NTNU

Innovation and Creativity

PowerScan user guide

Thomas Langerud and Jøran Vagnby Lillesand

Master thesis
Spring 2008

Norwegian University of Science and Technology
Faculty of Information Technology, Mathematics and Electrical Engineering
Department of Telematics

Contents

1 User Guide **5**

1.1 Requirements 5
1.1.1 Client 5
1.1.2 Virtualization servers 6
1.1.3 Usage 6
1.2 Environment Setup 6
1.3 User Interface 8
1.3.1 Graphical user interface 10
1.3.2 Command line usage 11
1.4 Malware sample scan 12
1.5 Malware sample execution 13
1.6 Malware sample analysis 14
1.7 Update AV definition files 15
1.8 Adding new anti-virus engines or tools 16
1.9 Saving console output 16
1.10 Editing the XML configuration file 16
1.10.1 The “File” menu 18
1.10.2 The “View” menu 18
1.10.3 The “Delete” menu 18
1.10.4 The “Host/VM” view 20
1.10.5 The “AVE” view 21
1.10.6 The “Tools” view 23
1.11 Understanding the XML configuration file 24
1.12 Redirection of console output 29
1.13 Understanding the properties file 29
1.14 PowerScan files 31
1.15 Understanding the log files 32

2 Configuration files **35**
2.1 Example XML config file 35
2.2 PowerScan XML Schema Definition (XSD) 38
2.3 Description of PowerScan's XML with respect to the XSD schema 41
2.4 Properties file example 45

Chapter 1

User Guide

This user guide describes how to set up and use the PowerScan analysis framework and is intended to be as detailed as necessary to enable a user to utilize the capabilities of the framework without knowledge of the source code. The XML configuration and properties files are described in detail.

1.1 Requirements

To be able to use the PowerScan framework, there are some requirements that will have to be fulfilled. The following are the system requirements for the different components.

1.1.1 Client

The client is the computer that is to be used to run the PowerScan jar-file, upload the malware samples and read the results. The following are the requirements to the client:

- Java Runtime Environment: Minimum JRE 6.
- Operating system: Only Microsoft Windows XP has been tested, although it will probably work on all Microsoft Windows platforms. The-

oretically, PowerScan should work on any platform as it is written in Java, but this has not been tested.

1.1.2 Virtualization servers

The virtualization servers are the servers hosting the virtual machines used by the PowerScan framework. The following are the requirements to the virtualization servers:

- Operating system: The system has been tested with Ubuntu Server Edition 7.10 with Linux kernel 2.6.22-14-server as host operating system, although any OS supported by VMware Server should work.
- Virtualization software: The system requires VMware Server 1.x or any other VMware Server version supported by VMware Vix. Has been tested with VMware Server 1.0.6 for Linux.

1.1.3 Usage

The following requirements apply for the usage of the PowerScan framework:

- When the jar file is to be executed, it should be executed with the directory containing the jar file as the working directory. This ensures that relative references to configuration files etc. will work. The shortcuts in supplied with PowerScan ensures the correct working directory on Windows platforms.

1.2 Environment Setup

The PowerScan malware analysis framework is made up of an analysis environment, an XML-formatted configuration file and a text formatted properties file in addition to the PowerScan jar-file running from a client workstation.

The malware sample scanning environment is made up of one or more hosts running VMware Server. By default VMware Server listens on port 902, but

it is possible to use customized port numbers. Once the server(s) have been set up, virtual machines need to be created and a Microsoft Windows OS installed. PowerScan requires that VMware Tools are installed on each of the virtual machines, which can be done via the VMware Server interface. The system supports each virtual machine (and thus Windows installation) running one anti-virus engine and/or one or more dynamic analysis tools. To avoid the tools interfering with each other, it is recommended to use only one anti-virus engine or a few tools on each virtual machine. After the anti-virus engine and/or analysis tools have been installed, the corresponding paths and parameters must be written into the XML config file.

The following is a step-by-step installation instruction:

1. Install VMware Server on hosts running x86 architecture OSs, such as Microsoft Windows or Linux/Unix. The test setup during the implementation in this thesis used Ubuntu Server, which provided very good performance and stability. Host system requirements can be found in section "Host System Requirements" of chapter 1 of the VMware Server Online Library[1].

2. A serial number is needed when installing VMware Server, this is issued by VMware when registering for download at *Download VMware Server - http://www.vmware.com/download/server*. Note that there are different serial numbers for Windows and Unix/Linux systems. Help on installation may be found in chapter 2 of "Administration Guide" on the VMware Server Online Library[1].

3. Install VMware Server Console on the computer that should function as the client. VMware Server Console is part of the "VMware Server Windows client package"[2]. Requirements for the client is also found in the "Administration Guide" on the VMware Server Online Library[1].

4. Create as many virtual machines as desired on each host. Instructions are given in chapter 2 of the "Virtual Machine Guide" on VMware Online Library[1].

5. Install Microsoft Windows XP[3] as guest OS on the virtual machines.

[1]VMware Server 1 online library - `http://pubs.vmware.com/server1/wwhelp/wwhimpl/js/html/wwhelp.htm`.

[2]VMware Server registration - `http://register.vmware.com/content/download.html`.

[3]Or other supported OS. Although only tested with Win XP, PowerScan should work with any Win NT OS. 64 bits guest OSs are also supported.

Help on this operation may be found in the "Choosing and Installing Guest Operating Systems" section of the "Guest Operating System Installation Guide" on the VMware Online Library[1]. This guide also lists known issues and compatibility between VMware products and various guest OSs.

6. Create a user with administrator access on each guest OS. Note that the users must have a password set in order to work with PowerScan.

7. Install the desired anti-virus engines and analysis tools on the designated virtual machines. This installation process is vendor specific, and is not described here.

8. Take a snapshot of each VM using the VMware Server Console, as shown in figure 1.6. The snapshot should, for performance reasons, be taken when the guest OS is running, the user logged[4] in and no windows open[5].

9. Once the virtual machines are set up, an appropriate config file and properties file must be prepared. An example XML file is shown in section 2.1. For an explanation of the XML, see section 1.11.

Note that the VMware environment need to be secured before uploading any suspected malware samples. In particular, the network feature of the virtual machine must be set up properly. Typically, before a malware sample is executed, it should be ensured that the virtual machine does not have access to the Internet or any other potentially unsecured networks. The network options, shown in figure 1.2, are available by double clicking the "network card" icon on the VMware Server Console status bar, as shown in figure 1.1. Before running a malware sample, the network should be set to "*Host-only*", so that the malware is unable to communicate with the Internet.

1.3 User Interface

The PowerScan framework currently supports two interfaces; a graphical user interface (GUI), a command line interface (CLI) and an application

[4]If not, the virtual machine must be powered on and the user logged in every time PowerScan is run.

[5]As open windows may cause real-time scanners to interfere with the on-demand scan operation.

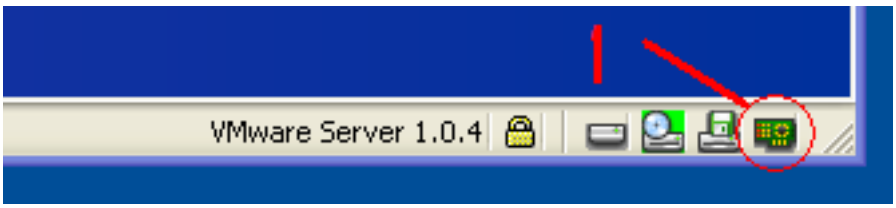

Figure 1.1: VMware Server Console status line.

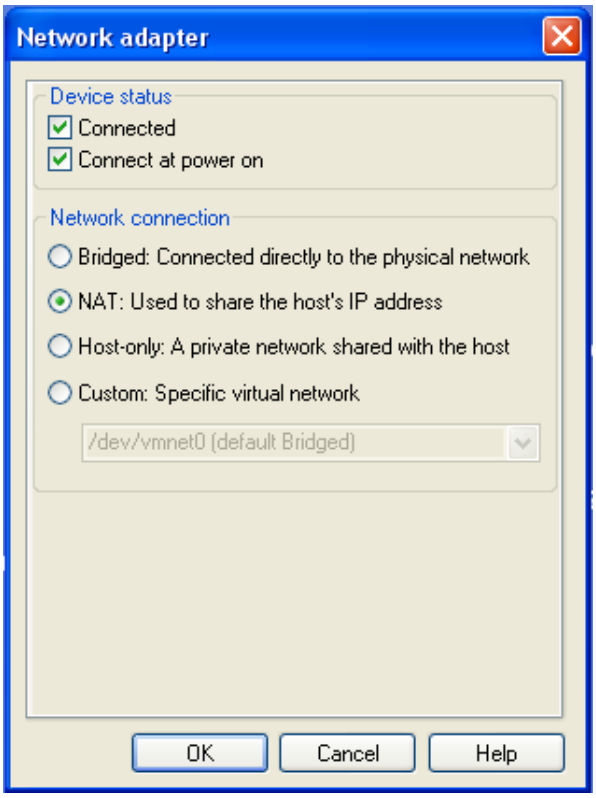

Figure 1.2: VMware Server network options.

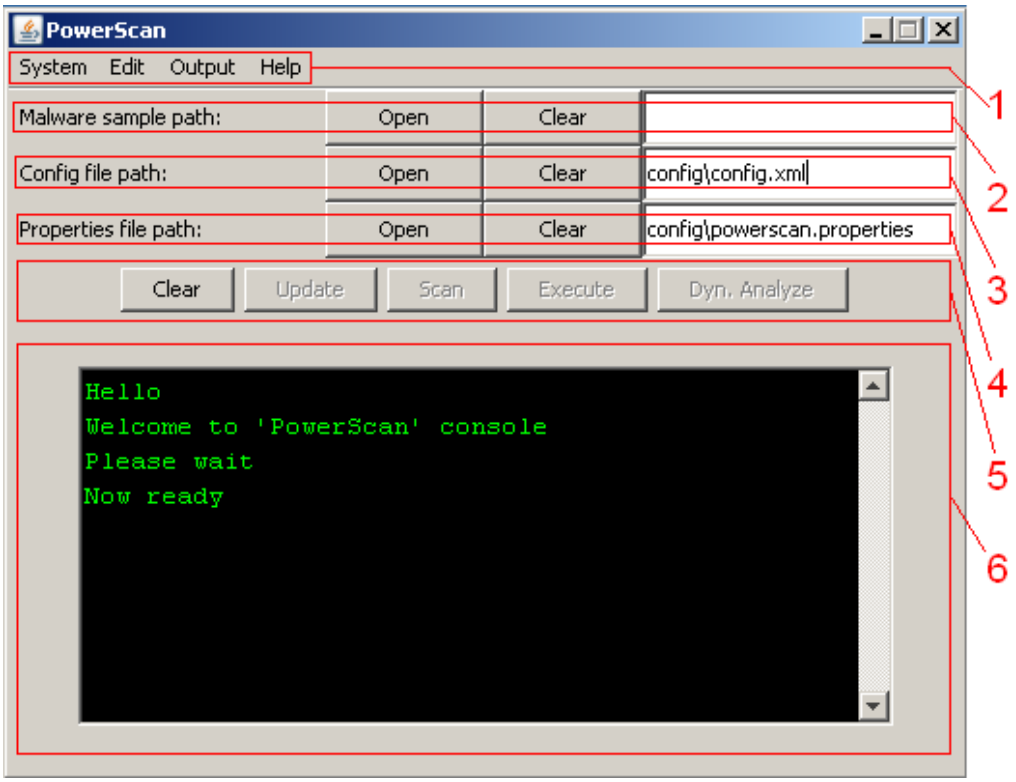

Figure 1.3: The GUI main window.

programming interface (API)[6]. The command line interface is invoked by default when launching the application. When using the CLI, all output from the program is written to stdout, meaning that it will usually be directed to the command line console. When using GUI, all output will appear on the console area of the GUI window.

1.3.1 Graphical user interface

To use GUI, the application must be started with the "-GUI" command line switch[7] or by using the included Windows shortcut. The various areas of the main GUI window are described below:

[6]The API can be called by using the PowerScan as a library and calling functions in the `edu.ntnu.item.jt.system.PowerScan` class.

[7]Meaning that PowerScan must be launched as `java -jar powerscan.jar -GUI`.

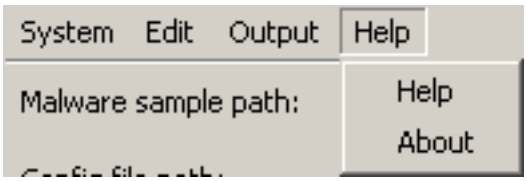

Figure 1.4: The GUI "Help" menu.

1. The menu line gives access to various options as shown in figures 1.4, 1.5, 1.7 and 1.8. The various menu options are described later in this document.

2. The *Malware sample path* area contains a field for manual path entry, a "Clear" button to clear the field and an "Open" button that allows the user to browse the client file system. This field must contain the path to the malware sample file to be analyzed (unless the operation to be performed is update).

3. The *Config file path* area contains a field for manual path entry, a "Clear" button to clear the field and an "Open" button that allows the user to browse the client file system. This field must contain the path to the XML config file described in section 1.10 of this document.

4. The file properties path area contains a field for manual path entry, a "Clear" button to clear the field and an "Open" button that allows the user to browse the client file system. This field should contain the path to the properties file described in section 1.13 of this document.

5. The buttons in this area are used to invoke the various operations of the PowerScan system.

6. The console text area is where the progress messages and result outputs are printed.

1.3.2 Command line usage

The following is the text shown when using the CLI:

```
usage: java -jar powerscan.jar [arguments].
 -ANALYZE <arg>        Perform dynamic analysis of the supplied sample
                       with all (if any) registered dynamic analysis tools
                           from XML.
```

```
-c,--CONFIG <arg>      XML Config file location. Default location:
                       config\config.xml
-EXECUTE <arg>         Executes the supplied malware sample on any virtual
                       machine supporting real-time anti-virus detection and
                            extracts results.
-GUI                   Determines whether to use a GUI.
-s,--SETTINGS <arg>    Properties file location. Default location:
                       config\powerscan.properties
-SCAN <arg>            Performs a surface scan of the supplied malware
                       sample with all scan engines configured in the XML
                           config file.
-UPDATE                Performs a update of all the AV engines listed in
                       the XML configuration file. If update can not be done
                            for a given scanner,
                       the user will be instructed on how to perform it.
```

Listing 1.1: The CLI help text

This shows that all operations are started by running the application with an appropriate switch. The location of the XML config file and properties file should also be given to the application using an argument. If not, the default locations are assumed.

1.4 Malware sample scan

Performing a malware sample scan in PowerScan means scanning the suspected malware sample with the on-demand scan feature of the installed anti-virus scanners.

Starting a malware sample scan using GUI is performed using the following steps:

1. Input the paths to the config and properties files in fields 3 and 4 shown in figure 1.3.
2. Input the path to the malware sample on the local system in field 2 shown in 1.3.
3. To read the configuration and initialize the system, choose “Start” from the “System” menu choice, shown in figure 1.5.
4. Once “Now ready” appears in the text area, press the “Scan” button.
5. When the operation finishes, the result will appear in the text area.

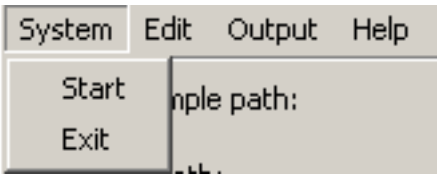

Figure 1.5: The GUI "System" menu.

Starting a malware sample scan using the CLI is done using the following command:

```
> java -jar powerscan.jar -c <configFilePath> -s <propertiesFilePath>
    -SCAN <malwareSamplePath>
```

Listing 1.2: Invocation of the PowerScan scan operation using the Command Line Interface.

1.5 Malware sample execution

Some malware samples may use packers and avoid detection by the on-demand surface scan, but might be detected when executed under surveillance of a real-time anti-virus scanner. Executing a malware sample in the test environment requires the following steps:

1. Input the paths to the config and properties files in fields 3 and 4 shown in figure 1.3.
2. Input the path to the malware sample on the local system in field 2 shown in figure 1.3.
3. To read the configuration and initialize, choose "Start" from the "System" menu choice in field 1, shown in figure 1.5.
4. Once "Now ready" appears in the text field, press the "Execute" button.
5. When the operation finishes, the result will appear in the text area.

To start execution of a malware sample using the command line, use the following command:

```
> java -jar powerscan.jar -c <configFilePath> -s <propertiesFilePath>
    -EXECUTE <malwareSamplePath>
```

Listing 1.3: Invocation of the PowerScan execute operation using the Command Line Interface.

1.6 Malware sample analysis

When dynamic analysis tools are installed as a part of the testing environment, they can be used to analyze a malware sample. To execute the installed analysis tools on the sample, requires the following steps:

1. Input the paths to the config and properties files in fields 3 and 4 shown in figure 1.3.
2. Input the path to the malware sample on the local system in field 2 shown in figure 1.3.
3. To read the configuration and initialize, choose “Start” from the “System” menu choice in field 1, shown in figure 1.5.
4. Once “Now ready” appears in the text field, click the “Dyn. analyze” button.
5. When the operation finishes, the result will appear in the text area.

One important difference between the scan and execute operation on the one hand, and the analysis operation on the other, is that while the scan and execute operations are executed on all the virtual machines in parallel, the analysis operation is carried out on one virtual machine at the time. This means that first, all the tools registered on one virtual machine is started and the malware sample optionally executed, before the system sleeps for the indicated amount of time (given in the properties file). The result files (if any) are copied back, and the virtual machine is reverted to snapshot. While the first virtual machine is reverted to snapshot, the tools registered with the next virtual machine are started, and a new sleep period is started. This is done to allow an operator time to interact with the analysis tool and/or malware sample at one virtual machine at the time. The interaction sleep period is set in the properties file, but the sleep can be skipped at any

time by pressing the "Cancel" button in the GUI process monitor or pressing the *Enter* key when using the command line.

When installing analysis tools, it is worth noting that some tools take the name of the executable to be monitored as a parameter, while others need to be started before the malware sample is executed on the system. In the configuration file, there is an element associated with the analysis tool that indicates whether the malware sample should be executed. For tools taking the path to the sample as part of their parameter, the placeholder "$samplePath" should be used in the parameter string to indicate insertion of the malware sample path.

To start execution of a malware sample using the command line, use the following command:

```
> java -jar powerscan.jar -c <configFilePath> -s <propertiesFilePath>
    -ANALYZE <malwareSamplePath>
```

Listing 1.4: Invocation of the PowerScan analyze operation using the Command Line Interface.

1.7 Update AV definition files

The definition files of the anti-virus engines needs to be updated relatively frequently. This operation includes taking snapshot of the system after definition files have been updated[8] To update the virus signature files, the following steps should be performed:

1. Input the paths to the config and properties files in fields 3 and 4 shown in figure 1.3.
2. To read the configuration and initialize, choose "Start" from the "System" menu choice in field 1, shown in figure 1.5.
3. Once "Now ready" appears in the text field, press the "Update" button.
4. When the operation finishes, the result will appear in the text area.

To initiate update of the anti-virus engines using the command line, use the following command:

[8]This is done automatically unless otherwise is stated.

```
> java -jar powerscan.jar -c <configFilePath> -s <propertiesFilePath>
    -UPDATE
```

Listing 1.5: Invocation of the PowerScan update operation using the Command Line Interface.

1.8 Adding new anti-virus engines or tools

Adding new tools to the analysis environment requires installing the anti-virus engine or tool on a virtual machine, and taking a snapshot after the installation is complete. The various configuration parameters must then be added to the XML configuration file to enable the framework to utilize the tool/scan engine. The installation in the guest OS should follow an ordinary Windows installation process, and is not described in any further detail here. Taking snapshot using the VMware Server Console is shown in figure 1.6.

1.9 Saving console output

When using the GUI, it is possible to save the text output that is currently in the console text area. This is done by choosing "Output" from the menu and then "Save", shown in figure 1.7. This option saves the text currently displayed in the text area to a file.

The similar operation when using the Command Line Interface would be redirecting the output to file by appending "`$>$ output.log`" to the given command so that for example the scan operation becomes:

```
> java -jar powerscan.jar -c <configFilePath> -s <propertiesFilePath>
    -UPDATE > output.log
```

Listing 1.6: Redirection of the update operation out using CLI.

1.10 Editing the XML configuration file

As the configuration file is written in XML, it is human-readable and could be edited using any text editor as described in section1.11. Another, hopefully

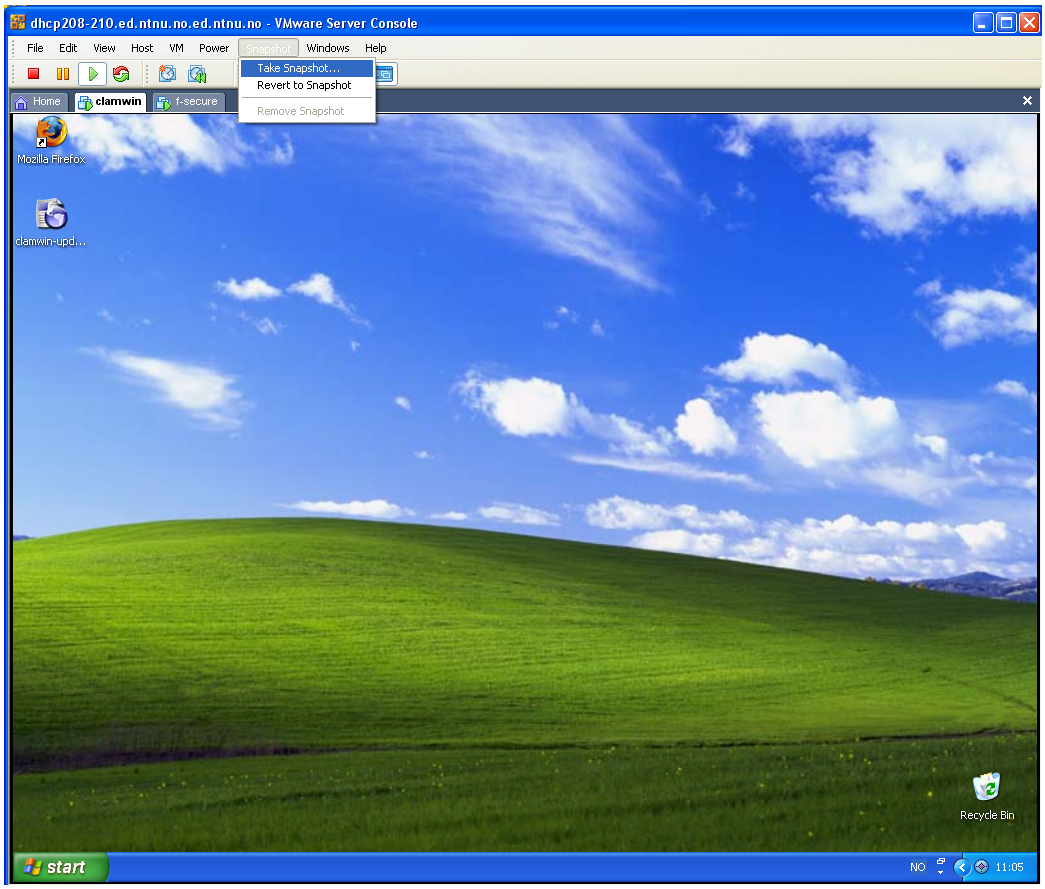

Figure 1.6: Taking snapshot using VMware Server Console.

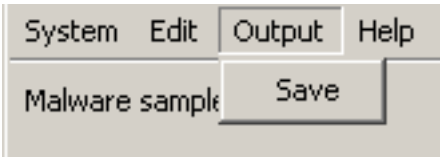

Figure 1.7: The GUI “Output” menu.

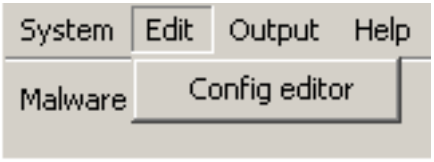

Figure 1.8: The GUI "Edit" menu.

more user friendly, means to edit the configuration is to use the built-in graphical config file editor. The config editor is launched from the "Edit" menu, and opens in a separate window. The main window is shown in figure 1.9. This figure shows the config editor in the "Host/VM" view.

The menus shown in field 1 of figure 1.9 are described in the following sections.

1.10.1 The "File" menu

The "File" menu offers the following choices, as shown in figure figure 1.10:

New config opens a new, empty configuration file.

Open config... launches a dialog box to select an XML file to edit.

Save config... launches a dialog box for saving the open XML file.

Exit closes the Config Editor.

1.10.2 The "View" menu

The "View" menu offers the following choices, shown in figure 1.12:

Host/VM view brings the user back to the host and virtual machine overview window.

1.10.3 The "Delete" menu

The "Delete" menu offers the following choices, shown in figure 1.11:

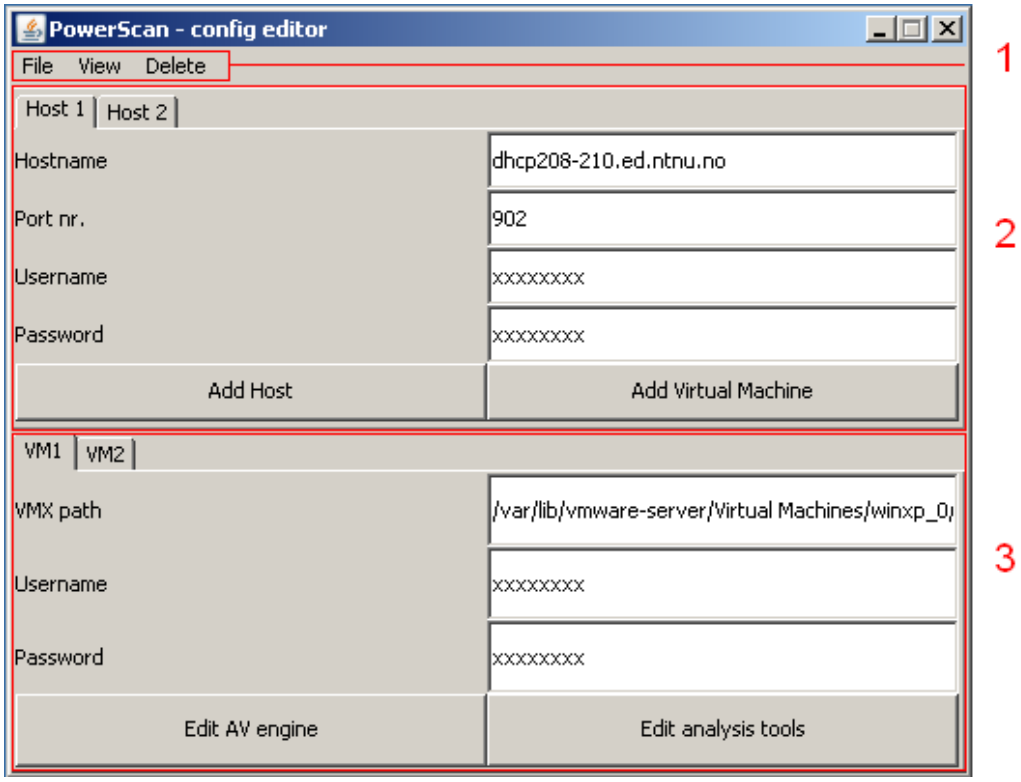

Figure 1.9: The config editor main window in "Host/VM view".

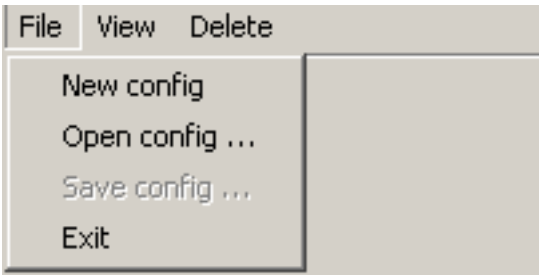

Figure 1.10: The config editor "File" menu.

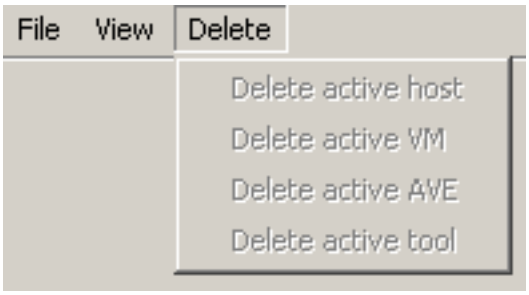

Figure 1.11: The config editor "Delete" menu.

Delete active host Only available in "Host/VM" view, this choice deletes the host that is selected in the upper half of the window, field 2 in figure 1.9.

Delete active VM Only available in "Host/VM" view, this choice deletes the virtual machine that is selected in the bottom half of the window, field 3 in figure 1.9

Delete active AVE Only available in "AVE" view, this choice deletes the currently active anti-virus engine.

Delete active tool Only available in "Tools" view, this choice deletes the currently active tool.

1.10.4 The "Host/VM" view

The "Host/VM" view shows the defined hosts in the upper half of the window, field 2 in figure 1.9, and the defined virtual machines defined for the

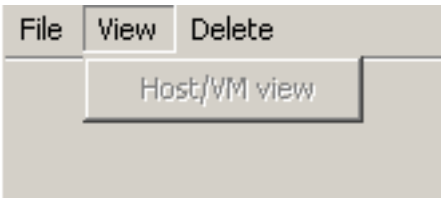

Figure 1.12: The config editor “View” menu

selected host in the lower part, field 3. It is always possible to return to this view by choosing the “View” menu on the menu bar and selecting “Host/VM view”. Field 2 also contains buttons to add a new host, or to add a new virtual machine to an already existing host. When a new host entry is created, a new, empty virtual machine entry is also created, as a VMware host offers no functionality in PowerScan unless it has a virtual machine registered.

When a host is selected in field 2, all the virtual machines that belong to that particular host are shown in the field 3. A virtual machine may run an anti-virus engine, and/or a set of analysis tools. If a host has an anti-virus engine or tools entry registered, an “Edit AV engine” or “Edit analysis tools” button is shown. If the host does not have an anti-virus engine or tool set registered, field 3 will have an “Add AV engine” and an “Add analysis tool” button.

Both the “Edit AV engine” and the “Add AV engine” button will change the editor to the “AVE” view, described in section 1.10.5. The “Edit analysis tools” and “Add analysis tool” buttons change to the “Tools” view, described in section 1.10.6.

1.10.5 The “AVE” view

The config editor “AVE” view is shown in figure 1.13. This view allows editing of all elements related to an anti-virus engine. In addition to the element fields, the view has a “Save changes” button that saves the changes made within the view. Note that this does not write the changes to file - the only way of permanently storing the changes is via the “Save config...” File menu choice! If the “AVE” view is left using the “Host/VM” choice on the “View” menu, the changes are discarded.

An entire “AVE” element can be deleted by choosing the “Delete active

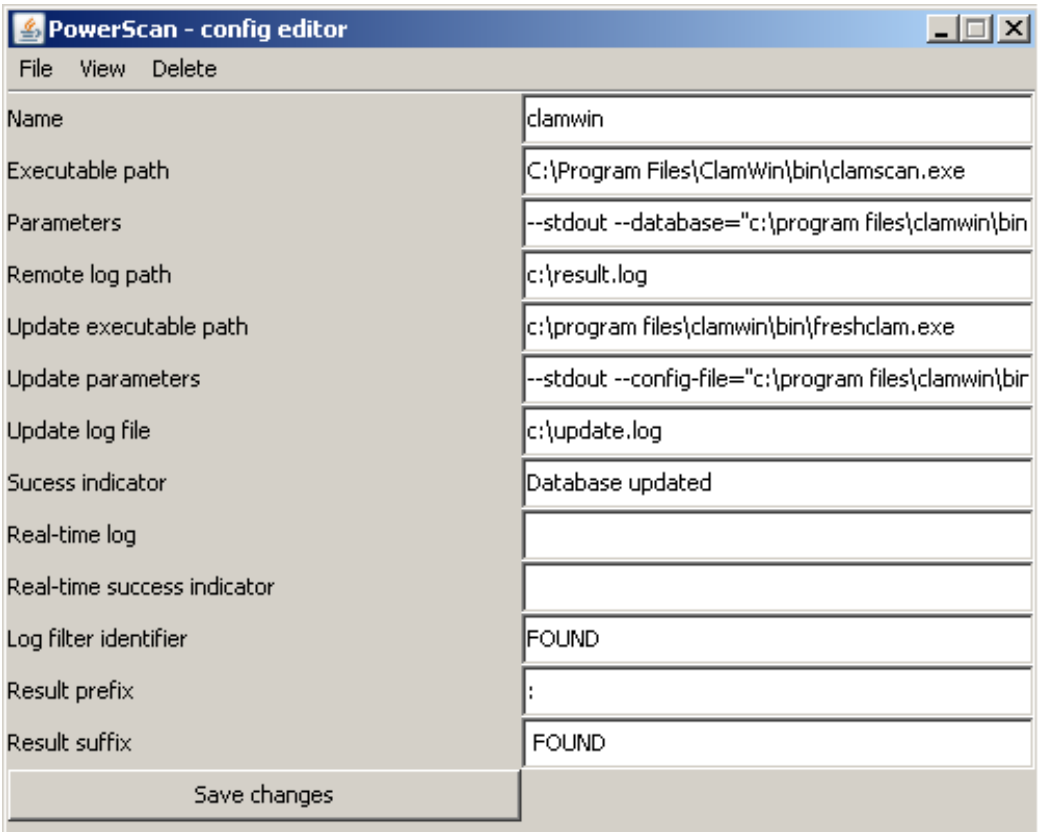

Figure 1.13: The config editor "AVE" view.

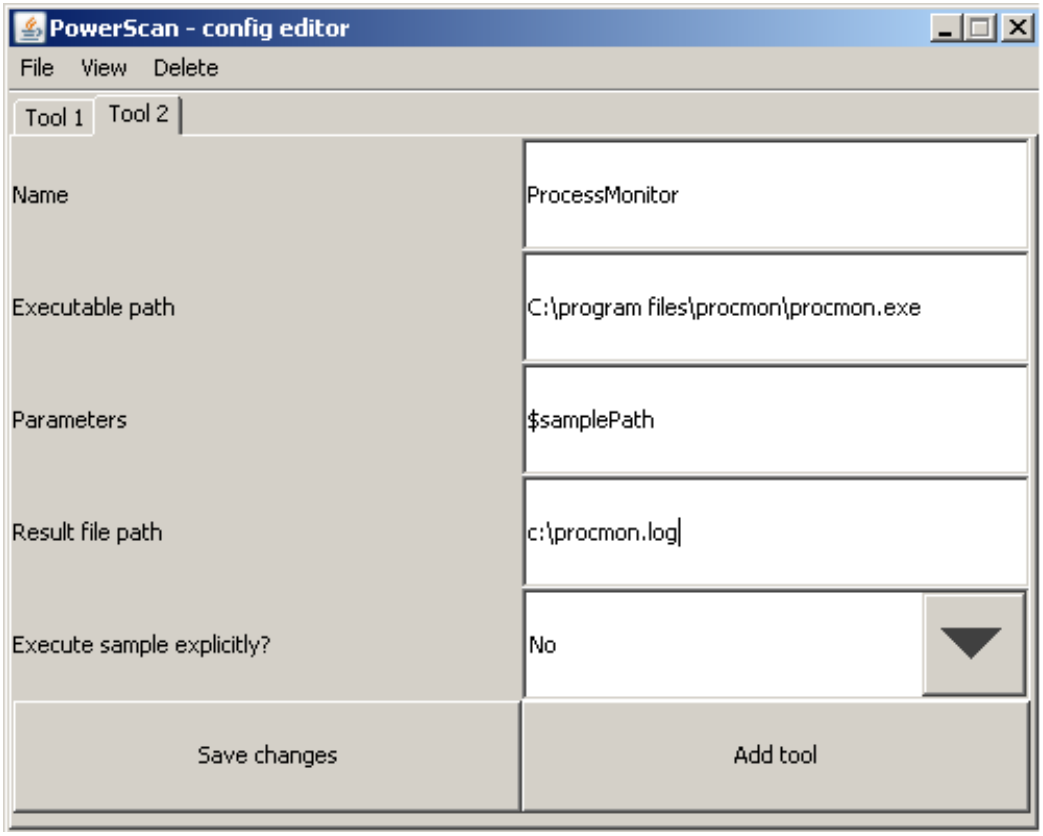

Figure 1.14: The config editor "Tools" view.

AVE" command on the "Delete" menu. If no changes have been made, or the changes have been saved by clicking the "Save changes" button, it is possible to navigate back to the "Host/VM" view by selecting the "Host/VM" command from the "View" menu.

1.10.6 The "Tools" view

The config editor "Tools" view is shown in figure 1.14. This view allows editing of all tools associated with a virtual machine. Switching between different tools is done by selecting the appropriate tab at the top of the view window. In addition to the element fields, the view has a "Save changes" button that stores the changes in the system. Note that this does not write the changes to file - the only way of permanently storing the changes is via the "Save config..." File menu choice! If the "Tools" view is left using the

"Host/VM" choice on the "View" menu, the changes are discarded. This view also has an "Add tool" button that adds a new tool element to the configuration.

The currently active tool can be deleted by choosing the "Delete active tool" command on the "Delete" menu. If no changes have been made, or the changes have been saved by clicking the "Save changes" button, it is possible to navigate back to the "Host/VM" view by selecting the "Host/VM" command from the "View" menu.

1.11 Understanding the XML configuration file

As mentioned earlier, PowerScan uses an XML structured configuration file to represent its execution environment (meaning the associated VMware hosts, virtual machines, installed scanners, tools and so on). Although the file can be manipulated in the config editor GUI, it is also possible to edit the XML directly using any text editor. This section describes the different parts that make up the XML, and how they should be interpreted.

The skeleton of the XML config file is shown below.

```
1: <PowerScan>
2:  <VMwareHostList>
3:    <VMwareHost host="">
4:      <hostPortNumber></hostPortNumber>
5:      <hostUsername></hostUsername>
6:      <hostPassword></hostPassword>
7:      <VM vmxPath="">
8:        <vmUsername></vmUsername>
9:        <vmPassword></vmPassword>
10:       <avEngine name="">
11:         <avExecutablePath></avExecutablePath>
12:         <avParameters></avParameters>
13:         <avLogFilePath></avLogFilePath>
14:         <avLogFilter>
15:           <avResultIdentifier></avResultIdentifier>
16:           <avResultPrefix></avResultPrefix>
17:           <avResultSuffix></avResultSuffix>
18:         </avLogFilter>
19:         <avUpdateInfo>
20:           <avUpdateExecutable></avUpdateExecutable>
21:           <avUpdateParameters></avUpdateParameters>
22:           <avUpdateLogPath></avUpdateLogPath>
23:           <avUpdateSuccessIndicator></avUpdateSuccessIndicator>
24:         </avUpdateInfo>
25:         <realTimeScan>
26:           <rtLogPath></rtLogPath>
27:           <rtResultIdentifier></rtResultIdentifier>
```

```
28:         </realTimeScan>
29:       </avEngine>
30:       <analysisTools>
31:         <dynamicAnalysisTool toolName="">
32:           <toolExecutablePath></toolExecutablePath>
33:           <toolParameters></toolParameters>
34:           <executeMalwareExplicitly></executeMalwareExplicitly>
35:         </dynamicAnalysisTool>
36:       </analysisTools>
37:     </VM>
38:   </VMwareHost>
39: </VMwareHostList>
40:</PowerScan>
```

Listing 1.7: Skeleton of the XML config file

The <PowerScan> element is the root element of the PowerScan XML configuration and signifies that the XML indeed is a PowerScan configuration file. It contains only one element; the <VMwareHostList>, which is, as the name implies, a list containing one or more <VMwareHost> elements. The <VMwareHost> element has a mandatory attribute "host" which contains the host name or IP address of a host running VMware Server. The <VMwareHost> element contains two elements, <hostUsername> and <hostPassword>, which are required to connect to the server. Further, a <VMwareHost> element contains one or more <VM> elements representing a virtual machine installed on the given host. The <VM> element has a mandatory attribute "vmxPath", which gives the path on the server to a vmx file that contains configuration info for the VM. This path is needed to be able to boot the VM without using the GUI-based VMware Server Console. A <VM> element contains a <vmUsername> and a <vmPassword> element which contain user credentials for a user that must exist on the virtual machine guest OS. In addition to the "vmxPath", <vmUsername> and <vmPassword> elements, a <VM> contains information about what tools are installed on the virtual machine. A tool in this context can be an anti-virus engine and/or one or more analysis tools. XML description of the installed tools of the virtual machine are then given within the <VM> element. The description of an anti-virus engine is shown below.

```
10:       <avEngine name="">
11:         <avExecutablePath></avExecutablePath>
12:         <avParameters></avParameters>
13:         <avLogFilePath></avLogFilePath>
14:         <avLogFilter>
15:           <avResultIdentifier></avResultIdentifier>
16:           <avResultPrefix></avResultPrefix>
17:           <avResultSuffix></avResultSuffix>
18:         </avLogFilter>
19:         <avUpdateInfo>
20:           <avUpdateExecutable></avUpdateExecutable>
21:           <avUpdateParameters></avUpdateParameters>
```

```
22:           <avUpdateLogPath></avUpdateLogPath>
23:           <avUpdateSuccessIndicator></avUpdateSuccessIndicator>
24:         </avUpdateInfo>
25:         <realTimeScan>
26:           <rtLogPath></rtLogPath>
27:           <rtResultIdentifier></rtResultIdentifier>
28:         </realTimeScan>
29:       </avEngine>
```

Listing 1.8: The AV engine element of the XML config file

As can be seen, the <avEngine> element has a "name" attribute that is used for identification of the engine. The value of this attribute could for example be set to "F-secure" or "ClamWin". Note that the element is only used for visual identification, and can theoretically be set almost any value. Then there follows four elements used when running an on-demand scan of a sample file.

- The <avExecutablePath>element on line 11 gives the path to the executable used to initiate the on-demand scan.
- The <avParameters> element on line 12 holds any parameters or arguments that should be passed to the executable. To insert the malware sample path in the parameters, use the string "$samplePath".
- The <avLogFilePath> on line 13 gives the path to a log file that is retrieved from the VM after the scan has completed.
- The <avLogFilter> element on line 14 contains strings to look for in the retrieved log file when parsing the result. These strings are used as follows:
  - The string given in the <avResultIndentifier> element on line 15 is used to fine the correct line in the log file. If this element is empty, the entire log file is printed.
  - The string given in the <avResultPrefix> element on line 16 is used to determine from which position in line the result is starting. If this element is empty, it is assumed that the result text starts at the beginning of the line. If the element is set, the text AFTER the prefix is extracted.
  - The string given in the <avResultSuffix> element on line 17 is used to determine at which position in the line the result text ends. If the element is empty, it is assumed that the result text

ends at the end of the line. If the element is set, the text starting with the given string is removed.

For example, the interesting line in a log file may look like:

VIRUS FOUND: eicar_test_file found in selected file(s)

A filter definition like:

```
14:         <avLogFilter>
15:           <avResultIdentifier>VIRUS FOUND:</avResultIdentifier>
16:           <avResultPrefix>: </avResultPrefix>
17:           <avResultSuffix>found</avResultSuffix>
18:         </avLogFilter>
```

Listing 1.9: The AV log filter element of the XML config file

would give the result *eicar_test_file*.

The next part of the <VM> element is used to hold information needed to run virus definition database updates on the anti-virus engines. The <avUpdateInfo> element contains the following elements:

- The <avUpdateExecutable> element on line 20 should contain the path to the executable that performs the update.
- The <avUpdateParameters> element on line 21 holds any parameters required by the update executable.
- The <avUpdateLogPath> element on line 22 holds the path to the log file where the result of the update operation is written. If this tag is not present (or empty), the update operation is started and no further action is taken[9].
- The <avUpdateSuccessIndicator> element on line 23 contains a string that is searched for in the log file to determine the result of the update operation. If the success indicator is found, the operation is assumed to have succeeded, and the line(s) containing the success indicator are printed. If the <avUpdateLogPath> element is present, but no <avUpdateSuccessIndicator>, the entire log file is printed for manual analysis by the user.

[9]This again means that the user will have to manually check the result of the update operation. This might be desirable for updates performed on engines without support for reporting the update result to stdout.

The final element within the <avEngine> element is used when a malware sample is executed in the OS installed on the virtual machine to see if the sample is detected by the real-time functionality of an anti-virus engine. Following the convention of the other elements, the <realTimeScan> elements contains a log file path and a result identifier. The <rtLogPath> element holds the path to the real-time scanner log file, and the <rtResultIdentifier> element a string used to find the line containing the result in the log file.

If a VM is used to perform dynamic analysis of malware, the <VM> element needs to contain an <analysisTools> element. Although it is possible for virtual machine to have both analysis tools and an anti-virus scanner, it is generally recommended that they are installed on separate machines, as they may interfere with each other. The <analysisTools> element is a list containing information about one or more tools installed on the virtual machine. The <analysisTools> element is shown below.

```
30:       <analysisTools>
31:         <dynamicAnalysisTool toolName="">
32:           <toolExecutablePath></toolExecutablePath>
33:           <toolParameters></toolParameters>
34:           <executeMalwareExplicitly></executeMalwareExplicitly>
35:         </dynamicAnalysisTool>
36:       </analysisTools>
```

Listing 1.10: The analysis tools element of the XML config file

The <dynamicAnalysisTool> element is the elements that represents one tool. As seen on line 31, the element has an attribute "toolName" that is used for identification[10]. The elements contained within the <dynamicAnalysisTool> elements are the following:

- The <toolExecutablePath> element holds the path to the tool executable.
- The <toolParameters> element holds the parameters that are supplied to the executable. An important feature with this string is that occurrences of the string "$samplePath" is replaced with the path to the malware sample on the virtual machine.
- The <executeMalwareExplicitly> element is a boolean value, holding either *true* or *false*. *true* indicates that the malware sample should be executed on the remote system after the tools have been started.

[10]As for anti-virus engines, the name is merely a visual identificator, and has no real impact on the execution.

Some tools do not require this, as they take the malware sample file as a parameter. In the latter case, the value should be *false*. If there is more than one tool installed on a single virtual machine, the malware sample is executed after the tools have been started if *any* of the tools have the <executeMalwareExplicitly> value *true*.

1.12 Redirection of console output

Some applications print their output to the command line console and use a log file format that is not human readable. This can make interpretation of the result hard. There is, however, a means for redirecting the command line console text output to a file that can be copied back to the client for further examination. This is done by running the application through the Microsoft Windows Command Prompt on the virtual machine. The Microsoft Windows Command Prompt executable, *cmd.exe*, has a switch that executes whatever is given after the switch and terminates. What makes this useful is that the Command Prompt allows for redirection of output from application run under it using the ">" operator. The following is an example XML configuration of a tool using output redirection[11]:

```
<dynamicAnalysisTool toolName="ipconfig">
  <toolExecutablePath>c:\windows\system32\cmd.exe</toolExecutablePath>
  <toolParameters>/C ipconfig.exe > c:\ipconfig.log</toolParameters>
  <toolResultFilePath>c:\ipconfig.log</toolResultFilePath>
  <executeMalwareExplicitly>false</executeMalwareExplicitly>
</dynamicAnalysisTool>
```

Listing 1.11: Redirection of Command Prompt output on a virtual machine

1.13 Understanding the properties file

The properties file is a simple text-based configuration file that is used to set global PowerScan constants. The format is a simple "key = value" scheme, where the values are parsed from string to other data types by the system, as shown in the table below. The following values are set in the properties file:

[11]Note that the parameters contain both the file to be executed (ipconfig.exe) and the > operator. Any tool can be launched via the Command Prompt in a similar manner.

Property	Type	Default	Description
vmware.tools- .timeout	int	60	Timeout for VMware Tools to start in the Virtual Machines (VMware Tools are required for operation of PowerScan).
scanner- .executionpath	string	c:\\	Specifies where in the virtual machine the malware should be copied (and executed).
snapshot.before.scan	boolean	false	Specifies whether snapshots should be taken before performing scans and malware execution. This is generally NOT recommended, as storing snapshots is a time consuming operation, which should only be performed when changes are made to the Virtual Machine.
polling.interval- .minor	double	0.250	Sets polling interval for minor operations. This indicates how often the system should check for completion of minor non-blocking tasks, such as file execution.
polling.interval- .major	double	1.000	Set polling interval for major operations. This indicates how often the system should check for completion of major non-blocking tasks, such as full scan operations.
scanner.timeout	int	60	Maximum time to allow for individual scan operations to finish.
update.timeout	int	600	Maximum time to allow for updating to finish.
full.scan.timeout	int	600	Timeout for full scan. Used to prevent entire system from crashing following failure in a single scan thread. Should be greater scanner.timeout value above, since copying of files etc. is included in this timeout in addition to the actual scan.
malware.execution- .timeout	int	25	Maximum time to allow malware to execute with real-time anti-virus scanner running in the background.
full.execution- .timeout	int	240	Maximum total time for malware execution operation (for all registered scanners). Used to prevent the entire system from crashing following failure in a single thread.
log.overwrite	boolean	false	Specifies whether log files should be overwritten when program is executed. If set to false, time and date are appended to log files. Note that log files may take up significant resources over time.
log.vmware	string	logs\\vmware.log	Path for the VMware log file, where VMware-related events are logged.
log.system.scanner	string	logs\\scanner.log	Path to the log file for Scanner operations.
log.system	string	logs\\system.log	Log path for system classes.
log.system.executor	string	logs\\executor.log	Log path for the executor class (dynamic analysis tools log).
executor.execution- path	string	c:\\	The path on the virtual machines where the malware sample is copied to and executed from when using the dynamic analysis functionality.
executor.localResult- Path	string	results\\	Directory on the client machine where log files should be copied when using the dynamic analysis functionality.

Continued on next page

Property	Type	Default	Description
executor.overall- .timeout	int	330	Sets the time the operator has to interact with the analysis tool or malware sample logging in to the virtual machine before the result is copied back to local system.
xml.xsd.path	string	config.xsd	Sets the path for the XSD file used to validate the XML config file

All times are given in seconds. Note that backslashes ("\") must be escaped by another backslash, such that `c:\program files\` becomes `c:\\program files\\`. Also note that directories given on the remote machine must already be created on the virtual machine, as Vix 1.0 and 1.1 do not support directory creation.

1.14 PowerScan files

The following files are found in PowerScan:

`PowerScan.jar` The PowerScan jar file. Contains the PowerScan code. Executed by running *java -jar PowerScan.jar* (requires that Java 6 or newer is installed on the client machine). Note that the PowerScan file must be executed with the root directory of PowerScan as the working directory. In other words, the working directory must contain the other files described in this section.

`PowerScan GUI` Microsoft Windows shortcut which launches the PowerScan graphical user interface.

`PowerScan CLI` Microsoft Windows shortcut which launches the PowerScan command line interface using `cmd.exe`.

`vix.dll` The Vix C library used for communication with VMware components such as VMware Server hosts and virtual machines.

`ssleay32.dll` & `libeay32.dll` The OpenSSL toolkit for SSL/TLS. Extension used by Vix to enable secure communication.

`userguide.pdf` User manual of the PowerScan framework - this document.

`eclipse_project.zip` Eclipse project containing the needed files to import PowerScan in Eclipse.

config\ Configuration related files.

- `powerscan.properties` Default properties file for PowerScan, defining constants.
- `config.xml` Default XML configuration file for PowerScan.
- `config.xsd` XML Schema Definition (XSD) file used to validate the XML configuration.

lib\ External framework library files.

- `jna.jar` Java Native Access framework, used to communicate with C libraries.
- `simple-xml-1.7.1.jar` Simple framework, used to serialize and deserialize configuration to and from XML.
- `commons-cli-1.1.jar` Apache Commons Command Line Interface framework, used to offer the command line capability of PowerScan.
- `xercesImpl.jar` Xerces Java Parser framework, used to validate the XML configuration file against XSD.

logs\ Log files created by the PowerScan framework.

src\ The PowerScan source code.

javadoc\ The PowerScan javadoc files.

1.15 Understanding the log files

The PowerScan framework creates plain text formatted log file when executing operations. Paths for the log files are given in the .properties-file, see section 1.13 for details. It is important to notice that the property *log.overwrite* determines whether the log files should be overwritten, or if new log files should be created every time the PowerScan program is executed. Not overwriting the files is smart to keep history, but the number of log files can easily become overwhelmingly large.

Log files are written to `$workingDirectory\logs`. Files with the following prefixes are created:

system Used by the XML error handler, XML reader and Connector classes. These files contain log messages containing information about functionality such as reading from configuration file, initialization and connections to the hosts.

executor Used by the Executor class. An Executor object is a representation of a virtual machine with analysis tools installed. Information in these files includes copying of files, execution of programs and snapshot handling.

scanner Used by the Scanner class. A Scanner object is a representation of a virtual machine with an anti-virus engine installed. Execution of files, on-demand scans and the copying of files to and from a virtual machine is logged in these files.

vmware Used by GuestOS and VMwareServer classes. Contains information about operations against virtual machines, such as power-ons, login attempts, copying of files and the taking of and reverting to snapshots.

Chapter 2

Configuration files

2.1 Example XML config file

```
<PowerScan>
   <VMwareHostList>
      <VMwareHost host="dhcp208-210.ed.ntnu.no">
         <hostPortNumber>902</hostPortNumber>
         <hostUsername>XXXXXXX</hostUsername>
         <hostPassword>XXXXXXX</hostPassword>
         <vmList>
            <VM vmxPath="/var/lib/vmware-server/Virtual Machines/winxp_0/
                Windows XP Professional.vmx">
               <vmUsername>XXXXXXX</vmUsername>
               <vmPassword>XXXXXXX</vmPassword>
               <avEngine name="clamwin">
                  <avExecutablePath>C:\Program Files\ClamWin\bin\clamscan.
                      exe</avExecutablePath>
                  <avParameters>--stdout --database="c:\program files\
                      clamwin\bin" --log=c:\result.log "$samplePath"</
                      avParameters>
                  <avLogFilePath>c:\result.log</avLogFilePath>
                  <avLogFilter>
                     <avResultIdentifier>FOUND</avResultIdentifier>
                     <avResultPrefix>: </avResultPrefix>
                     <avResultSuffix> FOUND</avResultSuffix>
                  </avLogFilter>
                  <avUpdateInfo>
                     <avUpdateExecutable>c:\program files\clamwin\bin\
                         freshclam.exe</avUpdateExecutable>
                     <avUpdateParameters>--stdout --config-file="c:\program
                         files\clamwin\bin\clamd.conf" --datadir="c:\program
                          files\clamwin\bin" --log="c:\update.log" </
                         avUpdateParameters>
                     <avUpdateLogPath>c:\update.log</avUpdateLogPath>
                     <avUpdateSuccessIndicator>Database updated</
                         avUpdateSuccessIndicator>
                  </avUpdateInfo>
```

```
      <realTimeScan>
         <rtLogPath></rtLogPath>
         <rtResultIdentifier></rtResultIdentifier>
      </realTimeScan>
   </avEngine>
   <analysisTools>
      <dynamicAnalysisTool toolName="IPconfig">
         <toolExecutablePath>c:\windows\system32\cmd.exe</
             toolExecutablePath>
         <toolParameters>/C ipconfig > c:\ipconfig2.log</
             toolParameters>
         <toolResultFilePath>c:\ipconfig2.log</
             toolResultFilePath>
         <executeMalwareExplicitly>false</
             executeMalwareExplicitly>
      </dynamicAnalysisTool>
      <dynamicAnalysisTool toolName="Netstat">
         <toolExecutablePath>c:\windows\system32\cmd.exe</
             toolExecutablePath>
         <toolParameters>/C netstat -a > c:\netstat.log</
             toolParameters>
         <toolResultFilePath>c:\netstat.log</toolResultFilePath>
         <executeMalwareExplicitly>false</
             executeMalwareExplicitly>
      </dynamicAnalysisTool>
   </analysisTools>
</VM>
<VM vmxPath="/var/lib/vmware-server/Virtual Machines/
    winxp_default/Windows XP Professional.vmx">
   <vmUsername>XXXXXXX</vmUsername>
   <vmPassword>XXXXXXX</vmPassword>
   <avEngine name="F-Secure 7.10">
      <avExecutablePath>c:\Program files\f-secure\Anti-Virus\
          fsav.exe</avExecutablePath>
      <avParameters>/REPORT=c:\result.log "$samplePath"</
          avParameters>
      <avLogFilePath>c:\result.log</avLogFilePath>
      <avLogFilter>
         <avResultIdentifier>Infection</avResultIdentifier>
         <avResultPrefix>: </avResultPrefix>
         <avResultSuffix></avResultSuffix>
      </avLogFilter>
      <avUpdateInfo>
         <avUpdateExecutable>c:\program files\f-secure\anti-
             virus\getdbhtp.exe</avUpdateExecutable>
         <avUpdateParameters>-url=http://avupdate.f-secure.com/
             updates/ -gui=1 -ver=FSAV6</avUpdateParameters>
         <avUpdateLogPath></avUpdateLogPath>
         <avUpdateSuccessIndicator></avUpdateSuccessIndicator>
      </avUpdateInfo>
      <realTimeScan>
         <rtLogPath></rtLogPath>
         <rtResultIdentifier></rtResultIdentifier>
      </realTimeScan>
   </avEngine>
</VM>
<VM vmxPath="/var/lib/vmware-server/Virtual Machines/winxp_1/
    Windows XP Professional.vmx">
   <vmUsername>XXXXXXX</vmUsername>
   <vmPassword>XXXXXXX</vmPassword>
   <avEngine name="AVG">
```

```
                <avExecutablePath>c:\Program files\avg\avg8\avgscanx.exe</
                    avExecutablePath>
                <avParameters>/REPORT=c:\report.txt /SCAN=$samplePath</
                    avParameters>
                <avLogFilePath>c:\report.txt</avLogFilePath>
                <avLogFilter>
                   <avResultIdentifier>identified </avResultIdentifier>
                   <avResultPrefix></avResultPrefix>
                   <avResultSuffix></avResultSuffix>
                </avLogFilter>
                <avUpdateInfo>
                   <avUpdateExecutable>C:\Program Files\AVG\AVG8\avgupd.
                       exe</avUpdateExecutable>
                   <avUpdateParameters></avUpdateParameters>
                   <avUpdateLogPath></avUpdateLogPath>
                   <avUpdateSuccessIndicator></avUpdateSuccessIndicator>
                </avUpdateInfo>
                <realTimeScan>
                   <rtLogPath>C:\Documents and Settings\All Users\
                       Application Data\avg8\Log\avgrs.log</rtLogPath>
                   <rtResultIdentifier>EID_Id_vir</rtResultIdentifier>
                </realTimeScan>
             </avEngine>
          </VM>
       </vmList>
    </VMwareHost>
    <VMwareHost host="129.241.208.158">
       <hostPortNumber>902</hostPortNumber>
       <hostUsername>XXXXXXX</hostUsername>
       <hostPassword>XXXXXXX</hostPassword>
       <vmList>
          <VM vmxPath="C:\Virtual Machines\Windows XP Professional\Windows
               XP Professional.vmx">
             <vmUsername>XXXXXXX</vmUsername>
             <vmPassword>XXXXXXX</vmPassword>
             <analysisTools>
                <dynamicAnalysisTool toolName="Norman Sandbox Analyzer">
                   <toolExecutablePath>C:\NSA>analyzer.exe</
                       toolExecutablePath>
                   <toolParameters> /d:c:\nsa /a:c:\nsa.log $samplePath</
                       toolParameters>
                   <toolResultFilePath>c:\nsa.log</toolResultFilePath>
                   <executeMalwareExplicitly>false</
                       executeMalwareExplicitly>
                </dynamicAnalysisTool>
             </analysisTools>
          </VM>
       </vmList>
    </VMwareHost>
 </VMwareHostList>
</PowerScan>
```

Listing 2.1: A sample XML configuration file

2.2 PowerScan XML Schema Definition

The following listing shows the PowerScan XML Schema Definition (XSD), which defines the rules for the PowerScan XML configuration file.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

<!-- Define the simple elements first -->
<xsd:element name="hostPassword" type="xsd:string"/>
<xsd:element name="hostUsername" type="xsd:string"/>
<xsd:element name="avResultIdentifier" type="xsd:string"/>
<xsd:element name="avResultPrefix" type="xsd:string"/>
<xsd:element name="avResultSuffix" type="xsd:string"/>
<xsd:element name="avUpdateExecutable" type="xsd:string"/>
<xsd:element name="avUpdateParameters" type="xsd:string"/>
<xsd:element name="avUpdateLogPath" type="xsd:string"/>
<xsd:element name="avUpdateSuccessIndicator" type="xsd:string"/>
<xsd:element name="avExecutablePath" type="xsd:string"/>
<xsd:element name="avParameters" type="xsd:string"/>
<xsd:element name="avLogFilePath" type="xsd:string"/>
<xsd:element name="vmUsername" type="xsd:string"/>
<xsd:element name="vmPassword" type="xsd:string"/>
<xsd:element name="toolExecutablePath" type="xsd:string"/>
<xsd:element name="toolParameters" type="xsd:string"/>
<xsd:element name="toolResultFilePath" type="xsd:string"/>
<xsd:element name="rtLogPath" type="xsd:string"/>
<xsd:element name="rtResultIdentifier" type="xsd:string"/>
<xsd:element name="executeMalwareExplicitly" type="xsd:boolean"/>
<xsd:element name="hostPortNumber" type="xsd:integer"/>

<!-- Define attributes -->

<xsd:attribute name="host" type="xsd:string"/>
<xsd:attribute name="name" type="xsd:string"/>
<xsd:attribute name="vmxPath" type="xsd:string"/>
<xsd:attribute name="toolName" type="xsd:string"/>

<!-- Define the complex elements -->

<xsd:element name="VMwareHostList">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element maxOccurs="unbounded" minOccurs="1" ref="VMwareHost"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<xsd:element name="VMwareHost">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element maxOccurs="1" minOccurs="1" ref="hostPortNumber"/>
      <xsd:element maxOccurs="1" minOccurs="1" ref="hostUsername"/>
      <xsd:element maxOccurs="1" minOccurs="1" ref="hostPassword"/>
      <xsd:element maxOccurs="1" minOccurs="1" ref="vmList"/>
    </xsd:sequence>
    <xsd:attribute ref="host" use="required"/>
  </xsd:complexType>
</xsd:element>
```

```
<xsd:element name="vmList">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element maxOccurs="unbounded" minOccurs="0" ref="VM"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<xsd:element name="avLogFilter">
  <xsd:complexType mixed="true">
    <xsd:sequence>
      <xsd:element maxOccurs="1" minOccurs="1" ref="avResultIdentifier"/>
      <xsd:element maxOccurs="1" minOccurs="0" ref="avResultPrefix"/>
      <xsd:element maxOccurs="1" minOccurs="0" ref="avResultSuffix"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<xsd:element name="avUpdateInfo">
  <xsd:complexType mixed="true">
    <xsd:sequence>
      <xsd:element maxOccurs="1" minOccurs="0" ref="avUpdateExecutable"/>
      <xsd:element maxOccurs="1" minOccurs="0" ref="avUpdateParameters"/>
      <xsd:element maxOccurs="1" minOccurs="0" ref="avUpdateLogPath"/>
      <xsd:element maxOccurs="1" minOccurs="0" ref="avUpdateSuccessIndicator
          "/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<xsd:element name="avEngine">
  <xsd:complexType mixed="true">
    <xsd:sequence>
      <xsd:element maxOccurs="1" minOccurs="1" ref="avExecutablePath"/>
      <xsd:element maxOccurs="1" minOccurs="1" ref="avParameters"/>
      <xsd:element maxOccurs="1" minOccurs="1" ref="avLogFilePath"/>
      <xsd:element maxOccurs="1" minOccurs="0" ref="avLogFilter"/>
      <xsd:element maxOccurs="1" minOccurs="0" ref="avUpdateInfo"/>
      <xsd:element maxOccurs="1" minOccurs="0" ref="realTimeScan"/>
    </xsd:sequence>
    <xsd:attribute ref="name" use="required"/>
  </xsd:complexType>
</xsd:element>

<xsd:element name="realTimeScan">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element maxOccurs="1" minOccurs="1" ref="rtLogPath"/>
      <xsd:element maxOccurs="1" minOccurs="0" ref="rtResultIdentifier"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<xsd:element name="analysisTools">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element maxOccurs="unbounded" minOccurs="0" ref="
          dynamicAnalysisTool"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

```
<xsd:element name="dynamicAnalysisTool">
<xsd:complexType mixed="true">
  <xsd:sequence>
    <xsd:element maxOccurs="1" minOccurs="1" ref="toolExecutablePath"/>
    <xsd:element maxOccurs="1" minOccurs="0" ref="toolParameters"/>
    <xsd:element maxOccurs="1" minOccurs="0" ref="toolResultFilePath"/>
    <xsd:element maxOccurs="1" minOccurs="0" ref="executeMalwareExplicitly"/
        >
  </xsd:sequence>
  <xsd:attribute ref="toolName" use="required"/>
</xsd:complexType>
</xsd:element>

<xsd:element name="VM">
<xsd:complexType mixed="true">
  <xsd:sequence>
    <xsd:element maxOccurs="1" minOccurs="1" ref="vmUsername"/>
    <xsd:element maxOccurs="1" minOccurs="1" ref="vmPassword"/>
    <xsd:element maxOccurs="1" minOccurs="0" ref="avEngine"/>
    <xsd:element maxOccurs="1" minOccurs="0" ref="analysisTools"/>
  </xsd:sequence>
  <xsd:attribute ref="vmxPath" use="required"/>
</xsd:complexType>
</xsd:element>

<!-- Define the overall structure -->
<!-- Starting with one VMwareHostList element -->
<xsd:element name="PowerScan">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element maxOccurs="1" minOccurs="1" ref="VMwareHostList"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

</xsd:schema>
```

Listing 2.2: The W3C XML Schema definition

2.3 Description of PowerScan's XML with respect to the XSD schema

This section describes the setup of PowerScan's XML configuration file, with respect to the defined XSD schema. It gives a description of the XML elements and their limitations, as they are described in the XSD. For the actual XSD schema and an example XML file, refer to the sections above.

Schema languages are useful, as they can be used to verify that the given XML config file is not only well-formed, meaning that it conforms to the XML standard, but also that it conforms to specified rules concerning which elements are optional, the number of allowed sub elements within elements and so on.

When initiated, the PowerScan framework needs to be told how the lab environment it is supposed to utilize is set up. The required information includes how many VMware hosts are supposed to be used, their IP address or host name, username and password for the host, how many VMware guest OS instances are running and which AV engines or analysis tools are installed on them and how the tools are to be executed. The layout of the XML file is described below:

The XML file starts with one root element called <PowerScan>. A <PowerScan> element is defined in XSD as follows:

```
<xsd:element name="PowerScan">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="VMwareHostList" minOccurs="0" maxOccurs="1" />
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Listing 2.3: XSD PowerScan element

This means that a <PowerScan> element contains one and only one element of the type <VMwareHostList>. The <VMwareHostList> element is also a complex element[1], defined as

```
<xsd:element name="VMwareHostList">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="VMwareHost" minOccurs="1" maxOccurs="unbounded" />
    </xsd:sequence>
  </xsd:complexType>
```

[1] A complex element is an element that contains other elements and/or attributes.

```
</xsd:element>
```

Listing 2.4: XSD VMwareHostList element

This shows that a <VMwareHostList> element is a list containing at least one but possibly an arbitrary high number of <VMwareHost> elements. The <VMwareHost> element contains the following elements:

```
<xsd:element name="VMwareHost">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element maxOccurs="1" minOccurs="1" ref="hostPortNumber"/>
      <xsd:element maxOccurs="1" minOccurs="1" ref="hostUsername"/>
      <xsd:element maxOccurs="1" minOccurs="1" ref="hostPassword"/>
      <xsd:element maxOccurs="1" minOccurs="1" ref="vmList"/>
    </xsd:sequence>
    <xsd:attribute ref="host" use="required"/>
  </xsd:complexType>
</xsd:element>
```

Listing 2.5: XSD VMwareHost element

As seen, a <VMwareHost> contains the password and username for the host, and an element called <vmList>, which is a list one or more <VM> elements. Each <VM> element represents a guest OS running Windows XP, and has the following elements

```
<xsd:element name="VM">
<xsd:complexType mixed="true">
  <xsd:sequence>
    <xsd:element maxOccurs="1" minOccurs="1" ref="vmUsername"/>
    <xsd:element maxOccurs="1" minOccurs="1" ref="vmPassword"/>
    <xsd:element maxOccurs="1" minOccurs="0" ref="avEngine"/>
    <xsd:element maxOccurs="1" minOccurs="0" ref="analysisTools"/>
  </xsd:sequence>
  <xsd:attribute ref="vmxPath" use="required"/>
</xsd:complexType>
</xsd:element>
```

Listing 2.6: XSD VM element

The last two elements of the <VM> shows that a guest OS can run one AV engine, or one or more analysis tools. An AV engine has the following configurable properties;

```
<xsd:element name="avEngine">
  <xsd:complexType mixed="true">
    <xsd:sequence>
      <xsd:element maxOccurs="1" minOccurs="1" ref="avExecutablePath"/>
      <xsd:element maxOccurs="1" minOccurs="1" ref="avParameters"/>
      <xsd:element maxOccurs="1" minOccurs="1" ref="avLogFilePath"/>
      <xsd:element maxOccurs="1" minOccurs="0" ref="avLogFilter"/>
      <xsd:element maxOccurs="1" minOccurs="0" ref="avUpdateInfo"/>
      <xsd:element maxOccurs="1" minOccurs="0" ref="realTimeScan"/>
```

```
    </xsd:sequence>
    <xsd:attribute ref="name" use="required"/>
  </xsd:complexType>
</xsd:element>
```

Listing 2.7: XSD avEngine element

The AV engines are used for on-demand scan, and the first two parameters are used to configure the command used to initiate the scan. The third parameter, <avLogFilePath>, is the path to the log or result file in which the result of the on-demand scan is written. This may be a standard log file that the AV engine uses, or the console output piped to a temporary file. The <avLogFilter> element is used to filter the result file looking for specific words that indicate a malware hit. It is defined as follows:

```
<xsd:element name="avLogFilter">
  <xsd:complexType mixed="true">
    <xsd:sequence>
      <xsd:element maxOccurs="1" minOccurs="1" ref="avResultIdentifier"/>
      <xsd:element maxOccurs="1" minOccurs="0" ref="avResultPrefix"/>
      <xsd:element maxOccurs="1" minOccurs="0" ref="avResultSuffix"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Listing 2.8: XSD avLogFilter element

At regular intervals, the AV engines need to update their definition/signature database. This is hard to schedule using automatic updaters, as the guest OSs are reverted to snapshot at unpredictable intervals (typically after each use). To deal with this, PowerScan supports automatic update of all registered engines. The <avUpdateInfo> element contains path to the update executable and parameters, a path to a log file and some word or words to look for in the log that indicate a successful update operation. Note that this element is optional, as some engines may not support automatic updates via the command line.

```
<xsd:element name="avUpdateInfo">
  <xsd:complexType mixed="true">
    <xsd:sequence>
      <xsd:element maxOccurs="1" minOccurs="0" ref="avUpdateExecutable"/>
      <xsd:element maxOccurs="1" minOccurs="0" ref="avUpdateParameters"/>
      <xsd:element maxOccurs="1" minOccurs="0" ref="avUpdateLogPath"/>
      <xsd:element maxOccurs="1" minOccurs="0" ref="avUpdateSuccessIndicator
          "/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Listing 2.9: XSD avUpdateInfo element

The <realTimeScan> element of the <avEngine> is, as the name indicates, configuration parameters for the use of real-time scan functionality. The malware sample is executed within the guest OS, and the real time scan log is parsed to look for indicators showing the the executable has been recognized as containing malware. Although real-time scanner alerts often are shown in GUI pop-ups, PowerScan relies on parsing the log file of the anti-virus solution in order to determine if an infection was noted. This is because the Vix framework does not support GUI interaction.

In addition to scanning the malware sample with an AV scanner, it is also possible to set up guest OSs running one or more analysis tools. These tools are often executed with the malware sample as a parameter, but in some cases the analysis tool is started before the sample is executed. The <VM> element contains an <analysisTools> element, which is a list containing one or more <dynamicAnalysisTool> elements. The <dynamicAnalysisTool> element contains the configuration parameters for an analysis tool, and is defined as

```
<xsd:element name="dynamicAnalysisTool">
<xsd:complexType mixed="true">
  <xsd:sequence>
    <xsd:element maxOccurs="1" minOccurs="1" ref="toolExecutablePath"/>
    <xsd:element maxOccurs="1" minOccurs="0" ref="toolParameters"/>
    <xsd:element maxOccurs="1" minOccurs="0" ref="toolResultFilePath"/>
    <xsd:element maxOccurs="1" minOccurs="0" ref="executeMalwareExplicitly"/
        >
  </xsd:sequence>
  <xsd:attribute ref="toolName" use="required"/>
</xsd:complexType>
</xsd:element>
```

Listing 2.10: XSD dynamicAnalysisTool element

This element contains a path to start the tool, a string that may contain parameters, a log file path to be able to copy back the result and an element called <executeMalwareExplicitly> which is used to indicate whether the malware should be executed explicitly, or if it is supplied as a part of the parameter string. If the malware path needs to be included as part of the parameters, it can be inserted using the string $malwarePath.

An example XML config file can be found in section 2.1, and the appurtenant XSD file in 2.2.

2.4 Properties file example

```
## VMware host setup
## Note that all times are given in seconds.

## Virtual machine setup
# Specify timeout for waiting for tools to start in Virtual
# Machines (tools is required for operation)
vmware.tools.timeout = 30

## Scanner parameters
# Specify the path on the virtual machines in which to execute the malware
    sample.
# Note that backslashes (\) will have to escaped by another backslash, such
# that c:\program files\ becomes c:\\program files\\.
# Also note that the directory must already be created on the virtual
    machine,
# as VIX 1.0 and 1.1 does not support directory creation.
# The execution path must end with a file separator (typically slash (/) or
    backslash (\)).
scanner.executionpath = c:\\

# Specify whether snapshots should be taken before performing scans.
# This is generally NOT recommended, as storing snapshots is a time
    consuming
# operation, which should only be performed when changes are made to the
    Virtual Machine.
# It may also lead to instabilities in the virtual machines and cause scan
    failures.
snapshot.before.scan = false

# Set polling interval for minor operations
polling.interval.minor = 0.250

# Set polling interval for major operations
polling.interval.major = 1.000

# Maximum time to allow for individual scan operations to finish.
scanner.timeout = 60

# Maximum time to allow malware to execute with real-time anti-virus scanner
     running
malware.execution.timeout = 25

# Maximum total time for malware execution operation (for all registered
    scanners)
full.execution.timeout = 240

# Maximum time to allow for updating to finish
update.timeout = 600

# Timeout for full scan. Used to prevent entire system from crashing
# following failure in a single thread.
full.scan.timeout = 600

# Determine whether log files should be overwritten when program is executed
    .
# Note that log files may take up significant resources over time.
log.overwrite = false

## Log files
```

```
# VMware calls
log.vmware = logs\\vmware.log

## Scanner log (scan logic)
log.system.scanner = logs\\scanner.log

## System log (main thread)
log.system = logs\\system.log

## Executor log
log.system.executor = logs\\executor.log

## Executor path to location of malware sample
# Note that backslashes (\) will have to escaped by another backslash,
# such that c:\program files\ becomes c:\\program files\\.
# Also note that the directory must already be created on the virtual
# machine, as VIX 1.0 and 1.1 does not support directory creation.
# The execution path must end with a file separator (typically slash (/) or
    backslash (\)).
executor.executionpath = c:\\

# Path to which the result files from the analysis tools should be copied
# on the local system
executor.localResultPath = c:\\test\\

# Sets the time in seconds for which the operator is allowed to interact
# with the malware sample/analysis tools
executor.overall.timeout = 30

# Sets the path for the XSD file used to validate the XML config file
xml.xsd.path = config.xsd
```

Listing 2.11: Example PowerScan properties file