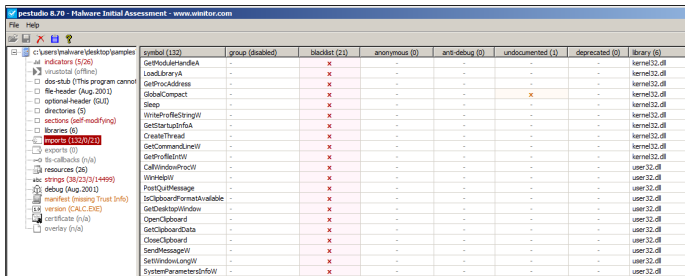


B. Imports

The executable imports 6 libraries in total, and in particular PESTudio marks as blacklisted 21 functions belonging to *Kernel32.dll* and *user32.dll*. The most interesting are:

- **getModuleHandleA**, to call external files, such as *dll* or *exe*;
- **loadLibraryA**, similar to that one above;
- **GetProcAddress**, to get the address of a procedure inside a library;
- **getStartupInfoA**, to get information about startup;
- **getCommandLineW**, to let the program execute code on the command line;

Moreover there are also imports of functions dealing with *Threads*, *Clipboard* and *Windows*. This suggests a suspicious behavior, due to the possibility of the program to access to all the libraries of the installed operating system. In the dynamic analysis we will see what actions are actually performed. The figure below contains all the blacklisted imports detected by PESTudio.



C. Strings

Regarding strings we notice that those in *.text*, *.data* and *.rsrc* sections are in clear; furthermore many of them are function calls or values used to modify system registers. Following them there are thousands of obfuscated strings, seemingly meaningless, which belong to the “*vpm0*” section. This is another clue of code obfuscation.

III. DYNAMIC ANALYSIS

The dynamic analysis consists of the execution of the malware in a controlled and isolated system to observe the behavior and the malicious actions performed.

To accomplish this task we used *VirtualBox*, a virtualization program that let us run a Windows 7 machine. VirtualBox handles the snapshots too, so that after every test the machine was restored to its initial state. The machine was also isolated from internet in order to avoid leaks.

The machine was equipped with various tools to monitor the execution of the malware. The main ones were:

- **regshot**, to detect files and registers alterations between a time lapse;
- **procmon**, to log system functions called by the malware;
- **fakenet**, to track internet traffic in a simulated network.

These programs were executed with administrator permission in order to detect access to restricted locations.

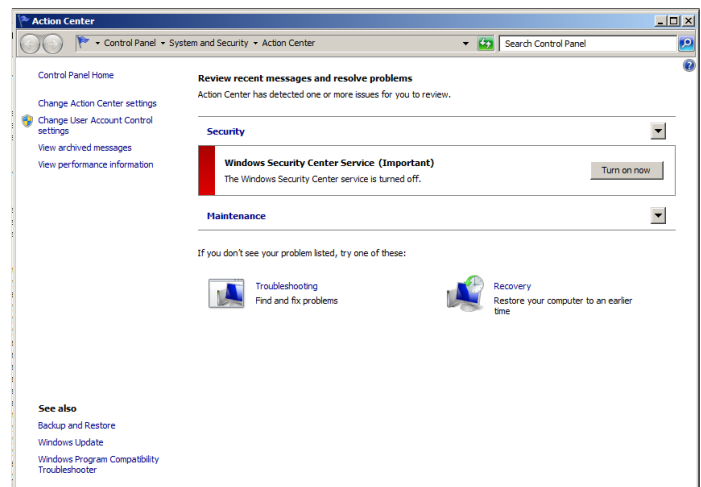
The tools can cooperate but they need to be executed in a certain schedule, in order to avoid conflicts among them. The analysis proceeded following these phases:

- 1) launch Fakenet;
- 2) launch and setup Procmon;
- 3) launch Regshot, setup path and run of its first shot;
- 4) start Procmon analysis and launch of the malware;
- 5) interaction with calculator by the GUI;
- 6) stop Procmon tracking
- 7) second Regshot shot;
- 8) stop Fakenet;

The first tests did not last too long: the GUI remained opened for just a few seconds. These tests did not give us evidence of malicious behavior, probably because of the poor performance of the host machine.

Then we ran other test leaving the malware run for a couple of minutes, leaving all the time needed to perform bad actions. Then we collect the logs and analyzed them one by one.

The first evidence of malicious behavior has been observed without any tools. After 20-30 seconds from the program execution the operating system prompted us a message about the *Windows security center*; it was disabled and, from the control panel, it could not be re-activated. This is a serious threat to the security of the machine, leaving it opened to many other infections.



A. Fakenet

FakeNet is a dynamic network analysis tool for malware analysts and penetration testers. It simulates a network logging the traffic generated by the machine. Every request is a file containing:

- Request type
- Destination URL
- Protocol
- User agent

The user agent field contains various informations about the local machine, such as architecture, operating system, product and so on. This string is

```
User-Agent: Mozilla/4.0
(compatible; MSIE 28;...
```

There are other numbers dealing with the versions that have not been written for simplicity.

The registered requests are over 900, whereas the destinations are 300, mainly Russian sites. Unluckily the body is empty, and we cannot know the expected result due to the presence of Fakenet, which builds a fake http page as response to every request.

Fakenet generated also a .pcap file, containing the captured packets. Examining it with *Wireshark* we noticed all the TCP streams related to the POST requests, and other packets irrelevant with the malware, such as DHCP, ARP and DNS. One frequent message was a request to *www.msftncs.com*, asking for a txt file called *ncsi*. This sort of message is generated by Windows as a heartbeat, keeping the machine constantly connected to the internet.

This whole internet traffic should not be present, due to the absence of any socket library imported by the executable. Thus, in order to send packets over the network the malware needs to load external libraries dynamically, leaving a trace on the system that we capture using *Procmon*.

B. Procmon

Process Monitor is an advanced monitoring tool for Windows that shows real-time file system, registry and process/thread activity. Procmon offers also a filtering function, and we set it to keep only the actions made by the malware.

We can group these actions in 3 main sets: dll, registry and file.

Dll stands for dynamic link library. These are particular files, shared among all the programs of the Windows operating system, that expose useful functions to programs which can call them to perform frequent operations. Every time that the malware loads a dll the system keeps a track of the action, thus using Procmon we were able to collect them all, identifying 46 different dlls used with the "Load Image" function. This primitive allows the program to load also other executables but, in our case, the only one loaded by the malware was the malware itself.

Overall we counted 46 dll files loaded, among which the most interesting are the cypher family, like *cryptbase* and *crypt32*, to handle cryptography and certificates.

In order to perform network operations the malware needs a library to handle internet connections. This is made possible

by the load of *ws2_32.dll*, which stands for web socket. This library provides the APIs (Application programming interface) needed to do POST request to all of the different URLs listed before.

Among the actions concerning with the registers Procmon logged many Open-Read-Close operations on many different ones, but only a few were written directly by the malware. These ones are about the "Language list" and "Internet settings". This last one manages the permission over the internet among the 4 different zones (Intranet, Trusted, Internet, Restricted), setting the value to 0 which means "Accept all". This is dangerous as exposes the machine to possible attacks over the net.

Procmon registered also accesses to thousands of file, and we noticed that many one of them were infected. The infection procedure followed these steps:

- Read the .exe victim file.
- Write of the content plus the infected part in a .vir file with the same name.
- Copy of the content of the .vir file to the .exe one changing the EOF location.
- Set of fake information on the executable such as creation and last access time.
- Delete the .vir file.

After the infection the victim had another section called *.vmp0*, having a similar size to the original one, which is both executable and writable. The infected files are hundreds and they are stored in many different locations. Many of them are common applications such as Windows Media Player, Internet Explorer, Windows Defender, Windows Mail, Windows Photo Viewer and other files in the System32 directory.

C. RegShot

Regshot is an open-source (LGPL) registry compare utility that allows you to quickly take a snapshot of your registry and then compare it with a second one - done after doing system changes or installing a new software product.

In our use-case we recorded the first shot just before the execution of the malware. Then we ran it for some minutes and, after the program exit, we took the second shot, saving the result of the comparison in a .txt file.

This file contains various information about registries files and folders. The downside of Regshot is that it takes care of all the system changes, but it cannot associate each change to the process that caused it, so there is no way to separate the malware generated events from the system ones. The only clue that we can use is a comparison to the logfile generated by Procmon.

Analyzing the output there are 6 new keys added to the registry, 4 related to error reporting, one to instrumentation and one to the Security Center. This last one is the most interesting because it is the trace left by the deactivation of the security center that we noticed during the malware execution. In fact among the values added to the registries there is "Enable Notifications" set to zero. Then we found all the changes reported by Procmon about the internet zones, where every value is set to 0 (Allow anything).

Another malicious evidence that we found is that the value “Start” of the *wscntc* service set to 4. This service is the security center, and the value 4 stands for disabled.

The weird thing is that Procmon did not log those events, so they are not made by the malware itself, but by another process. In order to validate this hypothesis we ran Procmon and executed the sample another time, keeping even the action performed by other processes. We discovered that those registry modifications had been made by “services.exe” and “dllhost.exe”. They are not malicious process, but they are likely to be corrupted by the malware to deactivate the security center.

The file attributes that are modified are those related to the infected files and thus they have a different creation time.

IV. REVERSE ENGINEERING

In order to analyze the source code of the malware we used a tool called IDA, which is a feature rich, cross-platform, multi-processor disassembler and debugger developed by Hex-Rays.

Unluckily our version of IDA could analyze the main procedure only until the address *0x101264D*. From that point on it can disassemble the instruction but it can not build the control flow graph of the program. To supply to this lack we choose to change tool in favour of *Ghidra*. It is a software reverse engineering (SRE) suite of tools developed by NSA’s Research Directorate in support of the Cybersecurity mission, that recently became open source.

The whole process of reverse engineering is very difficult due to the presence of *VMProtect* which obfuscates the core of the code.