反病毒“芯”技术

火绒反病毒引擎简介

目录

1. 反病毒引擎概述

• 反病毒引擎的定义

反病毒引擎负责扫描给定待扫描对象是否包含恶意代码。一般来说，反病毒引擎应至少具备以下属性：

1) 对于给定对象，评估是否包含恶意代码；
2) 能够明确描述该对象所包含恶意代码类型，如：木马、后门、蠕虫等；
3) 对于寄生类恶意代码（宏病毒、感染型病毒等），可以从宿主对象中剥离恶意代码，并还原宿主对象数据；

• 反病毒引擎的构成

符合上述定义的反病毒引擎，一般来说至少由以下几个模块构成：

1) 数据格式识别、分析模块
   负责对待扫描对象的格式进行识别和分析，为扫描核心提供足够的格式相关信息；
2) 反病毒特征库
   a. 本地特征库；
   b. 云特征库；
3) 扫描核心
   负责整个反病毒引擎的扫描逻辑，不同的扫描技术也由扫描核心来调度；

• 反病毒引擎的扫描技术

1) 特征扫描
   a. 全文哈希；
   b. 分段哈希；
   c. 局部敏感哈希；
   d. 关键数据
      i. 通过提取恶意代码中关键代码或数据片段来标识此类恶意代码；
      ii. 关键数据特征描述和方法多种多样，但至少会包含两类信息，即定位方法和匹配方法。例如，在文件偏移 0x100 至 0x200 之间，查找序列 AABBCCDDEEFF；
2) 启发式扫描
   a. 静态启发式扫描：通过提取待扫描对象的静态信息，通过启发式算法评估其恶意性；

b. 动态启发式扫描：通过提取待扫描对象的动态信息（例如：在虚拟沙盒中产生的行为），通过启发式算法评估其恶意性；

3) 动态行为分析

在反病毒引擎扫描时，通过在虚拟沙盒中动态执行待扫描对象，并捕捉其动态行为，并通过行为模式或启发式算法来评估其恶意性；

4) 基于大数据的统计分类

a. 支持向量机（SVM）；
b. 决策树；
c. 神经网络；
d. ……；

5) ……

- 反病毒引擎的解码（Decomposition）技术

当在当前待扫描对象中没有扫描到恶意代码时，扫描核心需要对该对象进行分析，并进行子对象拆分，以便对拆分后的子对象进一步进行扫描。通常，解码技术包括：

1) 解压缩；
2) 解安装包；
3) 解复合文档；
4) 其他数据解码；
5) 可执行文件脱壳；
6) ……；

2. 火绒反病毒引擎架构

• 火绒反病毒引擎架构

• 火绒反病毒引擎组件

总体来说，火绒反病毒引擎采用单内核（Monolithic Kernel）设计，并采用比较常规的模块划分方式，主要包括：

1) 火绒反病毒特征库
   与其他主流反病毒引擎并无二致，火绒反病毒引擎是一款特征驱动的引擎；
2) 火绒反病毒引擎扫描核心
   扫描核心负责全部扫描逻辑的调度，包括特征扫描、静态和动态启发式扫描，以及宏病毒和感染型病毒查杀等功能；
3) 火绒反病毒引擎 I/O 抽象层
   为扫描核心提供一致的 I/O 访问接口，同时提供相应的缓存机制来减少对物理磁盘的 I/O 访问；
4) 火绒反病毒引擎文档分析模块
   文档分析模块负责对不同格式分析以及解码的支持，包括各种压缩文档和邮件、PDF 等各种复合文档等；
5) 火绒虚拟沙盒
   火绒反病毒引擎在进行扫描时，会将待扫描对象置于火绒虚拟沙盒中进行受控虚拟执行，以此来实现脱壳、行为分析等动态分析技术；

3. 火绒反病毒引擎特性

• 广泛的平台支持

目前，火绒反病毒引擎支持以下操作系统平台：

1) Windows x86/x64;
2) Linux x86/x64;
3) Mac OS X x86/x64；

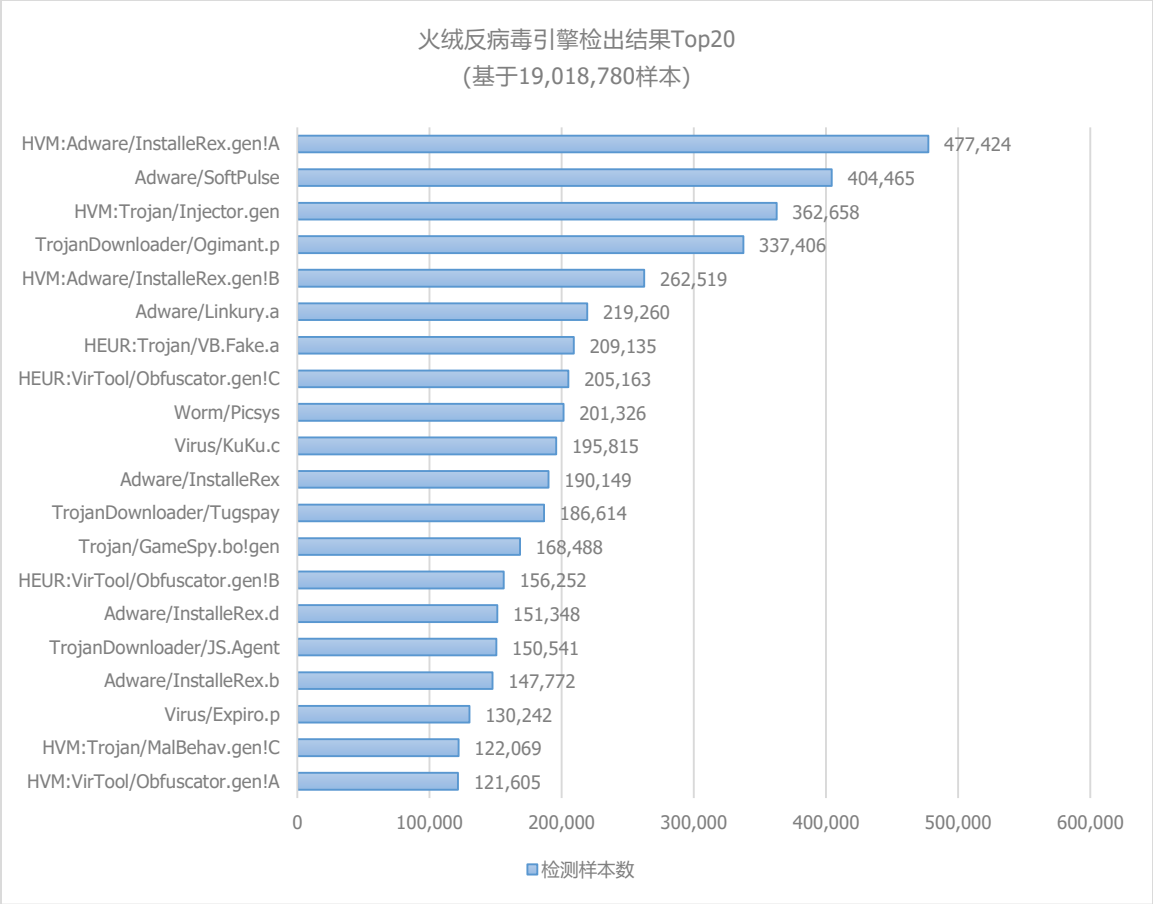
火绒反病毒引擎非常容易移植到 FreeBSD 等类 Unix 操作系统平台。

• 通用扫描（Generic Detection）技术

从 2007 年左右开始，以 Trojan/C2Lop（业内通常称其外层混淆器为 Swizzor）为代表的各种自定义壳和混淆样本大量涌现，到 2012 后，此类样本的快速迭代更呈现井喷之势。

火绒反病毒引擎凭借火绒虚拟化技术，对几乎所有待扫描 PE 样本均应用通用脱壳和动态行为扫描，用较少的记录，长期、有效地检出家族性样本。凭借火绒虚拟沙盒接近真实 CPU 的执行效率和高还原度的操作系统环境仿真，火绒反病毒引擎拥有了很强的抗干扰能力。

下面的图表展示了火绒反病毒引擎在 1900 万样本的量级上的检出结果。从中可以看到，单条检出样本数最高的扫描特征 HVM:Adware/InstalleRex.gen!A 命中了超过 47 万样本，而 2014 年流行的混淆下载器 TrojanDownloader/Ogimant.p 则单条记录检出了超过 33 万样本。



- 轻量化的设计

1) 轻巧的特征库

   a. 抽取恶意代码中的关键片段作为特征，类似生物病毒的 DNA 片段；

   b. 通过高度复用、重组恶意代码 DNA 片段来描述不同的恶意代码，最大限度减少特征库中的冗余数据；

   c. 通过极强的通用扫描技术，火绒反病毒引擎仅需要极少的扫描特征便可以检出同一恶意代码家族的不同变种；

2) 采用单内核（Monolithic kernel）而非微内核（Micro kernel）

   火绒反病毒引擎采用单内核的设计，引擎组件间的耦合度极高，不仅实现了轻量化的设计，还避免了微内核设计（例如基于类 COM 架构的引擎设计）所带来的效率损失；

- 丰富的文件格式支持

火绒反病毒引擎支持对任意文件类型的扫描。但对于不同的文件格式，仍有专门的分析模块进行分析和解码（Decomposition）的操作。目前此类专门的分析模块包括但不限于：

1) 可执行类型

   PE、NE、LE、MZ、COM、DEX、ELF、Mach-O、……；

2) 脚本类型

   HTML、JavaScript、VBScript、AutoIT、BAT、PHP、Lisp、……；

3) 压缩包类型

   RAR、Zip、7-Zip、Gzip、Bzip2、Tar、CAB、ARJ、……；

4) 安装包类型

   NSIS、SmartInstallMaker、……；

5) 复合文档类型

   OLE、Office Powerpoint 容器、Office 宏、SWF、MIME、MSO、PDF、……；

6) 其他数据类型

   Windows 快捷方式、Windows 注册表文件、……；

7) ……

- 多种扫描技术的应用

火绒反病毒引擎的扫描核心应用了多种扫描技术，并根据待扫描对象的类型配置相应的扫描策略。火绒反病毒引擎目前应用的扫描技术包括：

1) 传统特征扫描；
2) 静态、动态启发式扫描；
3) 基于虚拟沙盒的动态行为分析（行为沙盒）；

- 代码级修复能力

正如前文"反病毒引擎定义"中所述，从宿主对象中剥离恶意代码并还原宿主对象数据是反病毒引擎的基本属性之一。火绒反病毒引擎对于寄生类恶意代码拥有完善的解决方案，包括但不限于：

1) 宏病毒查杀
    a. Office VBA 宏病毒；
    b. Office Excel 公式宏病毒；
    c. ……；
2) 感染型病毒查杀
    a. 入口点模糊类（EPO）感染型病毒;
    b. 多态类（Polymorphism）感染型病毒；
    c. 变形类（Metamorphism）感染型病毒；
    d. ……；

- 基于火绒虚拟沙盒的通用脱壳（Generic Unpacking）

脱壳技术				
具体技术	静态脱壳	动态脱壳		
		虚拟机动态脱壳		调试动态脱壳
		指导脱壳	通用脱壳	
技术原理	在对壳代码完整分析的基础上，单独编写相应的识别、解码模块；	通过虚拟机的调试接口，模拟调试的过程动态还原原始代码和数据；	不对壳进行识别，通过虚拟机动态执行带脱壳程序，通过启发式分析判断停止条件；	通过调试被加壳的程序动态还原原始代码和数据；
优点	1. 可跨平台； 2. 执行效率可控； 3. 可以精确还原原始入口点；	1. 可跨平台； 2. 控制逻辑编写简单； 3. 可以控制精确定位到原始入口点； 4. 编写控制逻辑的复杂度和时间成本远远低于静态脱壳；	1. 可跨平台； 2. 无需编写任何控制逻辑； 3. 可以脱未知壳（包括各种自定义壳和混淆器）；	1. 控制逻辑编写简单； 2. 可以控制精确定位到原始入口点； 3. 编写控制逻辑的复杂度和时间成本远远低于静态脱壳；
缺点	1. 需要对每个壳单独分析和编码，复杂度和时间成本非常高； 2. 只能脱已知壳；	1. 仍需要对每个壳单独编写控制逻辑； 2. 只能脱已知壳；	1. 无法精确定位到原始入口； 2. 存在"跑"的太远，拖慢扫描效率的可能；	1. 被脱壳程序与脱壳程序需要运行在相同平台； 2. 存在代码"跑飞"的可能，导致本地系统被恶意代码感染；
适合反病毒引擎应用	√	√	√	x

火绒反病毒引擎应用	x	√	√	x
火绒应用的壳类型		1. UPX、FSG、MEW、Upack、PETite、MoleBox、Aspack、NSPack、Asprotect、PECompact 等；	1. ZProtect、VMProtect、Themida 等公开壳； 2. Trojan/FakeAV、TrojanDownloader/Upatre、Backdoor/Simda 等自定义壳； 3. 任何其他混淆器；	

上面的表格清晰地描述了不同脱壳技术的原理，以及各自的优势和劣势。

火绒反病毒引擎完全采用虚拟机动态脱壳的方式来处理壳的问题。另外，火绒反病毒引擎仅通过指导脱壳来解决少数公开壳，而对于包括病毒使用的自定义壳、代码混淆器在内的所有其他代码级对抗技术，均采用通用脱壳技术来解决。

- 基于火绒虚拟沙盒的动态行为分析（行为沙盒）

火绒虚拟沙盒可以跟踪和记录运行在其中程序的行为，火绒反病毒引擎通过行为记录，可以通过启发式分析算法对程序的恶意性进行评估。我们将虚拟沙盒的此类应用称为行为沙盒。火绒反病毒引擎以 HVM：开头的病毒名，均由火绒行为沙盒检出。

将动态行为分析引入反病毒引擎，需要满足两个条件：

1）虚拟沙盒的执行效率要足够高：能在有限的扫描时间内执行尽量多的代码，以释放更多的程序行为；

2）虚拟沙盒的仿真环境要足够逼真：能让运行在其中的程序"认为"运行在真实环境中，并释放真正的行为；

下面，我以 TrojanDownloader/Upatre 家族的两个样本（SHA1 分别为 51d17236fbf0236ccd2571e252a9f173f37702a4 和 a3c20b864669b19407d56e240281d88ca10c1c7f）为例，来作简单说明。

首先，通过静态分析，这两个样本的静态特征（入口代码、数据特征等）完全不同，很难通过静态特征来统一检出。之后，在火绒虚拟沙盒中运行，运行后展开的镜像数据虽然共性数据非常多，但关键数据却完全不同。这和表格中的行为序列表现相同，两个样本自复制的文件名、InternetOpen 的 Agent 名称、试图访问的网址等关键数据完全不同，详见下面的表格。最后，虽然关键数据不同，但通过观察很容易发现，两个样本的行为具有极大的同源性。火绒反病毒引擎在对待扫描样本进行扫描时，便是根据动态捕捉到的行为信息来进行模式识别、发现同源性，最终作同一认定（HVM:TrojanDownloader/Upatre.gen）。

样本	51d17236fbf0236ccd2571e252a9f173f37702a4
行为序列	KERNEL32!CreateFileW("c:\Windows\temp\cahst.exe", 0x40000000, 0x00000002, 0x00000000, 0x00000002, 0x00000080, 0x00000000) KERNEL32!WriteFile(0x00000010, 0x0036582c, 0x00015334, 0x0012f99c, 0x00000000) KERNEL32!CloseHandle(0x00000008) KERNEL32!GetTempPathW(0x00001000, "c:\Windows\temp\") shell32!ShellExecuteW(0x00000000, "open", "c:\Windows\temp\cahst.exe", NULL, "c:\Windows\temp\", 0x00000000) KERNEL32!ExitProcess(0x00000000) ... wininet!InternetOpenW("Updates downloader", 0x00000000, NULL, NULL, 0x00000000) wininet!InternetConnectW(0x00000001, "payrollbureaux.com", 0x00000050, NULL, NULL, 0x00000003, 0x00000000, 0x00000000)

样本	a3c20b864669b19407d56e240281d88ca10c1c7f
行为序列	KERNEL32!CreateFileW("c:\Windows\temp\dcare.exe", 0x40000000, 0x00000002, 0x00000000, 0x00000002, 0x00000080, 0x00000000) KERNEL32!WriteFile(0x00000010, 0x0036582a, 0x0000463a, 0x0012f970, 0x00000000) KERNEL32!CloseHandle(0x00000008) KERNEL32!GetTempPathW(0x00001000, "c:\Windows\temp\") shell32!ShellExecuteW(0x00000000, "open", "c:\Windows\temp\dcare.exe", NULL, "c:\Windows\temp\", 0x00000000) KERNEL32!ExitProcess(0x00000000) ... wininet!InternetOpenW("onlymacros", 0x00000000, NULL, NULL, 0x00000000) wininet!InternetConnectW(0x00000001, "sportcalgary.ca", 0x00000050, NULL, NULL, 0x00000003, 0x00000000, 0x00000000)

4. 本地传统引擎的价值

• 本地引擎 vs. 云引擎

目前，国内安全软件多数都是采用 OEM 引擎+云引擎的套路，即便是自研本地引擎+云引擎的安全软件，也基本都是云引擎为主、本地引擎为辅。究其原因，主要是本地引擎的发展需要持续、高额的技术投入，且包括启发式扫描、虚拟沙盒等在内的核心技术存在较高技术门槛。而对于云引擎，只需要解决样本采集和云端样本自动分析平台即可。样本采集自不必多说，国内厂商在云端多采用多引擎扫描的方式进行自动"分析"，所以只需要很少的技术投入便可以带来比较好的检出效果。所以，本地引擎和云引擎的区别，其实更多的是技术投入与资金投入的区别。

云引擎由于可以实时同步云端的计算结果，所以实时性较高。但由于网络带宽的限制，在有限的扫描时间内，云引擎只能在本地提取高度抽象的数据特征发送到云端进行匹配，所以一般云引擎会选择哈希类特征（通常是全文哈希）。而哈希类特征的检出能力与样本基本是 1：1 的关系，即一条哈希特征通常只能检出一个样本，所以恶意代码的快速迭代对云端的样本收集能力、分析处理能力以及云端平台的运营成本来说，都是不小的挑战。

而对于本地引擎，引擎本身技术能力的发展可以放大特征与样本之间的比例，即用少量的特征检出更多的样本。且引擎抗干扰能力越强，给恶意代码制造者进行免杀设定的技术门槛越高，相应的有效检出时效越长。

另外，本地引擎的核心技术可以被应用于云端自动分析平台，所以本地引擎技术的发展对云端的计算结果也有着积极的促进作用。

对于火绒来说，本地引擎作为整个反病毒引擎的基石，会一直持续地发展并优化下去。同时，在适当的时机火绒也会引入云引擎来辅助本地引擎以实现对恶意代码问题的快速响应。

• 传统引擎 vs. 统计引擎

近些年，国内多款安全软件都发布了各自的基于大数据或"人工智能"的反病毒引擎。这些引擎本质上都是基于统计学算法，通过对海量样本以固定方法抽取特征，并对特征进行统计、分析，进而产生计算模型。依照计算模型对待扫描样本进行分类，进而推测样本是否属于恶意分类。那么，我们不妨统称此类引擎为统计引擎。

在没有数据为依托的前提下，我不能对此类引擎的效果妄加揣测。但无论算法如何、样本如何选取，都躲不过一个重要条件，那就是对特征的抽取。我们不妨做如下假设：我们通过外貌、服饰、声音、举止等方面可以基本准确地判断一个人的性别，这就像扫描明文的恶意代码。如果将人关进房间，只有房间的颜色、外观等特征可见，而人本身相关的特征完全看不到，这就像是被混淆过的恶意代码。将大量的人随机安排进不同颜色，不同外观的房间，试问仅通过房间的颜色和外观，通过统计、分析，产生计算模型，并推测某一房间内人的性别是否可靠？我想答案是否定的。

我想表达的是，无论采用何种算法，扫描的本质仍然是特征。火绒反病毒引擎大力发展的如虚拟沙盒等各种核心技术，均旨在如何打开那一扇扇房门，即如何挖掘出真正的恶意代码特征。自 2007 年以后，超过 80%的恶意代码均有不同形式的加密、混淆，以对抗反病毒引擎的扫描。打开这些房门的钥匙，才是反病毒引擎的真正实力。

反病毒技术无论如何演变，总是离不开特征，特征码的时代不但现在没有终结，也永远不会终结，特征只是在以不同的形态存在着。

5. 评估反病毒引擎能力

• 反病毒引擎的检出能力

对于很多关心反病毒引擎的人来说，检出率似乎是最值得关注，甚至是唯一的评估指标。的确，由于检出率的可量化性，从安全爱好者或非安全从业人员的角度来看，检出率确实是最方便评估反病毒引擎能力的。但在这里我希望明确一些概念，以使这种评估更具可操作性、结果更具科学性。

1) 关于评估样本集
    a. 量级大

       对概率有基本认识的人都能理解，评估样本量级越大，评估的结果越有参考意义，反则亦然；行业内的评测，一般选择的样本量都在 10,000 左右，最少也不会少于 1,000；
    b. 样本类型丰富

       单一类型的样本（如 PE、ELF、PDF、MIME 等）只能评估引擎对单一类型样本的检出能力，样本类型越丰富对反病毒引擎检出能力的评估越全面；
    c. 样本质量可控

       这里指的样本质量包括损坏样本比例和白样本比例：
        i. 损坏样本

           除非刻意评估引擎对损坏样本的处理方式和结果，通常评估样本集不应包含损坏样本；
        ii. 白样本

           评估样本集可以包含白样本，但评估者应对白样本比例应有明确的把握，说白一些就是，如果评估者不清楚评估样本中白样本的数量，那么就没有办法计算真正的检出率。一些厂商为制造 100%检出率连评估样本集中白样本一起入库的案例也不胜枚举；

    只有当评估方法符合上述条件，下面我们将要提到的评估结果才具有意义。

2) 绝对检出率反应引擎的整体表现

    绝对检出率是指在评估样本集上，反病毒引擎检出样本数与总恶意样本数（即排除样本集中白样本后的样本总量）的比值。很明显，这个比例越大，说明反病毒引擎的整体检出效果越好；
3) 单位检出率反应引擎的核心能力

单位检出率是指绝对检出率与反病毒特征库体积的比值。该比值反应的是反病毒引擎的核心能力，比值越大，说明反病毒引擎的核心能力越强。当然，与上述评估条件所述，样本集越大，评估的结果越有参考意义。

提高引擎的绝对检出率固然很重要，但对单位检出能力的提升更具意义。国外厂商多采用新样本批量入库（本地或云特征库），并周期性进行特征整理，完善通用扫描特征的方法。火绒也在这条道路上进行着不断的尝试，力求用最轻巧的体积打造更高的绝对检出率。

- ## 反病毒引擎的解码（Decomposition）能力

通过压缩、加壳等方式对样本进行变换，并评估反病毒引擎对特定编码方法的解码能力。对于压缩包、安装包等比较常规的解码能力，我想对于绝大多数人来说都比较容易清楚地评估，所以本文不再赘述，下面我们来聚焦对加壳及混淆类可执行文件的解码能力。

正如前文关于本地传统引擎的价值中所说，目前绝大部分恶意样本都经过一定程度加密、混淆以对抗反病毒引擎的扫描，而这其中又以 PE 类恶意样本数量最为庞大，所以对 PE 混淆样本的处理能力对于反病毒引擎来说尤为重要。

评估反病毒引擎对混淆样本的解码能力，需要如下两个步骤：

1) 挑选能够被通用扫描特征检出的样本
   由于快速处理的需要，反病毒引擎通常会选择哈希类特征对收集到的样本进行快速处理，但由于脱壳过程是非精确还原的过程，所以这类特征往往在脱壳后便会失效。所以应选择能够被通用扫描特征检出的样本。针对 PE 样本，例如可以对样本进行 UPX 加壳（因为如果连 UPX 都脱不了那就没有必要继续测下去了:-(），如果加壳前后都可以检出并且病毒名相同，那么基本可以说明这个样本是被通用扫描特征检出的；
2) 有目的地选择壳或混淆器器对样本进行编码并测试；

火绒反病毒引擎特征库包含一条 PE 文件的通用扫描特征，有兴趣的朋友可以自行编译并测试，代码如下：

```
#include <stdio.h>

int main(int argc, char *argv[])
{
        printf("HUORONG ANTIVIRUS ENGINE TEST FILE\n");
        return 0;
}
```

- 案例：火绒反病毒引擎通用脱壳（Generic Unpacking）能力评估

下面，我们通过上述测试程序，评估火绒反病毒引擎的通用脱壳能力。注意，文中所涉及的测试用例并不完备，结果仅供参考。此案例权当抛砖引玉，有兴趣的朋友可以选择其他样本对其他反病毒引擎的通用脱壳能力进行评估，亦可选择多款反病毒引擎用类似方式进行横向评估。

1) 测试用例及结果

a. 将测试样本逐字节按如下公式编码：f(b) = (b + 1) ^ 0x88；

b. 将编码后的样本转换为 C 语言字节数组格式，数组名称为 mal，并保存为单独文件 mal.dat；

c. 测试用例、附加说明、预期结果以及测试结果，以表格的形式呈现：

	测试用例	附加说明	预期结果	测试结果
测试 0	原始样本	能够被检出的原始样本；	√	√

测试 1	#include <stdio.h>
	#include "mal.dat"
	
	int main(int argc, char *argv[])
	{
	FILE *f = fopen("c:\\1.bin", "w+b");
	if (f) {
	int i;
	for (i = 0; i < sizeof (mal); ++i)
	mal[i] = (mal[i] ^ 0x88) - 1;
	fwrite(mal, sizeof (mal), 1, f);
	fclose(f);
	}
	return 0;
	}
		1. 在内存中对编码过的数据进行解码；
		2. 创建 c:\1.bin 并写入解码后的数据；
		评估反病毒引擎是否具有最基本的通用脱壳能力；
		如果反病毒引擎具有最基本的通用脱壳能力，可以扫描内存中被还原的数据或释放的被还原文件；
			√	√

测试 2	#include <stdio.h>
	#include <stdlib.h>
	#include "mal.dat"
	
	int main(int argc, char *argv[])
	{
	FILE *f;
	
	exit(0);
	
	f = fopen("c:\\1.bin", "w+b");
	if (f) {
	int i;
	for (i = 0; i < sizeof (mal); ++i)
	mal[i] = (mal[i] ^ 0x88) - 1;
	fwrite(mal, sizeof (mal), 1, f);
		由于此测试评估的是反病毒引擎的通用脱壳能力，所以应排除静态启发扫描的情况；
		最常用的方法，就是在程序入口直接退出执行，若反病毒引擎仍可以检出，则表示该引擎对此样本并不是采用通用脱壳后扫描的方式检出，则无法进行后续测试；
			x	x

	fclose(f);
	}
	return 0;
	}			

测试3	#include <stdio.h>
	#include "mal.dat"
	
	int main(int argc, char *argv[])
	{
	FILE *f = fopen("c:\\1.bin", "w+b");
	if (f) {
	int i;
	for (i = 0; i < sizeof (mal); ++i) {
	char c = mal[i];
	c = (c ^ 0x88) - 1;
	fwrite(&c, 1, 1, f);
	}
	fclose(f);
	}
	return 0;
	}
		该测试与测试 1 的不同之处在于，不在内存中恢复原始数据，而是逐字节将解码后的数据写入 c:\1.bin 文件；
		该测试与测试 1 相比会产生更大的代码量和更多的 API 调用；
		该测试评估的是：
		1. 反病毒引擎是否有能力"跑开"更大的代码量和 API 调用；
		2. 反病毒引擎是否会扫描虚拟执行过程中释放的文件；
			√	√

测试4	#include <stdio.h>
	#include "mal.dat"
	
	static void trap(void) { trap(); }
	
	int main(int argc, char *argv[])
	{
	FILE *f;
	__try {
	trap();
	} __except (1) {}
	f = fopen("c:\\1.bin", "w+b");
	if (f) {
	int i;
	for (i = 0; i < sizeof (mal); ++i) {
	char c = mal[i];
	c = (c ^ 0x88) - 1;
	fwrite(&c, 1, 1, f);
	}
	fclose(f);
	}
	return 0;
	}
		该测试是在解码之前加入了一个无限递归产生的栈溢出，并在异常处理完成后再进行测试 3 的解码过程；
		该测试评估的是反病毒引擎虚拟沙盒的操作系统环境仿真对于异常处理的仿真程度；
			√	√

.....	.....	.....		

6. 附录

A. 恶意代码分类

火绒反病毒引擎对恶意代码采用如下分类：

英文名称	中文名称	定义
Trojan	木马病毒	通过网络或者系统漏洞进入您的系统并隐藏，破坏系统或正常软件的安全性，向外泄露用户的隐私信息。
TrojanDownloader	下载者木马	通过下载其他病毒来间接对系统产生安全威胁，此类木马通常体积较小，并辅以诱惑性的名称和图标诱骗用户使用。
TrojanSpy	盗号木马	此类木马会隐匿在系统中，并伺机盗取各类账号密码，对您造成财产损失。
TrojanDropper	释放器木马	通过释放其他病毒来间接对系统产生安全威胁。
TrojanClicker	点击器木马	在后台通过访问特定网址来"刷流量"，为病毒作者获利，并会占用被感染主机的网络带宽。
TrojanProxy	代理木马	在被感染主机上设置代理服务器，黑客可将被感染主机作为网络攻击的跳板，以躲避执法者的网络追踪。
Backdoor	后门病毒	秘密开放用于远程操控的端口和权限，或主动连接黑客的控制端，将被感染主机变为可以被黑客控制的"肉鸡"。
Worm	蠕虫病毒	通过可移动存储设备、网络、漏洞主动进行传播，此类病毒具有很强的扩散性。
Virus	感染型病毒	通过感染以寄生的方式将恶意代码附着于正常程序中，并通过被感染的程序进行传播。
Rootkit	内核后门病毒	对操作系统内核进行劫持，通过隐藏其他病毒进程、注册表、文件相关操作等方式与安全软件进行对抗。
Bootkit	引导后门病毒	在操作系统启动前或启动时对操作系统内核进行劫持，通过隐藏其他病毒进程、注册表、文件相关操作等方式与安全软件进行对抗。
OMacro	宏病毒	会感染您计算机上的 OFFICE 系列软件保存的文档，并且通过 OFFICE 通用模板进行传播。
DOC	恶意文档	文档中携带恶意代码，或文档结构有潜在被溢出攻击的可能。
Adware	广告程序	广告类灰色软件，不会直接破坏您的系统和文件，但是会弹出广告或存在欺诈等风险。
Constructor	病毒生成器	病毒作者可以通过此类程序批量生成新的病毒变种，以躲避安全软件的查杀。
HackTool	黑客工具	可以被黑客利用，用来控制用户计算机或发起网络攻击的工具程序。
VirTool	代码混淆器	通过代码变形、反跟踪、反虚拟机等技术手段，专门被病毒用来与安全软件进行技术对抗的恶意代码类型。
Exploit	漏洞攻击程序	利用软件漏洞进行攻击的恶意代码类型。
Joke	玩笑程序	不存在威胁系统安全的行为，但会通过动作、声音、图像等方式惊吓程序的使用者。
TEST	引擎测试程序	此程序不包含恶意代码，仅用来检验反病毒引擎是否正常工作。