第 7 章 壳

攻击者会避免在受害者处使用恶意软件的原始文件，一方面是因为反恶意软件产品可以通过静态签名很容易地实现检出，另一方面则是原始文件可能很大，这就需要更长的时间才能完成下载，减小恶意软件的文件大小是很重要的。

为了不分发原始文件，攻击者通常会在分发前对原始文件进行加密、加壳或者压缩操作。不过，加壳和加密并非是恶意软件独占的。与恶意软件相同，良性软件也使用这些技术对内部代码进行加密和混淆，防止他人破解软件后获取相关知识产权信息。与此同时，良性软件也希望通过压缩减小文件大小，以便用户能够快速下载。

在本章中将要介绍恶意软件使用的加壳程序、加密程序和安装程序，包括这几种不同方式的工作原理。通过加壳与非加壳文件的静态与动态信息比对，查看加壳带来的影响。手动分析样本时，介绍了如何使用 Process Hacker 和 Process Explorer 等各种工具进行脱壳，并查看脱壳后的数据内容。

加密和压缩

加密是一种通过密钥管控数据内容的方法，没有密钥就不能访问数据。加密的直接动因就是要向无权读取和理解数据的人隐藏数据内容，而混淆是加密的副作用，数据经过加密后看起来“乱七八糟”像是某种无意义的垃圾数据。

压缩是一种减少数据大小的方法。当然，压缩算法也会改变数据，带来的副作用也是混淆。如图 7- 1 所示，可以使用记事本创建一个名为 Hello.txt 的示例文本文件，并输入相应内容。利用压缩程序处理该文件并生成文件 Hello.zip 后，再使用记事本打开该压缩文件。如图中所示，文件的原始内容已经不可见，经过压缩后变成了人不可读的数据内容。

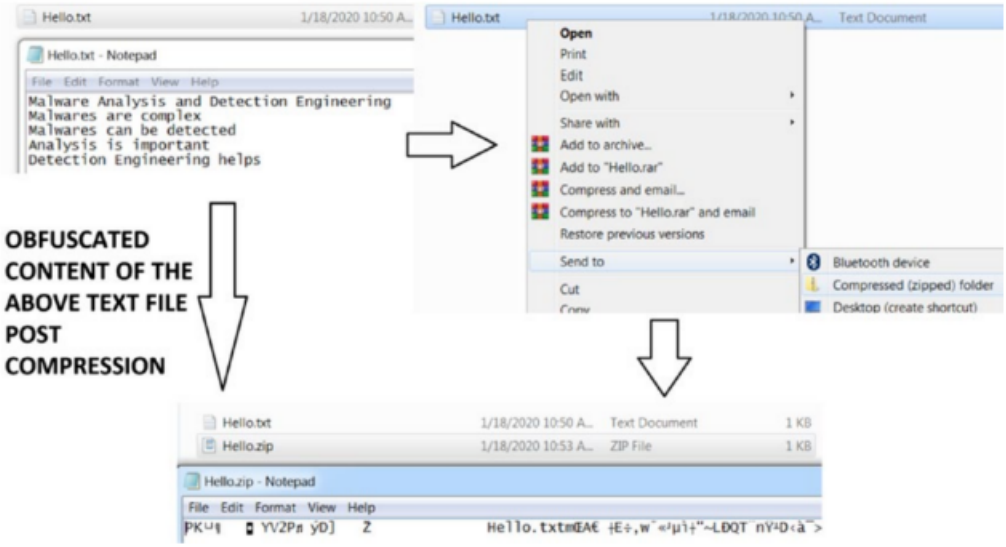

图 7- 1 压缩文本文件带来的副作用—混淆

业界多年来一直致力于开发牢不可破的加密算法来保护数据安全。尽管加密算法与压缩算法从不是为恶意目的而创造，但不幸的是它们一直在被攻击者所利用，许多恶意软件都会使用加密算法来隐藏部分或者全部代码与数据。密码学不是本书讨论的重点，希望读者在阅读本书时能够对密码学相关知识有基础的了解，例如 AES、Xtea、RC4 与 base64 都是恶意软件常用的加密和编码算法。

与良性软件相同，恶意软件也会使用压缩算法来对代码和数据中的某些部分进行压缩，LZMA、LZSS 和 APLib 都是恶意软件经常使用的压缩算法。

实际的恶意软件可以使用加密算法和压缩算法来对代码和数据中的某些部分进行加密与压缩，在进程的运行时进行解密与解压缩操作。加密与压缩现在已经成为分析和检测恶意软件的一种阻碍，研究人员为了克服这种阻碍必须开发对应的解密、解压缩算法，以此获得恶意软件的实际代码。

大多数恶意软件作者都不希望自行实现这些加密算法与压缩算法，他们希望能够将加密/压缩的操作与实际的恶意行为拆分开并行处理。将有关加密/压缩的操作都交给一个被称为“壳”的软件实现，该软件基于攻击者开发的原始恶意软件 Payload 文件，生成一个新的、经过加密和压缩的恶意样本文件。后续将会讨论这个被称为“壳”的软件是如何工作的。

壳

壳程序是用于压缩可执行文件的软件，不仅减少了样本大小，还对可执行文件的内容进行了混淆，隐藏了恶意软件的实际代码与数据。因此，恶意软件使用壳程序会带来两大优势：既可以减少文件大小，还能够混淆真实代码、数据与攻击意图。

壳是如何工作的

壳程序将一个 PE 可执行文件作为输入，最终输出一个新的、加壳的 PE 可执行文件。一个 PE 可执行文件主要分为两部分：文件头与段，段中存放程序所需的代码、数据和资源，这些段都是需要经过压缩以减少文件大小的主要部分。壳程序从要加壳的 PE 文件中获取文件头与段信息，最终生成包含压缩数据的新文件头与新的段。将新的文件头与新的段组合在一起，输出一个新的可执行文件。该文件经过加壳后，占用硬盘空间更少，同时也经过了混淆。整个逻辑过程如图 7- 2 所示:

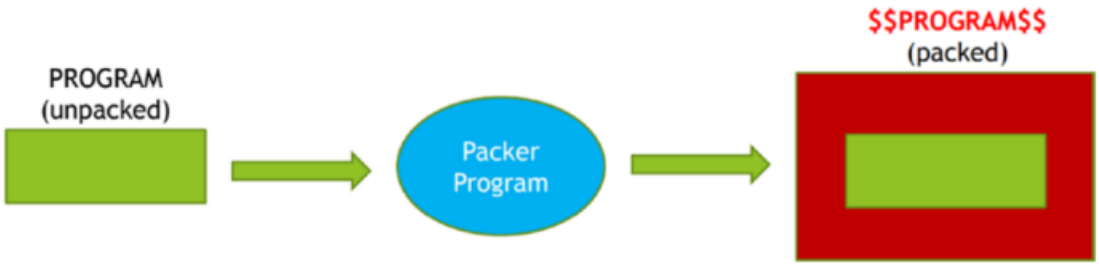

图 7- 2 加壳过程的逻辑可视化

新创建的、加壳的可执行文件中的代码与数据都经过处理，在运行时如何能够保证正确执行？在生成可执行文件时，加壳程序会在其中嵌入加载代码，也被称为 unpacking stub code。加载代码了解处理后的代码与数据在加壳文件中的位置，主要实现在获取处理后的代码与数据后，将原始 Payload 的未处理代码和数据加载进内存中。整体流程如图 7- 3 所示:

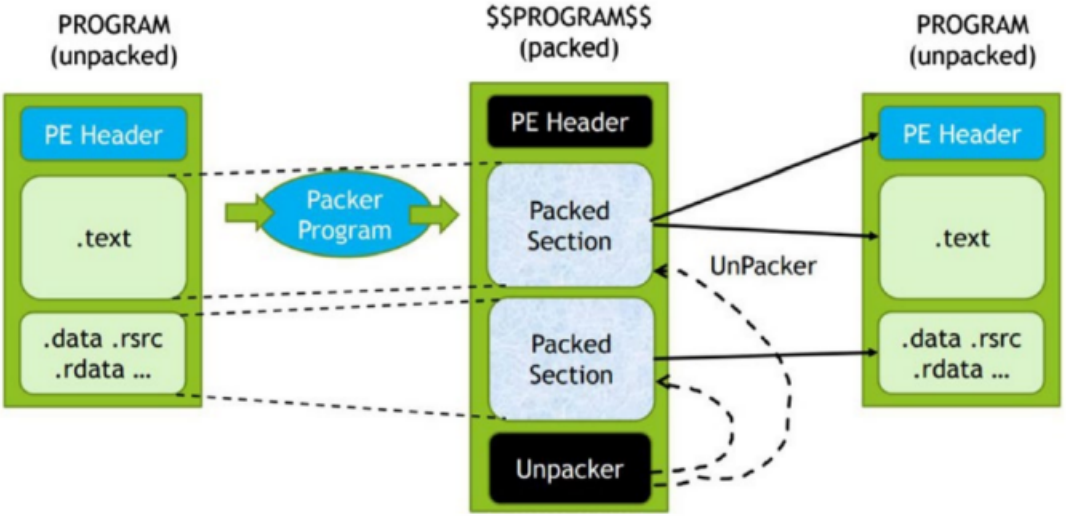

图 7- 3 加载代码的脱壳逻辑可视化

加载代码就像是围绕原始代码创建的、处于压缩状态的“保护壳”。当加载代码运行时，不仅会将压缩的代码和数据还原为虚拟内存中原始状态，还会将指令执行控制权也交还给原始的代码。

如前所述，使用壳程序进行加壳会改变恶意软件的“样子”。换句话说，恶意软件通过加壳进行了混淆。恶意软件还可以使用其他能够混淆代码与数据的工具，来对抗反病毒与其他反恶意软件的检测。加密壳与保护壳就是其中典型代表，下一节将会进行介绍。

加密壳（Cryptors）和保护壳（Protectors）

加密壳是被恶意软件使用，而良性软件不使用的一类壳。加密壳可以像压缩壳一样压缩或者加密代码，但最主要的还是通过更改恶意软件的外部特征使其看起来像合法软件，使恶意软件样本文件具有欺骗性。攻击者可能会更改图标、缩略图与版本信息，使文件看起来像是由正规组织创建的良性软件。例如，在野恶意软件中有很多都带有 Adob e PDF Reader 或 Internet Explorer 等常用应用程序图标。

保护壳可以通过将原始代码替换为执行等效操作的代码来对恶意软件进行混淆，阻碍检测与分析。例如，表达式(A + B)与(A*C/C+B)实现的功能相同，但第二个表达式相比第一个表

达式就更加难以阅读和分析，这也被叫做代码多态性。

在恶意软件领域，压缩壳、加密壳和保护壳之间的界限非常模糊，有时甚至都可以通用。许多恶意软件都会组合使用前述技术，并且会将这些技术与其他各种可以逃避安全产品检测、阻止分析人员分析的技术结合起来使用。如今，大多数压缩壳、加密壳与保护壳中都融入了多种技术，例如反调试、反虚拟机、反分析和其他阻碍分析的技术，整合进了加载代码中。

安装程序（Installer）

安装程序是软件加壳的另一种形式，也常被恶意软件所使用。安装程序除了加壳外，还能够为恶意软件提供安装选项。攻击者可以压缩并生成恶意软件的安装程序（Installer），预置的配置可以将恶意软件安装到特定的目录并执行。

恶意软件经常使用的安装程序是 MSI、Inno Setup 与 autoIT。良性软件使用安装程序的方式与恶意软件最大的区别就是，良性软件的安装程序会弹出用户可操作的界面，而恶意软件通常是静默安装并执行的。

加壳

本节将对一个简单的程序进行加壳，解释加壳的具体过程。UPX 是一个非常古老，但由于开源所以广为使用的加壳工具之一，可以通过 https://github.com/upx/upx/releases 进行下载。使用时可以将其添加到 PATH 环境变量中，就像在第 2 章中介绍的那样。此处使用 upx.exe 运行代码 7- 1 中的命令来对样本文件 Sample-7-1 进行加壳。

代码 7- 1 UPX 加壳命令

```
upx.exe -o Sample-7-1-packed Sample-7-1
```

图 7- 4 中显示了通过命令提示符执行该命令的结果，对应生成了加壳后的样本 Sample-7-1-packed。

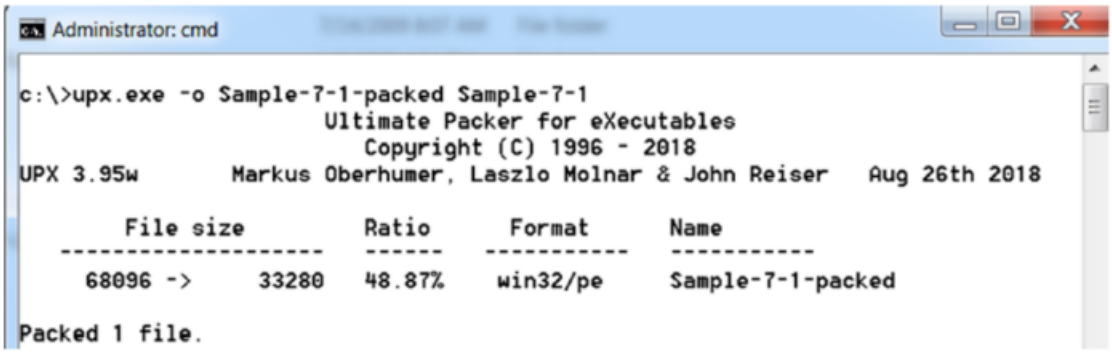

图 7- 4 执行 UPX 加壳命令对样本文件 Sample-7-1 加壳

执行命令后，生成了加壳后的样本 Sample-7-1-packed。加壳前后的样本大小比较如图 7- 5 所示，根据使用的 UPX 版本不同，输出的加壳样本大小可能与图中的大小略有不同。与原始文件相比，加壳后的样本文件更小，明显显示了 UPX 的压缩效果。

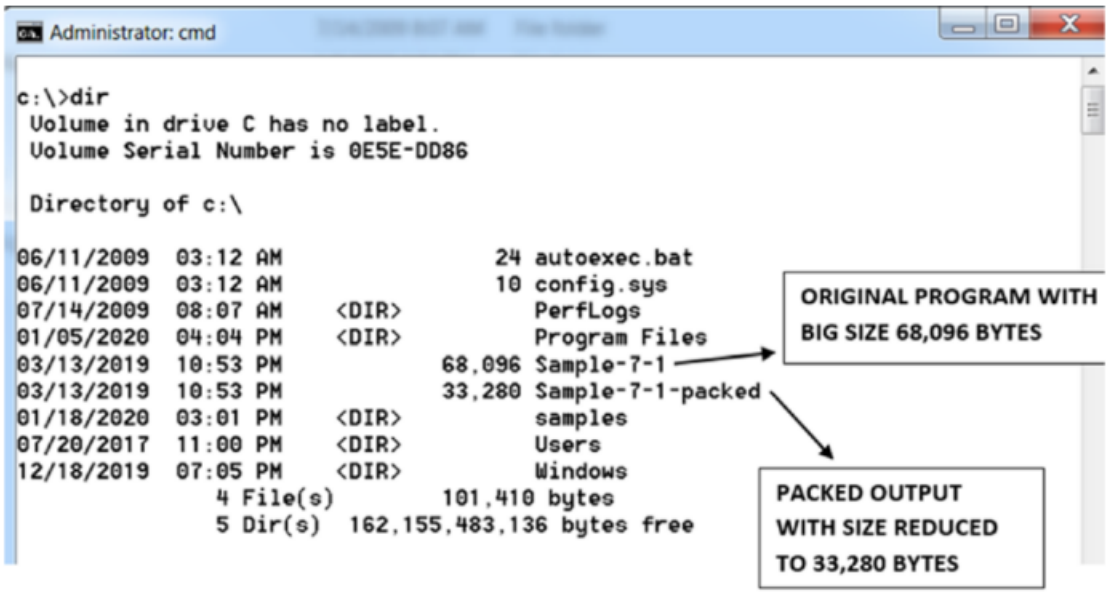

图 7- 5 比较加壳前后文件的大小

比较加壳和未加壳样本文件

加壳和压缩的副作用就是混淆。加壳前的原始文件 Sample-7-1 是一个 C 语言编写的程序，源码中包含 Hi rednet on the heap 的字符串，经过编译后该字符串也会进入最终可执行文件中。在 BinText 中分析样本文件 Sample-7-1 并搜索该字符串，能够确认该字符串确实存在于文件当中，如图 7- 6 所示。

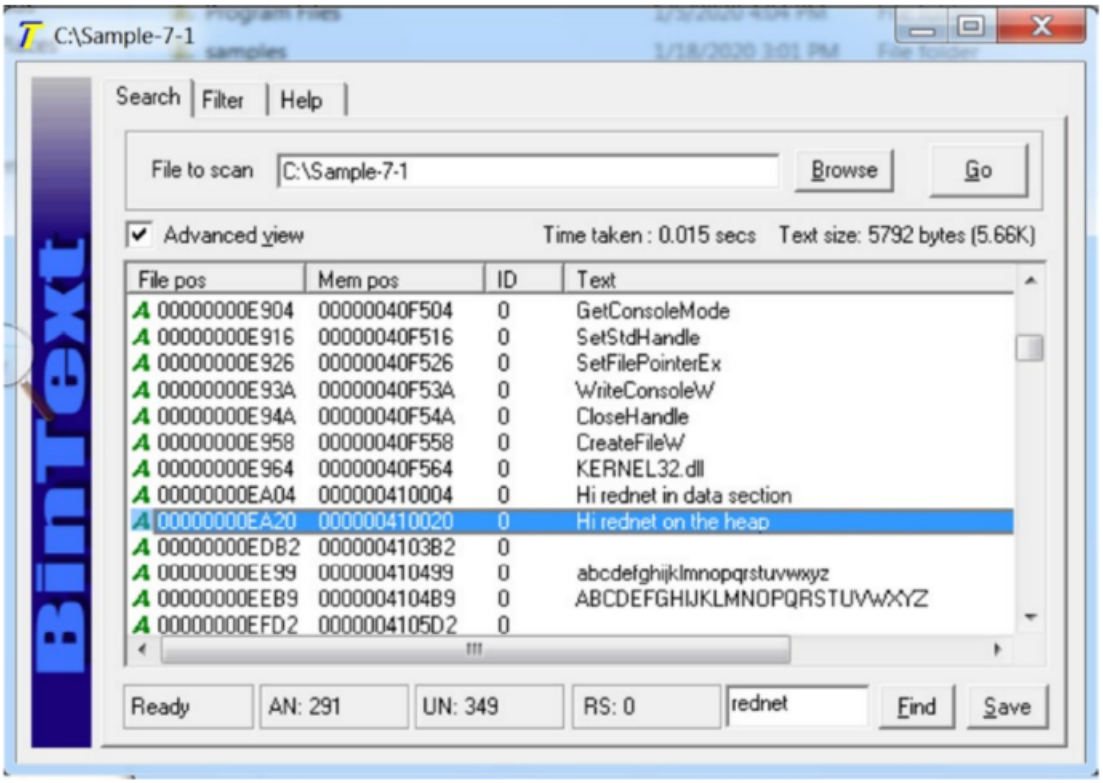

图 7- 6 未加壳文件 Sample-7-1 中存在特定字符串

加壳后的文件 Sample-7-1-packed 如图 7- 7 所示，加壳带来的副作用就是将数据混淆。

在加壳后的样本中已经搜索不到对应的字符串，即加壳造成的混淆使这些字符串不再可见。

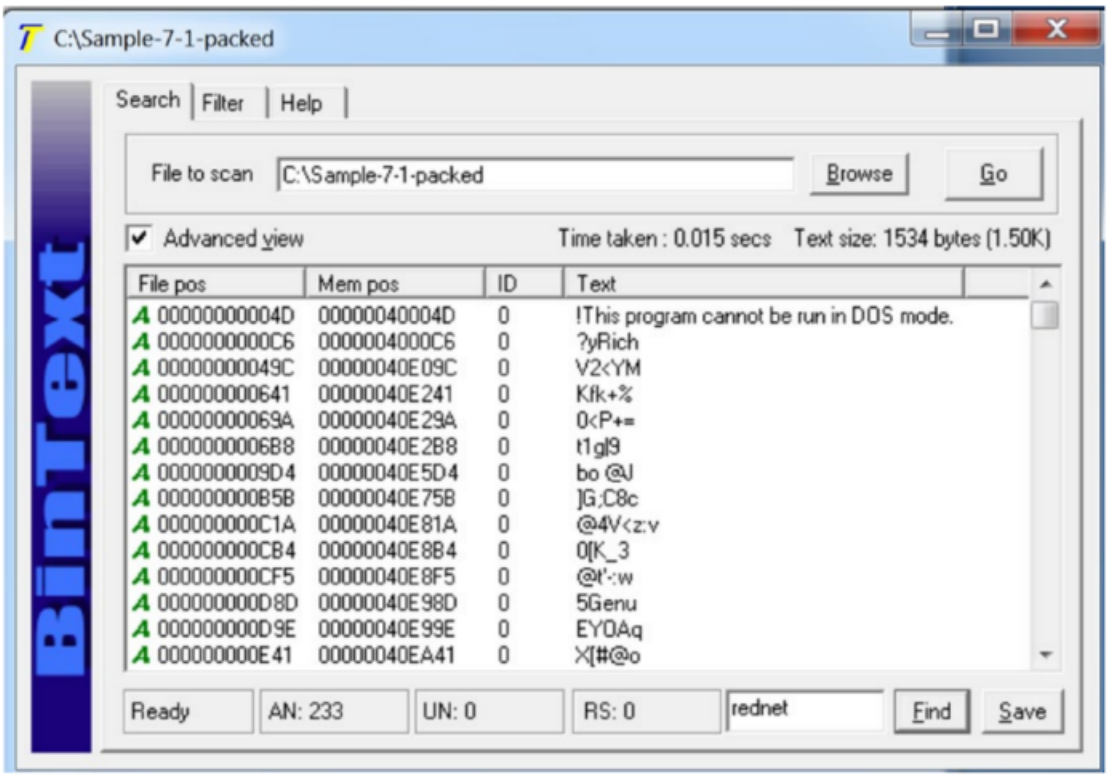

图 7- 7 加壳文件 Sample-7-1-packed 中不存在特定字符串

识别加壳文件

作为分析人员，每天会处理很多样本文件。并不是所有的恶意软件都被加壳了，也有一些恶意软件在没有被攻击者加壳的情况下就分发了。有些时候，也会针对其他分析人员脱壳后的恶意软件进行分析。分析时，最先需要确定的就是样本文件是否加壳。一些判断方法可以在静态分析时使用，不执行样本即可确定样本文件是否被加壳。也有一些判断方法需要动态执行恶意软件，分析其行为与属性判断样本文件是否被加壳。具体技术在后续将逐一进行介绍。

熵

熵是对数据随机性的度量。熵值是检测加密与压缩的常用特征，一方面因为在经过压缩与加密后，数据更像是随机的且具有更高的熵值。另一方面加壳前的原始文件性更差，熵值也相对较低。

分析人员经常计算文件的熵值来判断样本文件是否被加壳。如图 7- 8 所示，使用 PEiD 加载样本文件 Sample-7-1-packed 后可以得到该样本的熵值为 7.9。熵值越接近 8，越有可能是加壳的样本。读者可以自行使用 PEiD 加载原始未加壳样本文件，其熵值为 5.8。相比之下，加壳后的样本熵值较高为 7.9。

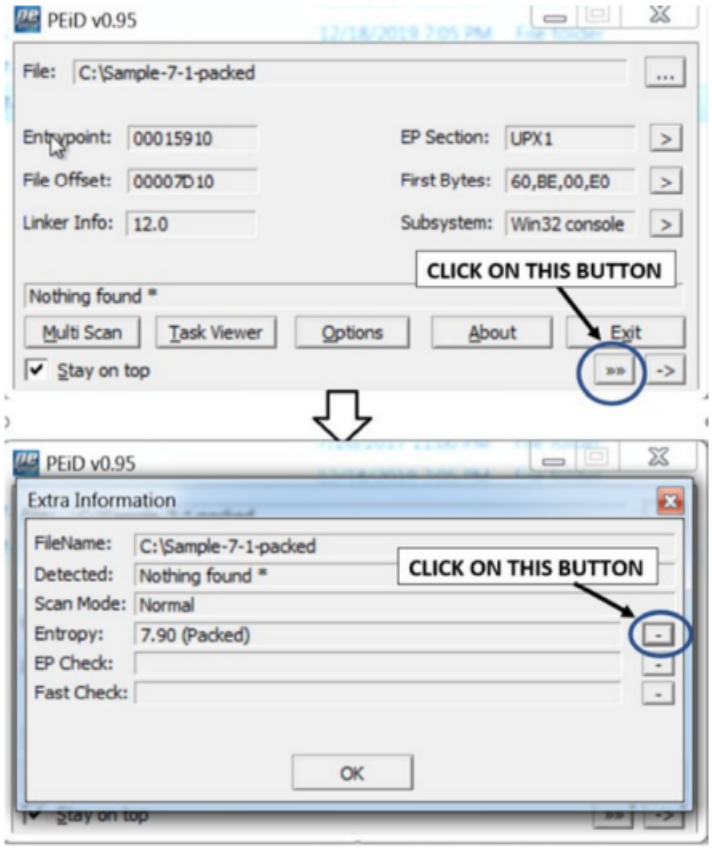

图 7- 8 Sample-7-1-packed 的熵值达到 7.9，可能为加壳文件

字符串

编写程序时需要在源码中使用许多字符串。特别的，恶意软件中的字符串有可能包含 C&C 服务器的域名和 IP 地址、逃避与对抗分析工具和虚拟机用到的特征名称、用于 C&C 通信的 URL 格式、与网络协议有关的字符串等。最终完成编译后，生成的可执行文件中也会包含这些字符串。如前所述，加壳会使这些字符串被混淆，接下来将会介绍如何通过字符串来识别加壳的样本。

静态分析文件中的字符串

前文介绍了加壳对可执行文件的影响，在图 7- 6 与图 7- 7 中，使用 BinText 加载分析样本文件 Sample-7-1 和 Sample-7-1-packed。根据 BinText 中显示的字符串，原始文件

中包含人类可读的字符串，例如 Hi rednet。但在加壳后的文件中并没有发现这些字符串，而是被一些垃圾字符串所取代。

分析恶意样本时，分析人员可以首先将文件加载到 BinText 或者其他任何能够查看字符串的同类工具。如图 7- 7 所示，样本中的大多数字符串都是经过混淆的垃圾字符串，其中没有任何有意义的单词和句子，这是样本被加壳的明显标志。

值得注意的是，一些字符串对于加壳与非加壳样本中都是存在的，这些无法帮助判断是否加壳的字符串应该被忽略。如图 7- 9 所示，还有 API 名称、导入 DLL、编译器代码字符串、语言环境和语言等公共字符串。分析处理更多的加壳恶意样本，将加壳文件的字符串与原始字符串进行比较，就能够知道哪些公共字符串应该被忽略。

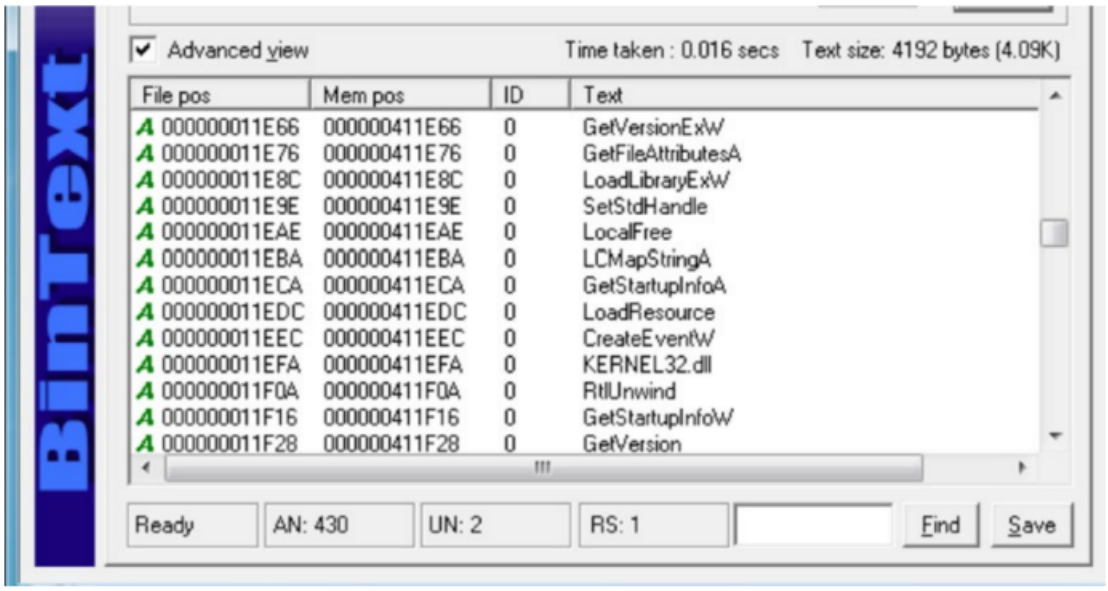

图 7- 9 在未加壳与加壳样本中都存在的公共字符串不能用于区分是否加壳，需要忽略

在 BinText 中加载分析样本文件 Sample-7-2，查看其字符串能够发现许多人类可读的非垃圾字符串。如图 7- 10 所示，能够找到 NOTICE、PRIVMSG、DCC SEND、PING、JOIN、#helloThere 等字符串，这些字符串都与 IRC 协议有关。进一步向下滚动查看其他字符串，包含 USER、NICK、C:\ marijuana.txt 等字符串。当然，字符串中还包含许多垃圾字符串，这也是正常的。即使没有加壳，许多正常的二进制指令也会被显示为垃圾字符串。在加壳的

样本中，应该几乎找不到如前所述那种人类可读的字符串，这也表明样本文件 Sample-7-2 没有被加壳。

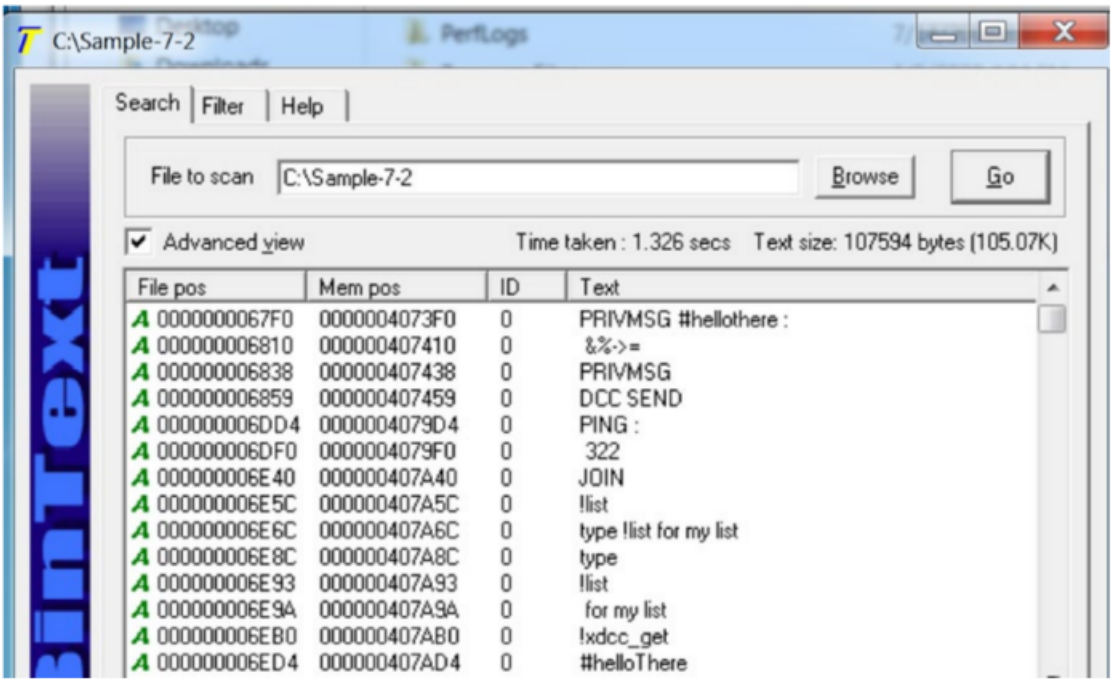

图 7- 10 文件 Sample-7-2 中包含可读字符串，该文件可能未加壳

接着分析示例样本库中的样本文件 Sample-7-3。如图 7- 11 所示，在 BinText 中加载分析该样本，可见字符串多为垃圾字符串，这表明该样本应该是被加壳的。

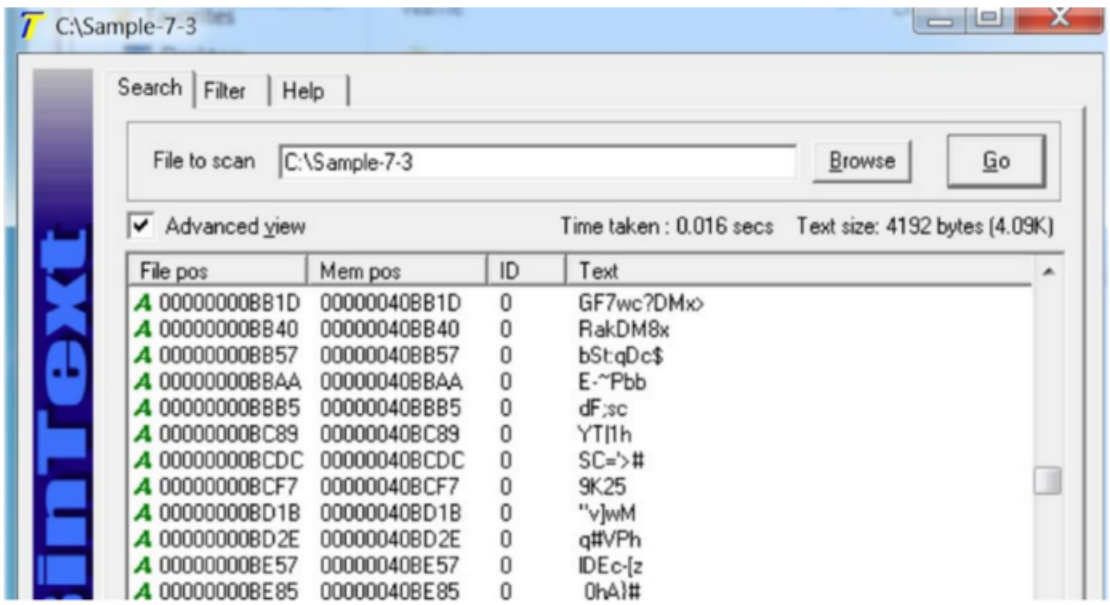

图 7- 11 文件 Sample-7-3 中包含垃圾字符串，该文件可能被加壳

动态分析内存中的字符串

与确认样本是否加壳的静态方法类似，依赖样本执行时在内存中字符串的动态方法也能够识别是否加壳。

如前所述，在加壳样本运行时，加载代码活在某个时间点被执行，将原始可执行文件的代码与数据释放到内存中。虚拟内存中的未加壳数据中，会含有属于原始文件 Payload 的所有字符串。如果样本运行时的虚拟内存中的字符串更具有可读性，而非利用 BinText 进行静态分析时发现样本中包含的大量垃圾字符串，则说明原始文件被加壳了。

恶意软件动态分配用于释放代码的内存区域与内存页面，就是在内存中查找字符串的目标位置。在 Process Hacker 中，此类页面显示为具有 Commit 属性的 Private 页面，并且不属于任何模块。另一个区域则是由示例样本文件的主模块 Image 所在的区域，相关知识在第 4 章中进行了介绍。

通过结合使用 Process Explorer 和 Process Hacker 与 BinText，将内存中的字符串与文件中的字符串进行比较。第 4 章中已经介绍了使用 Process Hacker 查看内存中字符串的方法。当然，也可以使用 Process Explorer 查看文件在内存中的字符串。例如将文件重命名为 Sample-7-1-packed.exe，并且双击执行创建进程。再在 Process Explorer 的窗口中双击该进程，在弹出的 Properties（属性）窗口中，选择 String 选项卡。该选项卡底部有两个按钮：Image 与 Memory，选择前者会显示文件中的静态字符串，选择后者会显示进程主模块的内存中的字符串，如图 7- 12 所示。

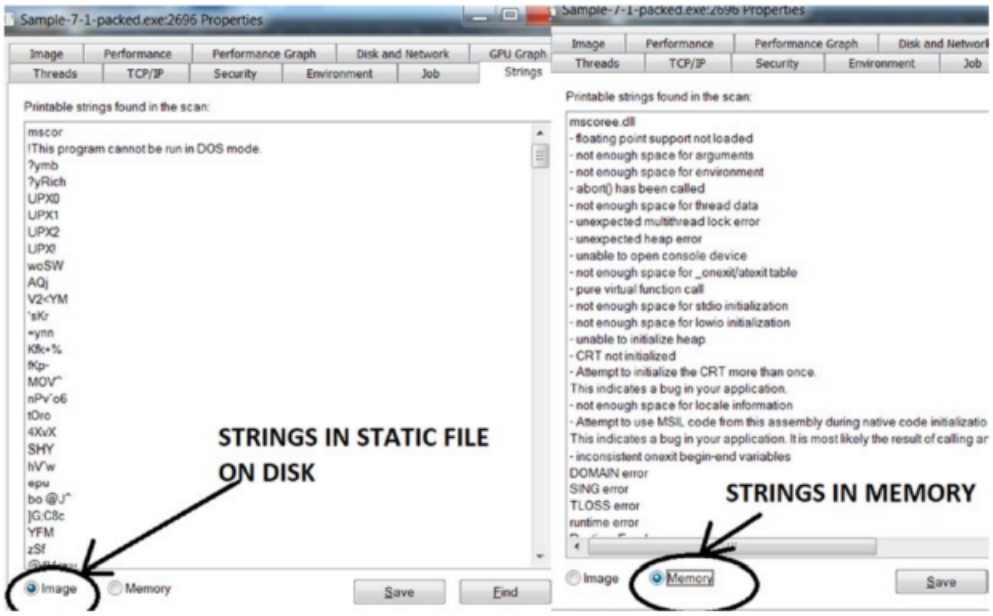

图 7- 12 利用 Process Explorer 查看文件中的字符串与进程中的字符串

如图 7- 12 所示，文件中的静态字符串与运行的进程中的字符串相比差异特别大。这可能表明该文件已经被加壳，并且会在运行时将实际的原始代码还原到内存中。利用“查找“功能可以对字符串进行搜索，例如搜索 rednet 字符串就可以发现，该字符串并不在静态文件中却存在于内存中，如图 7- 13 所示。

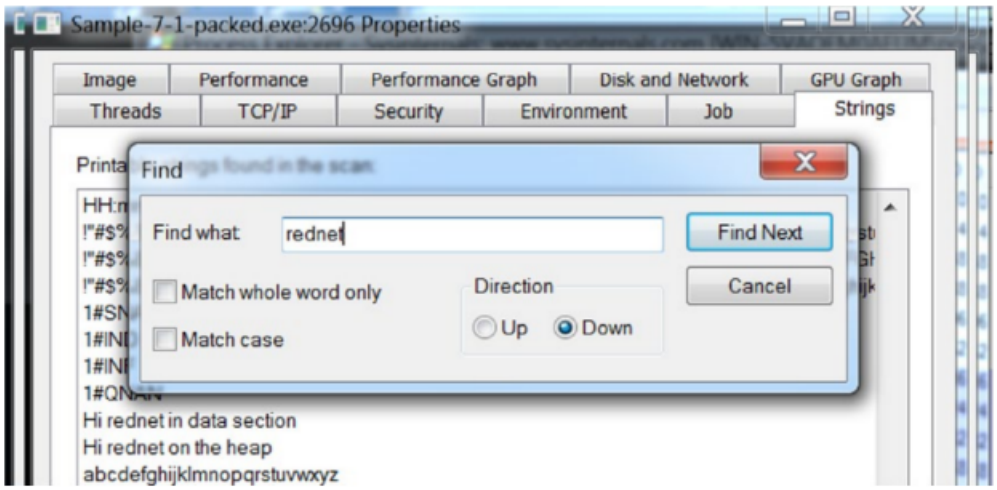

图 7- 13 通过 Process Explorer 确认特定字符串只存在内存中并不存在于静态文件中

值得注意的是，Process Explorer 只显示进程主模块中的字符串。这对分析人员来说可能是不够的，因为恶意软件可以将 Payload 释放到进程主模块之外的私有内存中。

与使用 Process Explorer 类似，也可以使用 Process Hacker 查看进程字符串。不过，Process Hacker 并没有像 Process Explorer 中的 Image 选项那样显示静态文件中的字符串的功能。

必须首先使用 BinText 静态查看文件中的字符串，再使用 Process Hacker 查看正在运行的进程内存中的字符串，最后将其与 BinText 查看的字符串进行比较。

Process Hacker 的一个好处是，它可以从整个进程的内存中查看字符串，而非只局限于进程的主主模块。但这样做带来的副作用就是会显示许多来自其他 DLL 文件的无关字符串，这些字符串也被加载到进程的内存中。当只想要查看内存中的字符串时，建议首先使用 Process Explorer，而非 Process Hacker。

Process Hacker 带来的另一个好处就是能够选择显示字符串的页类型。如图 7- 14 所示，Process Hacker 在进程的 Properties 窗口中为样本文件 Sample-7-1-packed.exe 启用了 Memory 选项卡。单击 Strings 选项即可根据 Private、Image 与 Mapped 页类型来选择要显示的字符串，这个必要的功能为分析人员带来了便利。

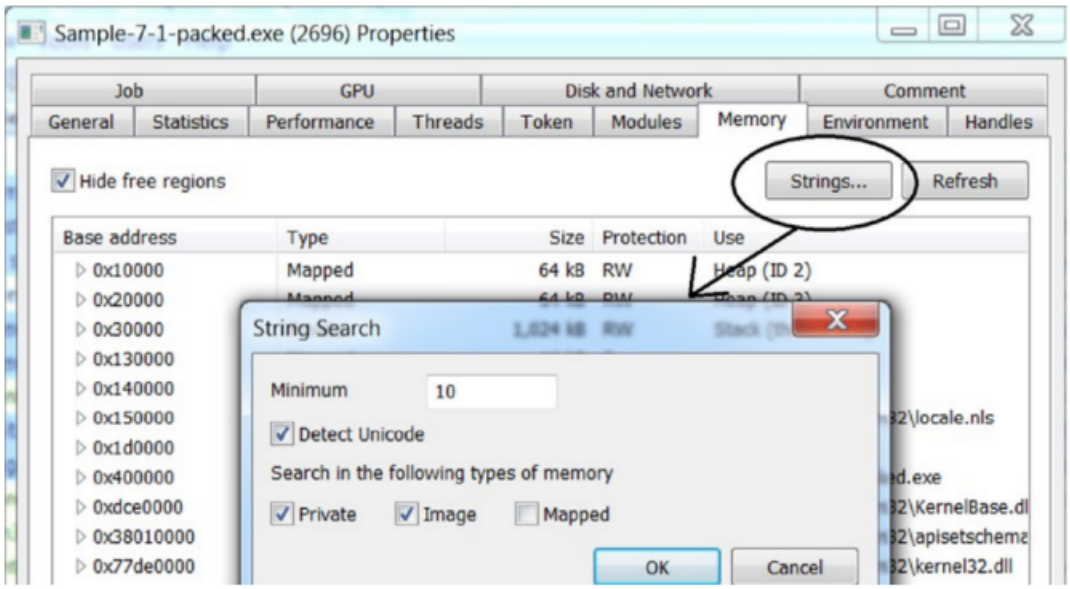

图 7- 14 在 Process Hacker 中选择要显示的字符串类型

恶意软件案例研究

如所示，使用 BinText 对样本文件 Sample-7-3 进行静态分析，获取样本文件中的字符串。根据静态字符串情况，可以确定该样本已经被加壳。

为了确认样本确实是加壳的，需要查看恶意软件在内存中脱壳的结果，运行样本文件并将内存中的字符串与在 BinText 中查看的字符串进行比较。重命名样本文件 Sample-7-3 为 Sample-7-3.exe 并双击执行，恶意软件为了隐藏自身，会使用名为 process hollowing 的技术注入另一个名为 svchost.exe 的进程中运行，在第 10 章中将对此技术进行详细介绍。此时查看 svchost.exe 进程的字符串，可以看到许多人类可读的字符串。与在已加壳文件中静态看到的垃圾字符串相比，这些字符串的可读性已经提高很多，这表明样本文件确实被加壳了。使用 Process Hacker 查看字符串，并且查看 Private and Image 页面，如图 7- 15 所示。

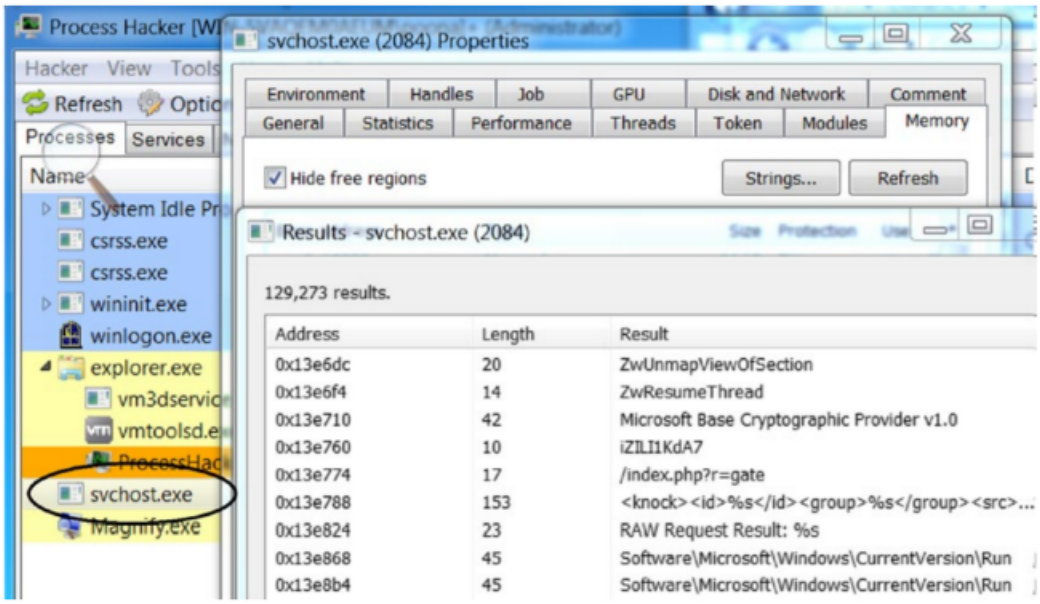

图 7- 15 使用 Process Hacker 查看文件 Sample-7-3 进程内存中的字符串

在后续的章节中，将会介绍如何利用内存中的字符串判定样本为恶意软件，并且确定其恶意软件类型与所属家族。

识别壳

前文介绍了如何判断样本文件是否加壳，在确定了样本加壳后就需要识别加壳的具体类型。实际上并不是总能够识别加壳的具体类型，一方面壳的类型非常多，另一方面是由于恶意软

件也会自行定制开发特殊的壳来处理恶意软件。对样本进行逆向工程时，确定壳的具体类型是非常有必要的。确定了壳的具体类型后，就可以在互联网上获取有关该类壳的更多信息，这对了解壳的工作原理以及掌握如何进行脱壳很有帮助，能够为编写自动脱壳工具提供支持。后续会介绍识别加壳的具体类型的一些具体特征，例如通过入口点代码、段名称等。

PEiD 工具

PEiD 是一个常用与识别壳类型的分析工具。如图 7- 16 所示，PEiD 检测到样本文件 Sample-7-1-packed 使用 UPX 进行加壳。

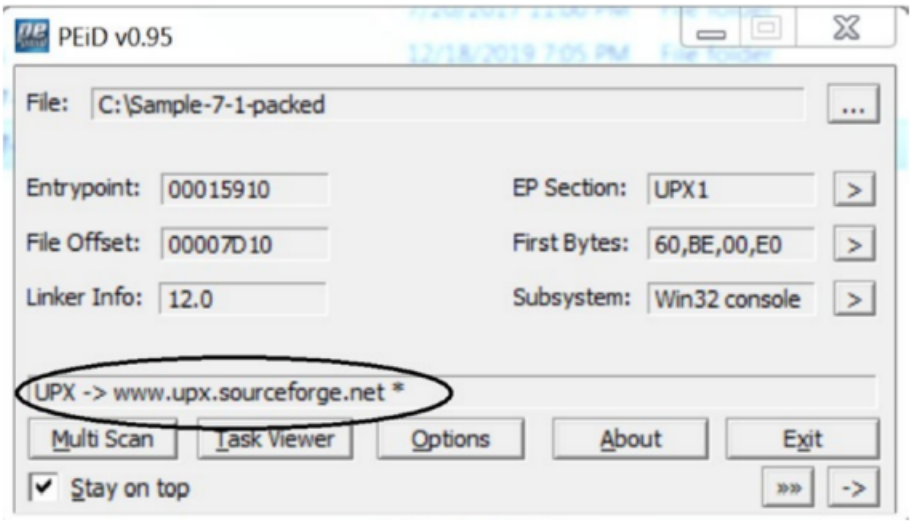

图 7- 16 使用 PEiD 识别文件 Sample-7-1-packed 使用 UPX 加壳

PEiD 就是依赖匹配入口点附近字节的签名来检测壳的。PEiD 用于识别壳的签名来自一个名为 userdb.txt 的签名数据库文件，该文件通常与 PEiD 可执行文件位于同一目录下。如果在使用 PEiD 分析样本文件 Sample-7-1-packed 时，没有像图 7- 16 那样检测到加壳的话，很有可能 PEiD 此时没有加载任何签名。读者可以自行从互联网上下载签名数据库文件，例如在 https://handlers.sans.org/jclausing/userdb.txt 中就提供了一个这样的签名数据库文件。签名数据库中不仅包含针对 UPX 壳的签名，也包含其他各种壳的签名。

在发现了新类型的壳时，使用者可以编辑签名数据库文件 userdb.txt 来新增或者删除用于

识别的签名。

入口点代码

大多数壳在入口点附近都具有固定模式的代码，所以可以通过加壳的 PE 文件的入口点附近代码来对壳进行识别。值得注意的是，入口点代码可能会因壳的版本而产生差异，并不是所有版本都是一成不变的。

如图 7- 17所示,样本文件Sample-7-1-packed的入口点附近代码为 60 BE 00 E0 40 00 8D BE 00 30 FF FF，这也就是在用户签名数据库中 UPX 壳的签名。可以通过在 PEiD 依赖的签名数据库文件 userdb.txt 中检索该签名进行交叉验证。如代码 7- 2 所示，userdb.txt 中识别壳的签名为匹配入口点附近字节 60 BE ?? ?0 4? 00 8D BE ?? ?? F? FF，其中的??是可以表示任意字符的通配符。

代码 7- 2 PEiD 使用的数据签名文件 userdb.txt 中识别 UPX 壳的签名

```
[UPX -> www.sourceforge.net]
signature = 60 BE ?? ?0 4? 00 8D BE ?? ?? F? FF
ep_only = false
```

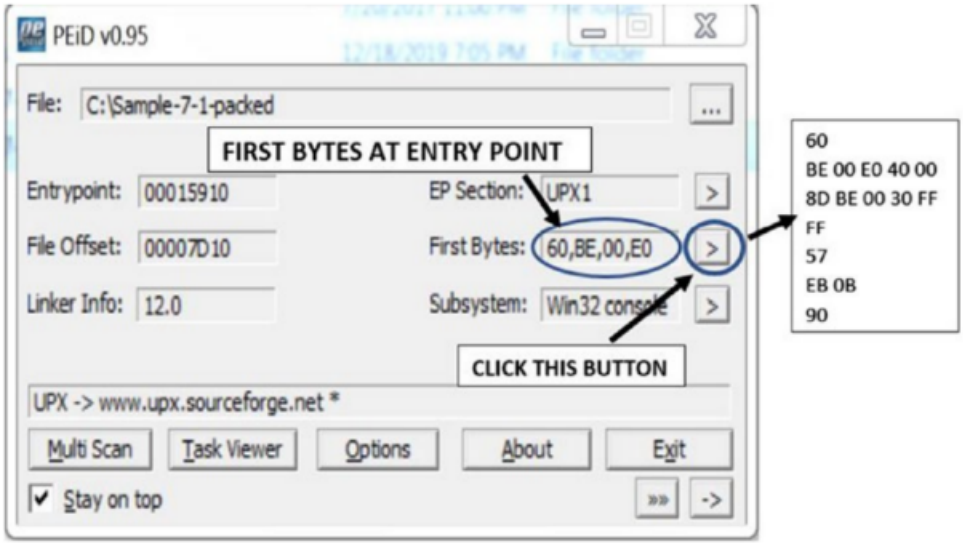

图 7- 17 文件 Sample-7-1-packed 的入口点附近代码

段名称

通常来说，加壳后的文件与原始文件区别很大，例如二者具有不同的段名称。许多壳都会使用特殊的段名称，能够通过某个固定的、可识别的模式来匹配发现所有同类加壳文件。例如，使用 CFF Explorer 加载分析样本文件 Sample-7-1-packed，能够看到以字母 UPX 开头的段名称，这就是 UPX 壳的典型特征，如图 7- 18 所示。

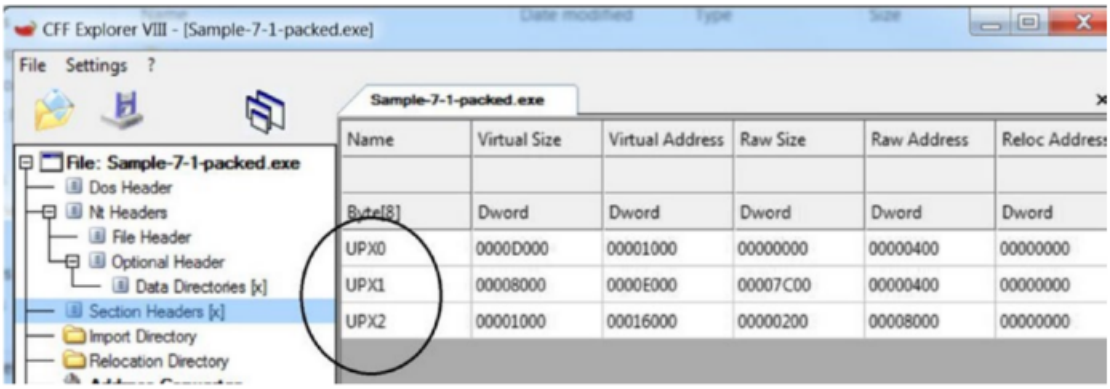

图 7- 18 UPX 加壳的文件段名称都是以 UPX 开头

表 7- 1 中，列出了常见壳与其加壳后的文件中包含的段名称。

表 7- 1 常见壳与加壳后文件包含的段名称

壳	段名称
UPX	.UPX0、.UPX1
Aspack	.adata、.aspack
Mpress	MPRESS1、.MPRESS2
NsPack	.nsp0、.nsp1、.nsp2
PECompact2	pec、pec、pec2
RLPack	.RLPack
Y0da Protector	.yP、.y0da
Upack	.Upack

VMProtect	.vmp0、.vmp1、.vmp2
Pepack	PEPACK!!
FSG	FSG!

定制壳

很多恶意软件开发者都会使用自定义加壳程序来对恶意样本进行加壳。在分析实际的恶意软件时，很多时候可能并不会遇到通用的壳，甚至在互联网也找不到任何有关如何脱壳的信息。样本文件的入口点代码与任何已知的加壳程序都不匹配，段名称也是这样，使用 PEiD 或者其他任何分析工具都不能识别出所使用的壳。

当然，不论样本文件利用什么壳来加壳，包括自定义壳在内，都几乎有通用的解决方案来对样本文件进行脱壳。在第 17 章中对此将进行介绍，可以使用这些技巧来对恶意软件进行脱壳处理。

关于壳的误解

对壳最常见的误解就是：如果发现文件被加壳，则该文件为恶意软件。加壳是一种良性软件与恶意软件都使用的技术，用以达成压缩和混淆的目的。如果认为加壳的文件就是恶意软件，分析人员很容易将良性加壳软件判断为恶意软件。另外，使用静态 Yara 规则实现检测的解决方案与签名数据库也十分常见，这些签名通过匹配加壳文件的特殊字节进行识别。这样做会导致误报与漏报，对检测会产生不利影响。

总结

本章中主要介绍了恶意软件为实现压缩与混淆而使用的技术——壳。不仅介绍了加壳的基本原理，还解释了如何使用 PEiD、CFF Explorer 以及自定义签名等各种工具来识别加壳的

样本文件。通过分析恶意样本，展示了如何使用 BinText、Process Hacker 和 Process Explorer 等工具，在静态分析或者动态分析时识别样本文件是否加壳。