TURING 图灵程序设计丛书

PEARSON

Surreptitious Software

Obfuscation, Watermarking, and Tamperproofing for Software Protection

软件加密与解密

- 领军人物专业解读安全新领域
- 全面权威的软件保护指南
- 精到的算法解说与代码完美结合

[美] Christian Collberg
Jasvir Nagra 著

崔孝晨 译

人民邮电出版社
POSTS & TELECOM PRESS

TURING 图灵程序设计丛书

Surreptitious Software

Obfuscation, Watermarking, and Tamperproofing for Software Protection

软件加密与解密

[美] Christian Collberg
Jasvir Nagra 著

崔孝晨 译

人民邮电出版社
北京

图书在版编目（CIP）数据

软件加密与解密 / (美) 科尔伯格 (Collberg,C.)，(美) 纳盖雷 (Nagra,J.) 著；崔孝晨译. -- 北京：人民邮电出版社，2012.5
（图灵程序设计丛书）
书名原文：Surreptitious Software:Obfuscation, Watermarking,and Tamperproofing for Software Protection
ISBN 978-7-115-27075-7

Ⅰ. ①软… Ⅱ. ①科… ②纳… ③崔… Ⅲ. ①软件加密②解密软件 Ⅳ. ①TP309.7

中国版本图书馆CIP数据核字(2011)第264440号

内 容 提 要

本书介绍了如何利用混淆、水印和防篡改等技术，来保护软件免受盗版、篡改和恶意逆向工程的危害，主要内容包括攻击者和防御者用来分析程序的各种主流方法，如何使用代码混淆技术使程序更难以被分析和理解，如何在软件中添加水印和指纹以标识软件的开发者和购买用户，等等。

本书适合各层次软件开发人员阅读。

图灵程序设计丛书

软件加密与解密

◆ 著 [美] Christian Collberg Jasvir Nagra
译 崔孝晨
责任编辑 卢秀丽
执行编辑 杨 爽 毛倩倩

◆ 人民邮电出版社出版发行 北京市崇文区夕照寺街14号
邮编 100061 电子邮件 315@ptpress.com.cn
网址 http://www.ptpress.com.cn
北京艺辉印刷有限公司印刷

◆ 开本：800×1000 1/16
印张：38.75
字数：1037千字 2012年5月第1版
印数：1-4 000册 2012年5月北京第1次印刷

著作权合同登记号 图字：01-2010-0392号

ISBN 978-7-115-27075-7

定价：99.00元

读者服务热线：(010)51095186转604 印装质量热线：(010)67129223

反盗版热线：(010)67171154

以我全部的爱、自豪和赞美，献给Louise和Andrew。

——Christian Collberg

献给Shwetha，你对我坚贞不渝的爱和支持，让一切皆有可能。

——Jasvir Nagra

译 者 序

《无隐录》上说："道与艺，具有正眼。得此而后工力有所施，否则毕世滑茫茫耳！"[①]。那么软件保护和破解的正眼又在哪里呢？按我的经验，刚学会使用调试器和反汇编器破开了一两个软件的"破解者"是最骄傲的。可不是嘛！你看那些原本不可一世的软件都败在了自己手下，这感觉……可是当他玩得越来越深入，就会遇到越来越多的反击措施。有些反击措施纯粹就是要把你玩死——程序中有时会充满成千上万个防逆向工程的"小把戏"（比如花指令），搞定一两个可能是轻而易举的事，但是在整个破解过程中你也许会遇到几千甚至几万个这样的小把戏！于是愉悦的心情变得沮丧，你不断地重复劳动着，一遍一遍又一遍……直到自己完全崩溃！也许你更聪明些，在彻底崩溃之前，会去"看雪"、"第八个男人"之类的软件保护技术网站逛一逛，下载一些脚本工具，运行一下，就能把这些小把戏统统搞定了。但是如果仅仅满足于这样一个状况，那么你终究只是一个脚本小子（script kid），修不成正果！真正的"黑客"或者"破解者"应该去追寻这些工具是怎么编写出来的，为什么它能工作得这么高效，为什么它就能自动地将程序中的那些恼人的东东全都识别出来，一一搞定，还不出错！如果你有志于这方面的探索，这本书就是不错的参考手册。书中详细地介绍了如何将数学和计算机科学中的一些原理、方法和技巧应用在程序破解中，它将使你功力大增。套用*Exploiting Software: How to break code*上的一句话："攻击计算机系统是一个科学过程，就好像一门艺术。事实上，相比其他随心所欲的游戏，挥舞着'科学方法'这把利器的攻击者占了游戏的上风。"[②]

如果你是个软件工程师，想要保护你的软件免遭破解，这些技巧是必不可少的。为什么呢？因为诸如如何自动理解程序的这些技术对于在软件中加入保护措施也是必不可少的。你一定不想因为添加软件保护措施而给程序引入新的bug，因此使用语义保留的代码转换技术非常必要。如何做到语义保留呢？只有使用"编译原理"中的数据流分析和控制流分析技术，自动理解程序才能做到这一点。而在设计一个同态加密算法[③]时，你又可以使用近世代数中同态域转换来达到目的。

本书的英文名是*Surreptitious Software*， surreptitious一词的意思是"保密的，隐瞒的"，所以surreptitious software的意思就是"隐蔽软件"。

本书本质上算是一本教科书，这既是它的优点也是它的缺点。说优点，是因为它巨细无靡地概述了这门学科的全貌，每章都给出了一些思考题，供读者进一步思考。对于有经验的读者，这将大大加

---

① 《无隐录》由明末清初吴殳著，是一本讲述我国古代枪法的武术专著。这段论述见该书自序。（书中脚注除特别说明之处之外，均为译者注。）

② 见该书中文版《软件剖析——代码攻防之道》第58页（清华大学出版社，2005年4月第1版）。

③ 也就是可以直接对密文进行某些运算，其效果就等价于将密文解密后对明文进行某种运算，然后再将明文加密回去的加密系统。详见本书第5章。

深其理论视野，帮助提高其自身功力。说缺点，是因为软件的保护和破解技术是一门实践性很强的艺术，如果读者没有丰富的实践经验就容易陷入空谈的境地，变成“理论家”。由于本书实际上是本教科书，基本上满足于对各种软件保护/破解技术的理论描述，缺少对各种平台下具体分析工具的介绍和案例支撑，因此没有破解经验的读者从中能够得到的收获就相对比较少了。本书假设读者已经拥有了一定的软件保护/破解经验，如果读者没有这方面的经验，可以先阅读一些基础教程，然后再来阅读本书，这样收获会更大些。这就好比小学生不能直接看大学的教材，但是当他长大以后读完高中课程，达到了一定水平，大学教材就是不得不读的。又正如《无隐录》所说：“夫枪难事也，纵得正眼，而造艺（诣）之高下，存乎工力之深浅，秘之[①]何为？”

有网友戏称：“术语这个玩意儿虽然能让同行之间说话更简洁，但是它最大的作用还是糊弄外行。”的确，译文中术语翻译的好坏也会影响到整本书是否易于理解。对于书中术语的翻译，我采用了如下原则。

对于那些有多个中文译名的术语，我选用其中最易于理解的。比如单词dominate，“龙书”（《编译原理（原书第2版）》，机械工业出版社，2009年1月）中将其译为“支配结点”，“鲸书”（《高级编译器设计与实现》，机械工业出版社，2005年7月）中将其译为“必经结点”。我认为术语“必经结点”（控制流流到结点*A*之前必然经过的结点）比较好，因为你能很方便地根据这个名称（望文）猜出它的作用（生义），而不需要任何编译原理方面的知识积累，所以在本书中我将单词dominate译为“必经结点”。

对于那些已有对应中文译名的术语，如果我认为中文译名不恰当，会给出我的译法并详述理由。比如第8章中的Planted Plane Cubic Trees，国内有些文献中译为“平面环路树”[比如《基于PPCT和基数*k*的动态图混合编码方案》（《计算机工程与应用》，2010，46（25））]，但是为什么这么译呢？特别是其中的“环路”实在是莫名其妙！难道就是因为cubic和circuit有点像？我给出的译名是“根延伸的平面三叉树”。其中，“根延伸的”（Planted）是指在二叉树的根结点上还延伸了一个结点，作为整棵树的根结点（书中附图中最高的结点是整棵树的根结点，而次高的结点是二叉树的根结点）；“平面”则是指这是一张平面图；而“三叉”（Cubic）则来自于这种图的基础图（亦称无向图）版本——在将这种图转换成无向图之后，每个结点（除了根结点和叶子结点）都正好只有3条边，所以称为“三叉”。（详见8.10.4节）

对于那些没有标准中文译名的术语，我尽量给出中文译名。比如Mobile Agent，我根据实际情况将其意译为“网上自动询价机器人”，方便读者理解（详见1.4.1节的第三小节）。

另外，对于极少的单词，我选择不将它翻译出来。比如trace，我就没有把它翻译出来，尽管翻译出来并没有错，但是我认为这类单词是非常常用的术语，在各种工具中也经常使用，最好还是不要翻译。

在翻译过程中，我也发现原书中存在一些错误。如图3-41，作者可能是为了说明函数起始指令（prologue）的重要性，让某条跳转指令多跳了一个字节，于是整个反汇编的结果就出现了一些问题，作者认为这是没有prologue指令给反汇编带来的问题，但是实际上这个解释实在是有些牵强——因为作者分析的程序原本就错了，根本不能正常执行！在翻译过程中遇到这类问题，我都是本着忠实于原文的精神，将原文原封不动地译出，然后再添加注说明我的观点供读者参考。

此外，书中还存在一些“低级”错误，比如将代码清单中的“int”打成了“in0”，这类问题可能

① 这个“之”指“正眼”。

是作者在书中加入的水印（因为作者在书中已经声明了这一点，而且在本书的电子版中我也发现作者对各行的行距进行了微调），也有可能是印刷错误，对于这类错误，我直接予以修正了。

原文中有些地方存有歧义，比如算法7.1中的单词dominate，你既可以把它理解成“某些结点的必经结点”，也可以把它理解成“在某些结点之前”。究竟应该怎样理解呢？对于这类情况，我都会去阅读相应的参考文献（因为参考文献的论述更为详细），做出判断。如这个例子中，通过阅读参考文献[59]，我确认后一种理解才是正解。

最后，在这本书中，Tamperproofing技术还包括一部分反调试、反模拟技术。但是为了整本书的逻辑连贯性，我还是将原文“Tamperproofing”直译为“防篡改技术”，但这一名称并不足以囊括其所指代的所有技术内容。请读者在阅读时注意。

我衷心地感谢人民邮电出版社图灵公司对本书出版所给予的大力支持，由于李松峰老师、傅志红老师的指导和大力推动，我才能完成本书的翻译工作。

在本书的翻译过程中，team509的wooshi等人提出了宝贵意见，特在此表示感谢。

还有，我要感谢我的爱妻，没有她在这段时间里对我的容忍，并挑起了家中所有的重担，我也不会用这么多时间完成本书的翻译。

本书阅读和翻译起来相对比较吃力，这是因为：一来作为一本准教科书，这本书的学究气比较重一点，措辞比较晦涩；二来本书涉猎了数论、图论、编译原理、通信技术等多门学科，穿插使用了这些学科中的术语和符号，使得翻译的难度也大了不少。再加上翻译时间仓促，书中存在错误在所难免，敬请读者不吝指正。

崔孝晨

2011年5月

前　言

隐蔽软件（surreptitious software）是近十年来计算机安全研究领域新兴的一个分支。在隐蔽软件的研究过程中不仅需要借鉴计算机安全方面的技术，还会用到计算科学其他领域的大量技术，如密码学、隐写术、数字水印、软件量度（software metric）、逆向工程以及编译器优化等。我们使用这些技术来满足在计算机程序中安全存储秘密信息的需求，尽管这些需求的表现形式千差万别、各不相同。本书中“秘密”一词的意思比较广，书中所介绍技术（代码混淆、软件水印和指纹、防篡改技术以及软件“胎记”等）的使用目的是防止他人剽窃软件中的智力成果。比如，软件中使用指纹技术可以用来跟踪软件是否被盗版，代码混淆技术能够加大攻击者逆向分析软件的难度，而防篡改技术则可以使别人很难制作软件的破解版，等等。

好了，现在我们来讲讲为什么需要阅读本书，谁使用隐蔽软件以及本书将会涵盖哪些内容。

为什么阅读本书

与传统的安全研究不同，隐蔽软件不关心如何使计算机免于计算机病毒入侵，它关心的是计算机病毒的作者是如何防止他人分析病毒的！同样，我们也不关心软件到底有没有安全漏洞，我们关心的是如何隐蔽地在程序中加入一些只有在程序被篡改时才会执行的代码。密码学研究领域中，被加密数据的安全性依赖于加密密钥的隐秘性，而我们现在研究的恰恰是如何隐藏密钥。软件工程中有大量的软件量度技术，以确保程序结构良好，本书中将使用同样的技术使程序复杂难读。本书中描述的很多技术都是基于编译器优化技术研究开发的算法的，但是编译优化的目的是使编译器生成个头尽量小、运行速度尽量快的程序，而使用本书中介绍的一些技术却会使生成的程序个头又大，执行起来又慢。最后，传统的数字水印和隐写术是想办法把要隐藏的信息藏到图像、音频、视频甚至纯文本文件中，而隐蔽软件则是把需要隐藏的信息藏到计算机代码中。

那么，为什么要阅读本书呢？为什么要了解一种不能防止计算机被病毒或者蠕虫攻击的安全技术？为什么要学习一种只会让代码体积变大而执行速度变慢的编译优化技术？为什么要把精力花在一种违反了密码学基本前提（即密钥是不可能被攻击者获得的）的密码学分支上呢？

回答是，传统的计算机安全和密码学研究成果有时并不能解决实际工作中遇到的且亟待解决的安全问题。比如，在本书中将展示如何使用软件水印技术防止软件盗版。软件水印是在程序中嵌入的唯一标识（类似信用卡的卡号或者版权声明），通过这个标识，程序的某个副本就和你（程序的作者）或者客户联系在了一起。要是你发现市场上在卖自己软件的盗版光盘，就可以通过在盗版软件中提取

的水印追查制作这个盗版软件的母版[①]当初是哪个家伙从你这里买走的。当给合作商提供新开发的游戏的测试版时，你也可以在测试版中加上数字水印。要是你感觉有人泄露了你的代码，就能（从众多的合作商中）找出肇事者，并把他送上法庭。

又比如，在程序的新版本中加上了某个新的算法，你当然不希望竞争对手也得到这个算法，并把它加到他们的软件中。这时，你就可以去混淆程序，使之尽可能变得复杂难懂，使竞争对手逆向分析软件时效率很低。而如果确实怀疑某人剽窃了你的代码，本书也会教你如何使用软件“胎记”证实你的怀疑。

再比如，你的程序中包含有某段不能为人所知的代码，并且你想确保没有这段代码程序就不能正常运行。例如，你肯定不希望黑客修改程序中的软件使用许可验证代码，或者可用于解密数字版权管理系统中mp3文件的密钥。第7章将讨论多种防篡改技术，确保受到篡改的程序停止正常运行。

听说你把密钥放在可执行文件里了？这主意实在太糟糕了！以往的经验告诉我们，任何类似“不公开，即安全”[②]的做法最终都将以失败告终，而且不管在程序中怎样隐藏密钥，最终它都逃不出一个足够顽强的逆向分析人员的手心。当然，必须承认你的做法也还是对的。本书中介绍的所有技巧都不能保证软件能永远免于黑客的毒手。不必保证某个东西永远处于保密的状态，也不必保证程序永远处于不可能被篡改的状态，更不需要保证代码永远不会被剽窃。除非这个研究领域有什么重大的突破，否则能指望的只是延缓对方的攻击。我们的目标就是把攻击者的攻击速度减缓到足够低，使他感到攻击你的软件十分痛苦或要付出过高的代价，从而放弃攻击。也可能攻击者很有耐心地花了很长时间攻破了你的防御，但这时你已经从这个软件中赚够了钱，或者已经用上了更新版本的代码（这时他得到的东西也就一钱不值了）。

比方说，你是一个付费频道的运营商，用户通过机顶盒来观看你提供的电视节目。每个机顶盒都是带有标签的——在代码的某个位置上存放了分配给每个用户的唯一标识（ID），这样你就可以根据用户的缴费情况决定是允许还是拒绝某个特定用户观看频道里的节目。可是现在有一个黑客团伙找到并且反汇编了这段代码，发现了计算用户ID的算法，并且在网上以低廉的价格把修改用户ID的方法卖给了网民。这时你该怎么办呢？你也许想到了使用防篡改的智能卡，不过这玩意儿并不像看上去那么难破解，这将在第11章中讲解。或者你可能想到要混淆代码，使之更难以被分析。或者你也可以使用防篡改技术使程序一被修改就自动停止运行。更有可能，你会混合使用上述各种技巧来保护代码。但是尽管使用了所有技术，你还必须要知道并且必须接受，你的代码仍然可能被破解，秘密仍会泄露（在这个案例里就是机顶盒里的用户ID仍然会被篡改）这一事实。怎么会这样呢？这只是因为“不公开，既安全”这个想法在根本上就存在漏洞。不过既然本书中介绍的所有技术都不能给你一个“完美并且长期的安全保证”，那么为什么还要使用这些技术，为什么还要买这样一本书呢？答案很简单，代码能顶住黑客攻击的时间越长，订阅频道的客户就越多，同时升级机顶盒的周期也就越长，这样你赚到

① 多数盗版软件的制作流程是这样的，盗版软件的制作人员先想尽办法搞到一套正版软件，然后由黑客将正版软件中的版权验证代码删掉，制成crack文件，或者把版权验证算法破解出来，写出注册机。把正版软件变成盗版软件。而一开始被搞来用于制作盗版的正版软件就称为母版。

② “security through obscurity”，就是通过把秘密藏起来不让人知道的方法保守秘密。在传统的密码学中这一做法一直是反面典型，因为使用这种方法保守的秘密理论上最终都会被攻击者发现。但是在本书中作者是部分地鼓励这种做法的，因为作者认为只要能够在足够长的时间里不让攻击者发现秘密，即使攻击者最终能够发现秘密，但这时早就没有必要再保守这个秘密了。

的钱和省下的钱也就越多。

就这么简单。

谁使用隐蔽软件

很多知名的公司都对隐蔽软件有浓厚的兴趣。事实上很难真正掌握有关技术在实践中具体被使用的程度（因为大多数公司在如何保护自己的代码一事上绝对是守口如瓶的），但是我们还是可以根据他们专利的申请和拥有情况把他们对隐蔽软件的感兴趣程度猜个八九不离十。微软公司拥有多个关于软件水印[104,354]、代码混淆[62,62,69,69,70,70,180,378]和软件“胎记”[364]技术的专利。Intertrust公司拥有大量与数字版权管理技术相关的组合式专利，包括代码混淆和代码防篡改专利。2004年，在微软与Intertrust之间的马拉松式官司落下了帷幕之后，微软向Intertrust支付了高达4.4亿美元的专利使用费，才获得了后者所有的专利使用许可。同年，微软也开始与PreEmptive Solution公司开展商业合作[250]，从而把PreEmptive Solution开发的identifier obfuscator（PreEmptive solution公司在该工具中拥有专利[351]）加到了Visual Studio的工具集里。而普渡大学科研成果的副产品Arxan，因其独创的防篡改算法专利[24,305]而成功地开办了一家公司。苹果公司拥有一个代码混淆方面的专利，估计是用于保护其iTune软件的。Convera，一家从英特尔公司独立出来的企业，则着力研究应用于数字版权管理的代码防篡改技术[27,268-270]。从加拿大北方电信公司中分离出来的Cloakware公司也是这个领域里最成功的企业之一。该公司拥有他们称为“白盒加密”的专利[67,68,182]，即把加密算法和密钥藏到程序代码中。2007年12月，Cloakware公司被一家主营付费电视业务的荷兰公司Irdeto以7250万美元的价格收购。即使是相对的后来者Sun Microsystem也已经提交了一些代码混淆领域的专利申请。

Skype的VoIP客户端也使用了类似Arxan[24]、英特尔[27]及本书中将要提到的[89]代码混淆和防篡改技术进行了防逆向工程加固。对于Skype公司来说，保护其客户端的完整性无疑是极其重要的，因为一旦有人成功逆向分析了其客户端软件，解析出Skype所使用的网络协议，黑客们就能写出廉价的能与Skype软件进行正常通信的程序（这样的话，人们就没有必要一定用Skype）。所以保持网络协议不公开则有助于Skype拥有一个庞大的用户群，这大概也是2005年易贝公司以26亿美元收购Skype的原因吧。实际上，使用隐蔽软件技术还使Skype公司赢得了足够多的时间，进而成为了VoIP技术的领军企业。即使这时Skype的协议被分析出来了（这一点黑客们确实也做到了，详见7.2.4节），黑客们也拿不出一个能够撼动Skype市场地位的类似软件了。

学术研究者从多种角度对隐蔽软件技术进行了研究。一些拥有编译器和程序语言研究背景的研究者，比如我们，会很自然地加入这一领域的研究，因为涉及代码转换的绝大多数算法都会涉及静态分析的问题，而这一问题则是编译优化技术的研究者再熟悉不过的了。尽管以前，密码学研究者大多不屑于研究“不公开，即安全”的问题，但最近一些密码学研究人员已经开始把密码学的相关技术应用于软件水印以及发现代码混淆技术的局限性上了。来自多媒体水印、计算机安全和软件工程方面的研究人员也已经发表了很多关于隐蔽软件的文章。遗憾的是，由于没有专门的刊物、学术会议（供研究人员相互之间进行交流），这一领域的研究进展被大大延缓了。事实上，为了使这些研究成果能被传统的学术会议和期刊接受，研究人员在不停地努力着，现在仍在努力。目前已经发表过隐蔽软件研究成果的学术会议有POPL（Principles of Programming Languages，程序设计原理）上的ACM专题研讨会、信息隐藏研讨会、IEEE的软件工程研讨会、高级密码学会议（CRYPTO）、ISC（Information Security

Conference，信息安全大会）以及其他一些关于数字版权管理的学术会议。随着隐蔽软件这一领域的研究越来越成为学术研究的主流，我们有望拥有专门针对于隐蔽软件的期刊、专题讨论会甚至是研讨会，只是可惜目前为止这一切都还没有实现。

军方也在隐蔽软件上花了很多精力（和纳税人的钱）。比如，Cousot公司拥有的软件水印算法[95]专利就归属于世界上第九大国防工程承包商法国Thales集团。下面是一段引自最新的（2006）美军招标文件[303]中有关AT（anti-tamper）技术①研究的文字。

> 现在，所有的美军项目执行部门（PEO）和项目管理方（PM）在设计和实现有关系统时，必须在系统中使用军队和国防部制定的AT策略。嵌入式软件现代武器系统的核心，是被保护的最重要技术之一。AT技术能够有效地保证这些技术不被他国（人）逆向工程分析利用。仅仅由标准编译器编译生成而不加AT技术防护的代码是很容易被逆向分析的。在分析软件时，逆向工程分析人员会综合使用诸如调试器、反编译器、反汇编器等很多工具，也会使用各种静态和动态分析技巧。而使用AT技术的目的就是使逆向工程变得更为困难，进而防止美国在技术领域的优势被他国窃取。今后还有必要向部队的PEO和PM提供更有用、更有效并且多样化的AT工具集……研发AT技术的目的在于提供一个能够抗逆向工程分析的高强度壳②，从而最大限度地迟滞敌方对被保护软件的攻击。这样美国就有机会维持其在高科技领域的优势或者减缓其武器技术泄密的速度。最终，美军就能继续保持其技术优势，进而保证其军备的绝对优势。

这份招标文件来自于美军导弹和空间程序（设计部门），专注于实时嵌入式系统的保护。我们有理由相信产生这份招标文件的原因是，美军担心射向敌方的导弹由于种种原因落地后未能爆炸，使敌方有机会接触到嵌入在导弹中负责引导导弹飞临目标上空的控制软件。

下面是另一段引自美国国防部[115]的文字。

> 进行主动式软件保护③（SPI）是国防部的职责之一，它必须开发和部署相关的保护技术，以保证含有国防武器系统关键信息的计算机程序的安全。SPI提供的是一种全新的安全防护方法，它并不（像传统的安全技术那样）保护计算机或者网络的安全，而只是加强计算机程序自身的安全。这种新方法能显著提升国防部的信息安全情况。SPI的适用范围很广，从台式机到超级计算机上面所有的程序都能使用SPI技术予以保护。它是（软件保护技术中）完整的一层，是"纵深防御"的一个范例。SPI技术是对网络防火墙、物理安全等传统安全技术的一个补充，

① 实际上，美军的术语"AT技术"中包含了几乎所有抗逆向工程的技术，其中除了包含本书中讨论的代码混淆和防篡改技术，还包括其他一些抗逆向工程技术，比如调试器检测技术——使程序在正常执行时表现正常，但一旦程序在调试器下运行就会立即崩溃。所以在这里我并没有按字面的意思把anti-tamper翻译成防篡改技术，因为这样太容易与本书中专指的防篡改技术的概念混淆了，而是直接使用了这个英文单词的缩写"AT"。

② 参见前面对于单词AT的翻译，在国内将把抗逆向工程技术应用到软件上的术语称为"加壳"，把去掉软件中的抗逆向工程技术称为"脱壳"，软件中的抗逆向工程技术也就被称为"壳"。所以在这里我把"layer"译为"壳"，这样对于国内有一定技术基础的读者来说，读起来更亲切些。

③ 从上下文和美军的使用实际情况来看，SPI技术指的是利用水印、软件固有特征等技术保护软件。前面讨论的使用代码混淆、防篡改等技术进行软件保护的方法是，让别人破不开你的软件。这属于被动式的保护（因为是等着被人来攻击你的软件）。而使用水印、软件固有特征等技术保护软件的方法是，当软件被攻破之后，根据有关特征进行责任倒查和追究。这属于主动式的保护（因为是主动找上门去）。所以在这里我将SPI译为"主动式软件保护"。

但是其实现并不依赖于这些传统的安全设备。现在SPI技术被部署在选定的HPC中心和150多家国防部机关以及其他由商业公司参与建设和维护的军事基地。广泛地部署SPI技术将会有效地增强美国和美国国防部对关键应用技术的保护。

上面这段话说明了什么？它说明美国国防部不仅关心导弹会不会掉到敌方领土上去，还关心在自己的安全系数和性能都很高的计算机中心运行的软件的安全。事实上，窃密和反窃密是防间谍机关和情报部门之间永恒的主题。比方说，一架战斗机上的某个程序需要更新一下，这时我们很可能就是用一台笔记本电脑连接到这架战斗机上进行更新操作。但是万一这台笔记本电脑不慎遗失了，或者干脆就被其他国家政府使用某种方法控制了，就像电影里常演的那样，这时会有什么情况发生呢？对方会马上把相关的代码拿去做逆向工程分析，并把分析的结果用于改进其战斗机中所使用的软件。更有甚者，对方会悄悄地在你的软件中加上一个特洛伊木马，并让飞机在特定的时间里从天上掉下来。如果我们不能绝对保证上述这一幕100%不可能发生的话，隐蔽软件至少可以作为安全防御的最后一道防线（至少还能做到事后的责任追究）。例如，飞机中的软件可以用有权访问相关软件的人的ID做一个指纹签名。要是哪天，在其他国家的战斗机上发现了这些代码，就可以立即对这些代码进行逆向分析，并进一步推算出谁是泄密事件的元凶。

什么？我听见你说，为什么我要对政府之间和商业巨头之间如何保护它们各自的秘密感兴趣呢？如果黑客破解了这些软件，他们也不过是通过自己的劳动换取一些微薄的利益而已啊。话虽如此，但是这些保护技术给你[①]带来的好处最终还是大于它给商业巨头带来的好处。理由是，对你来说，法律形式的保护措施（如专利、商标和版权）只有当你拥有足够的财力，能在法庭上把对方告倒的时候才会管用。换而言之，即使你认为某家大公司通过破解你的代码，剽窃了一个极有“钱途”的主意，你也无力通过那种马拉松式的官司在法庭上告倒微软，除非你有足够的经济实力能在这种财力的比拼中熬出头[②]。而在本书中讨论的保护技术（比如代码混淆和防篡改技术）则既廉价又好用，中小型企业和商业巨头均可使用。而且如果这时你去告这家大公司的话，也可以用水印或者软件“胎记”等技术，在法庭上当场拿出代码被剽窃的真凭实据来。

最后不得不简单地提一下另一类极其擅长使用隐蔽软件的人——坏蛋们。病毒的作者已经能非常成功地利用代码混淆的技术伪装病毒的代码，使之逃避杀毒软件的检测了。值得一提的是，人们使用这些技术（如保护DVD、游戏和有线电视）时经常被黑客破解，而黑客使用这些技术（如构建恶意软件）时，人们却很难抗击。

本书内容

隐蔽软件研究的目的是发明能够尽可能迟滞对手（逆向工程分析）进度，同时又尽可能地减少因为使用该技术，而在程序执行时增加的计算开销的算法。同时也需要发明一种评估技术，使我们可以说“在程序中使用了算法A之后，相对于原先的程序，黑客攻破新程序需要多花T个单位的时间，而新程序增加的性能开销是0”，或者最低限度我们也应该可以说“相对于算法B，使用算法A保护的代码更难被攻破”。特别要强调一下，隐蔽软件研究尚处于婴儿期，虽然我们在书中会把相关的保护算法和评估算法全都介

① 这个是指规模相对较小的软件开发商。

② 估计你是办不到的。——作者注

绍给大家，但是这门艺术的现状却还并不理想（到时候你可不能太失望啊）。

在本书中，我们试图把当前所有有关隐蔽软件的研究成果组织起来系统化地介绍给读者。我们力争每章内容涵盖一种技术，并描述这一技术的应用领域以及目前可用的算法。第1章将给出隐蔽软件这个领域的一些基本概念；第2章用对抗性演示的模式介绍黑客逆向分析软件时常用的工具和技巧，然后针对这些工具和技巧介绍如何防范黑客的攻击；第3章详细讲述黑客和软件保护方用于分析计算机程序的技术；第4章、第5章和第6章分别介绍与代码混淆有关的算法；第7章介绍与防篡改技术相关的算法；第8章和第9章分别介绍与水印相关的算法；第10章介绍与软件“胎记”相关的算法；第11章讲述基于硬件设备的软件保护技术。

如果你是位企业管理人员，只是对隐蔽软件的研究现状和这些技术怎么应用到你的项目中感兴趣，那么只要阅读第1章和第2章就够了。如果你是位拥有编译器设计背景的研究人员，那么建议直接跳到第3章开始阅读。但是之后的章节还是最好顺序阅读。这是因为……呃，还是举个例子吧，介绍水印技术的章节中会用到在代码混淆章节中介绍的知识。当然在本书撰写过程中，我们还是尽量使各章内容都能独立成章的，所以（如果你拥有一些背景知识）偶尔跳过那么一两章也未尝不可。如果你是一位工程师，想要使用有关技术加固你的软件，那么强烈建议你仔仔细细地阅读第3章的所有内容，如果有条件的话，还应该再搞几本编译原理方面的教材恶补一下“程序静态分析”的知识。然后你就可以随意跳到感兴趣的章节去阅读了。如果你是名大学生，把本书作为一门课程的教材来阅读，那么就应该一页一页地完整阅读本书，期末别忘了做好复习。

希望本书能够做到两件事情。首先，希望能向你，亲爱的读者，证明代码混淆、软件水印、软件“胎记”和防篡改等技术里有大量妙不可言的想法，值得你花点时间去学习，而且这些技术也可以用来保护软件。其次，希望本书能把本领域内当前所有有用的信息汇集在一起，从而为隐蔽软件的深入研究提供一个良好的起点。

Christian Collberg和Jasvir Nagra

2009年2月2日（土拨鼠日）

P.S. 实际上写作这本书还有第三个目的。要是在阅读本书时，你突然灵光闪现，冒出一个绝妙的主意，进而激发了你投身于隐蔽软件研究的雄心壮志，那么，亲爱的读者，我这第三个目的就算是达到了。请把你的新算法告诉我们，我们将把它加到本书的下一版里！

致　　谢

在此谨向下列各位朋友致谢，没有你们的帮助，本书是没法写好的：Amena Ali、Bertrand Anckaert、Gregory Andrews、Mike Atallah、Jan Cappaert、Edward Carter、Igor Crk、Mike Dager、Mila Dalla Preda、Saumya Debray、Fabrice Desclaux、Roberto Giacobazzi、Jon Giffin、Gael Hachez、Wulf Harder、William Horne、Mariusz Jakubowski、Yuichiro Kanzaki、Phil Kaslo、David P.Maher、Scott Moskowitz、Yoram Ofek、Reiner Sailer、Christine Scheer、Arvind Seshadri、Richard Snodgrass，以及参与CSc 620项目的研究员Clark Thomborson、Gregg Townsend、Tom Van Vleck和Glenn Wurster。

特别感谢中国科学院自动化研究所和王飞跃教授，在2006年8月至2007年7月的交流访问年中给予Christian Collberg的慷慨帮助。同时也特别感谢Yoram Ofek和RE-TRUST项目组在2008年夏季提供的帮助。

感谢Addison-Wesley出版社参与本书编辑出版的各位职员，诚挚地感谢Jessica Goldstein、Gary McGraw、Romny French、Elizabeth Ryan和Chuck Toporek。

最后，还要感谢我们的所有朋友和家人，在这本书的整个写作出版过程中，你们一直在鼓励和支持我们：Charlotte Bergh、Andrew Collberg、Melissa Fitch、Michael Hammer、John Hartman、Louise Holbrook、爸爸和妈妈、Ginger Myles、Qu “Jojo” Cheng、Shwetha Shankar、Haya Shulman、Snickers先生和Tudou。

版权声明

目　录