

软件加密进阶

今天提起智能卡加密已不再是什么新鲜话题了,应用了智能卡加密锁后的软件被破解的案例在智能卡加密锁的“不可破解”的光环的照耀下最终还是产生了。

当我们回顾历史,从DOS到windows,从第一代加密锁到今天的智能卡加密锁,加密锁厂商与破解者在长期的“交手”中都有了长足的技术进步。而真正需要掌握软件版权保护技术的大多数软件开发商们这些年在加密方面并没有太多的提高,这点完全可以从大多数网上见到的破解的文章中就可看出。

将结合加密锁进行保护而又可能被易于破解的加密方案简单划分成以下几类:

1. 软件关键点结合自定义算法,或将一部分关键点算法子程序移植到了锁中,但算法的变化过于简单。
2. 软件结合与锁信息相关的API(如种子码、user id、user 内存),然后在程序中各处关键函数埋藏加密点进行定值的比对。
3. 软件仅使用查找功能作为埋藏各处的加密点。
4. 少量的锁功能访问,使用结合一种纯软件的外壳加密工具进行保护。
5. 同时结合分类1、2、3和4。
6. 其它类别。

从以上这种分类方式来看,对于结合智能卡的加密锁来说被破解的情况主要就是集中在分类1。至于分类2、3、4在没有智能卡加密锁时也是有这种加密方案存在的。

对于以上方案可思考如下几个问题:

问题1:对于同一个软件,加密水平恒定的某一

个开发人员应用1至5几个方案的哪一个方案,对于同一个破解者(破解水平恒定)所需花费的周期最长。

问题2:同问题1,1至3中哪一个方案被破解的周期会最长。

对于问题1,答案是很显然的。一定是分类5,如果您在已结合了加密锁的方案上再结合上一种纯软件的保护技术一定是不会有什么副作用的。

对于问题2,可能大家会有些分歧,有人会说破解分类1所花的时间更长,有人会说破解分类1与分类2是一样的,因为这与被加密后的程序运行时的具体行为特征是有关的。

其实问题2确实很难回答,因为这里还有些条件并不明确,但如果可以使用进程外数据过滤拦截分析器(并且针对某款加密锁来说有这么个破解软件的话),并可利用回放数据方式进行破解,那么分类1与分类2根本就不会有什么差异,即破解分类1与分类2所花的时间可能是完全一样。

使用数据拦截方式进行破解,可以在不利用调试工具的前提下,只需按正常操作步骤在有加密锁的情况操作一次加密锁(只要能保证所有必要输入参数都能显现一次),则此后不插锁回放所记录的数据就可实现加密锁的破解。所以在这时分类1与分类2几乎是没什么区别的,只是回放数据的数据量上的差异。

数据拦截器大体上可分为两种:

1. “端口”级,在驱动或系统库的特定接口抓取数据。
2. API级,直接侵入某个要破解的进程中,找一些API的特征码,然后进行拦截。

虽然通过在硬件级对通讯数据进行加密可效地防止“端口”级的数据拦截，但由于开发商在调用API时的参数是不加密的，所以对于API级的数据拦截方式来说通讯层次的加密则根本不会有什么作用。

也许您对上面的逻辑关系还不是很清楚，简单说来就是一个破解者只要有一个针对某款加密锁的“端口”数据拦截分析器及一个API级的数据拦截分析器，那么几乎对于加密方案分类1到4都是攻无不破，而且不会花费什么气力。如果想真正发挥智能卡加密锁的作用，一定要输入与输出关系变化相对复杂的程序移植到加密锁中。

当然，一个“端口”数据拦截分析器再加一个API级的数据拦截分析器实现起来还是有些技术难度的，并且还需要非常了解某一款加密锁的输入及输出参数。并且API级的数据拦截分析器很容易就被一些纯软件的保护技术屏蔽。

但不排除任何一款加密锁在她的应用生命周期过长后，就会有相对的完美数据拦截分析器产生。

那么怎样才能有效地回避数据拦截分析器呢？可参考如下建议：

1. 算法的复杂，即输入与输出的变化范围要比较大，使得数据记录结果是一个天文数字。
2. 选用有硬件级实现通讯加密的加密锁。
3. 结合第三方纯软件加密技术与加密锁共用。
4. 定期了解所选用的加密锁是否已有通用的“端口”数据拦截分析器或API级的数据拦截分析器。
5. 自己动手学破解，查缺补漏。

建议1可能是也是众多开发商都非常了解的，但确实是一个很难平衡的问题，首先如果算法复杂了，软件的运行速度可能会变得非常慢。毕竟加密锁的运算速度不能等同于P4的CPU，同样的程序移植到加密锁后速度大多会与原来有很大的差异。

也可能是原来已经写好的程序，如果拆分，合

理选取一部分（算法中关键的多项式的一部分）放在加密锁中。理论上，这样就能很好的平衡速度与安全了，但实际上，大多开发商都不是数学或算法专家，很可能都是两天就要完成加这改方案，因此这条路也往往行不通。

建议2、3、4 开发商也只把自己的放置在一个非常被动的位置。如果买不到硬件级实现通讯加密的加密锁，如果第三方纯软件的加密接口也被找到了回绕方法了，开发商自身都没法改变什么。

建议5听起来无从下手，但却是我个人极力推荐的。

前面提到，如果有了一个结合了某一个款加密锁的数据拦截分析器（“端口”级或API级），将会使破解变得非常容易，完成这样一个程序应该至少不会少于500行代码，并且要花上很长的时间去开发及调试。

但是，如果已经知道那个数据拦截分析器进程名，这时我们只需不超过30行代码就可能令一个已知的数据拦截分析器失效。

下面我就介绍一下这种方法，当然如果破解者已知道了您是使用了以下我将要介绍方法，那么可能只要改一个字节就能回绕，也可能对于一些端口级的数据拦截器以下将要介绍的方法可能就根本就起什么作用。

不管怎样，毕竟加入这30行代码不花您什么气力就能打击一下刚开始学解密的人，令他中途就放弃，这不是我引入这30行代码的目的。我想真正编写一个好的难于破解的软件，要每周都学习这样30行代码，而且还要每年都不停地跟随主流破解技术的变化，这也许就是破解与加密的魅力所在…

简单说来就是只要您知道某个与数据拦截分析器工作时有关的进程名存在就成了，那么这种方法就还是可取的。如果数据拦截器是一个驱动则这段代码是完全不起作用的，但换另外30行代码还是可能

会起作用。叙述的原文引自《软件加密原理与应用》，因为几年前就写过了，对于下面的代码我就懒得再改动一行了，如果看明白了自然就应该能30行实现反数据拦截分析了：）。

反加载其实对反用户模式调试器、反内存补丁和反脱壳均有十分重要的意义。由用户模式调试器所加载运行的程序的父进程一般就是这个调试器。而在应用内存补丁时，也是通过一个小的Loader程序加载所要被补丁的程序，那么这个Loader也就成了被补丁的程序的父进程。脱壳程序也是用类似的方法加载所要脱壳的程序。在通常情况下，在开始菜单及资源管理器中启动的程序的父进程则一定是Explore.exe。

如果所要保护的程序能使用多种方法检测父进程是否为Explore，那么将是一种非常有效的反破解的方法。下面的ProcList函数的源码可在光盘上chap07\ProcList目录下找到。这个函数实现了列举系统中所有的进程，并显示所有进程的父进程及其本身的进程ID。示例是根据MSDN中与API CreateToolhelp32Snapshot（Tool Help函数）相关的示例Taking a Snapshot and Viewing Processes的源代码做了一些小的改动编写而成的。这段代码可以很好地工作于Windows 98，Windows 2000和Windows XP。

```

BOOL ProcList()
{
    HANDLE          hProcessSnap = NULL;
    BOOL            bRet         = FALSE;
    PROCESSENTRY32 pe32         = {0};

    hProcessSnap = CreateToolhelp32Snapshot
    (TH32CS_SNAPPROCESS, 0);
    if (hProcessSnap == INVALID_HANDLE_VALUE)
        return (TRUE);

```

```

    pe32.dwSize = sizeof(PROCESSENTRY32);

    if (Process32First(hProcessSnap, &pe32)) {
        do {
            printf("(%d-%d) %s\n", pe32.
            th32ParentProcessID,
            pe32.th32ProcessID, pe32.szExeFile);

        } while (Process32Next(hProcessSnap,
        &pe32));
        bRet = TRUE;
    }
    else
        bRet = FALSE;
    CloseHandle (hProcessSnap);
    return (bRet);
}

```

上面代码中的PROCESSENTRY32是一个包含了一个与进程相关的信息的结构体类型，其中的成员th32ParentProcessID记录的内容就是我们所要得到父进程的ID，而每个进程本身的ID则被存放在th32ProcessID这个成员中。szExeFile中存放的内容是每个进程的文件名，对于win9x系统这个文件名是包含全路径的名字，而对于winnt系列的系统则只包含可执行文件的名字（但对于系统核心进程来说这个名字就不等效于可执行文件的名字了）。

光盘上chap07\AntiLoader下面的示例实现了winnt下的反加载。主要原理是找到explorer.exe进程的ID并在程序中比较，当前进程的父进程是否为explorer，如果不是则说明可能被别的程序加载，并退出程序。不过由于szExeFile在windows9x下返回的内容是进程的全路径，所以程序无法工作于windows9x平台。但只要略加改进就可实现对两类平台的支持。