J Comput Virol
DOI 10.1007/s11416-006-0016-y

ORIGINAL PAPER

How to assess the effectiveness of your anti-virus?

Sébastien Josse

Received: 13 January 2006 / Revised: 11 February 2006 / Accepted: 16 April 2006

Abstract I will present an approach whose purpose aims at supporting and making easier and more relevant the choice of an anti-virus product. Among the qualities, which one can expect from an anti-virus product, appear classically the optimal use of the resources and the reactivity of the manufacturer, particularly concerning the viral signature base update. If these requirements are significant, other methodical and technical verifications may be required in order for an individual or a company to make their choice. In the Common Criteria evaluation scheme, a protection profile is proposed to help a software manufacturer to design a product that should be evaluated by an independent security evaluation laboratory. Protection profiles are written in accordance with the Common Criteria standard. Starting from a protection profile, we list some tests that could be carried out to validate the security requirements of an anti-virus product. Both use of a protection profile and the specification of tests seem to be a valuable basis to measure the confidence to grant an anti-virus product.

S. Josse (✉)
Laboratoire de virologie et de cryptologie
Ecole Supérieure et d'Application des Transmissions
B.P. 18, 35998 Rennes, France
e-mail: Sebastien.Josse@esat.terre.defense.gouv.fr

S. Josse
Silicomp-AQL,
1 rue de la Châtaigneraie,
Cesson-Sévigné, CS 51766, France
e-mail: Sebastien.Josse@aql.fr

1 Introduction

1.1 What is effectiveness?

End-users, when choosing an anti-virus, focus on classical qualities:

- Optimal use of the resources;
- Reactivity of the manufacturer.

Evaluation Lab analysts, when assessing the level of confidence to grant a security product, focus on:

- Conformity to standard criteria, shared by manufacturer and evaluation lab;
- Robustness against threat;
- Design and implementation quality, regarding a state-of-art in the field.

Starting from evaluator and manufacturer shared criteria, functional tests allow to validate mandatory security requirements.

Technological monitoring of threat, by using vulnerability database, like CVE [14], BugTraq [32] or Vigil@nce, enable to assess robustness of a security product, by developping adapted penetration tests (pen-tests).

Both functional tests and pen-tests permit to assess design and implementation quality.

Our definition of effectiveness cover these two points-of-view (End-user and Security evaluator):

The effectiveness of an anti-virus product rests on conformity to defined criteria, which corresponds to the state-of-the-art in the field and to an expression of need for security. A test bench must be used as a support for the analysis of security functions and mechanisms. The effectiveness of the anti-virus product is evaluated according to the robustness of its elementary mechanisms.

1.2 How to assess?

Choosing an anti-virus product may be difficult. One of the reasons is that internal mechanics of these products (in particular algorithms used and design features) are generally not well documented.

Consequently, how can one be sure to make the right choice?

Some companies grant anti-virus products with awards or quality certificates.

A first example is the VB100% Award [44]. To get this label, an anti-virus product must detect all viruses from a pre-set list (In-the-Wild[1] viruses) and must generate no false positives when analyzing a set of clean files.

A second example is the WCL Checkmark [46]. To get the WCL Checkmark, level 1, an anti-virus product must detect all In-the-Wild viruses and should not cause any false alarms when analyzing a set of clean files. To get the WCL Checkmark, level 2, the product must in addition get rid of all viruses in the Wild-List (which are capable of disinfection). Additional checkmarks can be provided by WCL, which assess the detection rate against specific malware types (WCL Trojan Checkmark, WCL Spyware Checkmark).

A third example is the ICSA Labs Certificate [28]. To obtain the certificate, an anti-virus product must satisfy all requirements in the primary criteria module, which correspond to its product category (for example, Desktop/Server anti-virus product category). Testing against the secondary criteria module (cleaning module) is optional. Virus Certification Test Suites Matrix maps anti-virus types towards several virus collections (In-the-wild, common infectors and Zoo).

Virus Bulletin, West Coast Labs, and ICSA Lab tests focus on detection rates. All these laboratories use (among additional virus collections) the well-known Wild-List for their tests. Let us focus on ICSA Labs.

ICSA Labs' own certification criteria are quite similar to the Common Criteria: anti-virus products satisfying the requirements in the primary module should be capabe of detecting and preventing the replication of viruses, reporting no false positives, logging the results of virus detection attempts, and performing necessary administrative functions. These requirements can be seen as a subset of functional Common Criteria requirements that are described later in this document.

However, ICSA Lab's approach does not take into account the insurance components related to the design choice or the quality of the product development. ICSA Lab certification criteria are based neither on fundamental design and engineering principles, nor on an assessment of underlying technology. Their approach is a black-box approach, which is results-oriented. The main advantage of ICSA Lab's approach is that test procedures are easily automated and repeatable for an anti-virus product. The main drawback is that the tests are not accurate enough to increase the evaluation assurance level, as they are not open to interpretation and analysis. It is at the same time a quality and a limitation. For example, these criteria are not precise enough to make a product source code review possible.

Moreover, ICSA Labs' certification criteria do not take the environment requirements of the target of evaluation into consideration. The scope of their analysis is limited to the product. Finally, it remains difficult to know which tests have been carried out according to the criteria chosen by these companies.

In this paper, an approach is described for making the choice of an anti-virus product easier and more relevant. Starting from a protection profile, we list some tests that should be carried out to validate the security requirements of any anti-virus product.

Our approach is complementary to those introduced by the laboratories previously mentioned. We focus on the robustness of security functions and mechanisms with regards to the specific vulnerabilities of this product category. We enlarge the scope of their evaluation by taking into account system environment and internal design.

1.3 Results

It appeared important to us, regarding to the current viral threat, to provide ourselves with tools to reproduce or simulate a viral attack. Thus, several characteristic viral mechanisms were implemented. It should be noted that these viral techniques are applied more and more often (by the researchers) to the validation of new viral intrusion behavioral detection techniques. These viral techniques were applied to the validation of security functions of the anti-virus products. Most complete achievements on this topic are: the implementation of an obfuscation engine and the implementation of a tool for injecting relocatable code remotely.

In the context of a sword-against-shield battle, it appeared important also to understand the constraints related to the implementation of an anti-virus product. Thus, several of the current components of an anti-virus product were specified and developed. Our most complete achievements on this point are: the implementation of an on-demand scanner of viral code and the implementation of a real-time monitor.

[1] The Wild-List is a list of viruses that are known to be active and widespread. Viruses of this list are usually called In-The-Wild viruses.

Finally, a family of tests (benchmark, functional tests, penetration tests) has been specified and has been used for improving the effectiveness of the security functions of an anti-virus product against the viral threat.

1.4 Structure of the document

Tools that have been adapted or developed within the framework of our analyses are described at the same time as they are used for the tests. Firstly, the analysis criteria used are presented, as well as the supporting elements of the selected tests. Secondly, the platform and the most representative tests are presented. Finally, we conclude with our achievements and results and provide future trends and topics to investigate.

2 Our approach

2.1 Analysis criteria

First of all, to compare products it is important to define a clear target, with regards to the category of the product and the characteristics of its environment. We used the scope defined by the protection profile: US Government Protection Profile Anti-Virus Applications for Workstations in Basic Robustness Environments [7].

A protection profile is an implementation-independent specification of information assurance security requirements. Protection profiles are a complete combination of security objectives, security related functional requirements, information assurance requirements, assumptions, and rationale. Protection profiles are written in accordance with the Common Criteria standard. In the Common Criteria evaluation scheme, a protection profile is made to allow a software manufacturer to design a product that should be evaluated by an independent security evaluation laboratory. Security analysts perform testing and security requirements rating. Several actors may take part in this process: security analysts, developers of the targeted anti-virus product, a third party in charge of arbitration. For example, in France, the third party is a governmental organization, which belongs to the Prime Minister's offices. The security evaluation laboratory must also be agreed by this third party.

This protection profile [7] specifies the minimum security requirements for an anti-virus application used on workstations in the US government in basic robustness environments. The target strength level (i.e., how well the target of evaluation can protect itself and its resources) of basic robustness environments is considered as sufficient for low threat environments or where compromise of protected information will not have a significant impact on mission objectives. This point is discussed in detail in CCEVS [7]. This protection profile is based on the Common Criteria, Version 2.2. The following figure (Figure 1) depicts the evaluation target and its environment.

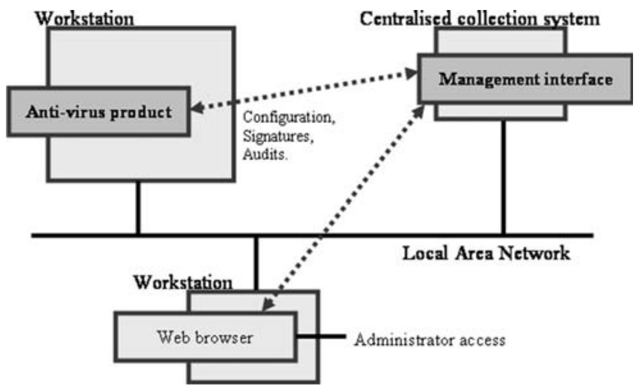

Fig. 1 Target of evaluation and its environment

This protection profile and Common Criteria have been used as a basis (method and terminology) for our analyses. For example, in this terminology, *virus* is used generically to refer to an entire suite of exploits of security vulnerabilities, such as Worms and Trojan Horses. The same term is used more specifically to refer to exploits that replicate themselves. The term *anti-virus* typically refers to measures used to counter the entire suite of exploits, not just the more specific definition of virus.

2.2 Selected tests

The analysis has been carried out of several off-the-shelf anti-virus products, which have been chosen according to their reputation or to the statute of the manufacturer on the software market. A set of tests, which were discriminatory enough to enable a comparative analysis of the products, has gradually been made.

The developed tests are intended to validate the security functions of an anti-virus product with regards to the security requirements defined in the protection profile. These requirements relate either to the security functions of the anti-virus product, or to its environment. Additional requirements, relating to the insurance components, make it possible to validate the quality of the software development and the technological awareness of the anti-virus product manufacturer. At the end of this section the security requirements (functional, environment, quality) and the corresponding validation tests are given. Tests are described in the second part of this document.

The tables (Tables 5, 6 in the Appendix) points out the principal security requirements relating to the security functions of an anti-virus product. A more detailed description of these requirements can be found in CCEVS [7]. The tests, which make it possible to validate these security functions (in comparison with the security requirements), are summarized in the following table (Table 1). A detailed description of these tests can be found in the second part of this document.

The table (Table 7, Appendix) points out the main security requirements relating to the environment of the target. The following table (Table 2) presents the tests, which permits the robustness of the security functions to be validated with regard to the requirements relating to the environment of the target.

The table (Table 8 in the Appendix) points out the main security requirements relating to the quality assurance components. The following table (Table 3) presents the tests, which make it possible to check that the requirements relating to quality are covered.

Table 1 Security functions requirements tests

Functional requirement	Test
FAV_ACT_EXP.1	Anti-virus actions
FAV_ACT_EXP.1.1	TST_MEM
FAV_ACT_EXP.1.2	TST_VXDAT
FAV_ACT_EXP.1.3	TST_MAIL
FAV_SCN_EXP.1	Anti-virus scanning
FAV_SCN_EXP.1.1	TST_MEM
FAV_SCN_EXP.1.2	TST_VXDAT, TST_HEURISTIC,
FAV_SCN_EXP.1.3	TST_FMT, TST_DOC,
FAV_SCN_EXP.1.4	TST_UNICODE
FAU	Security audit
FAU_GEN.1-NIAP-0347	TST_AUDIT
FAU_SAR.1	
FAU_SAR.2	
FAU_SAR.3	
FAU_STG.1-NIAP-0429	
FAU_STG.NIAP-0414-NIAP-0429	
FCS	Cryptographic support
FCS_COP1	TST_INTEG
FMT	Security Management
FMT_MOF.1	TST_UPDATE, TST_AUDIT
FMT_MTD.1	
FMT_SMF.1	
FMT_SMR.1	
	Protection of the security functions
FPT_SEP_EXP.1	TST_INSTALL, TST_INTEG

Table 2 Environment requirements tests

Environment requirement	Validation test
FAU	Security audit
FAU_STG.1-NIAP-0429	TST_AUDIT
FPT	Protection of the security function
FPT_ITT.1	TST_INSTALL, TST_INTEG,
FPT_RVM.1	TST_UPDATE
FPT_SEP.1	
FPT_STM.1	

Tools have been implemented to refine our understanding of the products. The latter series of tests also enable us to increase our level of comprehension of the various operations that are likely to be performed by any anti-virus product.

2.3 Supporting elements of the tests selected and the analysis criteria used

2.3.1 Theoretical framework

The problems of virus detection are, from a formal point of view, rather frozen:

- There is no system of detection/eradication which makes it possible to identify a program as being malicious without errors [13];
- There is no algorithmic transformation which makes it possible to make the code of a malicious program semantically unintelligible [3].

As the underlying theoretical models (viral set model, obfuscator model) today have not been seriously questioned, we consider the problems of virus detection theoretically frozen.

2.3.2 Brief history of viral threat

I will give here a short overview of viral threat evolution:

- Encrypted: virus consists of a constant decryptor and the encrypted virus body. Virus body is constant;

Table 3 Assurance requirements tests

Quality requirement	Validation test
ACM	Configuration management
ACM_CAP.2	TST_INSTALL
ADO	Delivery and operation
ADO_DEL.1	TST_INSTALL
ADO_IGS.1	
ADV	Development
ADV_FSP.1	TST_INSTALL, TST_HEURIST,
ADV_HLD.1	SRC_CODE_REVIEW
ADV_RCR.1	
AGD	Guidance documents
AGD_ADM.1	TST_INSTALL
AGD_USR.1	
ALC	Life cycle support
ALC_FLR.2	TST_UPDATE
ATE	Tests
ATE_COV.1	TST_MAIL, TST_HEURIST,
ATE_FUN.1	TST_MEM, TST_VXDAT,
ATE_IND.2	TST_AUDIT
AVA	Vulnerability assessment
AVA_MSU.1	TST_FMT, TST_UNICODE,
AVA_SOF.1	TST_DOC, TST_INTEG
AVA_VLA.1	

- Oligomorphic: different versions of virus have different encryptions of the same body;
- Polymorphic: virus contain a polymorphic engine for creating new keys and new encryptions of its body. Virus body is constant;
- Metamorphic: obvious next step. The virus mutates its body too.

Metamorphic mutation techniques:

- Register swap: different register name;
- Substitutions: instruction sequences replaced with other instructions that have the same effect, but different opcodes;
- Permutation engine: instructions are reordered within a basic bloc (intra-procedural transformation) or subroutines are reordered (inter-procedural transformation);
- Dead junk code inserted in unreachable code areas;
- Alive junk code, more robust by use of open predicates;
- Code integration: virus integrates/merges itself into the instruction flow of its host, by using a disassembler and an engine which enables to rebuild/re-generate the host binary.

Armored viruses: slowing down the AVers work, by using classical or tricky software protection techniques, specific cryptographic architectures and key management infrastructures:

- Delayed code;
- Environmental Key Generation [22];
- Environmental Code Generation [2];
- Anti-disassembler;
- Anti-debug;
- Anti-dump;
- Anti-emulator: huge random code block insertion at entry-point. Emulator executes code for a while, does not see virus body and decides the code is benign. When main virus body is finally executed, virus propagates;
- Integrity checking by using White-Box Technology;
- Interleaved cipher layers;
- Etc.

2.3.3 AVers strike back

We give here a short overview of anti-virus reactions as regard to the viral threat evolution:

- Encrypted/Oligomorphic viruses: signature based scanner;
- Polymorphic viruses: heuristic engine and emulation;
- Metamorphic viruses, armored viruses: expert systems, by using model checking.

The architecture of the detection systems must evolve and progress, from the simple pattern search engine to an expert system,[2] which formulate its diagnosis (i.e., answer the question: is this program viral?) only after having correlated several detection models [21]. The evaluation of security products dedicated to viral detection must be adapted to this theoretical and technological context.

Constant technological monitoring must be maintained, concerning:

- The viral threat and its development;
- Anti-virus product bypassing modes: propagation vectors, stealth techniques, protection techniques, polymorphism;
- New antiviral techniques;

Adapted tests have to be specified. The specified tests must take into account:

- The new viruses and the algorithmic innovations [16];
- The new antiviral techniques (is the anti-virus product state-of-the-art?);
- The corrections of the published vulnerabilities (Conformity of the anti-virus product);

The specified tests must also report:

- The performance of the selected and implemented algorithms by the anti-virus product (naive, worked out, complexities. Is the anti-virus product user-friendly?);
- The resistance to the black box analysis (security of implementation, robustness of the anti-virus product) [23].

Both the use of a protection profile (validated by the methodological approach of the Common Criteria) and the specification of tests (which give an account of the cover of the security requirements) seem to be a valuable basis to measure the confidence to grant an anti-virus product.

[2] An expert system is able to carry out a diagnosis by comparing information, which results from one system with a priori knowledge available on this system (as a human expert would do). An expert system is made up of at least a knowledge base, a facts and rules base, and of an inference engine. The inference engine chooses the rules according to the observed facts.

2.4 Description of the tests platform

We have positioned our anti-virus tests in relation to a fixed attack context: the techniques were studied on a PC Intel/MS. Windows platform, because of its widespread distribution (however, the majority of the tests produced for this platform can be transposed without difficulty to other platforms). The test platform must comprise:

- Virtualisation software, like VMware [45];
- Network probe tools;
- System diagnostic tools [12];
- Various tools possibly dedicated to static and dynamic code analysis.

Regarding the Workstation environment parameter setting, the operating system must be re-installed and updated. No other antivirus must be installed.

3 Tests

We describe in this part the tests set which makes it possible to validate the security requirements relating to the security functions of an anti-virus product, its environment and the components of insurance relating to its design. Table 4 below presents a summary of the tests, which are described in more detail way in the rest of the document.

3.1 TST_INSTALL (Installation of the anti-virus product)

3.1.1 Goal

This test table is divided into two parts:

- A part dedicated to recovery of information on the installation of the anti-virus product (interactions with the API and NT register). These tests are carried out during the installation;
- A part which makes it possible to test the robustness of the installation and its robustness against low level viral attacks (file system filter drivers, for example). These tests are carried out after the installation.

3.1.2 Principle of operation

We automated the recovery of useful information at the time as the installation of the product by implementing techniques of listening to and bypassing calls according to the API functions. There are many methods that allow the code injection into the system environment or into a process context, as well as the interception and the redirection of API Win32 function calls towards the injected code [6,18]. Trojan horses and spyware very often implement these furtive techniques. These techniques are also sometimes useful for tracing system calls (in particular I/O towards the file system and the register base) made by a viral program or to make certain protection software programs inoperative. These protections have been implemented for example by armoured viruses [22]. It should be noted that these techniques of instrumentation and interception of calls to API Win32 functions are also applied to the behavioral detection of viral intrusion – cf. DOME, Detection of Malicious Executable [15], and even to the implementation of real-time control engines [48]. We use some of these techniques to automatically trace system calls carried out during the installation of the anti-virus product, in particular those taking part in the installation of filtering drivers (use of the detours tools [6]).

We have also evaluated the robustness of the installation by testing (enable/unload or uninstall) the filter drivers of the anti-virus product (using fltmc and devcon tools [27,47]).

3.1.3 Analysis of the results

After the first step, we should obtain a snapshot of the anti-virus product installation and a trace of its interactions with several system components during installation process. After the second step, we should have a good idea of the robustness of the anti-virus product, concerning its drivers stack and register keys.

3.1.4 Other ideas

A lot of information can be retrieved by analysing the interactions of the installation process with the core components of the system (Kernel API, COM bus, etc.). Tricky tests could be carried out to bypass the filter or minifilter kernel mode driver of the anti-virus product (instrumentation of the driver stack), and thus assess more completely the robustness of installtion. Other techniques can be used to instrument the API, like call replacement in anti-virus installation function binary code or break-point trapping. The main constraint is that the TST_INSTALL test bench remains automatic and could apply to any anti-virus product.

3.2 TST_HEURISTIC (heuristic engine)

3.2.1 Goal

The objective of this test is to validate the correct operation of the heuristic engine of a given anti-virus product.

This test also enables us to better understand the actions performed by the heuristic engine.

3.2.2 Principle of operation

We consider a S_0 population of healthy programs of the system. We generate a $S_1 = T(S_0)$ population of healthy programs (Win32 executable files) having properties generally considered as suspect by any anti-virus software regarding:

- The structure of the executable file (header, sections of data and code);
- The executable file's section code.

The T transformation was firstly implemented by using an adapted version of Y0da's packer [49]. This transformation is actually performed by the obfuscator described below, because its modularity is better. The executable files generated by the T transformation have characteristics incompatible with the code usually generated by a compiler of high-level language:

- Sections are not correct (for example the last section is executable or the first section is accessible in writing, sections are incorrectly aligned or with strange names);
- Header is not correct (incorrect values in the header, unusual entry point).

We considered a V_0 population made up of Win32 viruses. We built a population $V_1 = O(V_0)$ by application of an obfuscator O. We proceeded, as for the validation of the SAFE (Static Analyser For Executable) viral detection engine [9], with an adaptation of the Mistfall engine [50,51]. Let us note that this idea was also used to validate the effectiveness of the SAVE viral detection engine [8], even if the authors did not automate the application of the obfuscation transformations (the transformations were applied manually, using a simple hexadecimal editor). The viral engine Mistfall implements a disassembler (LDE) and a permutation engine (RPME). We instrumented this engine so that it carries out more complex obfuscation transformations [29]. We coupled it with a generator of junk code and a base of opaque predicates.

3.2.3 Analysis of the results

The correlated analysis of the S_1, V_0 and V_1 populations makes it possible to draw several conclusions on the anti-virus product:

- Its capacity to analyze polymorphic viruses;
- its capacity to analyze Entry-Point Obscuring (EPO) viruses [25].

It is also possible to see which are the heuristic ones used, as well as the limits imposed on the level of the parameter setting by the manufacturer of the anti-virus product (depth of the analysis, etc).

3.2.4 Other ideas

We did not explore the possibilities offered by the generators of viral code (Virus Generator Kits [33]). Their use could supplement our analysis (generation ex nihilo of a viral population V_2. Nor did we explore the possibilities offered by the fact of having the source code of certain viruses. We could, starting from these programs, build a V_3 population by application of an obfuscator O like Cloakware for the languages C/C++ and Java [11] or SandMark for the Java language [31]. For example, the LLVM compilation chain [30] seems well suited to the application of obfuscation transformations on C source code.

3.3 TST_MEM (Read-write memory analysis/worm blocking)

3.3.1 Goal

The objective of this test is to validate the operation of the anti-virus product, regarding the read-write memory. This test makes it possible to check that a virus present in the memory will be well detected by the anti-virus product.

3.3.2 Principle of operation

We use a server process program authorizing the exploitation of a buffer overflow [24]. This type of vulnerability is used as a worm propagation vector. We inject relocatable[3] viral code from a client program.

3.3.3 Analysis of the results

The anti-virus software must detect the presence of a data-processing infection in the memory. Some antiviruses have stack/heap overflow monitoring capabilities. This functionality can be validated.

[3] def: remote code injection, exploitation of a buffer overflow, relocatable code, viral shellcode

3.3.4 Other ideas

The techniques used for remote code injection can be improved to account for new generation exploits. Using LSD or Metasploit framework for pen-tests can be a complementary approach.

3.4 TST_FMT (Management of the various file formats)

3.4.1 Goal

Some anti-virus products do not correctly analyze certain file formats. They are classified unknown instead of being decoded and analyzed. An attacker can thus employ illicit files in order to pass on a virus. This test enables us to check if the anti-virus product suitably manages the various formats of file and compression, in particular when these formats are corrupted intentionally.

3.4.2 Principle of operation

We use GNU Win32 software to archive, compress or pack a virus recognized by the anti-virus product (ARC, ARJ, BASE64, BZIP2, GZIP, LHA, LZO, ms-compress, SHAR, TAR, UPX, UUE, ZIP file formats or extensions). We carry out the analysis of the directory containing the packed, archived and/or compressed files.

Case of UPX format: UPX format [34] is used to compress a program and uncompress it before its execution. Some anti-virus products cannot recognize such custom UPX files as viral. A remote attacker can thus create a virus, compress it in UPX and then apply a treatment to it so that it is still recognized by the system as being a UPX file, but no longer by the anti-virus product (it is an evasion attack). The antivirus software can consider the format as being unknown or the file as being corrupted and not declare the presence of a virus to the user. Consequently, the now confident user may decide to run it. We have adapted the executable UPXRedir in order to be able to transform a UPX file into a file recognized as UPX by the system but not by the anti-virus product (when this one does not manage this format in a correct way). Thus the test consists of compressing a viral executable file with UPX format, ensuring that the anti-virus product detects the virus in this format and then corrupting the executable file before again subjecting it to the analysis.

Case of the Zip format: By using a custom Zip file [39,40], an attacker can circumvent some anti-virus software. The Zip files are compressed files containing one or more documents and/or directories. A header precedes each one of these documents. In the same way, a summary of all the files appears at the end of the archive. An attacker can modify the file sizes appearing in the header and the summary. Thus, an additional document can be hidden in the file. WinZip and the Windows decompression utility extract the hidden file. However, some anti-virus products do not detect it. Any attacker can thus create a file containing a virus that will not be detected before reaching the user's workstation. We have developed a tool that corrupts an archived file with the Zip format, while zeroing the Uncompressed Size fields of the Local File Header structures and the Central Directory of the Zip file.

3.4.3 Analysis of the results

All the formats or encoding must be supported. Case of intentionnaly corrupted file formats: some anti-virus products do not properly inspect a file using the illicit Zip and UPX format, and are thus vulnerable to this evasion attack.

3.4.4 Other ideas

Zip and UPX formats are not the only formats that raise these problems. The following formats are also subject to evasion attack: RFC822 [35], MIME [36], LHA [38], URI [41], ANSI ESC [42], RAR [43]. As a next step, we envisage building necessary tools to make sure that these formats are well managed by the anti-virus product.

3.5 TST_DOC (Filtering of the document viruses)

3.5.1 Goal

This test checks that an anti-virus product suitably detects the embedded viral code of an Office or HTML document.

3.5.2 Principle of operation

Users often reactivate the execution of macros, scripts or the applet in an Office application or a navigator, to make the default configuration of the software more flexible. We have adapted the conformity tests suggested by eSafe to validate its script-filtering engine [20].

3.5.3 Analysis of the results

The anti-virus software must be able to notify the presence of viral code embedded in a document.

3.6 TST_MAIL (Analysis of incoming and outgoing e-mails)

3.6.1 Goal

This test validates functions analyzing incoming/outgoing e-mails. Anti-virus product will monitor processes attempting to send or receive e-mails and may block traffic upon detection of a virus.

3.6.2 Principle of operation

The scenario of the test is simple: send and receive an e-mail containing an attached viral file. Initially, the procedure described above can be carried out with the EICAR test file [19]. The EICAR test file is a test file allowing vendors to calibrate their systems. It should be noted that a user without particular technical knowledge could carry out this test very easily. Secondly, the procedure described above can be repeated with other viruses in attachment (highly compressed virus file, intentionally corrupted archive formatted virus files, viral code embedded in a document, etc.).

3.6.3 Analysis of the results

According to the protection profile description of anti-virus actions [7] and the state-of-the-art in the field of anti-virus protection, we expect the anti-virus product to detect and disinfect or remove the attachment.

3.6.4 Other ideas

Using the remote injection of relocatable code client/server to validate security functions analyzing incoming/outgoing e-mails, by using SMTP and POP/IMAP protocols.

3.7 TST_VXDAT (Viral base analysis)

3.7.1 Goal

The objective of this test is to validate the completeness of the viral signature base of the anti-virus product.

3.7.2 Principle of operation

We have a virus base (about 30,000 uniquely identified viruses, at the time of analysis). We set up the scanner in order to target the analysis on the directory containing the viral base. In order to facilitate the maintenance of the viral base, we have developed a real-time monitor, which blocks the execution of the viral code (in the case of accidental access to a viral program) and alerts the administrator of the viral base.

This real-time monitor embeds three components:

- A client program, which recovers the I/O carried out from user space towards the file systems and carries out their analysis while applying:
  - The Aho/Corasick pattern matching algorithm [1, 10] taking into consideration viral base signatures;
  - The FDA/NFA pattern matching algorithm taking into consideration a list of rational PCRE expressions [26];
- A kernel mode driver, which makes it possible to intercept the IRP and the fast I/O, communicated by the I/O manager towards the file systems drivers (FAT, NTFS, RDR[4]).

Once a virus is detected, the kernel driver blocks the access to the file system and the user is notified by the client program. The client program proceeds with the loading of the signature base, with the initialization of the motif search structures, with the opening of a communication channel towards the driver and with the creation of threads. Each thread connected to the driver recovers data from it and carries out the analysis (by pattern matching). Upon virus detection, the thread raises an alarm. Then the client program sends a message to the driver that results in imposing a blocking of the I/O request by the driver.

The driver is registered at the NT executive, and then creates a port communications server on which it can receive connection requests from the user mode. It includes in the port a security descriptor limiting access to the administrator. It can then intercept all the packets (IRP, Fast I/O) coming from the I/O manager towards the file system drivers (FAT, NTFS, RDR). When an I/O request towards the file system is performed in user space, the driver retrieves information on the file, reads the beginning of this file, passes the contents into user space and sends an analysis request. If the file is suspect, then the I/O request is blocked. If the file is not suspect and the I/O request corresponds to a file opening in write mode. Then the file will be analyzed again. The context is recovered. If a new analysis is required, the driver reads the beginning of the file again, passes the contents into the user space and sends a new analysis request. If the file is suspect, then the writing request is blocked.

[4] RDR File Systems are NT Redirector, which permit Workstation services to access remote files towards I/O Manager. Redirectors are above TDI Transport Driver Interface, in a user mode/kernel mode representation of WinNT layered network architecture.

This real-time monitor locks the files. A file opened by a process will thus not be able (before being closed) to be read and executed by another process [17]. This program carries out on-demand analysis of whole or part of the viral base, with regards to the signature base.

3.7.3 Analysis of the results

The analysis of the virus base is carried out with several configuration profiles (heuristic engine enabled/disabled, analyze of archived files, default configuration, etc.). The detection rates obtained are a valuable information, which allows assessments of the completeness of the virus signature database of the product and to compare, on a result-oriented basis, several anti-virus products.

3.7.4 Other ideas

Neither did we explore the possibilities offered by the virus collection management tools, nor did we explore the possibilities offered by the use of an emulator, in order to check automatically that virus executables of a collection effectively run and correspond with their public description. For example, the command line tool TTAnalyze [4] seems well suited for analyzing the interactions of a virus executable with the Windows system. The virus executable is run under a QEMU based emulated environment [5]. The dynamic analysis (on a virtual processor) analyses self-modifying viruses, including packed and obfuscated executables. The QEMU emulator engine monitors the execution of the program and automatically gathers status information and interactions with the host system.

3.8 TST_UNICODE (Unicode support)

3.8.1 Goal

This test checks that the anti-virus product implements the Unicode format suitably. The incorrect implementation of this format is at the origin of certain evasion attacks. The objective of this test is to check if the anti-virus product detects viruses in long NT file system paths. Few anti-virus products may not be able to detect a virus that has an access path that is too long [37].

3.8.2 Principle of operation

The maximum size of the pathname is limited to 260 bytes for the directories and files created using the standard functions of API Win32. This maximum size is extended to 32,767 bytes for directories and files created using the Unicode functions of API Win32. Some anti-virus products are unable to detect viruses located in the directories whose path has a size greater than 260 bytes. Thus a virus may use this limitation to put a file which cannot be erased, and which may contaminate the machine each time it boots up. In a general way, the anti-virus products that do not use the API Win32 Unicode functions are not able to detect viruses located in tree structures of more than 260 bytes. We have developed an executable file which confines the viral code to the deepest part of a tree structure whose size is customizable. This executable file carries out the disinfection of the file system. Each time the viral code is hidden too deeply in the tree structure of the file system, some anti-virus products are no longer able to detect its presence.

3.9 TST_AUDIT (Audit files)

3.9.1 Goal

The objective of this test is to validate the correct operations of the audit function of an anti-virus product, regarding the types of events and informations that will be recorded. This test aims at examining the robustness of the audit function against modification of destruction of the stored audit records.

3.9.2 Principle of operation

We can, for example, build a test to check the audit files rotation function of the anti-virus software, when this one imposes a limited size on the logs files. The scenario of the test is simple: fill the log files and then check automatic rotations.

3.9.3 Analysis of the results

The following events will be recorded:

- Start-up/shutdown of the audit security function;
- Selection of an action by the user;
- Action taken in response to the virus detection.

The audit security function will record (within each audit record) the following information at least:

- Date and time of the event;
- Event type;
- Subject identity (for audit events resulting from actions of identified users);
- Outcome (success or failure) of the event;
- Action selected by a user or action taken in response to the detection of a virus;

- Description of the detected virus, file or process identifier where the virus was detected.

The audit security function should protect the stored audit records from unauthorized modification/deletion via the audit security function interface (the environment (OS) is responsible for preventing unauthorised modification/deletion of the audit file via OS administration interfaces). The audit security function shall provide the following options to prevent audit loss (if the audit trail is full):

- Ignore auditable events;
- Prevent auditable events, except those taken by the authorised user with special rights;
- Overwrite the oldest saved audit records.

3.10 TST_INTEG (Integrity of the viral signature base)

3.10.1 Goal

This test aims at examining the robustness of the signature file. The signature file of the product is often written in a proprietary format. It should not be possible to modify the contents of the file.

3.10.2 Principle of operation

We check the resistance of the product to restarting when one deteriorates the integrity of the viral signature base. The procedure is simple: we corrupt or affect the integrity of the viral signature base. The objective is to check the mechanisms that guarantee the viral signature base integrity and to check the behavior of the software, in terms of failure recovery. We perform several actions:

- Modification of the attribute: last modification date. Writing in one of the files of the viral signatures base;
- Deletion of one of the signature base files;
- Deletion of one of the signature base file entries;
- Substitution of an old base to the new one;
- If several versions of the signature database are present in the installation directory of the product, remove and/or modify the most recent version.

3.10.3 Analysis of the results

In all cases, the refusal to load the base must be accompanied by an alarm or a notification. If a sound (but less recent) version is loaded, the anti-virus product must clearly announce the fact that it did not manage to reload the viral signature base or that it is running in degraded mode. These tests thus validate the integrity auto-tests of each file of the viral signature base, the system of on-error recovery and the alarm system of the anti-virus product.

3.11 TST_UPDATE (Product updating)

3.11.1 Goal

An anti-virus product can be updated manually or by using the automatic update function. For the latter, the update is done on demand or in a scheduled way, at a frequency chosen by the user.

3.11.2 Principle of operation

We use a network analyzer (ethereal) to trace how the automatic updates are carried out. The analysis of the Ethernet frames provides a better understanding of the update process: client request, server response listing the definition files available, their signatures and the name of a server which can provide them, finally downloading the definition files, when necessary. An analysis of the symbols (strings) of the update executable file may confirm the fact that the name of the server is hard-coded in the program.

3.11.3 Analysis of the results

We are waiting for the use of a protocol that authenticates the client, the update server, with possibly mutual authentication. We expect (after downloading of the update files of the signature base and possibly of the anti-virus product analysis engines) an integrity check on the downloaded files. It may be necessary to restart the operating system or the anti-virus product. The (graphical) Man to Machine Interface must announce the modifications made into the viral signature base. The events related to the update of the product must be logged.

4 Conclusion and future works

We have proposed in this paper an approach and tools in order to help a single user or a company to be able to choose an anti-virus product off the shelf with a certain degree of confidence.

Both use of a protection profile (validated by the methodological approach of the Common Criteria) and the specification of security requirements validation tests seem to be a valuable basis to measure the confidence to grant an anti-virus product.

The main limitation induced by this approach is the objective interpretation of the tests. Even if the tests of our platform are automated, the results are sometimes firstly designed to enable the deepest investigation of the product internals. The verdict is not a binary “pass/do not pass”, but rather an input used by the security analyst to issue his comments, as regards the way the internal characteristics of an anti-virus product meet the state-of-the-art in the field of host-based intrusion detection systems and the security requirements of its product category.

Our approach may obviously be optimized. The test set must evolve on a regular basis, in order to take into account the technical developments of the two protagonists involved. A constant technological watch must thus be conducted and especially look for:

- Evolution of the viral threat, at the technical and algorithmic level;
- New techniques and anti-virus software architectures;
- Vulnerabilities related to the design and the implementation of the anti-virus product and its environment.

The protection profile (used as a methodological basis in this document) must also evolve to reinforce the level of security requirements and thus to increase the corresponding degree of confidence. A relevant test set will have to enable its implementation. Current Evaluation Assurance Level of a product claiming conformance to this PP is EAL2, which does not require an analysis of the source code. Functional and black box testing are enough in order to validate corresponding security requirements. Results demonstrate the potential intrinsic information that can be extracted about anti-virus internals, by grey box testing. In order to be able to establish a greater Assurance Level, more advanced testing tools and learning algorithms have to be specified and implemented. The author focus actually on fault injection systems and Grey Box analysis algorithms.

Appendices

Table 4 Functional and penetration tests

Test	Description
TST_INSTALL	Installation of the anti-virus product (recovery of information on the installation and test of its robustness)
TST_HEURISTIC	Heuristic engine (recovery of information on the heuristic engine and its characteristics)
TST_MEM	Read-write memory analysis / worm blocking (validation of the ability to check memory and to protect the host against virus propagation vectors)
TST_FMT	Management of the various file formats (validation of the ability to manage various file formats, in particular when these formats are corrupted intentionally)
TST_DOC	Filtering of the document viruses (validation of the ability to detect embedded viral code of an Office or HTML document)
TST_MAIL	Analysis of incoming and outgoing mails (functional validation of the ability to analyze SMTP traffic and e-mails’ attached files)
TST_VXDAT	Viral base analysis (validation of the completeness of the viral signature base)
TST_UNICODE	Unicode support (validation of the correct implementation of Unicode format)
TST_AUDIT	Audit files (functional validation)
TST_INTEG	Integrity of the viral signature base (validation of the robustness of the viral signature base)
TST_UPDATE	Product updating (recovery of information on the product updating process)

Table 5 Functional requirements (1/2)

Functional requirement	Description
FAV_ACT_EXP.1	Anti-virus actions
FAV_ACT_EXP.1.1	Upon detection of a memory-based virus, the anti-virus security function will prevent the virus from further execution.
FAV_ACT_EXP.1.2	Upon detection of a file-based virus, the anti-virus security function will perform the actions specified by the administrator. Actions are administratively configurable on a per-workstation basis and consist of: clean the virus from the file, quarantine the file, and delete the file.
FAV_ACT_EXP.1.3	The anti-virus security function will actively monitor processes attempting to access a remote system using TCP or UDP remote port 25 (SMTP) and block traffic from unauthorized processes.
FAV_ALR_EXP.1	Anti-virus alerts An alert shall be displayed on the screen of the workstation on which the virus is detected and on the screen of the administrator (upon receipt of an audit event). The alerts will continue to be displayed until they are acknowledged by the user or by the administrator.
FAV_SCN_EXP.1	Anti-virus scanning The anti-virus security function will perform real-time scans for memory-based viruses and real-time, scheduled and on-demand scans for file-based viruses. The administrator shall perform scheduled scans at the time and frequency configured by the administrator. The workstation user shall perform manually invoked scans.

Table 6 Functional requirements (2/2)

Functional requirement	Description
FAU	Security audit The anti-virus security function will be able to generate an audit record with suitable information, to provide the user and administrator with the capability to read all audit information, to perform and sorting of audit data. The stored audit records will be protected from unauthorised deletion, modification or loss (if the audit trail is full).
FCS	Cryptographic support
FCS_COP1	The anti-virus security function will calculate a message digest to verify the integrity of the signature files in accordance with a NIST FIPS 140-2 approved cryptographic algorithm.
FMT	Security management Management of security functions behaviour and data, specification of management functions and security roles. Protection of the security functions
FPT_SEP_EXP.1	Partial security functions domain separation.

Table 7 Environment requirements

Environment requirement	Description
FAU	Security audit
FAU_STG.1-NIAP-0429	Protected audit trail storage
FDP	User data protection
FDP_RIP.1	Subset residual information protection
FIA	Identification and authentication
FIA_AFL.1	Authentication failure handling
FIA_SOS.1	Verification of secrets
FIA_UAU.2	User authentication before any action
FIA_UAU.6	Re-authenticating
FIA_UID.2	User identification before any action
FPT	Protection of the security function
FPT_ITT.1	Basic internal security function data transfer protection
FPT_RVM.1	Non-bypassability of the security policy
FPT_SEP.1	Security function domain separation
FPT_STM.1	Reliable time stamps
FTA_SSL.1	Security function-initiated session locking
FTA_TAB.1	Default anti-virus product access banners

Table 8 Quality requirements

Quality requirement	Description
ACM	Configuration management
ACM_CAP.2	Configuration items
ADO	Delivery and operation
ADO_DEL.1	Delivery procedures
ADO_IGS.1	Installation, generation, and start-up procedures
ADV	Development
ADV_FSP.1	Informal functional specification
ADV_HLD.1	Descriptive high-level design
ADV_RCR.1	Informal correspondence demonstration
AGD	Guidance documents
AGD_ADM.1	Administrator guidance
AGD_USR.1	User guidance
ALC	Life cycle support
ALC_FLR.2	Flaw reporting procedures
ATE	Tests
ATE_COV.1	Evidence of coverage
ATE_FUN.1	Functional testing
ATE_IND.2	Independent testing - sample
AVA	Vulnerability assessment
AVA_MSU.1	Examination of guidance
AVA_SOF.1	Strength of anti-virus product security function evaluation
AVA_VLA.1	Developer vulnerability analysis

References

1. Aho, A.V., Corasick, M.J.: Efficient string matching: an aid to bibliographic search. Commun. ACM **18**(6), 333–340 (1975)
2. Aycock, A., DeGraaf, R., Jacobson, M.: Anti-disassembly using Cryptographic Hash Functions. In: Proceedings of the 15th EICAR Conference (2005)
3. Barak, B., Goldreich, O., Impagliazzo, R., Rudich, S., Sahai, A., Vadhan, S., Yang, K.: On the (im)possibility of obfuscating programs. In: Advances in Cryptology – CRYPTO '01, vol. 2139 of Lecture Notes in Computer Science, pp. 1–18, Santa Barbara (2001)
4. Bayer, U.: TTAnalyze: a tool for analyzing malware. Master's Thesis, Technical University of Vienna (2005)
5. Bellard, F.: QEMU, a fast and portable dynamic translator. In: Proceedings of USENIX 2005 Annual Technical Conference, pp. 41–46 (2005)
6. Brubacher, D., Hunt, G.: Detours: binary interception of Win32 functions. In: Proceedings of the 3rd USENIX Windows NT Symposium, pp. 135–143, Seattle (1999)
7. CCEVS: US Government Protection Profile Anti-Virus Applications for Workstations in Basic Robustness Environments. Version 1.0. (2005) http://niap.nist.gov/cc-scheme/pp/PP_VID10053-PP.html

8. Chavez, P., Mukkamala, S., Sung, A.H., Xu, J.: Static analyzer of vicious executables (SAVE). In: Proceedings of the 20th Annual Computer Security Applications Conference (ACSAC'04), pp. 326–334 (2004)
9. Christodorescu, M., Jha, S.: Static analysis of executables to detect malicious patterns. In: Proceedings of the 12th Usenix Security Symposium, pp. 169–186 (2003)
10. ClamAV (2006) http://www.clamav.net/
11. Cloakware (2006) http://www.cloakware.com/
12. Cogswell, B., Russinovich, M.: Sysinternals. (2006) http://www.sysinternals.com/
13. Cohen, F.: Computer viruses. Doctoral dissertation, University of Southern California, California (1986)
14. Common Vulnerabilities and Exposures (2006) http://www.cve.mitre.org/
15. Cunningham, R.K., Khazan, R.I., Lewandowski, S.M., Rabek, J.C.: Detection of injected, dynamically generated, and obfuscated malicious code. In: Proceedings of the 2003 ACM Workshop on Rapid Malcode (WORM), Washington, DC, pp. 76–82 (2003)
16. Dagon, D., Kolesnikov, O., Lee, W.: Advanced Polymorphic Worms: Evading IDS by Blending in with Normal Traffic. Georgia Institute of Technology, Technical Report (2005)
17. Das, A., Miretskiy, Y., Wright, C.P., Zadok, E.: Avfs: an on-access anti-virus file system. In: Proceedings of the 13th USENIX Security Symposium (2004)
18. Detoisien, E., Dotan, E.: Cheval de Troie furtif sous Windows: mécanismes d'injection de code. MISC Magazine n∘10 (2003)
19. European Institute for Computer Anti-Virus Research (EICAR) (2006) http://www.eicar.org/
20. eSafe eSafe test page. (2006) http://www.esafe.com/home/csrt/eSafe_Demo/TestPage.asp
21. Eskin, E., Schultz, M.G., Stolfo, S.J., Zadok, E.: Data mining methods for detection of new malicious executables. In: Proceedings of the 2001 IEEE Symposium on Security and Privacy. IEEE Computer Society, Washington, DC (2001)
22. Filiol, E.: Strong cryptography armoured computer viruses forbidding code analysis. In: Proceedings of the 14th EICAR Conference, pp. 216–227 (2005)
23. Filiol, E.: Malware scanning schemes secure against black-box analysis. In: Proceedings of the 15th EICAR Conference (2006)
24. Frej, P., Ogorkiewicz, M.: Analysis of Buffer Overflow Attacks. (2004) http://www.windowsecurity.com/
25. GriYo: EPO: Entry-Point Obscuring. (2006) http://vx.netlux.org/lib/vgy01.html
26. Hazel, P.: Perl-Compatible Regular Expressions. (2003) http://www.pcre.org/
27. IFSKit: Installable File System Kit. (2006) http://www.microsoft.com/whdc/devtools/ifskit/
28. International Computer Security Association Labs (2006) https://www.icsalabs.com/
29. Josse, S.: Techniques d'obfuscation de code: chiffrer du clair avec du clair. MISC Magazine n∘20, pp. 32–42 (2005)
30. Low Level Virtual Machine (2006) http://llvm.cs.uiuc.edu/
31. SandMark (2006) http://www.cs.arizona.edu/sandmark/
32. Security Focus Bugtraq (2006) http://www.securityfocus.com/bid/
33. Szor, P.: Advanced Code Evolution Techniques and Computer Virus Generator Kits. Addison Wesley, Reading (2005)
34. Ultimate Packer for eXecutables (2006) http://upx.sourceforge.net/
35. Vigil@nce: Outlook accepts messages whose format does not respect the RFC 822 (Standard for the format of ARPA Internet text messages). BUGTRAQ-5259, CVE-2002-0637. (2006a) http://vigilance.aql.fr/
36. Vigil@nce: Incorrect analysis of MIME messages. BUGTRAQ-9650, CVE-2004-2088. (2006b) http://vigilance.aql.fr/
37. Vigil@nce: Incorrect Unicode support. BUGTRAQ-10164. (2006c) http://vigilance.aql.fr/
38. Vigil@nce: Incorrect analysis of LHA files. BUGTRAQ-10243, CVE-2004-0234, CVE-2004-0235. (2006e) http://vigilance.aql.fr/
39. Vigil@nce: Incorrect analysis of ZIP files when they are protected by a password or have several levels of overlap. BUGTRAQ-11600, BUGTRAQ-11669, BUGTRAQ-11732, CVE-2004-2220, CVE-2004-2442. (2006g) http://vigilance.aql.fr/
40. Vigil@nce: No disinfection of ZIP file. BUGTRAQ-11448, CVE-2004-0932-0937, CVE-2004-1096. (2006h) http://vigilance.aql.fr/
41. Vigil@nce: Incorrect analysis of the data integrated into a URI. BUGTRAQ-12269, CVE-2005-0218. (2006i) http://vigilance.aql.fr/
42. Vigil@nce: Incorrect management of the files containing ANSI escape characters (these sequences can disturb display during the consultation of the audit files by the administrator). BUGTRAQ-12793. (2006j) http://vigilance.aql.fr/
43. Vigil@nce: Incorrect analysis of RAR files. BUGTRAQ-13416, CVE-2005-1346. (2006k) http://vigilance.aql.fr/
44. Virus Bulletin (2006) http://www.virusbtn.com/
45. Vmware (2006) http://www.vmware.com/
46. West Coast Labs (2006) http://www.westcoastlabs.org/
47. WinDDK: Windows NT Driver Devel Kit. (2006) http://www.microsoft.com/whdc/driver/WDK/
48. Winpooch (2006) http://winpooch.sourceforge.net/
49. Y0da: Yoda's packer. (2006) http://y0da.cjb.net/
50. Z0mbie: About Permutation (RPME). (2001a) http://vx.netlux.org/lib/
51. Z0mbie: Automated Reverse Engineering: Mistfall Engine. (2001b) Retrieved from: http://vx.netlux.org/lib/