



eEye Digital Security®

Anti-Virus Heuristics

Drew Copley



X'con 2005

X'con 2005



XFOCUS TEAM BEIJING.CHINA 2002-2005

```

if (argc < 3)
{
printf ("\nwww.get - determine what httpd version a site is running\n");
printf ("punkis@attrition.org ==-\n");
printf ("arg[1] = %s\n", argv[1]);
return 1;
}
host = argv[2];
sport = atoi(argv[2]);
hostinfo = gethostbyname(host);
if (!hostinfo)
{
fprintf(stderr, "Host: %s\n", host);
exit(1);
}
servinfo = getservbyname("http", "tcp");
if (!servinfo)

```

VULNERABILITY IS OVER



演讲者介绍

- ◆ 我的名字叫Drew Copley，是eEye Digital Security公司的高级安全研究员。
- ◆ 近几年来，我的工作职责是从研究者和安全QA的角度，研究各种新旧的技术和相应的解决方案。
- ◆ 在计算机安全方面，我的主要兴趣在于攻击和防御技术，并且重点在于实现技术细节而非它所取得的效果。
- ◆ 在计算机安全方面我有多年的经验，有时候象个专家，有时候仅仅象个技术狂热者，有时用真名，有时用笔名。尽管我用过真名发布过一些资料，但我通常首选用匿名。
- ◆ 我对AV的研究来自为我们的产品做架构设计方面的初步研究。



议题介绍

- ◆ 这次演讲的主要目标是重新定义你对保护技术的看法。
- ◆ 这是一次高层次的，详尽的演讲。因为缺少文档、对启发式（技术）的误解，所以我认为做详细的介绍是有必要的。这次演讲将间接的介绍许多新的和旧的观点，这是介绍观点的一种非常有效的方式。
- ◆ 面对保护技术，我们能做什么，我们已经做了什么，这是两个非常不同的东西。
- ◆ 近几年来，许多恶意软件研究者已经意识到，事实上大部分主流AV产品已经不能满足用户的需求。
- ◆ 对金钱的渴望，平庸的能力，偏见，传统，外观和内部质量的抗争，已经使得这个行业的想像力越来越枯竭。



为什么启发式反病毒会让你感兴趣

- ❖ 从任何战斗技术来看，如果你无法防御，那么你将无法进攻，如果你无法进攻，那么你也将无法防御。我们通常把这些技术分为“防御性的”和“进攻性的”，但最后这些通常是无法很清楚区分的。
- ❖ 正如大家所知，“启发式”意思是“研究分析”。但是，“启发式”在本文中的真正意思是使得防御软件在检查潜在的恶意软件时更加智能化。
- ❖ 启发式是软件动态学习的过程，几乎在所有的防御产品中这都是很有用的，但这里我们只讨论它在防御恶意软件领域的应用。



定义启发式模型

- ◆ 任何启发式模块都会包含多种模块，虽然它们都可以称为启发式模块，但事实上，它们是独立的：如模拟执行模块，处理打包/加密文件的模块。
- ◆ 最适宜的启发式模块应该是签名模块，它有简单的动态检测能力：用于替代仅仅进行字节匹配的方式。
- ◆ 在这个模块里面存在一个基本的签名引擎，但它也会使用其它的动态签名功能来检测攻击种类和恶意软件家族。



反病毒的缺陷：没用启发式

- ❖ 依赖静态的二进制签名，是现代反病毒产品的主要缺陷。他们把“智能分析”中的“智能”去掉了（只剩下“分析”）。
- ❖ 近几年来，因为这个缺陷，脚本小子已经战胜了已有的反病毒解决方案。
- ❖ 如果说销售一份软件带来的仅是一小部分收入，那么病毒库的更新带来的就是滚滚财源了。

启发式病毒检测不需要更新，即使在许多更新模型中，比起只进行静态签名的系统来它的更新要少得多。



反病毒的缺陷：通过物理安全来做比喻

- ❖ 物理安全和计算机安全共同发展，它们之间的差别越来越难区分。
- ❖ 在继续之前，我们需要使用一个重要的比喻：把你的反病毒产品看作机场的安检。
- ❖ 如果你仅仅依赖签名，这意味着安全人员仅仅需要检查目标是否在罪犯名单中。
- ❖ 这就象拿着相片来做匹配，不使用X光机，不搜身检查武器，不察言观色。
- ❖ 事实上，象这样的机场安检，未知的罪犯将带着火箭筒通过安检，声名狼藉的罪犯也可以戴顶帽子就顺利通过。
- ❖ 最好的情况下，你会根据罪犯的相貌来做检查，同时你也会检查是否有武器，检查DNA，察言观色，另外你还有检查炸弹的警犬，检查毒品的警犬，X光机——这就是我们正在讨论的为基于普通签名的反病毒系统添加启发式检测功能。



启发式检测代理的难题

- ❖ 启发式检测代理的首要难题是：打包/加密的文件。
- ❖ 最初的启发式检测技术在对付恶意软件时工作得很好，直到打包/加密方式出现。这使得主流的AV产业把启发式检测从默认的产品中去掉。（需要说明的是，主流的AV产品没有继续发展启发式检测——一般只是在默认情况下禁用它）
- ❖ 任何人拥有大量的木马/rootkit代码都不是神秘的事情。据Symantec的AV研究者Peter Szor估计，大约有90%的恶意软件都是打包/加密过的。（参考资源：“The Art of Computer Virus Research and Defense”）



迷惑文件难题的讽刺

- ❖ 仅基于签名的AV系统的首要难题也是打包/加密。
- ❖ 这是一个在地下工作了好几年的方法：某些人在地下发布一些木马代码，AV公司基于这些代码生成签名，脚本小子把这些代码进行打包/加密。之后这些签名就再也无法检测到这个木马，因为它已经完全变成另外一个文件了。
- ❖ 这个木马的版本将永无止境。



欢迎来到恶意软件变种的世界

- ❖ 恶意软件的改变为我们带来了变种的难题。
- ❖ 脚本小子得到木马/rootkit后，他们会通过打包/加密或修改一些代码后重新编译（例如Hacker Defender和BO2K这些开源代码）的方法来手工改变它。这是最常见的恶意软件变种产生的过程。



恶意软件变种：自伪装

- ❖ 到目前为至，恶意软件的族谱上已经有了自动变种的类型。虽然这种类型的比较少见，但近年来有增长趋势。
- ❖ 软件能够通过几种方法来修改自身，所以固定的签名对它无效。
- ❖ 通常，设计者在编写这种恶意软件时会犯一些错误，所以我们能够利用它来写一些普通的签名。
- ❖ 一般情况下，这种类型的攻击设计用于蠕虫。不管如何，已经有很多间谍软件包含了蠕虫代码，这种情况有增长的趋势。



间谍软件：目的和将来

- ❖ 最初的时候，“间谍软件”这个术语仅指商业的间谍软件产品。目前，已经使用这个术语来形容所有对用户进行监控的软件，包括木马、rootkit等。我这里引用的术语“间谍软件”指的都是后者。
- ❖ 对付间谍软件就象对付其它恶意软件一样，最重要的是要有一个合适的反恶意软件代理。信息系统与其中的信息一样重要。信息被盗取比被摧毁更加严重。
- ❖ 有两个主要标准会影响间谍软件使用的增长：第一，被盗取数据的市场需求扩大；第二，肉鸡的市场需求扩大。



间谍软件变种：常见的情节

- ❖ 间谍软件变种的常见情节是：定期手工改变软件的二进制代码。
- ❖ 影响趋势的几个因素
 - ∅ 代码是开源的，并且它已经被修改和重新编译。
 - ∅ 直接对二进制代码进行编辑
 - ∅ 打包/加密
 - ∅ 某些人编写间谍软件的一些原始代码
 - ∅ 代码由间谍软件产生



间谍软件变种：十六进制编辑

- ❖ 对间谍软件进行十六进制编辑是一种很古老的方法
- ❖ 这种情况并没有很多改变。现在有更多的开源间谍软件，但很多使用间谍软件的人并不知道如何进行编码，甚至不知道如何编译它。仅仅对二进制文件进行编辑通常会比较快且比较容易。
- ❖ 主流的AV产品对二进制文件改变的检测已经好了很多。
- ❖ 地下已经有越来越多的产品，设计用于为人们修改二进制文件，它比“间谍软件生成器”还要多。



经打包/加密的恶意软件：PE文件格式

- ❖ PE文件是win32可执行文件的格式。
- ❖ For these purposes, the main things to be aware of is that there are these sections of w32 files:
 - ∅ 导入表，文件中使用的API将会出现在这里
 - ∅ 导出表，文件输出的API将会出现这里。通常情况下这个出现在DLL中，不在可执行文件中。
 - ∅ There is the preliminary shell data, and then the Entry Point (EP) of the code
 - ∅ There is a definition of the sections in which the file is divided



经打包/加密的恶意软件：速览

- ◆ 打包/加密文件的方式有很多种
- ◆ 基本的主意是把原始文件的壳用一个新壳来替代
- ◆ 原始文件的**sections**将被改变
- ◆ 原始文件的数据被加密或（并）打包
- ◆ 在这过程中原始文件的输入表会被打乱
- ◆ 打包/加密的代码将替代原始代码来运行，它会
把内存中的原始代码进行解密/解压缩，然后把它
加载到内存中并运行。



对付打包/加密的恶意软件的基本方法

- ❖ 最基本的方法是在内存中找原始文件的完整拷贝，然后把它dump到文件中。
- ❖ 这个过程一般被称为寻找原始入口点(Original Entry Point)。
- ❖ 目前有很多工具能够完成这个工作
- ❖ 在处理这种事情时，Ollydbg是很多恶意软件研究者和软件破解者最喜欢的工具，因为它支持可执行的插件，并且有很多开源的插件能够用于对付许多不同种类的打包/加密格式。



自动破解打包/加密的二进制文件

- ❖ 在这里任何人想问的第一问题都会是“为什么这个过程不能是自动化的并且应用到现代的AV产品中？”
- ❖ **This is where modern heuristics, to a large degree is, and where much of the work has been going towards, inevitably, to a large degree.**
- ❖ **Major problems exist for tackling this problem, the foremost of which is that in this just mentioned process of cracking these binaries you see that a debugger is the primary ingredient here, therefore the file is being run on the system.**



AV中的模拟器和沙箱

- ❖ 自动进行破解打包/加密文件的基本思路是：让系统自动完成，同时保护它不受到潜在恶意代码的威胁
- ❖ 进行API hooking是其中一种办法，即拦截替换它的API调用，在恶意软件与系统之间充当缓冲区
- ❖ 处理这个难题的一个更复杂的方法是模拟整个系统
- ❖ 最后，你会看到这两种方法是类似的，他们最后做的都是相同的工作。



本地API Hooking和自动分析

- ❖ 在自动分析恶意软件时，用API Hooking来代替纯模拟执行会存在很多问题。
- ❖ 用API hooking替代完全系统模拟执行时会有问题，如：当文件进行磁盘和注册表访问时，如典型的安装文件，你将会看到注册表键和文件被创建。Hooking这些调用进行分析意味着在通知应用程序有文件和注册表键被创建后，允许恶意软件继续访问这些对象。
- ❖ 事实上，这个问题准确的描述了为什么完全的系统模拟执行是非常有用的。



静态启发式和打包/加密的恶意软件

- ◆ 这里分为“静态启发式”和“动态启发式”：动态启发式描述的是模拟模型。静态启发式描述的仅是处理文件的方法，非真正的启发式。(Terms documented, perhaps coined, by Peter Szor)
- ◆ 调试器模型和反汇编模型在这里是类似的。
- ◆ 从最简单的层面来说，静态启发式引擎仅仅进行常见的解压缩/解密的操作，它会有一个数据库来保存解压缩/解密文件的方法，然后应用算法选择合适的方法来对文件进行自动处理。
- ◆ 对付已知的打包/加密算法时这种方法很有效，但对于未知的打包/加密算法就无能为力了。这就是大多数启发式引擎转向动态启发式的主要原因。



调试器和反汇编器

- ◆ It is worthwhile noting the debugger and disassembler parallels here to automated heuristics, if only because anyone who has examined malware binary code uses these
- ◆ 在这里主要的组件仅是把OP codes解析为适当的汇编指令，然后执行正确的操作—让启发式或任何自动化分析引擎偏离你所期望的操作和模拟执行你所期望的操作，让它（恶意软件）相信自己已经在运行。
- ◆ 有很多可行的方法能够攻击这种类型的系统。一个有趣的、老的方法是使用未文档化的Intel汇编代码，这样会导致模拟器不能正确处理，但真正的系统可以。



非恶意软件的打包/加密

- ❖ 打包文件带来的好处是尺寸减小，但是解压时总会消耗一些CPU时间，并且磁盘现在是很便宜的。
- ❖ 打包文件和加密文件带来的迷惑级别是不一样的
- ❖ **Encryption of files, of course, is limited, as long as you have access to the process, because the key eventually will be processed and the decryption algorithm used... the key, of course, does not have to be within the binary, or even on the system**
- ❖ 正常的软件仍然会继续会加密/打包，主要的原因就是通过隐藏一些信息来提高安全性。它至少能够阻挡一些低层次的攻击，在一定程序降低被破解的可能性。



File Obfuscation Outside of Packers/Encryptors

- ❖ 当然，并非所有的迷惑技术都依赖打包/加密。一些很好的应用程序有反破解和反欺骗的机制，它们依赖的是动态运行的加密和/或压缩方法。
- ❖ 一些在线的视频游戏客户端就使用了这种技术
- ❖ **The more serious problem with dynamic only style heuristic engines is hidden in this – how do you deal with functionality that is not immediately run when the process starts? How do you deal with functionality that requires certain user or automated interaction to actually come into play?**



静态启发式和文件分析

- ◆ Most spyware operates in this fashion: it runs, it sets itself up to embed within the system, and this historically means it sets itself up by trojaning binaries, or more commonly, by setting registry or other “run at reboot” data
- ◆ 有许多方法能做到这些，但是始终存在一个问题，如果系统重起或不管因任何原因引起的程序崩溃，二进制程序将永远无法再次运行。
- ◆ 有许多方法来构建“**sleeper agents**”，我最近为我们的“**Vice**”写了一篇关于这个的文章。
- ◆ 最后，你必须对所有文件做一些静态启发式分析，即使文件明显经过打包或加密。



静态启发式和文件分析，第二部分（寻找打包/加密文件）

- ◆ 任何启发式引擎需要做的第一件事情都是检查文件是否经过打包/加密。
- ◆ **EP in first section of file – Entry Point not in first section of file, good indicator of not being packed, though some NT kernel files are like this which means updating with patches/OS versions**
- ◆ **Manual Inspection can quickly show difference between packed/encrypted files versus regular files... this has led to entropy checks**
- ◆ **The “Zero Order” Entropy check (PEiD)... PEiD introduced zero order entropy check...evading this could be difficult but highly useful for malware**



Static Heuristics and File Analysis, Part II (Packed/Encrypted File Detection Evasion)

- ❖ **If a packed/encrypted file can evade detection, this can lend a lot of value against heuristic engines... this kind of technology should be expected to grow**
- ❖ **Obfuscation in binary as packed/encrypted detection evasion... that is, not using packed/encrypted techniques directly, but obscuring the file innately (Covered later)**
- ❖ **Ultimately, it is very difficult to evade complete detection, though through layers of redirection it is possible to have code which can evade entropy checks and at the same time hide in dangerous malware behaviors**



传播和监控代码示例

- ❖ **Examples of spreading code would be: internet connectivity, scanning code, file/directory traversal code combined with file infection code, and the like**
- ❖ **Scanning code itself is generally best caught at the network level, and this is probably best evade by extremely slow scanning techniques – this detection method is common with some honeypot systems**
- ❖ **监控代码经常使用几种hooking技术，它需要调用到不少唯一的API，如writeprocessmemory、openprocess、几种windows messaging API、createremotethread，安装服务的API等。**
- ❖ **”可疑的调用”本质上不代表什么，但它们可以为进一步的检测提供参考。**



不调用API来绕过检测

- ◆ 一个更具迷惑性的文件传染和隐藏技术包括绕过可用的API
- ◆ 底层NTFS引擎，设备对象技巧。我创建了一个底层的磁盘访问引擎，使用它能够绕过基本的磁盘操作API。对于恶意软件和检测挂接了这些API的恶意软件来说都是很有用的。
- ◆ 物理内存对象技巧。有许多的直接访问物理内存对象的技巧，第一个披露的是来自sysinternals的Russinovich，之后是CrazyLord发布在Phrack上的文章。
- ◆ **Detection of Object manipulation... should be noted there are simple and direct detection techniques on these, for instance, NTFS structure detection, the fact this obscure code is obscure, etc... DiamondCSS processguard blocks access to Physical Memory object, etc**



启发式引擎和签名

- ◆ **As you can therefore see, there are a wide range of “bad behaviors” which are not yet known, therefore it is impossible for an Heuristic agent to be completely knowledgeable in what is “good” and what is “bad”**
- ◆ **象人类具有的基本智慧一样，使系统具备区分好坏的能力。**
- ◆ **不管如何，这些判断都可以编程到系统里面。**
- ◆ **Inevitably, it is wrong, however, to think of a heuristic system as a reasoning Artificial Intelligence System, but rather as a more advanced Signature Analysis System... one which requires continual signature updates – it should be noted this is not unlike how we humans operate as well, however... many past and current heuristic solutions do not take this route**



种类和家族

- ◆ 这里有两个概念：“种类”和“家族”
- ◆ 哪种启发式能对付攻击种类和malware家族
- ◆ 从malware的发展史来看，它们保留并利用了很多相同的技术
- ◆ One of the most ludicrous examples of spyware techniques recycled is simply in the various restart methods used by trojans – and rarely used by other applications... for years there were few automated ways to protect against this behavior, even though trojan after trojan used the same techniques
- ◆ People rarely write code with no influences, and generally, malware has key code taken from other malware – the heuristic attack simply attacks the key code methods, above all



动态加载API

- ❖ 最常见的迷惑API输入表的方法仅仅是通过动态加载API
- ❖ 动态加载API本身需要几个关键的API（这里不考虑绕过API），例如loadlibrary。
- ❖ **A popular method - I believe found by the famed 29a group – for dynamic API calling involves obfuscating the API names called themselves... this adds an initial layer of redirection to sorting out dynamically called APIs**
- ❖ **Almost anywhere in all of this where you see one level of redirection used, understand that usually even greater obfuscation can be found by adding additional layers**



Anti-Emulation Tricks: Slow and Obscure

- ❖ A really interesting technique was presented by Nico Brulez in a Honeynet reverse engineering challenge – this involved making a virtual machine within the application itself, to obfuscate analysis... for which he wrote a virtual machine to interpret his own assembly
- ❖ Another interesting technique used in some malware in the past has been brute forcing the encryption algorithm used by the binary, a technique dubbed “RDA” or Random Decryption Algorithm (as documented by Peter Szor, as found in a virus sample – I could not find credit for the author of this method)
- ❖ Where you have only pure emulation and no static analysis in an heuristic engine, you then can be faced with processes that do nothing for very long periods of time but simply decrypt themselves... therefore, actually getting to the dangerous calls themselves can be found to be exceedingly difficult



系统级API Hook和启发式技术

- ◆ Hook技术和启发技术研究在这些年来基本上是同步发展的
- ◆ 从某种程度上说，hook并没有错，只是变了味道，它已经成为了恶意软件和检测技术的核心
- ◆ API保护类“防火墙”有许多类似启发式AV引擎的相同问题，本议题不详细深入这部分内容。
- ◆ 我确信能同时实现API保护和启发式引擎，在我们的IPS：**Blink**中就是这样做的，而且将继续下去——通过多层保护机制，就能确保虚假内容不起作用，并且同时对新旧未知攻击提供很高等级的保护——对相同类型攻击的保护



旁注：变形保护软件和系统完整性解决方案

- ◆ 这已经变得普遍了，如已经出现的木马Optix、Hacker Defender以及29a制作的一些，通过成为反-反病毒引擎来直接供给防御系统
- ◆ 这类引擎可能采用不同的攻击或检测特征编码来对付不同的AV、FW等解决办法
- ◆ 直接产生的问题就是：在系统上，先到者胜！
- ◆ 对于防护机制来说可以采取两种办法取胜：系统完整性引擎，比如基线系统或者anti-hooking检测采取的方法，……等（暂时还没想到其他的），反恶意软件引擎自身采用变形技术来避免被检测
- ◆ 另一方面，反-反恶意软件这样的代码本身就直接表明了它就是恶意软件



自动化智慧：善与恶的审判

- ❖ 对启发式保护系统最有兴趣的其中一点就是：努力给软件应用赋予智慧。
- ❖ 你是怎样判断“善”与“恶”的呢？
- ❖ 以道德哲学的论点来说，要判断什么是好，什么是坏，本身就有很多含糊不清的问题
- ❖ 正如前面所说：好与坏的判断通常就是主观的，这对于物理安全来说是个问题，对于启发式AV引擎来说也是一个问题



不明确的实例

- ❖ 几乎任何行为都很难判断是恶性的、良性的
- ❖ 启发式引擎必须定义什么是“犯罪”和“证据”
- ❖ 扫描，比如这可能是网络安全扫描器、或者可能是蠕虫，也可能是P2P产生的
- ❖ 操作密码文件、关键文件，既可能是恶意行为，也可能是正常的操作
- ❖ 又如：调试器通常会访问一些隐藏的API，间谍软件也会这样干



可疑迹象

- ◆ 应用程序产生某些不可能发生的行为
- ◆ 从某个观点来说“主观性”是客观现实，这个观点是本质上的，对于观察潜在的恶意软件来说这个观点应该受到重视：它是否应该具有这样那样的权限，或者本就不应该？
- ◆ 这并不是全新观点，但却是基本计算机安全的基础，处于不同的条件下才论安全，就象上面的观点间接表明的那样
- ◆ 好的检测恶意软件的原则就是判断“举证”的原则，比如人类道德，证据有不同的价值等级，并且多甚于少，在某个转换点，如果你明白什么是好什么是坏，那么你就能更准确地做出判断



基本木马实例

❖ 这里是一个典型基本木马的负面行为列表

- ∅ 设置自身在重启动时自动运行
- ∅ 建立隐蔽的通讯或者控制通道
- ∅ 能进行广泛的基本系统控制
- ∅ 试图收集密码或其他机密信息
- ∅ 开启一个端口
- ∅ 有隐蔽代码
- ∅ 有间谍代码、插入进程，插入窗口
- ∅ 有权限提升代码



结论

✦ 这里的多数结论，你应当已经得出了即便没有注意到。即使你不同意前面我提出的某些观点，你现在应当明白这些事情：

- ∅ 使基于签名的AV系统更聪明是必然趋势
- ∅ 变形恶意软件包括手动编辑的，这是真实存在，但是可战胜的问题
- ∅ 物理安全模型对于计算机安全模型来说很有用.....但通常被忽略了



结论，第二部分

- ◆ 打包/加密文件即使启发式的主要问题，也是恶意软件的主要缺陷，创造恶意软件的趋势会是彻底的和躲避性的打包/加密技术
- ◆ “启发式”是一种有错觉的术语，实际上启发式存在的问题在签名系统上也存在——启发式系统实际上是一个签名系统，一个更多动态签名的系统
- ◆ 基础的证据和怀疑对于启发式系统是至关重要的——这是误报错报系统和精确判断系统的技术区别关键点
- ◆ 我们必须注意在定义中出现的不确定界定，比如变形恶意软件应该定义那些手工改变的，并且启发式系统的不同模块，比如模拟模块或静态分析模块不应该定义包含了启发系统本身
- ◆ 最好的启发系统不应该试图做所有事情，而应该是整个完整安全系统得一部分.....这是避免误报的一种方法，不至于让某个模块或系统有太多压力
- ◆ 误报是任何系统都存在的BUG，并且降低了系统的功能.....它们并不应该被任何系统接受，但应该是一个需要考虑的可克服的问题



结论，第三部分

- ◇ Most mainstream AV has not been keeping up with attacks from the wild, when they have always had the capabilities to do far better
- ◇ Even in such a new industry, as computer security, tradition can quickly be created and kept and this must be fought against and avoided to produce results
- ◇ A solid evidence and suspicion based heuristics systems can operate, but it must be programmed well by heuristic signature coders... and it must not avoid basic dictates of the criminal justice world to operate, but must embrace them
- ◇ Defensive software must embrace offensive software and the reverse is true, otherwise there is a dangerous disconnect
- ◇ Many of the conclusions made in this paper have used common sense style arguments with readily observable evidence supporting them for the purposes of winning the ideas
- ◇ Code which is absolutely malicious in nature means code which is more provably “bad”, such as anti-anti-virus code, spying code, spreading code
- ◇ Various other conclusions, understanding about current heuristics and future heuristics... such as the usefulness of understanding “perspective” and “subjectivity” within a “moral system”, and viewing the heuristic system as a moral system... and the classification of heuristic protection possible, that being against “classes of attack” and “families of malware”



Various Credits

- ◆ Foremost, I must credit all of the malware writers out there who have greatly aided the advance of the science of the defensive arts, premier among these groups for full disclosure, open source type projects have been 29a... also the various contributors to the rootkit.com projects... it is, however, impossible to fully credit the works of all of these people... further it is also impossible to credit all of the malware researcher's works out there whose work has influenced this
- ◆ Peter Szor's book, mentioned in this speech, and his various other writings have been extremely helpful as a coherent cataloguing of the techniques used by malware authors -- I have marked out where his work has been useful herein
- ◆ Mark Russinovich's paper on AV engines and general low level papers have been extremely instrumental and educational in many of these areas over the years (I did not, however, base my NTFS system on his work, if anyone is wondering, but due credit goes there to the Linux NTFS team for their extensive documenting of the binary format of the NTFS file system.)



Credits, Continued

- ❖ Credits further mentioned, “CrazyLord’s” paper in Phrack on Physical Memory exploitation and PEiD’s Zero Order entropy method – their exact method used was found in their forums
- ❖ Nico Brulez’s Honeynet challenge was referenced in this work, especially regarding the usage of a virtual machine in terms of AV systems
- ❖ Thanks to Derek Soeder for his extensive help in proofreading this document
- ❖ Most of the conclusions I came to first through experimentation and prototyping, through work in this field, on both the offensive and defensive sides of things: many conclusions I later found supported elsewhere. My initial API analysis tool I wrote and released as shareware three years ago. My initial interest in heuristics dates back to some underground work some seven years ago or so.

