浅析反病毒引擎

江海客（seak@antiy.net，http://www.antiy.net）

2002-12

一、 引言

峰会之后的一些讨论，让我考虑作这样一个“补充发言”，这依然是从工程化角度出发的东西，希望从一个 AVER 的角度，能和 hacker 们进行一些交流。

何谓反病毒引擎？反病毒引擎就是指依赖于一个特定的数据描述集合来完成计算机病毒检测和清除的一组程序模块。当然，这个特定的数据描述集合，就是病毒库。两者互相依赖，相辅相成。

二、 反病毒引擎的产生

在 80 年代中后期，针对 X86 系统的病毒产生并发展，由于当时病毒总量有限，而且依赖于一定地缘性流行。因此早期的计算机病毒的对抗和处理，往往依赖于那些一对一的免疫程序或者专杀工具。

这种方式一般需要知道某种病毒即将流行，之后通过免疫程序设置感染标志欺骗病毒，或者从微机使用者，发现或者猜测有某种病毒开始，之后使用特定的专杀工具清除。效率比较差。

之后，一些开发专杀工具的程序员将工具作了简单的集成，添加了一个类似如下的字符界面：

1. Scan and Kill Brain
2. Scan and Kill pingpong
3. Scan and Kill stone
4. Scan and Kill Jerusonlem
5. Exit

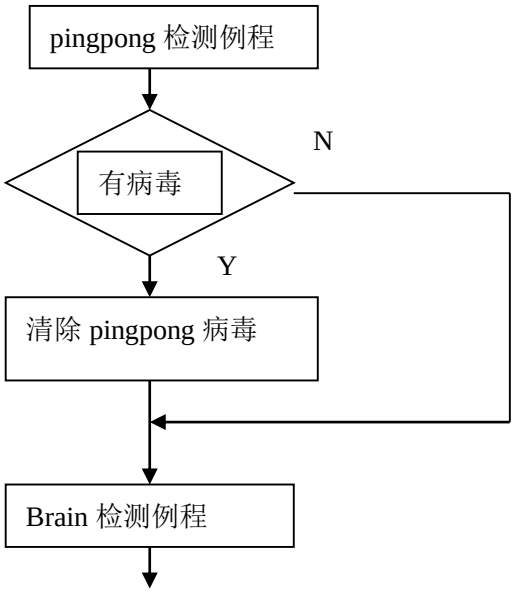
Press 1-5 to Continue….

这种工具只是将若干个专杀工具置于一个统一界面调度下而已，并没有实质性的进展。

随着病毒数量的增多，上述方式显然不再适合。一些初步具有商业软件风格的反病毒软件开始产生，这些软件可以自动完成对软件可查杀的所有病毒的检测，但早期的一些反病毒软件还只是简单的把若干个选择执行的模块串联起来而已。类似如下的流程。

比如一个完成对 Boot 检测的流程：

©安天实验室 版权所有



这种串接结构还不能称为反病毒引擎，它实际上做了大量的重复工作，耗费了资源，而且造成程序的结构混乱，不方便调试。

病毒引擎的最终出现，来自对病毒共性的提取，从而形成一组或者几组比较规范的用于病毒描述数据结构。

我们看看试图把上面两个病毒的检测，描述为一个规则，通过分析病毒体，我们分别提取了 15 字节特征串作为检测病毒的依据。为了让当今的网络安全研究者容易理解，让我们借鉴一下 snort 规则的风格。

alert boot (msg"virus-Brain"; content:" |A1 13 04 2D 07 00 A3 13 04 B1 06 D3 E0 8E C0|";)

alert boot (msg"virus-pingpong"; content:" |A1 13 04 2D 02 00 A3 13 04 B1 06 D3 E0 2D C0|";)

不要被两个串的部分近似所迷惑，那不是我们需要的。我们关心的是如何做工程化的处理。试图在特征码的近似性上找到窍门只会让我们绕得更远。对于这种早期的，非变形的病毒来说，特征码匹配是绰绰有余的。对于这一类病毒，我们已经可以做归一化处理，只要我们设计一个内容匹配算法。这个算法是的时间复杂度应该是和记录条数呈非线性关系的，那就能保证我们的高速检测了。

这样，我们就初步实现了把病毒引擎和数据库分离开来，我们可以把那些规则的集合称为反病毒库了。

但这样依然不够，查毒虽然有规律可以寻找，那么杀毒呢？如果杀毒也能找到一定的规律，从而变成一种描述，那么对很多病毒的处理就变成了一个通用的处理引擎+处理参数。这样软件的规范性和可维护性就会进一步增强。

我们知道，最理想的杀毒过程，是病毒感染的逆过程。除了少数病毒的感染是覆盖式而不具备可逆性外，多数病毒的感染过程都是可逆的。因此我们可以简单的总结总结出这个逆过程中，具有共性的一些操作：

- 进程中止
- 扇区读写（物理、逻辑）

©安天实验室 版权所有

- 文件读写
- 内存缓冲区中一段 2 进制数据的插入
- 内存缓冲区中一段 2 进制数据的清除
- 内存缓冲区中一段 2 进制数据替换

…

但也有观点认为，好的处理引擎，应该和文件读写方式无关，应该完全基于数据的操作，这样才能实现直接的平台移植，比如工作于 Linux 服务器的反病毒程序，可以公用一套 2 进制库，而不需要任何改写。因此 I/O 应该独立的一层，因此只要有一些数据处理的函数。我们可以进一步提炼总结，从而形成一组标准的模块，通过一组数据定义 作为操作指令，传递给这组标准模块，完成部分病毒的清除，当然对于那些复杂的病毒，还是需要独立的小模块处理。这样整体的结构化已经好了很多，稳定性、质量和可调试性都得到加强。

假定以下是 3 种不同的感染 MBR 的病毒的响应参数。

Respond:”M 0,0,7”

Respond:”O”

Respond:”R 0057”

第一组参数， Move 指令，表示对该种病毒处理，需要将 0 道 0 头的第七扇区，搬回 MBR 的位置。

第二组参数，Overwrite 指令，表示 MBR 可以直接用一个无毒的标准 MBR 代码区来重建，从而实现杀毒。

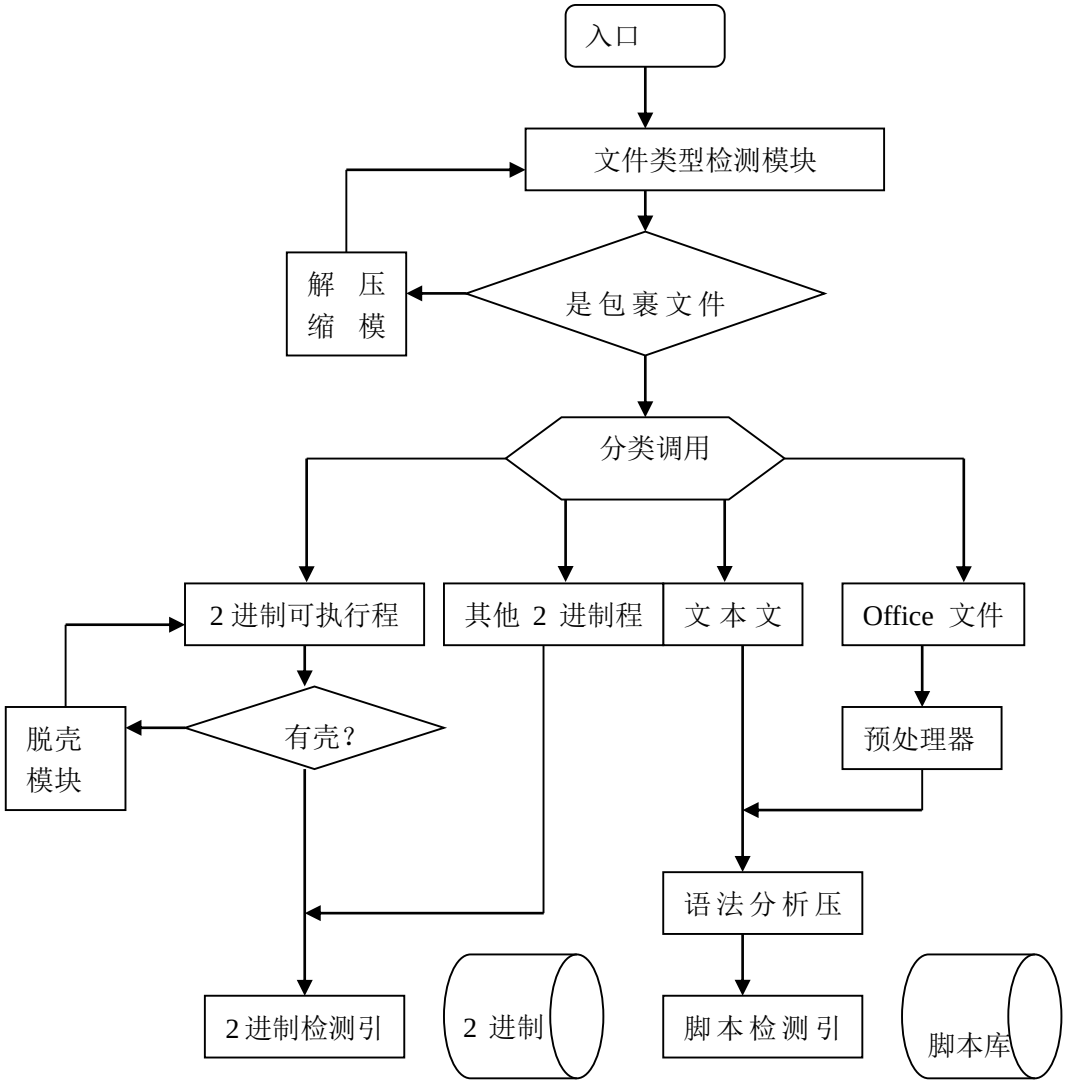
第三组参数，Run 指令，表示该病毒比较复杂，需要执行一个独立的小杀毒模块，执行 0057 号模块。

当然，以上这些只是借助一些简单的病毒所举的例子。用来描述反病毒引擎的产生过程，与现有的商用引擎相比，没有任何可比性。

三、 解读商用引擎

当今的商用反病毒引擎，基本上是针对文件对象、内存对象、引导扇区对象进行病毒扫描的。让我们以一个简略化的文件检测流图，来看一看引擎的工作过程。

©安天实验室 版权所有



对于文本类型文件，主要是进行脚本病毒检测，目前有 vbs、js、php、perl 等多种类型的脚本病毒，这要交给语法分析压缩器去处理，语法分析压缩器的结果再交给检测引擎做匹配处理。

部分反病毒软件的宏病毒检测是交给脚本处理引擎完成的，通过 office 预处理器提取出宏的 basic 源码，之后同样交给语法分析压缩器。

但仔细分析，这个示例的流程就会发现很多问题：

1. 自解压文件如何处理？其自身作为可执行程序由感染病毒的可能，而其包裹数据中，仍然可能有压缩的病毒。

2. 2 进制程序起点，但最终需要检测数据未必是 2 进制程序，比如一个批处理程序经过了如下的变化，bat2com、com2exe、pklite。最终还原出的是一个批处理文件，应该交给脚本引擎去检测。而文本起点，但最终的监测数据也未必一定是文本，比如有的病毒作者通过 bat 文件去加载 2 进制可执行程序。他将可执行程序，变成 16 进制字符，通过 echo 将其输出到文本文件，再通过管道将其传递给 DEBUG，通过 debug 去加载执行。显然，如果能将这样的数据还原为 2 进制数据，检测就可靠的多。因此，两个流程实质出现了交叉，这说明，分类处理点设置的位置是有问题的。

©安天实验室 版权所有

3. 2 进制病毒的检测路径上依然是利用匹配的方式，那么对于变形、加密的病毒如何处理的问题并没有解决。其中还需要一个虚拟机的模块置于 2 进制检测引擎之前。同样的，在脚本检测的路径上，同样可能需要一个虚拟的脚本调试器。
4. 异常处理问题，如解压缩失败如何处理、脱壳失败如何处理。
5. 反病毒处理的流程，应该是可干预、可定义的，是否扫描压缩包内部，是否扫描所有文件，都应该允许用户定义的。

如果一个反病毒引擎真的这样去实现，显然问题还会很多。因此，这只能作为一个形象化的说明，真正的去开发一个反病毒引擎，还需要更为全面细致地分析工作，和对病毒深入地了解、以及良好的技术前瞻力。

总结一下，商用反病毒引擎一般应该有以下模块：

1. 文件类型检测模块。
2. 解压缩模块。
3. 脱壳模块。
4. office 文件预处理模块。
5. 脚本语法分析压缩模块：
6. 未知病毒检测模块。
7. 特征匹配模块。

四、 流派和工程化问题

引擎与库的分离，并不是说两者是可以无关的，相反，如果我们打一个比方来描述的话，引擎好比一种口径的枪管，而病毒库好比弹匣，反病毒记录如同子弹，你是不能用 5.56 的枪去打 7.62 毫米的子弹的。反病毒引擎是围绕病毒数据定义方式来组织的。这种组织方式有深刻的历史演变色彩。从而形成了不同的流派或者风格。

当今，扫描引擎中，有不同的处理思路和技巧，以 AVP 为代表的以效力优先，通过预制大量的脱壳器、解包器，实现极为彻底的检测。而以 Norton 等为代表的以效率优先，则更多地考虑扫描速度和降低系统地占用。两者孰优孰劣，目前还不能最后定论。而类似 F-Secure 这种为了提高查杀能力双引擎反病毒软件也已经产生了多年。

有些朋友认为，只要 crack 了某个国外厂商的病毒库，用其特征码，不就可以自己做一个引擎，写一个查毒软件了么？其实，多数软件库中并没有特征码，而是存放一种形式的描述或者签名。如果不借助足够多的样本，根本就无法还原。更何况，这种描述，在一些反病毒软件中，是针对脱壳器或者虚拟机还原的后的内存结果，而并不是存在于静态的病毒体之上。对于这种数据组织，除非你连同人家的引擎一并拿来，否则并没有使用价值。实时上这种方式一方面可以缩短纪录的长度，减少反病毒软件的体积。另一方面，对于保护厂商的劳动和减少伪样本（fake）的出现，也有现实的意义。

反病毒引擎的出现是一个工程化结果，引擎与调度框架的独立，把病毒检测彻底从命令行参数、gui 界面、目录树递归、实时监控等等中解放出来，变成一个通用的接口。反病

©安天实验室 版权所有

毒引擎与反病毒软件调度框架的独立，以及反病毒引擎与病毒库的分离，都是反病毒发展的必然。

但我们希望工程化不是一时的解决方案，是科学化指导的工程化，我们引擎的规划是高屋建瓴的，能预留对未来的支持，不用频繁改动，不会因为一种异常的出现，来破坏引擎的规范化。

有一个鲜明的例子，可以说明，引擎设计时所需要的全面和谨慎，我们在 2000 年曾经发现了 McAfee Virus Scan 引擎的一个严重 bug，当用户选择，扫描压缩包的时候，如果自解压文件自身可执行部分感染了病毒，不能被检测。这个 bug 我们虽然提供给了 NAI 办事处，但是长时间未得到响应，我们认为，也许这个问题早就被 NAI 发现，但看上去一个很小的调整，将对整个引擎结构产生影响。

反病毒引擎的出现，是反病毒历史上的革命，他在泛科学化的空泛争论中挽救了反病毒的事业，并将其带入了工程化轨道。同时，于是我们不必关心每一个病毒的表象，不用自己冒充高手，去用 Debug 或者粗糙的工具去对抗病毒，我们尽可以享受实时监控给我们的透明保护，或者品位扫描器帮我们查杀病毒时的那种快感。

©安天实验室 版权所有