

# Open-source Security Solution for False Antiviruses Removing

C. Pop, A. Naaji, and M. Popescu

**Abstract**—One of the goals of existing strategies in companies and organizations regarding the security of information systems is to protect computer networks against attacks caused by viruses. For detection, and if necessary, removing computer viruses, specialized companies are increasingly concerned with the development of advanced antivirus products. However, although there have been made efforts in this regard, in recent years a new threat appeared, related to false antivirus programs for which there are still insufficient protection tools. The purpose of our software is to improve existing products in dealing with false antiviruses or false alerts, which may pose serious threats to computer safety. Once infected, these programs block access to the system, overwhelming the antivirus software. The application is created based on the fact that the same false antivirus makes the same files, the file is encrypted and sent again “into the wild” as 0-Day Malware file. We developed a open-source program that assists the existing security solution in cleaning the computer from false antiviruses or false alerts and provides an option to help the user with an alternative, combining commands from the terminal (*command prompt*) with a *console application* type, which means it can be run directly from the console.

**Keywords**—Antivirus software, false antivirus, security solutions, rogue, fake antivirus, fake alert.

## I. INTRODUCTION

ENSURING the security of information systems is one of the most important present challenges. Thus, most companies and organizations are concerned with development of administrative strategies and operational plans related to this issue [1]. In this respect, one of the major preoccupations is to protect computer systems against viruses.

In computing, the term “virus” is applied to various software applications, which can be found in the literature under the generic name of malware [2]. Computer users employ the term “virus” instead of “malware”, as it is a much better-known term. False antiviruses or *rogues* (the name is derived from the fact that false antiviruses appear and then shortly disappear), belong to Trojan category [3]. Generally, false antiviruses have a lifespan that ranges from a few hours

to one week. False antiviruses are classified into two categories: *FakeAV* and *Fake Alert*.

*FakeAV* antiviruses are copies of legitimate security programs or some other programs that seem to be security programs. They can disguise themselves as: antiviruses, anti-spyware, anti-adware, and, in some cases, even as computer maintenance programs [4].

One method for obtaining false antiviruses is by copying legitimate antiviruses programs from their official sites, using some key elements (logos, statistics etc.). In order to attract visitors, these sites usually post video guides, presenting the efficiency, purchasing methods, as well as free technical support.

*FakeAlert* applications are alerts warning computer users that their PC is infected. A recurrent method consists of creating a false *Security Center* and then warning the user that the antivirus is not activated or not installed [5]. Another one is to prompt messages in the tray area, warning of security issues. Recently, a new method appeared, displaying a window that warns the user that there is a security issue and that a PC scan is recommended.

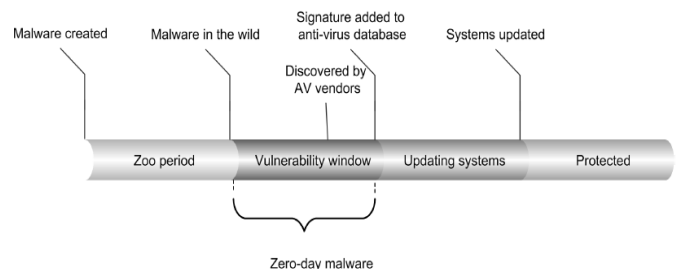
*0-Day Malware* is a virus that manages to infect the computer in the absence of signatures or other detection methods featured in the antivirus. The lifetime of such a virus is shown in Fig.1 [2], where:

*Zoo period* is the period that has elapsed since the virus was created and up until it was launched into the *wild*;

*Vulnerability window* is the period since the virus was spread on the Internet and up until a signature was created;

*Updating systems* is the period after signatures were uploaded and installed;

*Protected* is the period subsequent to the deletion of the virus.



C. Pop is with “Vasile Goldis” Western University, 310025 Arad, Romania (e-mail: pop\_catalin88@yahoo.com).

A. Naaji is with Department of Computer Science, “Vasile Goldis” Western University, 310025 Arad, Romania (+40-257-214505; e-mail: anaaji@uvvg.ro).

C. Popescu was with Department of Computer Science, “Vasile Goldis” Western University, 310025 Arad, Romania. (e-mail: popescu.marius.c@gmail.com).

Fig. 1 lifetime of a virus from the perspective of a malware research

As regards infection methods, the most common of them is by downloading the executable file. In time, these have evolved from using *FakeCodecs* (required for viewing video files), to *toolbars* form, to the current *exploits* or *IFrames*.

On the sites of false antiviruses, there are usually certain prizes to attract users, as well as certificates attesting the rate of confidence for that product.

Currently, the most widely used infection methods are:

- hijacking official antivirus pages and redirecting them towards infected ones;
- false FlashPlayers, Windows or even codec updates;
- using exploits;
- using other programs or Play-Per-Install (PPI).

An example for how a false antivirus acts, is the one that uses the Search Engine Optimization Poisoning technique [6]. The actions of the false antivirus consist of the following steps:

- User looks for news on the incident using a popular search engine;
- User clicks a malicious link that connects to a malicious site;
- User is redirected several times, which ends with the download of *troj\_fakeav.smry*;
- User sees bogus alert of nonexistent system infection.

Considering that an antivirus with classic signature detection cannot find *0-Day Malware* on time, we have created an open-source application with a *General Public License* (GPL), so that anyone could contribute to its development.

Some of the existing possibilities for false antiviruses analysis are presented in section II. The section III describes the structure and the features of our software. Section IV presents the techniques we want to implement as future developments for obtaining a competitive and flexible security tool.

## II. EXISTING TOOLS FOR FALSE ANTIVIRUSES ANALYSIS

Before creating a security solution for false antiviruses, it is necessary to highlight some existing tools used in various situations, as well as their analysis methods [7].

The research was conducted in a laboratory consisted of two computers: one virtual and the other dedicated.

The virtual computer was implemented using the *Oracle VM VirtualBox* application [8], available both on *Windows* and *Linux* operating system.

The research performed in this paper uses the *Windows XP* operating system, with no installed *updates*. After installing *Windows XP*, the virtual hard drive is copied onto *VirtualBox*, to avoid re-installing the entire system.

The dedicated computer is only used for tests, as a second option, since nowadays most viruses cannot run on virtual computers. In this case we need a platform for analysis. As in the case of the virtual computer, the operating system is installed only once, then a *backup* application is used. The resulting file is copied to an external storage device (in our case a *USB* flash drive).

In domain of information security, the computer system that is expressly set up to attract programs and/or attacks is named *honeypot* [5].

The tools we use are free or *open-source*, and depend on the type of analyzed *malware*. In the case of false antiviruses, the chances for a computer to be usable are minimal. In this case we need an *Ultimate Boot CD* (UBCD) [9]. It contains a wide range of applications, arranged into categories, running from a CD-ROM and requires approximately 100 Mb. Upon request, UBCD offers a customizing possibility, by adding or removing applications. If the computer starts, then the analysis mode is more effective, as the behavior may be observed in real time. The tools used for this analysis are:

- *RegShot* [10], an open-source program, with the possibility to create *snapshots* and to compare registries; this helps observe the changes occurring in the computer.
- *Autoruns* [11], a free utility that can view and change programs that start on computer startup.
- *ProcMon (Process Monitor)* [12], a free utility that provides real-time information on active processes.
- *Rootkit Revealer* [13], a free program providing advanced *rootkit* detection.
- *GEMER* [14], a free program allowing *rootkit* search in key parts of the operating system; this program is more advanced than *Rootkit Revealer*.

The analysis mode is always the same, regardless of the computer the analysis is run on. Thus, the following steps are needed:

- creating a backup of registries, which are saved on an external drive;
- creating several administrator accounts;
- running the file to be analyzed;
- restarting the computer.

If the computer does not start, *UBCD* is used to make a copy of registries, which will be compared to the first copy, using *RegShot*. Then the log is analyzed to establish why the PC does not start (in most cases, the *winlogon.exe*, *userinit.exe* files are copied and the computer starts). From the report provided by the *RegShot* tool, we may already add a few files from the list of infected files, which are to be deleted. *OpenRogueRemoval* may also delete some rootkits, which are best detected if specialized software like *GEMER* and *Rootkit Revealer* are used. Another way to detect rootkits is by using a *Live CD* with a *Linux*-type distribution, but this method requires advanced computing knowledge. For a more detailed analysis, to check whether other files have remained, we use *autoruns* and *procmon*. The files we find are entered to the list of infected files [15].

## III. APPLICATION PRESENTATION

False antiviruses make the same files from one version to another; the only difference is, in most cases, the name. This was first observed in the case of *Zbot* (a very dangerous botnet).

The application has four components [16]: *scan*, *view startup*, *view active processes* and *check system files*. The program is written in the C++ programming language and designed for use on the following operating systems: *Windows XP*, *Windows Vista* and *Windows 7*.

1. *Scan*. Upon scanning, the user has the option of scanning a default file or creating a new file. That file is recommended to be of the text type (\*.txt), as it is easier to use. The user may enter the destinations of files he wishes to delete into a file; it is recommended for deletion to occur in *safe mode* with *command prompt*, because, if the file is active, then normal deletion is not recommended and, in some cases, not possible (it runs in the memory, or is injected into various processes – a very good example is the *explorer.exe* application).

2. *View startup*. To view programs on computer startup, instead of the direct registry interrogation method, a *Windows* command is used:

```
wmic startup get caption,  
command>StartupLog.txt
```

Using the previous command, logs of different formats may be generated (in this case of the *.txt* type) only by changing the extension *Startup.\*(txt, doc, html, etc.)*.

3. *View active processes*. This option, an alternative to the *task manager*, intended for *command prompt*, lists active processes, their name and *Process ID* (PID), having the purpose to access processes, even if the computer is infected. In many cases, after the computer has been cleaned, the *task manager*, as well as *regedit* and *msconfig*, are not accessible because the *malware* restricted their access, and this option is an alternative.

4. *Check system files*. This option requires the operating system installation CD, it is recommended to be used only if the computer is greatly infected and only after disinfection, or in cases where the operating system has problems starting. This option uses a *Windows* command and is compatible with *Windows XP*, *Windows Vista* and *Windows 7*. It also entails a disadvantage for users with *Windows updates on*: if *updates* change viral files (in the system), with this option those files will be replaced, and downloaded on the next *update*. As in any program written in C++, the libraries we wish to use are imported. The application uses the following libraries [17]:

```
#include<stdio.h>  
#include<stdlib.h>  
#include<conio.h>  
#include<iostream>  
#include<windows.h>  
#include<string>  
#include<tlhelp32.h>  
#include<fstream>  
#include "resource.h"  
using namespace std;  
int remove(const char *pathname);  
#define IDI_ICON 101
```

A *resource.rc* file must be created; this is a file called *resource file* and is used to create icons, version, etc. In our

case, the *resource file* must include *resource.h* and be attached to the application. The file must also contain the name of the icon file (name.ico), and its size must be 32 x 32 pixels:

```
resource.rc: #include "resource.h"  
IDI_ICON ICON DISCARDABLE "icon.ico"
```

*IDI\_ICON* is an identifier, *ICON* is a resource, *DISCARDABLE* is an attribute, in case of error using the previous icon, "icon.ico" is the name of the icon. The icon must be in the working directory in order to be used.

*The menu of the application* contains five components, the last one being the exit option. The menu, as well as all options, is built so that, after each operation, the program is closed. For using a new option, or even the same one, we must re-launch the program. This option was added because the program is designed to also run on *UBCD* or on similar CDs. Since these CDs run in *RAM*, after each program or option run on that CD, we must run a program entitled *free mem*, which frees memory. Efforts were made so that, after each option, the memory would not be too loaded, and to offer the possibility to run this program without difficulty. It is recommended to run this program after scanning with *OpenRogueRemoval*.

The structure of the menu is:

```
void optionOne();  
void optionTwo();  
void optionThree();  
void optionFour();  
int main()  
{ Menu - GUI }
```

The *main* section only contains the interface and the selection options.

```
void optionOne()  
{ Scanning engine }
```

Option one, or the *scan mode*, contains the module by which files are deleted.

```
void optionTwo()  
{ Startup }
```

Option two *lists and ends active processes*.

```
void optionThree()  
{ Active processes }
```

Option three lists *startup programs* and writes them into a *log file*.

```
void optionFour()  
{ SFC - System File Checker }
```

Option four *checks system files*, but it requires the operating system installation CD. The interface is drawn in the *main* section, using only the *cout* and */n (new line)*, */t (tabulator)*

instructions. Before drawing the interface, the *system* ("cls") command is used, where *cls* is a command used in the *command prompt*, to clear the screen. For adding a name instead of the file destination, in our case "OpenRogueRemoval – Utility for removing false antiviruses", the "title" command was used. The interface being created, the selection option is added - first, adding it before *system*("cls").

```
char sel;
do
{ system("cls");
system("title OpenRogueRemoval - Utility for
removing false anti-viruses");
Continue interface ...
```

The selection option *Sel* is the character which the user enters to select the desired option, using the *switch* instruction. The user must enter a number between 1 and 5 to select an option.

```
Interface ...
cout <<"\tSelect desired option:";
sel=getch();
switch(sel)
{
case '1': optionOne(); break;
case '2': optionTwo(); break;
case '3': optionThree(); break;
case '4':
optionFour();
break;
case '5':
exit(1);
}
}
while(true);
```

In the next paragraphs we will describe the components of the menu.

#### A. Scan

The working principle of the scanning engine is to extract the information from a file and then to delete the file. For *OpenRogueRemoval* to delete this *malware*, the engine must read the entire line, regardless of the characters it contains, and to delete all. For spaces or special characters, *optionOne* is created after the *main* section.

```
void optionOne()
{
system("cls");
string lineRead="";
char fileCondemned[100];
char nameFileRules[24];
ifstream f(nameFileRules);
```

After defining all variables, the possibility to enter the file from the keyboard is created. The entering form will be *name.extension* (for example *rules.txt*).

```
cout <<"Enter file name + extension: ";
cin >>nameFileRules;
```

If the file is opened successfully, we use *getline()* [17] from the specified file (entered from the keyboard).

```
f.open(nameFileRules, ios::in);
if (f.is_open())
{
while(f.good())
{
getline (f,lineRead);
```

The file is browsed through, *chars* are transformed into *strings*, so that the file can be deleted; the *remove()* function [18], from the *stdlib.h* library only accept *strings*.

```
for (int i=0; i<=lineRead.length();i++)
{
fileCondemned[i]=lineRead[i];
}
```

In order to use the *remove()* function from the *stdio.h* library, immediately after *using namespace std*, one must add:

```
int remove(const char *pathname);
```

If, on deleting the file, an error occurred, then the message "deletion was not possible" will be displayed.

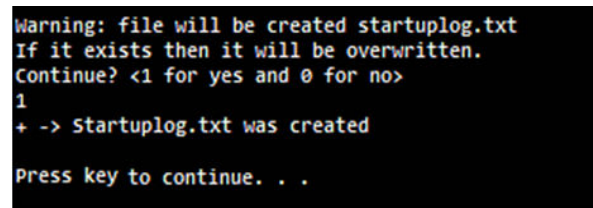
```
if(remove(fileCondemned) == -1)
{
cout <<"Deletion of file"<<fileCondamned
<<"was not possible"<<endl;
}
```

The successful deletion message is displayed, after *system* ("pause") is entered, for the user to be able to view in detail if the files were deleted successfully or not.

```
cout <<"Deletion of file"<<fisierCondamnat
<<"was successful"<<endl;
}
system("pause");
f.close();
}
else
cout<<"Error opening the file:
"<<f<<endl;
}
```

#### B. Startup

This module is merely informative and creates a *log* file, in which it lists all *startup* entries (Fig. 2).



```
Warning: file will be created startuplog.txt
If it exists then it will be overwritten.
Continue? <1 for yes and 0 for no>
1
+ -> Startuplog.txt was created
Press key to continue. . .
```

Fig. 2 the startup interface

The purpose of this module is to offer an alternative to existing products. In case the computer is infected with a false antivirus and blocks access to viewing startup entries, this module offers the possibility to view using *safe mode* with *command prompt*, by creating optionTwo() from the menu.

```
void optionTwo()
{
    system("cls");
    int startupselect;
    cout<<"Warning: the Startuplog.txt file
will be created. If it already exists, it will
be overwritten. Continue? (1 for yes 0 for
no)\n";
    cin>>startupselect;
    cout<<"+-->";
}
```

The user has to choose between two variants: if the file exists, overwrite it, if not, create it.

```
if (startupselect == 1)
{
```

```
        system("wmic startup get
caption,command > StartupLog.txt");
        cout<<"StartupLog.txt has been
creat\n";
        system("pause");
    }
    exit(1);
}
```

The *wmic* command can create *log* files not only in the current directory, for example:

```
wmic startup get caption,command>
C:\StartupLog.txt
```

In this case, the *log* file is created in C:\. The reason why the *log* file was chosen to be created in the same place as the application is that the file can be created regardless of the device from which the program is run (USB flash drive, external hard drive), and to avoid incompatibilities. This module requires administrator rights in order to generate and create the file. An example of generated file (on a clean computer) is:

<i>Caption</i>	<i>Command</i>
Sidebar	%ProgramFiles%\Windows Sidebar\Sidebar.exe /autoRun
Sidebar	%ProgramFiles%\Windows Sidebar\Sidebar.exe /autoRun
DAEMON Tools Lite	"C:\Program Files\DAEMON Tools Lite\DTLite.exe" -autorun
ctfmon.exe	C:\WINDOWS\system32\ctfmon.exe
Malwarebytes'Anti-Malware(reboot)	"C:\ProgramFiles\Malwarebytes'Anti-Malware\mbam.exe" /runcleanupscript
COMODO Internet Security	"C:\Program Files\COMODO\COMODO Internet Security\cfp.exe" -h
SunJavaUpdateSched	"C:\Program Files\Common Files\Java\Java Update\jusched.exe"

where:

- Caption is the name of the application and
- Command is destination and parameters (for example - h is hide, run in hidden mode).

Unlike *msconfig*, this command also displays the *Sidebar*. In this case it is displayed twice because the computer has two users.

### C. Active Processes

This module offers the possibility to view and end a process. It does not work if running from an antivirus CD or similar. This option can be used on an infected computer by creating beforehand two or three administrator accounts. In *Windows XP*, the following procedure must be completed: right click on *OpenRogueRemoval* → *Run as* → enter the name and password of an administrator account (different from the one that is logged in). The *OpenRogueRemoval* application will display the active processes. For this option we created optionThree().

```
void optionThree()
{
```

```
    system("cls");
    int processelect;
    int pidselect;

    SetConsoleTextAttribute(GetStdHandle(STD_
OUTPUT_HANDLE), 15);
    cout<<endl<<"List of Activ Processes"
<<endl;
```

This interface only uses colors 15 (white) and 8 (grey).

```
HANDLE WINAPI CreateToolhelp32Snapshot(
    DWORD dwFlags,
    DWORD th32ProcessID );
```

The handler *CreateToolhelp32Snapshot* [19] is used to make a *read-only* copy of objects from the system memory.

```
HANDLE
hSnapShot=CreateToolhelp32Snapshot(TH32CS_SNAP
PROCESS, 0);
    BOOL WINAPI Process32Next(
        HANDLE hSnapshot,
        LPPROCESSENTRY32 lppe);
```

*Process32Next* takes over the information from the next process [20] and *hSnapShot* is a *handler* on the *snapshot*.

*TH32CS\_SNAPPROCESS* includes the list of processes in the *snapshot* and *lppe* is a pointer to the structure *PROCESSENTRY32*. The function returns *true* if the data on the first process have been copied into the *buffer*.

```
PROCESSENTRY32* processInfo=new
PROCESSENTRY32;
    processInfo->dwSize=sizeof
    (PROCESSENTRY32);

while(Process32Next(hSnapShot,processInfo)!=
FALSE)
{
    SetConsoleTextAttribute(GetStdHandle(STD_O
UTPUT_HANDLE), 8);
    cout<<endl<<"=====
===== ";
    SetConsoleTextAttribute(GetStdHandle(STD_O
UTPUT_HANDLE), 15);
    cout<<endl<<"+-->      Name:"<<processInfo->
szExeFile;
    cout<<endl<<"+-->      PID:"<<processInfo->
th32ProcessID;
}

```

*ProcessInfo* is the member of the *dwSize=sizeof (PROCESSENTRY32)* structure which provides information on processes and browses through to the last process. Then, the name and PID are listed (*szExeFile* and *th32ProcessID* are used). The listing is composed of all processes running at that moment, as it is only a “copy” of processes. It is not done in real time and if, for example, a process was ended after this option has been interrogated, it will be listed as it being there. The possibility to end a process was also implemented. Ending is made according to the PID of the process, listing in this module being identical to that in the *Task Manager*.

```
SetConsoleTextAttribute(GetStdHandle(STD_OUT
PUT_HANDLE), 8);
    cout<<endl<<"=====
===== ";
    SetConsoleTextAttribute(GetStdHandle(STD_O
UTPUT_HANDLE), 15);
    cout<<endl;
    cout<<"[ 1 ] End a process"<<endl;
    cout<<"[ 0 ] Exit"<<endl<<"+--> ";
    cin>>processselect;
    if (processselect==1)
    {
        cout<<"End the process with the PID \n";
        cout<<"==> ";
        cin>>pidselect;
    }

```

To end this process, the *process* handler is used to “open” the second process (in order to end the first one), and close the *handler*.

```
HANDLE process = OpenProcess(PROCESS_
TERMINATE, FALSE, pidselect);
    TerminateProcess(process,0);
    CloseHandle(process);
    cout<<"[!] The Process "<<pidselect
<<"has been ended"<<endl;

```

}

If the user only wishes to view the processes, without ending them, an exit option has been introduced as well.

```
if (processselect==0)
{
    exit(1);
}
system("pause");
exit(1);
}

```

Fig. 3 present the interface for ending the processes according to their PID.

```
+ -> PID: 2640
+ -> Name: thunderbird.exe
+ -> PID: 3100
+ -> Name: mspaint.exe
+ -> PID: 3452
+ -> Name: SearchProtocolHost.exe
+ -> PID: 3096
+ -> Name: SearchFilterHost.exe
+ -> PID: 3568
+ -> Name: OpenRogueRemoval.exe
+ -> PID: 2688
+ -> Name: conhost.exe
+ -> PID: 2152
[1] Finish a process
[0] Exit
+ -> 1
Finish the process with PID
==> 2688

```

Fig. 3 ending the process according to PID

#### D. Check System Files Section

This option, in order to be functional on *Windows XP*, *Windows Vista* and *Windows 7*, uses the command *sfc*, provided by *Microsoft*.

The option has three parameters:

- /scanow - scans all system files immediately;
- /scanonce - scans all system files once;
- /scanboot - scans all system files each time the computer is restarted.

All parameters require administrator rights.

Fig. 4 present the interface with the message requesting administrator rights for using *sfc* utility.

```
Enter the filename + extension: 1.txt
Delete the file C:\Documents and Settings\All Users\Start Menu\Antivirus 2011 AVG\AVG Antivirus
2011.ink was successful
Delete the file C:\Documents and Settings\All Users\Start Menu\Antivirus 2011 AVG\Uninstall.ink
was successful
Delete the file C:\Program Files\AVG Antivirus 2011\avg.exe was successful
Delete the file C:\Windows\system32\iesafemode.exe was successful
Press key to continue. . .

```

Fig. 4 example of using *sfc* utility (without permission)

In the next section we used the `/scannow` parameter, preceded by a pause.

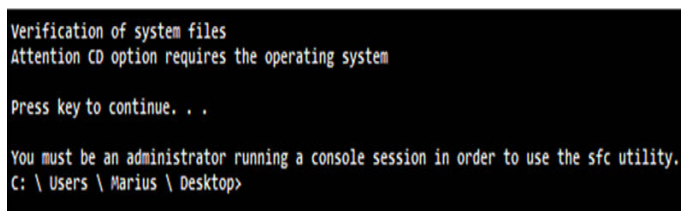
```
void optionFour()
{
    system("cls");
    cout <<"Checking system files \n\n"
    <<endl;
    cout <<"Warning: This option requires the
    operating system installation CD
    \n\n"<<endl;
    system("pause");
    system("sfc//scannow");
    exit(1);
}
```

In the following example, we can see how the locations of false antivirus AVG 2011 were introduced. The specific file is *FakeAV\_AVG2011.txt*.

```
C:\Documents and Settings\All Users\Start
Menu\AVG Antivirus 2011\AVG Antivirus
2011.lnk
C:\Documents and Settings\All Users\Start
Menu\AVG Antivirus 2011\Uninstall.lnk
C:\Program Files\AVG Antivirus 2011\avg.exe
C:\WINDOWS\system32\iesafemode.exe
```

In case the user is infected with the false antivirus AVG Anti-Virus 2011, both on *Windows XP* (in this case), and on *Windows Vista* or *Windows 7*, the destinations may be added into a single file. The only difference is the username in the destination. In this case, for uninstalling the fake antivirus using *OpenRogueRemoval* we have to enter in *safe mode* because the malware is injected into the *explorer.exe* file.

Fig. 5 presents the interface corresponding to removing the false antivirus AVG 2011, using our software.



```
Verification of system files
Attention CD option requires the operating system

Press key to continue. . .

You must be an administrator running a console session in order to use the sfc utility.
C: \ Users \ Marius \ Desktop>
```

Fig. 5 removing the false antivirus AVG 2011, using *OpenRogueRemoval*

#### IV. FUTURE DEVELOPMENT DIRECTIONS

The software for detecting and removing false antiviruses could be improved by adding some facilities, such as:

- Changing the program so that the user does not have to open the rule file and enabling him to enter the username or administrator name himself (the program would detect it automatically).

- Adding the option to run in real time, adding in startup and minimizing in the *tray* or starting with the `-h` attribute and, if it

finds a file that is in the list, warn the user with the message "Found potential virus".

- Adding a new option for checking whether the *hostfile* has been modified, and if it has, then display a message to the user. The *hostfile* restricts access to some sites, and viruses change this file and add to it the addresses of security products. This way, the "victim" cannot download a program to disinfect the computer.

- Adding the *open-source ClamAV* antivirus, with the specification that the entire interface and all options will be modified so that they may run in *safe mode* with *command prompt* (the menu will feature another scanning portion with *ClamAV*).

#### V. CONCLUSIONS

The application is intended for users with advanced knowledge in computing, providing the possibility to create rules for different false antiviruses. In this paper we presented a single example (AVG Antivirus 2011) of a list for a false antivirus. These lists, intended for one type of false antivirus, may contain different versions of the same antivirus (in most cases, viral components are the same, except for the targeted user and operating system). The user must only change the username, or create a user with administrator rights.

As regards the *startup*, it provides more information than the *msconfig* file and has the advantage that it can be used without an interface (from the *command prompt*).

Upon listing processes, the application offers a great advantage, as viruses disable *msconfig*, *task manager* and *regedit*, and if the computer has another user with administrator rights, this second user can run this program and end that process.

Checking protected files (system files or viral components), in case a virus makes changes or is injected, is a pretty good solution.

An antivirus with the classic signature detection cannot detect this malware on time. This program is provided as an alternative. The fact that the program is *open-source* with a *General Public License*, so anyone can contribute both rule files and improvements, is very important. Another advantage of this application is that it can run both in *safe mode* and from an anti-virus CD or similar. This was the starting point in creating the false antivirus software.



This software could be added to a security product in the form of a module, and on each *update* of classical definitions. It would also *update* these rules. The advantage is that this module will require very little memory (maximum 1 Mb) and it is the “*heuristic*” variant of that security solution. The application does not have false alarms (*False Positive*). The application should be extended for other viruses that have the same behavior and the same files, from one version to another. It would be added to a security solution as a “*cloud*” module; when finding a file from the list, it would send it to a laboratory, to analyze it together with all the files with it interacts.

#### REFERENCES

- [1] J. A. Ruiz-Vanoye, O. Díaz-Parra, I. R. Ponce-Meddelin, and J.C. Olivares-Rojas, “Strategic planning for the computer science security”, *WSEAS Transactions on Computers*, issue 5, vol. 7, pp.387-396, May 2008.
- [2] H.M. Halvorsen, “0-Day malware”, Project of the Norwegian University of Science and Technology, 2008, pp. 28-35.
- [3] P. Wang, X.C.You, and B.X. Fang, A, “A large network malicious code detection system:VDS”, *WSEAS Transaction on Information Science and Applications*, issue 4, vol .1, pp. 994-1003, October 2004.
- [4] P.Zsor, *The Art of Computer Virus Research*, Boston: Addison Wesley Professional, 2005, pp. 36, 434.
- [5] A. Sanabria, “Malware analysis environment design and architecture”, SANS Institute, USA, 2007.
- [6] Detcraig. (2011, June 26). Cross-Border Korean Shelling Searches Lead to FAKEAV. TrendMicro, Available: <http://community.trendmicro.com/t5/Web-Threat-Spotlight/Cross-Border-Korean-Shelling-Searches-Lead-to-FAKEAV/ba-p/20928>.
- [7] H. M. El-Bakry, and N. Mastorakis, “Fast virus detection by using high speed time delay neural networks”, *Proceedings of the 10th WSEAS International Conference on Neural Networks*, pp.169-183, 2009.
- [8] Oracle VM VirtualBox. Available: <http://www.virtualbox.org/wiki/Downloads>.
- [9] Ultimate Boot CD. Available: <http://www.ultimatebootcd.com/>.
- [10] RegShot. Available: <http://sourceforge.net/projects/regshot/>.
- [11] M. Russinovich, and B. Cogswell (2011, June 20). Autoruns for Windows. Available: <http://technet.microsoft.com/en-us/sysinternals/bb963902>.
- [12] M. Russinovich, and B. Cogswell (2011, June 20). Process Monitor. Available: <http://technet.microsoft.com/en-us/sysinternals/bb896645>.
- [13] B. Cogswell, and M. Russinovich (2006, November 1). RootkitRevealer. Available: <http://technet.microsoft.com/en-us/sysinternals/bb897445>.
- [14] GMER. Available: <http://www.gmer.net/>.
- [15] S.P. Correll, “Business of Rogueware”, PandaLabs Research, Spain, 2009.
- [16] C. Pop, M. Popescu M., and A. Naaji, “Security Solution for False Antivirus Detection”, *Recent Researches in Communications and Computers*, pp. 227-231, July 2012.
- [17] *String library. Getline Function*. Available: <http://www.cplusplus.com/reference/string/getline/>.
- [18] C++ library. Remove function. Available: <http://www.cplusplus.com/reference/clipboard/cstdio/remove/>.
- [19] *CreateToolhelp32Snapshot*. Available: <http://msdn.microsoft.com/en-us/library/ms682489%28v=vs.85%29.aspx>
- [20] *Process32Next*. Available: <http://msdn.microsoft.com/en-us/library/ms684836%28VS.85%29.aspx>.