细粒度可嵌入的反病毒引擎
——我们的理想与理解

江海客

Seak@antiy.net

提纲

- AV 辩证法面临的挑战
- 细粒度处理
- 可嵌入引擎

XFOCUS TEAM
from internet,for internet

AV 辨证法面临的挑战

X'CON 2004

AV 辩证法

- 反病毒决不是简单的技术对抗，整个的AV体制，包含着很多的逻辑的、法理的因素。以及工程规划的因素，有很多具有共性的基本原则
- 客观来说，这些共性原则首先被从实践中总结形成，回头来又指导着反病毒引擎乃至反病毒工具的设计。
- 1995年，我们曾总结了反病毒体制的最基本的共性原则44条，内部戏称为AV辩证法。

一些重点条目

- 计算机病毒归根到底是一种程序。
- 计算机病毒的特征码是从程序体或者程序体的某种处理结果上，选取的可以唯一确定计算机病毒类别的标识。
- 计算机病毒最根本的判据应该是程序特征码或其他的内容相关特性。
- 有害或主观有害是一个文件被提取特征加入病毒特征库的唯一原因。
- 计算机病毒的有害，是指程序包含着用户所不期待的的对信息系统的影响。
- 一个确定的程序是否应该被反病毒软件检测，是基于明确的标准。

- 反病毒软件的工作目的是保证系统数据安全和系统正常运行，反病毒操作不应该相反的结果。
- 计算机病毒的清除过程是感染的逆过程
- 用户对于反病毒产品拥有的权利：

  定义权：反病毒软件可以默认的设置，但用户具有定义进行何种模式的检测以及是否清除的权利。

  知情权：用户有权指导反病毒软件在系统中做过什么。

  备份权：反病毒工具应该提供用户对带毒文件备份的手段。

- 应识别包裹中的病毒，在具有算法授权的情况下，可以清除包裹中的病毒，但不能删除包裹本身。
- 病毒监控的基本模式应该以阻止带毒文件的运行（获取系统控制权）为目的（即前报原则）

AV辩证法所面临的挑战

- 随着应用环境与病毒技术的不断发展，我们当时的总结很多已经走入了矛盾之中。
- 这些条件产生的根本原因是信息系统进一步复杂化和用户需求的复杂化。

挑战一：判定困惑

- 条目：计算机病毒最根本的判据应该是程序特征码或其他的内容相关特性。
- 反例：红色代码留下的CMD后门。
- 问题：传统的反病毒技术解决是与非的问题，而判定是非的唯一判据是程序内容。但在特定的条件下，有害与无害与具体的环境，具体的权限发生联系，是是非的判据复杂化。Y=f(C)的问题，是否变成了y=f(C,P,R,...)的问题？

挑战二：范围悖论

- 条目：一个确定的程序是否应该被反病毒软件检测，是基于明确的标准
- 反例：Worm.Dvldr 中的psexec工具。
- 问题：x-file概念的出现，是另外一个意义上的判定困惑。反病毒软件的手到底应该伸多长。到底应该被反病毒软件检测的标准是什么？目前很多反病毒产品都加入了adware的检测，这是否是合理（法）的？

挑战三：包裹谜团

- 条目：识别包裹中的病毒，在具有算法授权的情况下，可以清除包裹中的病毒，但不能删除包裹本身。
- 反例：DIY蠕虫（如口令蠕虫）、使用zip格式发送或者保存得的蠕虫（如netsky一些变种）
- 问题：传统反病毒软件的基本假定是，包裹文件是正常文件，但包裹中可能含有病毒。而DIY蠕虫的自解压包裹就是蠕虫体。该如何处理？

挑战四：垃圾文件/无活性样本

- 条目：有害或主观有害是一个文件被提取特征加入病毒特征库的唯一原因
- 反例：民间评测引发的危机
- 问题：病毒软件的攀比效应，一个无意义的文件，一家公司添加后，会带动其他公司的添加，为了获得民间评测的高分，是否有价值？同时，病毒纪录的不断增加也会降低扫描速度，同时如何取舍检测性能与数量？同时，数万条DOS病毒特征，是否可以从规则集中摘掉？

挑战五：职责问题

- 条目：计算机病毒的清除过程是感染的逆过程
- 案例：遗留后门问题/蠕虫杀而复来
- 问题：反病毒软件是否要负责恢复病毒对系统的所有修改？是否应该负责处理漏洞？这是一个收敛的工作吗？

挑战六：时机选择

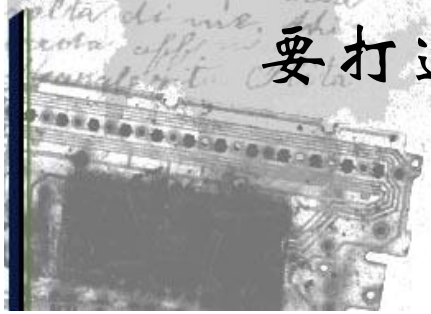
- 条目：病毒监控的基本模式应该以阻止带毒文件的运行（获取系统控制权）为目的
- 案例：关于文件评估技术的争论
- 问题：在现有技术对于未知PE病毒，特别是对于未知特洛伊木马/后门等程序存在困难的情况下，基于行为判定的后报技术是否可以被提倡？如果有一定的误报风险，是否允许应用？

挑战七：主动遏制

- 条目：用户对于反病毒产品拥有的权利...
- 案例：扫描蠕虫对网络的影响
- 问题：病毒已经从“谁感染谁受害”，向“谁感染谁害人”演化，那么是否允许有关部门从远端不经用户许可清除病毒，什么样的手段是可以接受的？技术问题，法律问题？

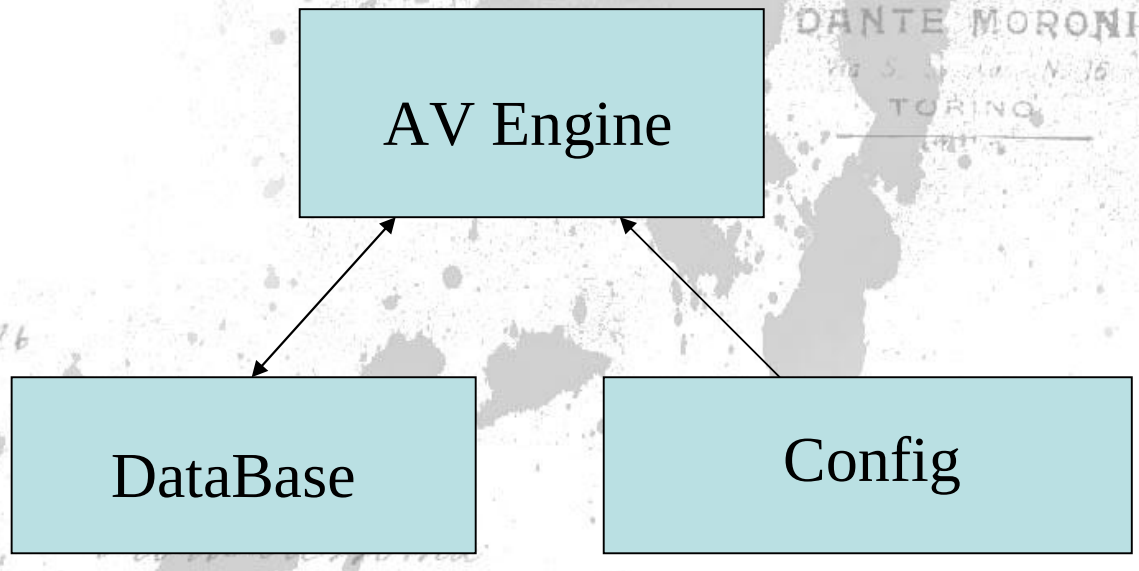
问题的提出与解决

- 对AV的挑战可以分为三类，技术挑战（如加密、变形、EPO等等）、架构挑战（如宏病毒、脚本病毒等等）和逻辑挑战。
- 上述任何一个事例都不是什么技术难题，而是对传统反病毒引擎基本体制和基本逻辑产生了微妙的影响。
- 我们需要的不是程序编写中的权宜之计，而是要打造更灵活、更合理的引擎架构。



细粒度处理

引擎三要素



我们认为，引擎的三要素，是引擎、病毒库和配置，引擎依赖于病毒库进行检测，同时又根据配置的定义进行工作。以前我们，更多的关注引擎程序，今天我们也要把更多的目光投向配置，看它能为我们提供什么？也重新审视一下我们认为是体力活的病毒库，看看创造的潜力是否依然存在！

	类型一	类型二	类型三	类型四
序号	√	√	√	√
模块号	√	√	√	√
病毒名	√	√	√	√
特征码首字			√	√
offset1+Sign1			√	√
offset2+Sign2			√	√
文件类型旗标				√
处理参数	√		√	√
处理模块名			√	√

传统的引擎

- 基于上面的库工作，通过类型三、四的纪录，检测95%的病毒（特征码检测）。通过类型一、二检测剩余5%比较特殊的病毒（独立模块检测）。
- 通过处理参数进行超过80%的病毒处理，通过处理模块处理剩余20%的病毒。

传统引擎控制基本点

- 对象控制：检测哪些对象。
- 行为控制：如何处理。
- 效力控制：检测强度。

传统的配置方法

- 引擎内部流程控制
- 内部开关（包括调试开关）
- INI控制（开放开关）

例：对象控制

- Memory=Yes;是否检测内存
- Sectors=Yes ;是否检测引导扇区
- Files=Yes ;是否检测文件系统
- Packed=Yes ;是否检测壳
- Archives=Yes ;是否检测包裹
- MailBases=Yes ;是否检测邮件文件
- MailPlain=Yes ;是否检测编码文件
- FileMask=2 ;扩展名检测范围
- UserMask= ;用户定制扩展名
- Exclude=No ;是否不检测定制的扩展名
- ExcludeMask= ;不检测扩展名的定义

例：行为控制

- InfectedAction=0 ;是否清除病毒
- InfectedCopy=No ;是否备份病毒
- InfectedFolder=Infected ;备份目录
- SuspiciousCopy=No ;是否备份可疑文件
- SuspiciousFolder=Suspicious ;备份目录
- Report=Yes ;是否生成日志
- ReportFileName=Report.txt ;日志文件名

例：效力控制

- Warnings=Yes ;近似警告开关
- CodeAnalyzer=Yes ;代码分析开关
- RedundantScan=Yes ;多重扫描开关

That's enough?

- 对于传统的反病毒工具工作环境，这样的控制粒度是足够的。
- 但对于更复杂的环境，会如何呢？

应用案例1

- 同一个引擎，作为单机反病毒软件使用和作为邮件服务器反病毒模块会有什么样的不同呢？
- I-Worm.Nimda.e是一个感染式的蠕虫，在Local处理的时候，应该作为一个PE感染式文件处理，但对邮件服务器，则可以抛弃。
- Win95.CIH是一个感染式病毒，当检测到这个病毒，无论对于Local还是邮件服务器，则都需要进行感染式病毒的清除操作，还原出原有文件。
  - 这样区别的性质在于，Win95.CIH没有mail发送自身的特性，如果出现在网上，应该是用户主要发送的可执行程序。而Nimda则反之。
  - 这种处理粒度的要求，已经对不同类别的病毒在不同环境下进行不同处理，超出了传统引擎控制粒度的能力。

应用案例2

- 一台网络病毒检测设备，有若干响应模块。
- 这些响应模块在何种策略下工作？
- 一些邮件蠕虫发件人是随机的，反馈式发送会造成什么？
- 一些邮件蠕虫的收件人是bot生成的，追加式发送会造成什么？

- 消息告知
- 追加式邮件告知
- 反馈式邮件告知
- 断连接
- …

用户的要求

	Smtp 检测				POP3 检测			
	收件人为构造	收件人为非构造	发件人为真	发件人为假	收件人为构造	收件人为非构造	发件人为真	发件人为假
反馈式			有效	无效			有效	无效
追加式	无效	有效			有效	有效		

应用案例3

- 联动是旁路设备实现响应行为的有效方法
- OPSEC、TOPSEC

- 对扫描蠕虫和邮件蠕虫的不同处理！
- 扫描蠕虫感染节点IP的封闭是简单的，但对邮件蠕虫如果采用相同的方式，可能会造成不停的重发，形成对设备的DoS。
- 网络中是否有代理服务器？

That's not enough!

- 上面的问题，并非都是应该由引擎解决的，但都或多或少与引擎所提供的信息发生联系。
- 传统引擎的控制能力的限制、信息粒度的不足，暴露无遗。
- 问题如何解决？

XFOCUS TEAM
from internet,for internet

细粒度的可嵌入引擎

X'CON 2004

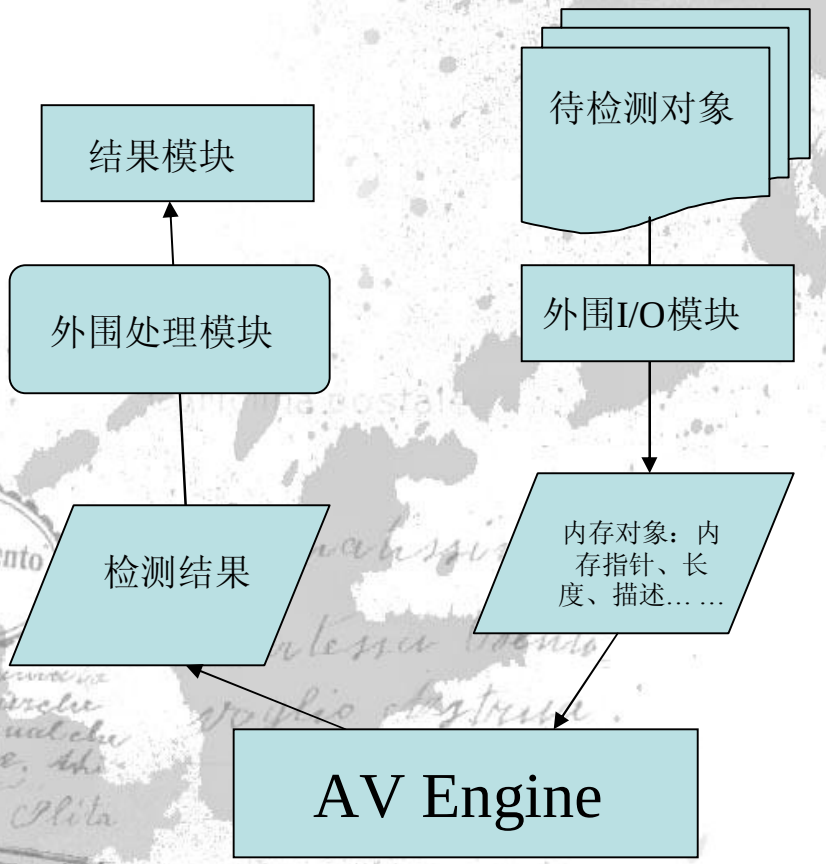
问题的提出

- 网络安全层次的下移（CISCO SDN、SP2），意味着病毒过滤体制会不断延伸（嵌入）的各层次设备上。
- 上述的讨论是为了反病毒引擎适应更复杂的使用环境的准备。
- 嵌入设备或其他应用环境的反病毒引擎是细粒度的引擎

可嵌入引擎的应用

应用形态	说明
防火墙反病毒模块	可以基于网络引擎为包过滤防火墙搭建基于线速检测的病毒过滤模块、也可以为基于文件引擎为应用代理、透明代理以及支持流过滤的防火墙搭建基于文件流的病毒过滤模块。
路由器反病毒模块	通过高速的包级别扫描，为路由设备添加病毒过滤能力。
交换机反病毒模块：	通过高速的包级别扫描，为交换、共享设备添加病毒过滤能力。
IDS病毒检测查件：	通过网络引擎扩展**IDS**的网络病毒检测能力。
GAP反病毒模块：	为网闸设备扩展病毒过滤能力。
邮件系统病毒保护：	为邮件服务嵌入病毒检测能力。
独立反病毒软件：	客户只要编写外围界面，就可以开发自己的反病毒软件。

基本要求一——内存引擎



一个高度可嵌入的反病毒引擎，必然是一个与I/O无关的内存引擎。

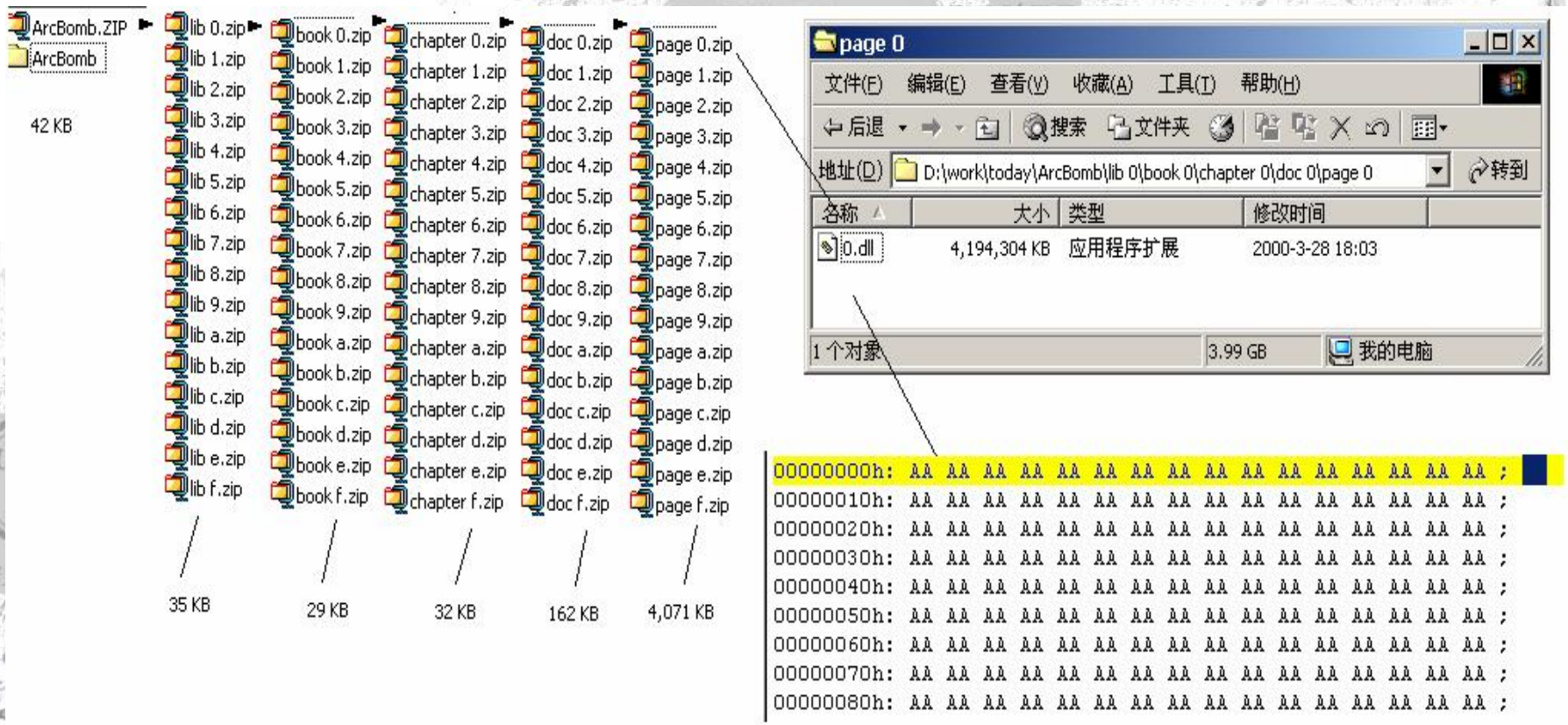
内存引擎的接口DEMO

```
/* 扫描参数结构 */
typedef struct _AVLF_SDK_SCAN_PARA
{
char * pBuffer;                    /* 待检测缓冲区指针 */
unsigned long ulSize;   /* 待检测内存的大小 */
const char * pDescription;         /* 描述信息 */
int   bUnpack;                                  /* 是否解包 */
int  bKill;                        /* 是否杀毒处理 */
int  bKilled;                      /* 是否杀毒成功 */
} AVLF_SDK_SCAN_PARA,*PAVLF_SDK_SCAN_PARA;

/* 设置收报人 */
AVLEACHSDK_API int AVLF_SDK_SetReciver(IReportReciver *pReciver);

/* 扫描：返回0未发现病毒，返回1发现病毒，病毒详细信息在收报人类接收 */

AVLEACHSDK_API int AVLF_SDK_Scan(PAVLF_SDK_SCAN_PARA pParamter);
```

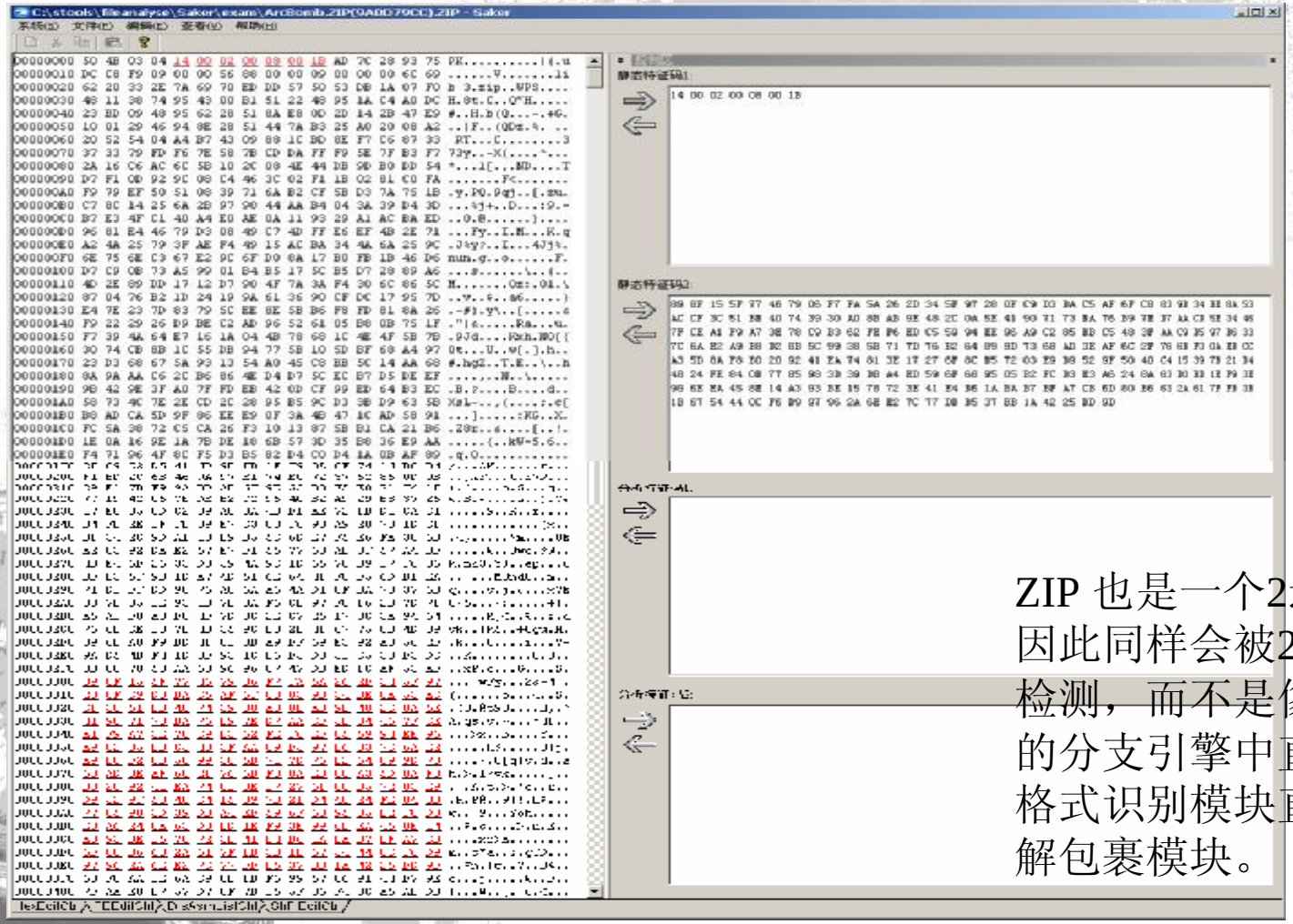
基本要求二——递归引擎

- 现代的反病毒引擎已经从以格式识别模块为先导的分支引擎，演化为递归引擎。
- 在递归引擎中，被检测对象可以同时具有多个标志，并被每个标志对应的模块分别检测到。
- Mcafee检测自解压包裹bug
- archbomb.zip的例子。

例：Archbomb.zip如何DoS AVware



如何检测ArchBomb.zip



sign1

Offset: 4h

Length: 7h

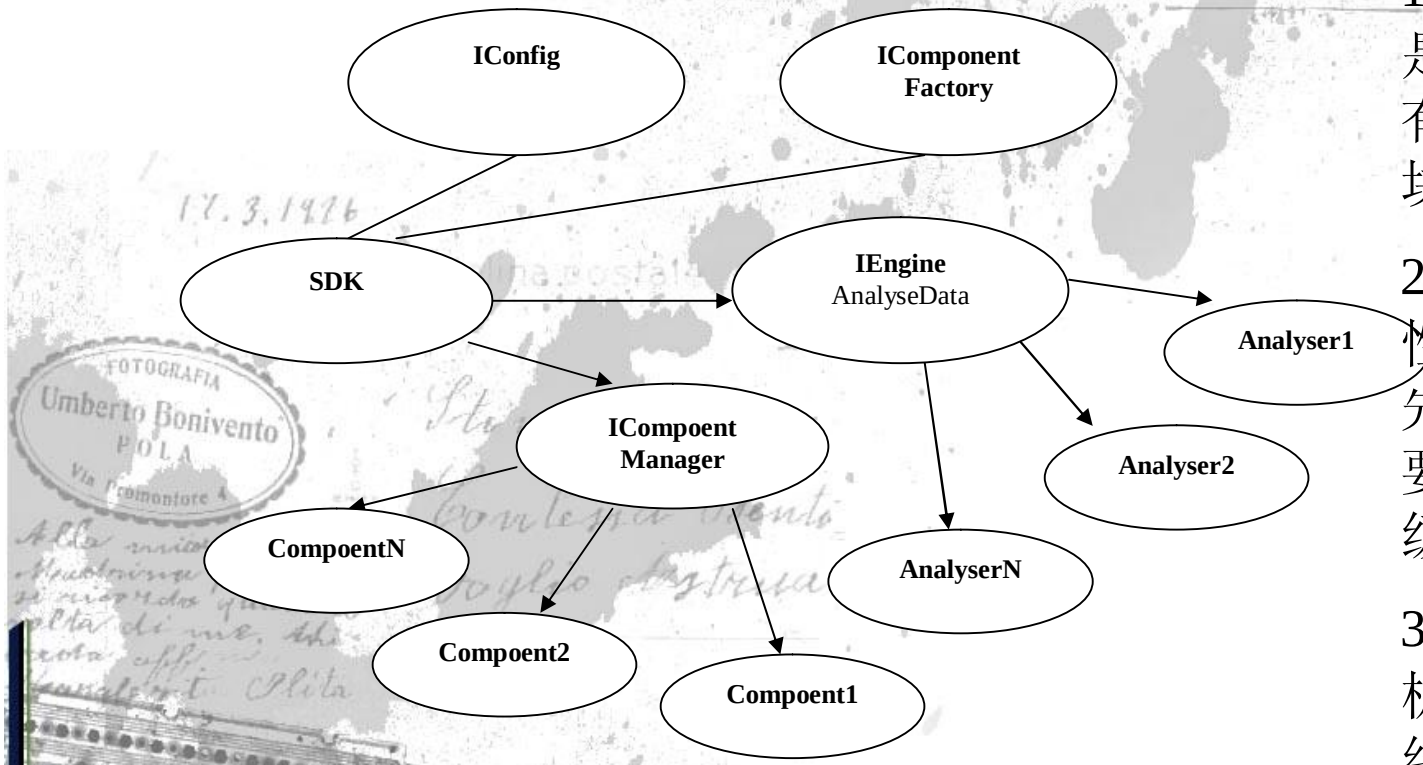
Sign 2

Offset:300h

Length:F0h

ZIP 也是一个2进制流，因此同样会被2进制引擎检测，而不是像在传统的分支引擎中直接送被格式识别模块直接送给解包裹模块。

图例——递归引擎



1、分析器之间是并列的，并没有绝对的先导模块。

2、结果有优先性，发现病毒优先级最高，而需要进一步预处理级别最低。

3、原则上，分析器串性工作，结果优先的引擎前置。

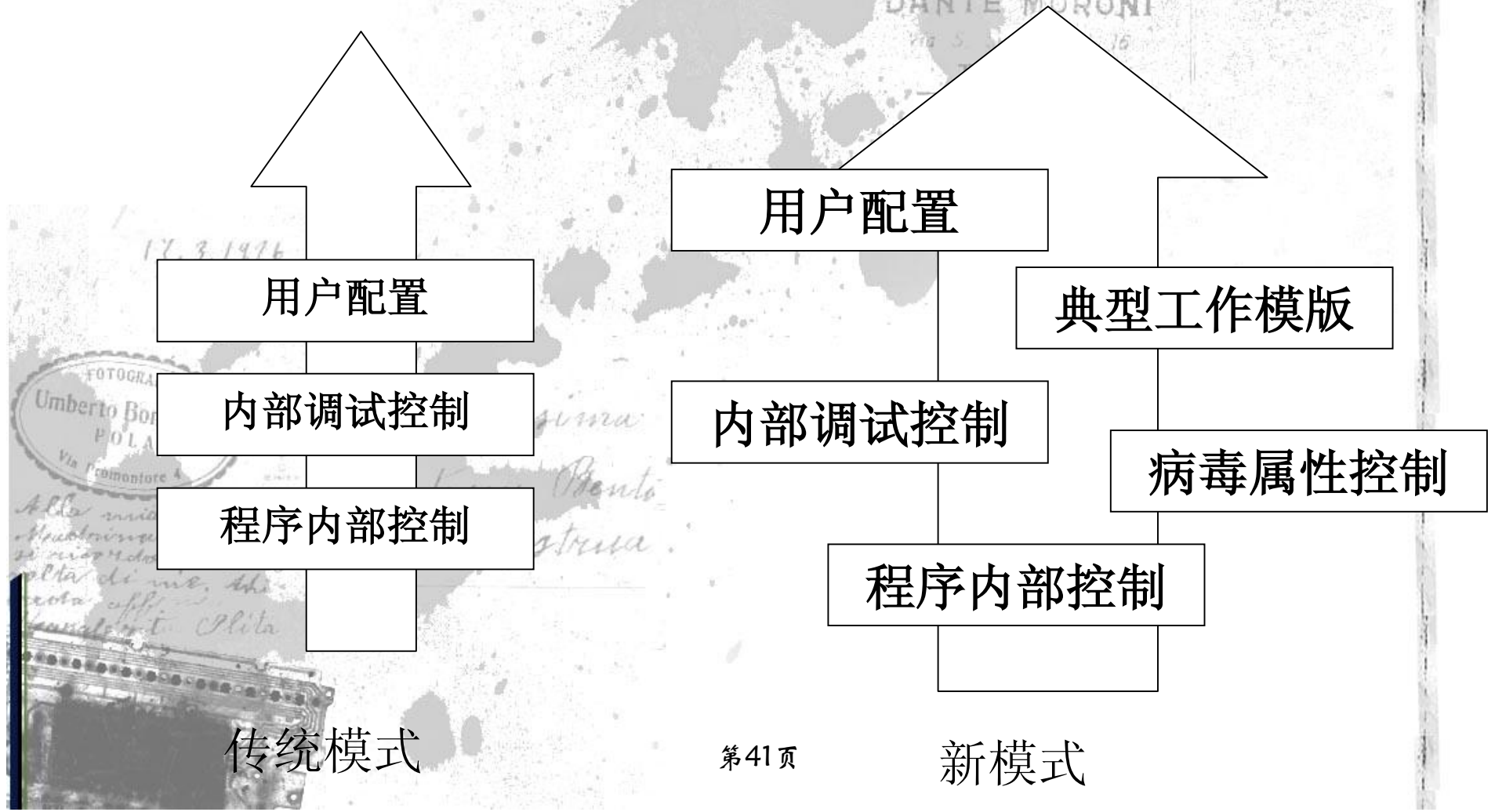
基本要求三——可移植引擎

- 传统的可移植性是基于coff文件结构的特点。
- 但在新的条件下，工作环境可能是x86架构，也可能是PPC这种非x86架构。
- 那些x86汇编编写的模块称为移植的障碍。

基本要求四——高可控引擎

- 细粒度要求的本质是什么？
- 在不同环境下的病毒处理，不仅与病毒的感染性有关，而且与病毒的“特性”有关！
- 控制粒度需要可以到达病毒个体，病毒库需要提供更多的信息！
- 病毒库的更多信息产生了特性控制。

新的控制层次


用户配置
内部调试控制
程序内部控制
传统模式
用户配置
典型工作模版
内部调试控制
病毒属性控制
程序内部控制
新模式

传统病毒库与细粒度引擎病毒库

```
struct vxdb
{
char name[255];
char fword[4];
char offset1[4];
char crc1[8];
char offset2[4];
char crc2[8];
…
};
```

```
struct tgvxdb
{
char name[255];
char fword[4];
char offset1[4];
char crc1[8];
char offset2[4];
char crc2[8];
…
int vxattribute ;
};
```

基本要求五——精确处理引擎

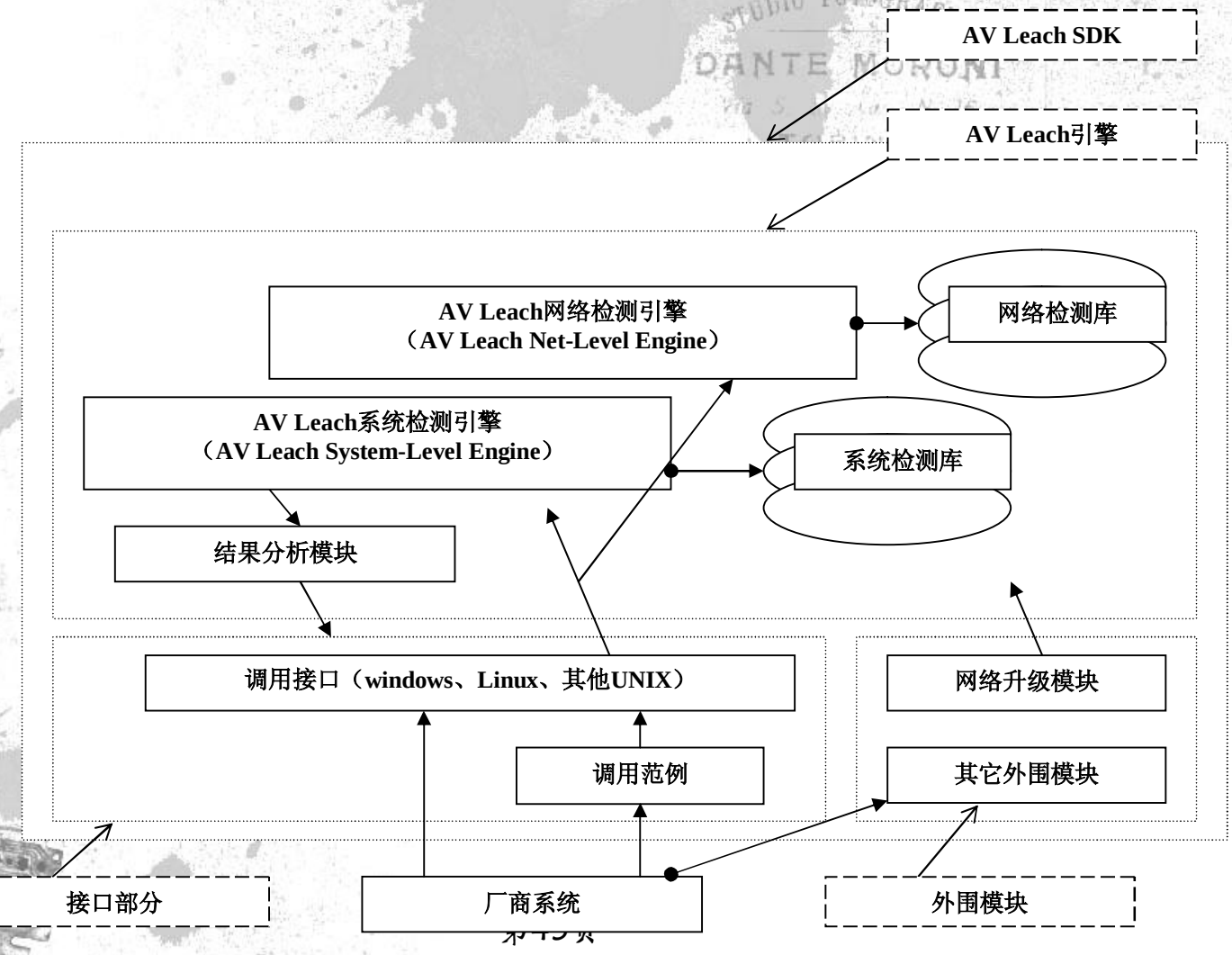
- 完美的逆过程，是一个理想。
- 细粒度引擎要求结束反病毒公司“不分析”病毒的时代！

如何处理

- 清com尾
- 清com头
- 清exe尾
- 清ne尾
- 清pe尾
- 删除文件
- 拷贝数据块
- 移动数据块
- 插入数据块
- 修改数据块
- 删除数据块
- 填充数据块
- 截数据尾
- 截数据头

- 左图是兄弟AV公司使用的感染式病毒清除参数集，相反，对于更容易处理的非感染式病毒我们一度反而没有这样的耐心。
- 我们需要同样细腻的系统处理宏定义脚本。
- 工作是否不收敛？

我们的模型


AV Leach SDK
AV Leach引擎
AV Leach网络检测引擎
（AV Leach Net-Level Engine）
网络检测库
AV Leach系统检测引擎
（AV Leach System-Level Engine）
系统检测库
结果分析模块
调用接口（windows、Linux、其他UNIX）
网络升级模块
调用范例
其它外围模块
接口部分
厂商系统
外围模块

演示病毒分析机

我们的理想与理解

- AV辩证法不是一成不变的，是发展的动态的规则。不仅需要我们的总结，也需要我们的补充和突破。
- 我们相信自己的理解，我们执著自己的理想
- 感谢焦点的兄弟们！谢谢大家！
- 欢迎mail to me seak@antiy.net