晰的虚拟经济体制。因为在游戏里没有真正地保护财产的法律效应，你累积的财富可能在一夜之间转瞬即逝。
- 开启杀毒软件和防火墙，保持时常更新。
- 不要以管理员身份去运行一个游戏，也不要装在根目录下。这个我们以后会解释……
- 不要以管理员身份去运行一个游戏，也不要装在根目录下。[⊜]
- 请阅读本书。

10.3　入侵网络游戏

至此，本书已经谈论了很多要点，从技术性问题比如机器人、作弊器、逆向工程、资源改装到法律问题和经济问题。我们经常会拿一个游戏来举例，以便更好地解释一些技术问题，我们必须强调，本书所谈到的技术问题并不是特例的，是可以放到任何一个网络游戏的例子中的。

⊖ 请阅读我们的另一本书《入侵软件》（*Exploitting Software*（Addison Wesley，2004））。

⊜ 请注意有些游戏需要安装在根目录下或者需要使用管理员权限。

同时，也别忘了本书谈论的大多数技术问题也不仅仅局限于应用在网络游戏领域。如今的软件已经朝着大规模结构的方向发展，所以对网络游戏的案例研究更具备参考价值。我们今天所学到的关于网络游戏的安全问题，会对其他软件行业中的相关领域产生很大的影响。

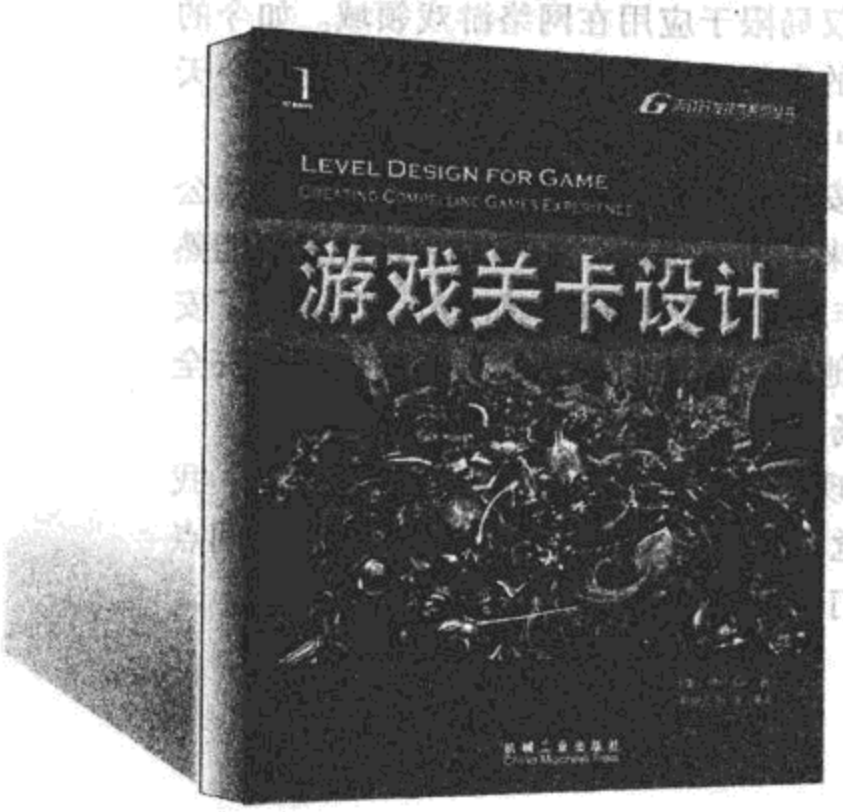
我们真诚地希望，本书所谈到的内容能为网络游戏安全问题的研究作出贡献。如果游戏公司能更重视到这一问题，他们可以看到他们的收入在未来会所有保证。如果游戏设计者们能熟悉这些攻击行为，他们就可以在安全问题上，在游戏制作之初便做得更好。如果玩家们理解了安全机制对于他们所处的游戏世界会产生多么大的影响，他们会对他们所爱的游戏提出更多安全方面的需求——于是便可形成一种无形的力量来推动市场发展，推动网络安全问题的进步。

当我们开始对这个领域的研究充满兴趣时，我们发现这一领域的探索是神奇而美妙的。我们尝试将内容叙述得尽可能清晰。在这个过程中，我们觉得十分高兴，对于这本书在这个时间点推出时所描述的这些内容，我们也很满意，我们希望你们也一样。谢谢！

一本打开的书，
一扇开启的门，
通向科学圣殿的阶梯，
托起一流人才的基石。

HZ BOOKS
华章图书

游戏开发技术系列丛书



谁教会了《魔兽世界》的关卡设计？

游戏关卡设计

作者：Phil Co

书号：7-111-19872-7

定价：35.00元

- 《半条命》作者倾心写就
- 盛大公司专业团队翻译
- 暴雪总裁极力推荐

游戏设计基础

作者：Andrew Rollings; Ernest Adams

网络游戏安全揭密

作者：Greg Hoglund; Gary McGraw

OpenGL游戏编程

作者：徐明亮

书号：978-7-111-22670-3

定价：55.00元

勤勤肯肯，夯实图形编程基础
兢兢业业，掌握游戏学习技巧

游戏开发核心技术：剧本和角色创造

作者：Marianne Krawczyk

书号：978-7-111-21197-6

定价：49.00元

我认为游戏艺术的真正标志是有人承认自己在第17关时哭了。

——史蒂芬·斯皮尔伯格

游戏工业导论

作者：Michael E. Moore; Jennifer Sward

欲知更多华章计算机图书出版动态，敬请您访问华章IT官方博客：http://blog.csdn.net/hzbooks

专业成就人生
立体服务大众

HZ BOOKS

www.hzbook.com

填写读者调查表　加入华章书友会
获赠精彩技术书　参与活动和抽奖

尊敬的读者：

感谢您选择华章图书。为了聆听您的意见，以便我们能够为您提供更优秀的图书产品，敬请您抽出宝贵的时间填写本表，并按底部的地址邮寄给我们（您也可通过www.hzbook.com填写本表）。您将加入我们的“华章书友会”，及时获得新书资讯，免费参加书友会活动。我们将定期选出若干名热心读者，免费赠送我们出版的图书。请一定填写书名书号并留全您的联系信息，以便我们联络您，谢谢！

书名：　　书号：7-111-(　　)

姓名：	性别：☐男　☐女	年龄：	职业：
通信地址：		E-mail：	
电话：	手机：	邮编：	

1. 您是如何获知本书的：

☐朋友推荐　☐书店　☐图书目录　☐杂志、报纸、网络等　☐其他

2. 您从哪里购买本书：

☐新华书店　☐计算机专业书店　☐网上书店　☐其他

3. 您对本书的评价是：

技术内容	☐很好	☐一般	☐较差	☐理由_____
文字质量	☐很好	☐一般	☐较差	☐理由_____
版式封面	☐很好	☐一般	☐较差	☐理由_____
印装质量	☐很好	☐一般	☐较差	☐理由_____
图书定价	☐太高	☐合适	☐较低	☐理由_____

4. 您希望我们的图书在哪些方面进行改进？

_____

_____

5. 您最希望我们出版哪方面的图书？如果有英文版请写出书名。

_____

_____

6. 您有没有写作或翻译技术图书的想法？

☐是，我的计划是_____　☐否

7. 您希望获取图书信息的形式：

☐邮件　☐信函　☐短信　☐其他_____

请寄：北京市西城区百万庄南街1号　机械工业出版社　华章公司　计算机图书策划部收

邮编：100037　电话：（010）88379512　传真：（010）68311602　E-mail: hzjsj@hzbook.com