第 17 章　安全警报

现在，你已经知道 PowerShell 是多么强大，但是你也会突然意识到一个问题：这些已存在的强大功能会不会造成一些安全隐患？答案是“可能会”。在本章中，我们会帮助你了解 PowerShell 将如何影响环境的安全，同时会讲解如何配置 PowerShell 才能取得安全和强大功能上的平衡。同时，本章几乎所有的内容都是关于 Windows 版本的 PowerShell；大部分这些功能在 macOS 与 Linux 中都不存在，这是由于这些功能并没有与 macOS 和 Linux 的传统 Shell 体验保持一致。

17.1　保证 Shell 安全

自从 2006 年年底 PowerShell 发布以来，微软在安全和脚本方面并没有取得很好的名声。毕竟那个时候，VBScript 和 Windows Script Host(WSH)是两个最流行的病毒和恶意软件的载体，它们经常成为臭名昭著的“I Love You”“Melissa”等其他病毒的攻击点。当 PowerShell 团队宣布他们创造了一种新的、能提供前所未有强大的功能与可编程能力的命令行 Shell 语言时，我们认为，警报来临，人们将对这种新的命令行 Shell 避之不及。

但是，没关系。PowerShell 是在比尔·盖茨先生在微软发起的一个“可信赖计算计划”之后才进行开发的。在微软公司内部，该计划产生了很积极的效果：每个产品部门都要求配备一名资深软件安全专家，该专家会参与到设计会议、代码复审等工作中。该专家被称为产品的“安全伙伴”（并不是我们编造的）。PowerShell 产品的“安全伙伴”是经由微软出版的《编写安全代码》（*Writing Secure Code*）的其中一位作者，该书描写了如何编写不易受攻击者利用的软件。我们可以保证 PowerShell 与其他产品一样都是安全的——至少默认情况下，都是安全的。当然，你也可以修改这些默认值，但是当你进行操作时，请不要只考虑软件的功能，也要注意安全问题。这也就是本章要帮你完成的事情。

17.2 Windows PowerShell 的安全目标

我们需要明确，当谈及安全时，PowerShell 会做什么，又不会做什么；最好的办法是列出一些 PowerShell 的安全目标。

首先，PowerShell 不会给被处理的对象任何额外的权限。也就是说，PowerShell 仅会在你已拥有的权限主体下处理对象。比如，如果通过图形用户界面操作，你没有在活动目录中创建新用户的权限，那么在 PowerShell 中你也无法创建该用户。总体来说，PowerShell 仅仅是你使用当前权限来完成某些操作的一种实现方式而已。

其次，PowerShell 无法绕过既有的权限。比如，想为你的用户部署某个脚本，并希望该脚本能完成某些操作——正常情况下这些用户会由于权限不足无法完成的某些操作，那么该脚本同样不能运行。如果希望用户可以完成某些操作，那么必须给他们赋予对应的权限；PowerShell 仅能完成这些用户凭借现有权限执行命令或者脚本可以完成的工作。

设计 PowerShell 安全系统的目的并不是为了阻止用户在正常的权限下输入并运行某些命令。该思想使得欺骗用户输入很长的、较为复杂的命令变得更加困难，因此 PowerShell 不会应用超过该用户当前拥有的权限之外的安全设置。从过去的经验我们知道，欺骗用户运行一段可能包含恶意代码的命令是非常简单的事。这也就是为什么 PowerShell 的大部分安全设置都被设计为阻止用户运行一些未知的脚本。“意外”这个部分是非常重要的：PowerShell 的安全并不旨在阻止一个已确定用户运行的脚本，只是为了阻止用户被欺骗运行来自不受信任来源的脚本。

补充说明

下面讲到的内容超出本书范围，但是还是希望你能知道存在其他一些方法可以使得用户在其他凭据（而非自有凭据）下运行某些命令。通常称这种技术为脚本封装。它是一些商业脚本开发环境的一个特性，比如 SAPIEM PrimalScript(www.PrimalTools.com)。

创建一个脚本之后，你可以使用打包程序将这个脚本放入到一个可执行文件(.EXE)中。这并不是编码学中的编译过程：这个可执行文件并不是独立的，它需要在 PowerShell 安装之后才能执行。你也可以通过配置打包程序，将可用的凭据加密到可执行文件中。这样，如果有人运行该可执行文件，其中的脚本会在指定的凭据下被执行，而不依赖当前用户的凭据。

当然，被封装的凭据也不是百分之百安全。被封装的文件中都会包含用户名以及对应密码，尽管大部分打包程序都会进行用户及密码的加密。准确地说，针对大部分用户而言，他们都无法发现用户名以及对应的密码；但是针对一个熟练的加密专家来说，破解出用户名以及密码是很简单的一件事。

PowerShell 的安全并不是针对恶意软件的防护。一旦在你的系统上存在恶意软件，那么恶意软件可以做你权限范围内的任何事情。它可能使用 PowerShell 去执行一些恶意命令，也有可能非常轻易地使用多种其他技术损坏你的电脑。一旦在你的系统中存在恶

意软件，那么你就被“挟持”了。当然，PowerShell 也并不是第二道防御系统。此时，首先你需要杀毒软件来阻止恶意软件进入你的系统。对大部分人而言，可能忽略这样一个重要的概念：即使恶意软件可能借助 PowerShell 去完成一些危害行为，也不应该将恶意软件问题归咎于 PowerShell。杀毒软件必须阻止恶意软件运行。再次申明，PowerShell 设计出来并不是为了保护一个已经受损的系统。

17.3 执行策略和代码签名

PowerShell 中第一个安全措施是执行策略。执行策略是用来管理 PowerShell 执行脚本的一种计算机范围的设置选项。正如本章前面所讲，该策略主要用作防止用户被注入，从而执行一些非法脚本。

17.3.1 执行策略设置

默认设置是 Restricted，该策略会阻止正常脚本的运行。也就是说，默认情况下，你可以使用 PowerShell 进行交互式执行命令，但是你不能使用 PowerShell 执行脚本。如果你尝试执行脚本，你会得到下面的错误。

```
无法加载文件 C:\test.ps1，因为在此系统中禁止执行脚本。有关详细信息，请参阅 "get
-help about_signing"。
所在位置 行:1 字符: 11
+ .\test.ps1 <<<<
    + CategoryInfo          : NotSpecified: (:) [], PSSecurityException
    + FullyQualifiedErrorId :RuntimeException
```

你可以通过运行 Get-ExecutionPolicy 命令来查看当前的执行策略。另外，如果你想修改当前的执行策略，可以采用下面 3 种方式之一。

- 运行 Set-ExecutionPolicy 命令。该命令会修改 Windows 注册表中的 HKEY_LOCAL_MACHINE 部分，但是需要在管理员权限下才能执行该命令，因为一般用户没有修改注册表的权限。
- 使用组策略对象（GPO）。从 Windows Server 2008 R2 开始，Windows PowerShell 相关的设置已经包含在内。如果因为某些原因你不得不继续使用老版本的域（我们为你哀悼），可以通过访问网站 http://download.microsoft.com，之后在搜索框中搜索 PowerShell ADM 进行查找。

  如图 17.1 所示，我们可以在“本地计算机策略”→用户配置→管理模板→Windows 组件→Windows PowerShell 中找到 PowerShell 的设置选项。图 17.2 展示了我们将该策略设置为“启用”的状态。当通过组策略对象来配置时，组策略中的设定会覆盖本地的任何设置值。实际上，如果你试图运行 Set-ExecutionPolicy，命令可以正常执行，但是会返回一个警告。该警告会告知由于组策略覆盖的原因，

新修改的设定值不会生效。

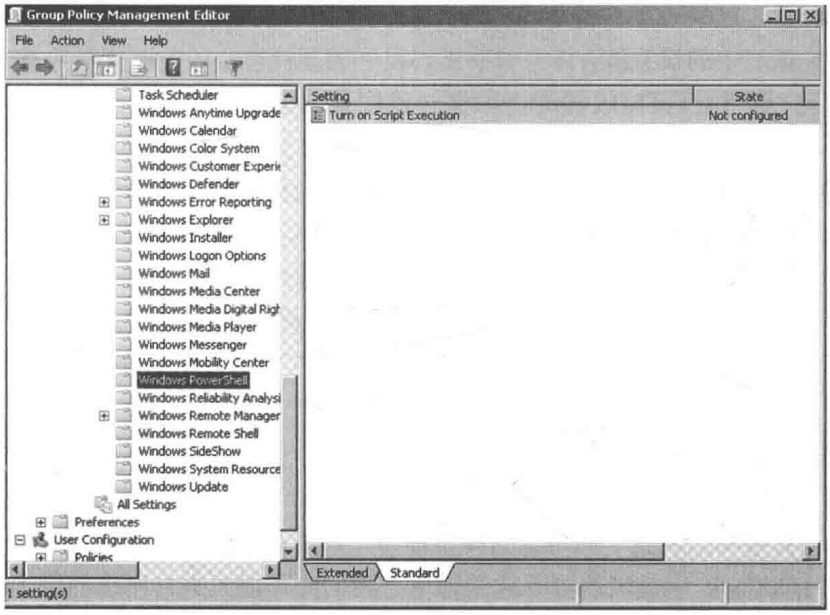

图 17.1 Windows PowerShell 设置在组策略中的位置

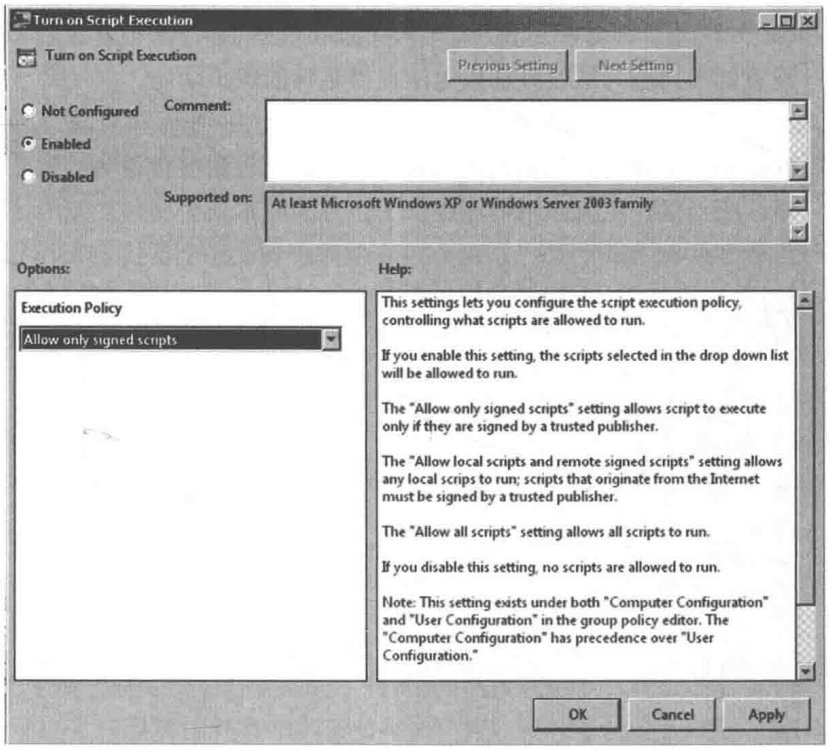

图 17.2 在组策略对象中修改 Windows PowerShell 的执行策略

- 通过手动运行 PowerShell.exe，并且给出-ExecutionPolicy 的命令行开关参数。如果采用这种方式，那么命令中指定的执行策略会覆盖本地任何设置和组策略中的设置值。

你可以将执行策略设置为 5 种值（请注意：组策略对象中包含下面列表中的 3 个选项）。

- Restricted——这是默认选项，除微软提供的一部分配置 PowerShell 的默认选项的脚本外，不允许执行其他任何脚本。这些脚本中附带微软的数字签名。如果修改这些数字签名，那么这些脚本就再也无法运行了。
- AllSigned——经过受信任的证书颁发机构（CA）设计的数字证书签名之后的任意脚本，PowerShell 均可执行。
- RemoteSigned——PowerShell 可以运行本地任何脚本，同时也可以执行受信任的 CA 签发的数字证书签名之后的远程脚本。“远程脚本”是指存在于远端计算机上的脚本，经常通过通用命名规则（UNC）方式访问这些脚本。我们也会将那些来自于网络上的脚本称为“远程脚本”。Internet Explorer、Firefox 和 Outlook 中提供的可下载的脚本，我们均可视为来自网络的脚本。在某些版本的 Windows 中，会区分网络路径以及 UNC 路径。在这些场景中，本地网络中的 UNC 都不会认为是“远程”。
- Unrestricted——可以运行所有脚本。我们并不是很喜欢或不建议使用这个设置选项，因为该设置选项无法提供足够的保护功能。
- Bypass——这个特殊的设定主要是针对应用程序开发人员，他们会将 PowerShell 嵌入到他们的应用程序中。这个设定值会忽略已经配置好的执行策略，应当仅在主机应用程序提供了自身的脚本安全层时才使用该选项。你最终告诉 PowerShell 的是“别担心，安全问题我已经全部搞定”。

等等，什么？

你是否注意到，我们可以在组策略对象中设置一种执行策略，但是也可以使用 PowerShell.exe 的一个参数来覆盖该设定？通过 GPO 控制的设定能被轻易覆盖，这样有什么好处呢？这里主要是体现了执行策略被设计出来的一个目的：防止不知情的用户无意中运行一些匿名脚本。

执行策略并不是为了阻止用户去运行某个已知的脚本。如果真是这样，那么执行策略就不算是一种安全设置。

事实上，一个聪明的恶意软件开发者可以更容易直接访问.NET Framework 的函数，而不是费力去使用 PowerShell 作为媒介。或是用其他方式，如果一个未经授权的用户拥有你计算机的管理员权限执行任意代码，你已经是在劫难逃了。

微软强烈建议在执行脚本时使用 RemoteSigned 执行策略，并且仅在需要执行脚本的机器上采用该策略。根据微软的建议，其他计算机应当继续保持 Restricted 的

执行策略。微软解释道：RemoteSigned 策略在安全性和功能之间取得了较好的平衡；AllSigned 相对更严格，但是它要求所有脚本都需要被数字签名。PowerShell 社区作为一个整体是更开放的，在到底哪种执行策略较优的问题上，存在大量的意见。就当前而言，我们会采纳微软的建议。当然，如果你有兴趣，你可以自己研究该主题。

现在，我们可以深入讨论数字签名的话题了。

注意：多个专家，包括微软的一些开发人员，都建议使用 Unrestricted 作为执行策略。他们觉得该功能并没有提供一个安全层，并且你也不应该相信该设置可以将任何危险的行为隔离开。

17.3.2 数字代码签名

数字代码签名，简称为代码签名，是指将一个密码签名应用到一个文本文件的过程。签名会显示在文件末端，并且类似下面的形式。

```
<!-- SIG # Begin signature block -->
<!-- MIIXXAYJKoZIhvcNAQcCoIIXTTCCF0kCAQExCzAJBgUrDgMCGgUAMGkGCisGAQQB -->
<!-- gjcCAQSgWzBZMDQGCisGAQQBgjcCAR4wJgIDAQAABBAfzDtgWUsITrck0sYpfvNR -->
<!-- AgEAAgEAAgEAAgEAAgEAMCEwCQYFKw4DAhoFAAQUJ7qroHx47PI1dIt4lBg6Y5Jo -->
<!-- UVigghIxMIIEYDCCA0ygAwIBAgIKLqsR3FD/XJ3LwDAJBgUrDgMCHQUAMHAxKzAp -->
<!-- YjcCn4FqI4n2XGOPsFq7OddgjFWEGjP1O5igggyiX4uzLLehpcur2iC2vzAZhSAU -->
<!-- DSq8UvRB4F4w45IoaYfBcOLzp6vOgEJydg4wggR6MIIDYqADAgECAgphBieBAAAA -->
<!-- ZngnZui2t++Fuc3uqv0SpAtZIikvz0DZVgQbdrVtZG1KVNvd8d6/n4PHgN9/TAI3 -->
<!-- an/xvmG4PNGSdjy8Dcbb5otiSjgByprAttPPf2EKUQrFPzREgZabAatwMKJbeRS4 -->
<!-- kd6Qy+RwkCn1UWIeaChbs0LJhix0jm38/pLCCOo1nL79E1sxJumCe6GtqjdWOIBn -->
<!-- KKe66D/GX7eGrfCVg2Vzgp4gG7fHADFEh3OcIvoILWc= -->
<!-- SIG # End signature block -->
```

签名中包含了两部分重要信息：一是列出了对脚本签名的公司或者组织；二是包含了对脚本的加密副本，并且 PowerShell 可以解密该副本。要理解这部分信息的工作原理，你需要了解一些背景知识。当然，这部分背景知识也会帮助在你的环境中决定该采用何种安全策略。

在创建一个数字签名之前，你需要拥有一个代码签名的证书。这些证书也被称为第三类证书。这些证书均由商业 CA 签发，比如 Cybertrust、GoDaddy、Thawte、VeriSign 等公司。当然，如果可能的话，你也可以从公司内部的公钥基础设施（PKI）中获取到该证书。正常情况下，第三类证书仅会签发给公司或者组织，而不会发给个人。当然，在公司内部可以签发给个人。在签发证书之前，CA 需要验证接收方的身份——证书类似一种数字识别卡，该卡上列出了持有者的姓名以及其他详细信息。比如，在签发证书给 XYZ 公司之前，CA 需要验证 XYZ 公司的授权代表人提交了该请求。在整个安全体系中，验证过程是其中最重要的环节，你应当仅信任能出色完成验证申请证书的公司身份工作的 CA。如果你对一个 CA 的验证流程不熟悉，那么你不应该信任该 CA。

应当在 Windows 的 IE 属性控制面板（也可以在组策略中配置）中配置信任关系。在该控制面板中，选择 Content 标签页，然后单击 Publishers 按钮。在弹出的对话框中，选择“受信任的根证书颁发机构”标签页。如图 17.3 所示，你可以看到计算机信任的 CA 列表。

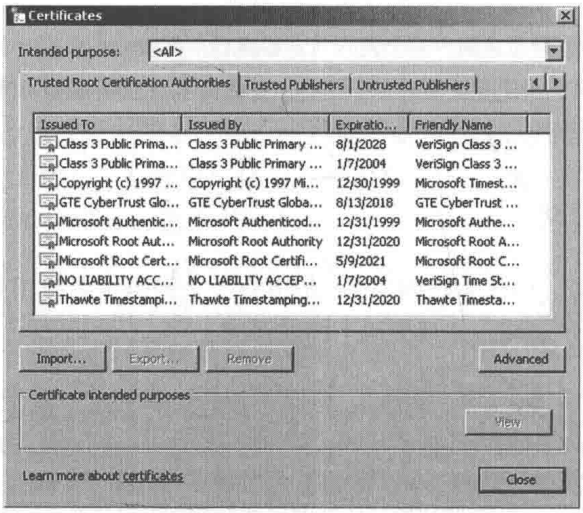

图 17.3 设置计算机的“受信任的根证书颁发机构”选项

当你信任一个 CA 之后，你也会信任该 CA 签发的所有证书。如果有人使用一个证书对恶意脚本进行签名，那么你可以通过该证书去查找该脚本的作者——这也就是为什么已签名的脚本相对于未签名的脚本更加值得“信任”。但是如果你信任一个无法很好验证身份的 CA，那么一个恶意脚本的作者可能会获取一个虚假的证书，这样你就无法使用该 CA 的证书去做追踪。这也就是为什么选择一个受信任的 CA 是如此重要。

一旦你获取了一个三级证书（具体而言，你需要一个包装为带有验证码的证书——通常 CA 会针对不同的操作系统以及不同的编程语言提供不同的证书），之后将该证书安装到本地计算机。安装之后，你可以使用 PowerShell 的 `Set-AuthenticodeSignature` Cmdlet 将该数字签名应用到一段脚本。如果需要查看更详细的信息，你可以在 PowerShell 中执行 `Help About_Signing` 命令。许多商业的脚本开发环境（PowerShell Studio、PowerShell Plus 以及 PowerGUI 等）都可进行签名，甚至可以在你保存一段脚本时进行自动签名，这样使得签名过程更加透明。

签名不仅会提供脚本作者的身份信息，也会确保在作者对脚本签名之后，不会被他人更改。实现原理如下。

（1）脚本作者持有一个数字证书，该密钥包含两个密钥：一个公钥、一个私钥。

（2）当对脚本进行签名时，该签名会被私钥加密。私钥仅能被脚本开发者访问，同时仅有公钥能对该脚本进行解密。在签名中会包含脚本的副本。

（3）当 PowerShell 运行该脚本时，它会使用作者的公钥（包含在签名中）解密该签名。如果解密失败，则说明签名被篡改，那么该脚本就无法被运行。如果签名中的脚本副本与明文文本不吻合，那么该签名就会被识别为损坏，该脚本也无法被运行。

图 17.4 描述了当执行脚本时，PowerShell 处理的整个流程。在该流程中，你可以看到为什么 `AllSigned` 执行策略在某种意义上说更加安全：在该种执行策略下，仅有包含签名的脚本才能被运行，也就意味着，你总是能识别某段脚本的作者。如果需要执行某段脚本，那么就会要求对该脚本进行签名。当然，如果你修改了该脚本，你也就需要对该脚本重新签名（可能稍显烦琐）。

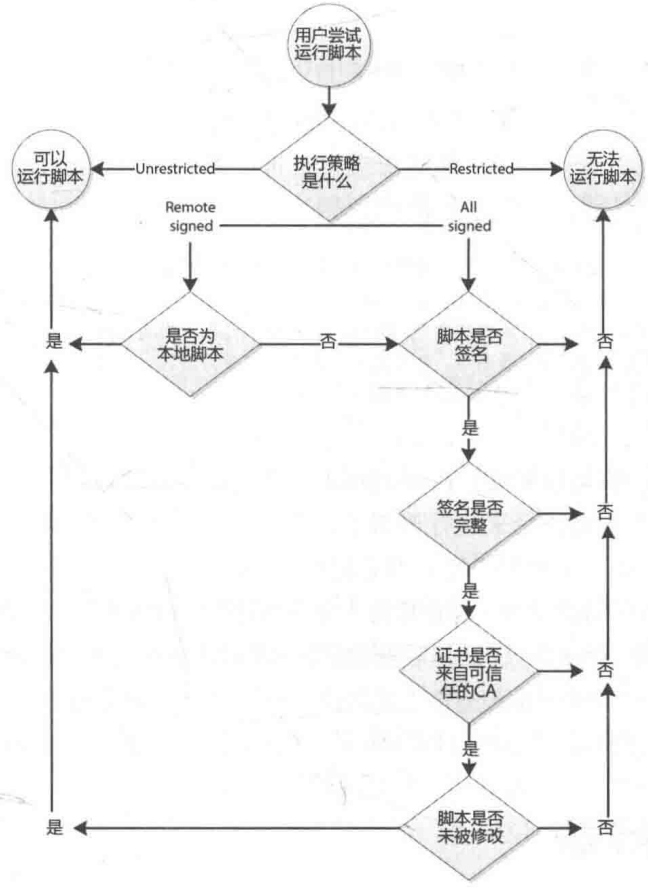

图 17.4 尝试执行脚本时 PowerShell 的处理流程

17.4 其他安全措施

PowerShell 包含另外两种总是一直有效的重要安全设置。一般情况下，它们应该保持默认值。

首先，Windows 不会将 PS1 文件扩展名（PowerShell 会将 PS1 识别为 PowerShell 的脚本）视为可执行文件类型。双击打开 PS1 文件，默认会使用记事本打开进行编辑，而不会被执行。该配置选项会保证即使 PowerShell 的执行策略允许执行该脚本时，用户也不会在不知晓的情况下运行某段脚本。

其次，在 Shell 中不能通过键入脚本名称执行该脚本。Shell 不会在当前目录中搜索脚本，也就是说，如果有一个名为 `test.PS1` 的脚本，切换到该脚本路径下，键入 `test` 或者 `test.PS1` 都不会运行该脚本。

比如下面的例子。

```
PS C:\> test

无法将“test”项识别为 Cmdlet、函数、脚本文件或可运行程序的名称。请检查名称的拼写，如果包括路径，请
确保路径正确，然后重试。
所在位置行:1 字符:5
+ test<<<<
    + CategoryInfo          : ObjectNotFound: (test:String) [], CommandNo
   tFoundException
    + FullyQualifiedErrorId :CommandNotFoundException

Suggestion [3,General]: 未找到命令 test，但它确实存在于当前位置。Windows PowerSh
ell 默认情况下不从当前位置加载命令。如果信任此命令，请改为键入 ".\test"。有关更
多详细信息，请参阅 "get-help about_Command_Precedence"。
PS C:\>
```

如你所见，你可以发现，PowerShell 会检测该脚本，但是会给出警告信息：必须通过绝对路径或者相对路径来运行该脚本。因为 `test.PS1` 脚本位于 C:目录下，所以你可以键入 `C:\test`（绝对路径）或者运行 `.\test`（指向当前路径的相对路径）。

该安全功能的目的是为了防止称为“命令劫持”的攻击类型。在该攻击中，它会将一个脚本文件放入到一个文件夹中，然后将它命名为某些内置的命令名，比如 `Dir`。在 PowerShell 中，如果你在一个命令前面没有加上其路径——比如运行 `Dir` 命令，那么你很明确运行的这个命令的功能；但是如果运行的是 `.\Dir`，那么你就会运行一个名为 Dir.PS1 的脚本。

17.5 其他安全漏洞

正如本章前面所讨论，PowerShell 的安全主要在于防止用户在不知情的情况下运行不受信任的脚本。没有什么安全措施可以阻止用户向 Shell 手动键入命令或者拷贝一个脚本的全部内容，然后粘贴进 Shell 中（尽管以该种方式运行脚本，可能不会有相同的作用）。恶意脚本很难让用户去手动执行，以及指导用户如何去做，这也就是为什么微软并没有将该种场景作为一个潜在的攻击因素。但是请记住，PowerShell 并不会给予用户额外的权限——用户仅能做权限允许的事情。

某些人可能会通过电话联系用户或者发送邮件方式，让用户打开 PowerShell 程序，然后键入一些命令，最后损坏他们的计算机。但是这些人也可以不通过 PowerShell 而是其他方式去攻击某些用户。说服一个用户打开资源管理器，选择 Program Files 文件夹，然后按键盘上的 Delete 键是非常容易的(当然，视你自己的真实情况，也可能比较困难)。在某些方面，比起让用户执行相同功能的 PowerShell 命令，这会更加容易。

我们会指出这一点，是因为人们总是倾向于对命令行以及其看起来具备无限多的功能及功能延伸感到焦虑不安，但是事实上，你和你的用户如果通过其他方式无法完成某些工作，那么在 PowerShell 中你也是无法完成的。

17.6 安全建议

正如前面提到的，微软建议针对需要运行脚本的计算机，将 PowerShell 的执行策略设置为 `RemoteSigned`。当然，你也可以考虑设置为 `AllSigned` 或者 `Unrestricted`。

`AllSigned` 选项相对来说可能比较麻烦，但是如果采用了下面两条建议，那么该选项会变得更加方便。

- 商业 CA 针对一个代码签名证书，每年最多收费 900 美元。如果你没有一个内部的 PKI 可以提供免费的证书，那么你也可以自己制作。运行 `Help About_Signing` 可以查询如何获取以及使用 MakeCert.exe，该工具可以用来制作一个本地计算机信任的证书。如果你仅需在本地计算机运行脚本，那么这种方式是较快免费获取一个证书的方式。根据你所使用的 PowerShell 版本，你还可以使用一个名称为 `New-SelfSignedCertificate` 的 cmdlet，也能完成同样的工作。
- 通过我们上面提及的编辑器去编辑一段脚本，这些编辑器在你每次保存这些脚本时对脚本进行签名。通过这种方式，签名过程更加透明以及自动化，这样对用户来说更加方便。

正如前面所讲，我们都不太建议你去修改.PS1 文件名的关联性。我们曾经看到过某些人修改了 Windows 的一些设置，将.PS1 视为一种可执行文件，也就意味着，你可以通过双击一个脚本来执行它。如果采用这种方式，那么我们就回到使用 VBScript 时的糟糕日子，所以你需要避免该问题。

另外需要指出的是，我们在本书中提供的脚本都没有经过数字签名。这些脚本可能会在不知情的情况下被修改，最后脱离本意。所以在运行这些脚本之前，你应该花费一定的时间去检查它们，理解它们实现的功能，并且确保它们与本书中对应的脚本相吻合(如果可能的话)。我们之所以不对这些脚本进行签名，就是为了让你花费这部分时间来完成这些工作：你应该养成这个习惯，不管该脚本来自于多么受信任的作者，都对那些从网上下载的脚本进行检查。

17.7 动手实验

注意： 对于本次动手实验来说，你需要运行基于 Windows 的 PowerShell v3 或更新版本 PowerShell 的计算机。

在本章的动手实验中，你的任务非常简单——正因为如此简单，所以我们并没有提供一个示例方案。我们需要你通过一些配置选项使得 PowerShell 可以执行脚本。通过 `Set-ExecutionPolicyCmdlet`，我们建议的值是 `RemoteSigned`。当然，你也可以选择 `AllSigned` 这个值，但是对本书后面章节的动手实验环节来说可能就不太适合了。你还可以选择 `Unrestricted` 执行策略。

即便如此，如果在生产环境中使用 PowerShell 工具，也请保证你选择的执行策略的设定值符合贵公司的安全规则与流程。我们不想你为了本书以及其动手实验而陷入某种困境。