Testing of “Dynamic Detection”

Presented at the AVAR 2007 Conference in Seoul, South Korea

Maik Morgenstern, Andreas Marx

*Please visit our website for current contact information: **http://www.av-test.org***

1 Introduction

As malicious software threats have changed, the anti-malware products have had to evolve, too. Simple **signature-based detection** proved to have downsides when having to deal with thousands of new malware samples daily. It takes time to process a sample and make a signature available. In the meantime a customer’s system remains completely unprotected from that particular threat. Hence **heuristic detection** was introduced. This technique also performs static scanning of malware, but doesn’t rely on signatures: instead, more generic patterns and techniques are applied. However, success of this technique isn’t always guaranteed, so again, some more sophisticated approaches are needed to detect and block malware. An example is the so called **shields feature** which monitors certain system areas and hook points for suspicious changes or actions.

All of these approaches share a certain limitation: they have only a partial view of the situation. To overcome this limitation, a technique called **behavior blocking** has been introduced by different vendors. Instead of reviewing only one file in respect of a few isolated criteria at a time, a systemic, holistic view is considered. This includes system monitoring as it was done before, but now correlates it with other information. It also broadens the view to include other files and processes which are connected to the inspected file. Additionally, a few more generic criteria are taken into account. These may include the method by which the malware arrives on the system (the introduction vector) as well as its execution type. All in all these data provide a comprehensive context and plenty of information on what to base a decision as to whether the file is malicious and should be blocked, or whether it is clean and can be permitted to execute as it wishes.

This paper describes testing methodologies for these new behavior-based approaches. In addition to a few general comments about testing, a basic outline of the testing procedure will be presented. Next the potential problems in this testing procedure will be identified and different solutions will be discussed. This will finally lead to a comprehensive framework, laying the foundation for testing behavior-based detection and prevention approaches. Because of the wide variety of different products and techniques, this paper doesn’t cover all of them. Instead it primarily targets desktop security software, rather than enterprise or corporate solutions. Nor are sandbox-based techniques covered, though the testing methodology described here can easily be extended to also include them.

2 Different views on testing

Before talking about the details of testing "dynamic detection" a few points about the different requirements and opinions on testing have to be made. There are several stakeholders involved in various types of testing and all of them have different opinions on testing and are looking for different kinds of results:

- the end user (home or corporate users, reading tests published by magazines for example)
- magazines and other companies that commission tests (excluding anti-malware vendors)
- anti-malware vendors (either commissioning a test or reviewing/commenting on another test)
- the tester (professional or amateur)

The average end user isn't interested in too many technical details, and the details of test setups are of little interest to him. Nor does he care about fancy test strategies, or whether all the technological features of a software product have been exhaustively tested. Straightforward answers to simple questions are wanted: "Am I safe or not?", "Is this software protecting me from the dangers that currently threaten me?"

Magazines (or companies commissioning tests) usually ask for particular criteria to be tested. They have constraints regarding time and money, which often results in the actual tester being given very little time (and not much flexibility) to perform the tests. On the other hand, the organizations commissioning tests want complex, up-to-date and unique results. It is also very important that every single part of the whole test can be reproduced. In case an anti-malware vendor is unhappy with a result and doubts it, the magazine and therefore the testing lab, has to be able to reproduce and prove its results. This can be supported by a well documented and scientifically sound testing methodology, which shall be presented in this paper.

In addition, for both types of customers, the results in one test have to be comparable across the range of products tested. This is most often the main goal of this type of test tests; rather than a review of a single software product, the goal is to compare different products. This requires the same standardized test approaches, the same test sets and environments, etc. for all the different products in one comparative test.

Anti-malware vendors have quite different requirements. They want their products to be tested exhaustively. Every little feature and option which will raise detection and cleaning rates should be taken into account. Reviews are often required to be very technical and go into extreme detail. However, different products have different options. While some focus on static heuristics, other products will focus on dynamic detection, all to a different extent. It is therefore difficult to compare different products, when focusing on all the special features specific to an individual product.

Clearly, someone is needed to find the best compromise between all these conflicting requirements. This is the task of the tester, who is responsible for finding the most suitable test setups to meet varying requirements and specifications. In order to do that, four different main types of tests can be considered:

- Full review of a single product
- Full comparison of several products
- Review of a specific feature of a single product
- Comparison of a specific feature of several products

What magazines usually specify (and what the end user looks for) is the "full comparison of several products" which will assess and compare the overall performance of a range of products. Tests like these require a test setup and strategy which can be applied consistently and fairly to all the products being compared. This means that not all specific features of a particular product might be fully considered in every test: rather, the most significant features common to all tested products will be considered. This especially applies to tests where only certain aspects of the software are reviewed. Of course, significant features shouldn't be disregarded just because competitors don't offer them. The most important and applicable features addressing the selected threat space should always be considered.

All of the products have to be tested in the same way, under the same conditions, with the same set of samples, to make sure the results are really comparable. As mentioned before, all of these tests have to be reproducible. The constraints regarding time and money might lead to a smaller, carefully chosen test set which will be processed in a simple, yet meaningful way. "Recreating the reality" is not an option because of the mentioned time and resource constraints, but performing tests which yield realistic and accurate results is the goal. Automation and reliability of the tests are important keywords here.

The next type of test we consider is the comparison of many products regarding one (or several) particular features. In this instance, test setups and strategies might differ between products. Different products may use different approaches for dealing with the same threats, and these differences will have to be considered when testing special features. However since the test addresses only a small portion of the products functionality (which might be implemented/used very differently throughout the different products), rather than the overall performance, it is not necessarily meaningful to compare results without additional clarification.

When only one product is under test, though, the test setups and approaches can be fully customized to meet the "needs" of the tested product. This will enable the tester to utilize and test all the functionality the product has to offer.

With all that said it becomes clear that there are the following (sometimes competing, and sometimes opposite set of) requirements, needs and wishes:

- Easy-to-read reviews with a clear message vs. very technical reports looking at every single function
- Tests that are completed quickly and cheaply vs. tests that are as complex and realistic as possible (of course, this takes time and this, in turn, costs money)
- Tests that are reproducible vs. tests that are quick and easy to do
- Test results should be comparative vs. tests that consider all the special features of a single software product
- And probably more…

Clearly, an abstraction of the different test types is needed, finding the balance between a reasonable level of completeness and realistic results yielded, and the need to respect resource restraints when establishing these test setups. It is, of course, possible to test every product in a more-or-less realistic environment and test setup. But this would take more time than any customer is willing to wait or to pay for, and this approach might make it impossible to compare the different products meaningfully and reproduce the gained results (because they all had to be tested differently). The goal can neither be to recreate reality to its fullest extent nor to provide an environment which is tailored to the best advantage of a single product, but rather to establish an environment which reproduces real life situations, as far as that is possible, and delivers results which map to the results to be expected in real life.

Different layers of abstraction can always be used, depending on for whom the test was designed, what constraints they have imposed, and what kind of test is being performed. All the approaches considered here are proposals, laying the suggested foundation for testing dynamic detection mechanisms. Of course, it's always possible to work out customized test setups and strategies. These points should be kept in mind, when reading the following sections.

3 New challenges in testing anti-malware products

This part of the paper will define the consensus proposals on how ideal tests of behavior-based techniques should be carried out and will then present a basic approach to the actual testing. One of the problems is obvious: when using the term "ideal", we implicitly accept that some real-life obstacles will have to overcome when actually performing tests. These will be ignored for now, but will be discussed in section 4.

The new approaches of anti-malware products pose a special challenge in testing. It is relatively easy to test signature-based detection, given a test set which fulfils both quantity and quality criteria. However detection rates based on these criteria don't necessarily reflect the complete abilities of the tested software: the entire range of static technologies implemented in the product should be taken into account as well. Testing this functionality is not much harder than before. Signature-based detection has to be neutralized, either by freezing the virus signatures or rolling them back to an older date. Having done so, simple scanning tests can be performed again and the anti-malware products will apply their heuristic or other generic scanning techniques for detection. However, even this is only a small part of a meaningful and useful testing. To test dynamic detection adequately, much more effort is necessary.

Besides the base approach to testing dynamic detection some other topics also merit discussion. These include the actual measurement of success in detecting and blocking threats, but also the problem of false positives and noise and finally the issue of performance assessment.

3.1 “Ideal” testing strategy

Modern anti-malware products don’t rely solely on static file scanning only anymore as we have learned above. Instead they also monitor the system for malicious changes or actions with different types of shield technologies. There are basically two approaches to testing these technologies: (1) simulation of malicious actions and changes or (2) execution of malware that performs these actions.
Performing tests the first way might yield misleading results, since the tools used to perform the actions may be known, clean good tools and therefore on a whitelist or not monitored for other reasons (such as a purist insistence on not flagging non-malware as malicious). Additionally, these tools may not reflect real malware behavior and cannot be used reliably to determine the detection capabilities of a security product. While, they can be of help when trying to determine the kinds of actions that are monitored and considered, these results can vary in accuracy and security software must not be penalized for not detecting or blocking the actions carried out by simulation tools.
On the other hand, doing it the second way, the malware might already be blocked by other detection technologies, so the actual behavior-based mechanisms are never actually activated. While this is no problem in terms of the products ability to protect overall, it can become problematic for testing when the tester tries to review mechanisms that are not signature-based.

Because of these issues, testers sometimes choose to test basic functionality with simple simulation tools. However this does not necessarily result in a meaningful test and therefore shouldn’t be considered. In order to be meaningful, real detection capabilities have to be tested with different examples of real malware which will trigger the behavior-based blocking mechanisms.

After the choice for real malware has been made, there is the question on how to work with these test samples. Traditional approaches to malware detection can easily be tested in an artificial lab set up and no special effort usually has to be made to simulate real life scenarios. This, however, is not true when testing the dynamic detection mechanisms. This is due to the combined and complex approaches used by the products. While on one hand simple static heuristics are applied and single shield technologies monitor changes, the whole state of the system and the actions and processes in progress are taken into account as well. This approach starts with the method of introduction, the type of execution and the operating system in use, patch levels and other active software. It also combines information about different shields and the files and processes involved. Certain other points such as network traffic might be considered by the anti-malware products as well.

It is easy to understand, that an artificial lab environment won’t meet the requirements for properly testing this functionality. Instead, two points have to be kept in mind. The first is the real life environment. A real OS running on a real PC has to be used. Virtualization solutions are not the best option here, because some malware samples check whether they are running in a virtual environment and change its behavior accordingly. Even where such checks aren’t carried out, some malware behavior might be affected by disparity between the test environment and real hardware. However, virtual environments can be used as secondary test systems, to evaluate differences in behavior between real and virtual systems. Additionally, some kind of network access has to be provided, either it’s the real internet or some form of simulated environment, depending on the needs and the risk of the tested malware. When using a real internet connection, the network traffic of the malware used in the test should be

filtered and restricted to prevent local spread and damage beyond the perimeter of the test environment. In the case of a simulated network, the most important protocols have to be implemented in order to enable the malware to communicate.
The second topic of discussion is the actual test procedure. The malware should be introduced in a similar way as it would be introduced and executed on the user's PC in the real world. Detection via signature should be prevented, so that non-signature mechanisms are actually triggered, when the aim is to test only these. However, all the functionality of the suite stays enabled throughout the whole test: to the extent to which this is possible without nullifying proactive testing.

Before actually starting the test, a set of malware samples has to be selected. One problem here is signature detection, when the intention is to review only the non-signature part of the software. This type of detection therefore has to be circumvented in some way that does not destabilize the product or otherwise place it into a potentially invalid condition. Additionally, a statistically significant number of samples have to be chosen and relevant malware types should be covered. It is also important to prioritize relevant samples over less relevant samples. These points are important because tests are required to give a general answer extrapolated from a relatively small number of samples (compared to the thousands and thousands of samples circulating in the wild). Choosing too few or insufficiently relevant samples could greatly distort the result so that it does not reflect real-world detection. Finally, it is also necessary to determine exactly what the malware samples do, in order to be able to tell whether the security software detected and/or blocked all actions, in addition to understanding the safety concerns of running the sample.

Following on from the comments above, the rough outline below is for an "ideal" test setup which tests behavior-based mechanisms only. This might include simple or different approaches such as system shields and sandbox techniques, depending on how the software incorporates these features:

1. Real hardware is used. Not brand new, not outdated, not uncommonly highly specified PC types
    a. Virtual machines may be used to allow comparison of results on real machines with those obtained on virtual systems
2. The base system runs a recent version of the Operating System targeted by the test
3. The patch level should reflect the pre-agreed scenario
4. Products will be tested with the default settings
5. Use malware samples which are not detected by signatures (if that is required)
6. A high volume of malware samples should be used and many different types of sample covered, depending on the test specifications
    a. Real malware as currently found "in the wild" will be used (no artificial setups, real e-mails, malicious websites and so on are to be used as sample sources)
    b. Or if not possible: A perfectly simulated internet with recent samples is used (which are introduced and executed as they would be in a real internet-connected situation)
7. Introduce the malware sample via the desired introduction vector
8. Record the actions of the security software, and compare this record to actions of the clean base system
9. Assess the product's success in detection, reporting and blocking

Table 1: "Ideal" test setup

The same steps should be performed for False Positive testing, but using known clean software instead of malware of course. More details on this will follow later in this section.

3.2 Measurement of Success

The next issue to think about is the definition of success. When considering a full scale test of security software, utilizing all of its features, there are several layers of functionality to consider. The first layer is the arrival of the malware on the system. This may happen via exploits, which make use of vulnerabilities in certain software products. Other vectors such as e-mail or peer-to-peer networks are utilized as well. It may be possible to block this kind of introduction vector already, for example when there is certain knowledge about the source being malicious or the exploit being used is known. The next layer addresses how we deal with a malicious file that already arrived on the system. At this stage, signature-based or other static scanning mechanisms would kick in and might detect this file. If neither of those two layers catches the malware, it is finally the turn of the behavior blocking mechanisms which are our main focus in this paper. As soon as the malware is executed, its behavior is monitored and actions identified as malicious or all the actions taken by the malware can be blocked.

The first two layers are relatively easy to deal with from a tester's point of view, while the last one can be tricky. Below is a table pointing out the main issues, while details are considered later.

State of malware	Detection	Comments for testing
Arrival on the System	Identify exploits, malicious packets, malicious web sources etc.	Easy to test: either malware arrives on the system or else it doesn't
Before Execution	Signature-based (or other static) detection	Easy to test, either malware is detected and the execution is blocked or it isn't
After Execution a) before doing damage b) after doing damage	Behavior-based (dynamic) detection	Harder to test, since malware is being executed and might perform actions on the system, and these have to be reviewed one by one

Table 2: Malware states, detection and testing details

In order to correctly identify the success of blocking the malware a number of prerequisite facts need to be established:

1. The clean state of the operating system has to be known.
2. The changes of the malware to the system and other actions have to be known (when there is no security software installed)
    a. The modifications and behavior should also be reviewed after a reboot (since some malware either doesn't survive reboots or exhibits different behavior when it executes after a reboot)
3. The state of the operating system after the malware has been executed in the presence of the tested security software

This information can be used to assess which malware actions and changes have been blocked and which haven't. This includes obvious issues such as created and changed files as well as registry entries. These can be checked by comparing "before and after" snapshots of the system. But it also extends to network traffic, both incoming and outgoing. Additionally, it

can be recorded, whether an action has been completely blocked, or whether it was detected and reversed after it was taken.

Comparing possible changes and actions with actual changes and actions is the most important part of measuring the success of the blocking mechanisms. Two things have to be kept in mind when doing this.
It is important to differentiate between malicious actions and changes and other, non-malicious actions and changes. When a malware uses self extracting RAR archives for example, there will be registry keys which are RAR specific. Such keys should not be counted as missed detections or disinfections when they are not blocked or not reversed.
Similar things can be said about file-system changes and registry entries which can do no harm by themselves. There should be a kind of risk rating system for the different actions and changes. When malware performs several actions and also creates a directory it is indeed unpleasant if this only this directory is missed by the security software, but no real threat to the system exists. Hence, this miss should be noted, but the product should also get a reasonable rating. It should be penalized less, for example, than if it had failed to remove a malicious executable.
The most important points to consider are the disabling of the malware and the avoidance of malicious actions. Removing all of the malware artifacts is “only” a secondary step, however still a very important one, since users expect their security software to keep their system clean.

It is also desirable to track how much information the security software presented to the user, and whether it required any interaction. This becomes important when thinking about options like “Allow program to run?” or “Allow program to perform action XYZ?”. When the software delegates key decisions to the user – making the user an integral part of the security product and process, if you will – this can hardly be counted as a positive result. The main goal for the security software should be to secure the users data and protect the computer system. To reach this goal, it would be perfect when the anti-malware software can decide reliably on its own, whether a file is malicious or not and take the required actions. However, when it is not possible to decide whether the actions should be stopped or not, the user can still be informed about what is happening and asked for a decision on what action is desirable. The security application can then also present a risk rating of the potentially malicious file so as to help the user make his decision. Additionally, the action taken by default in this scenario should reflect that risk rating, if used.

3.3 False Positive and Noise

There are several ways to involve the user into the blocking process:

- The software may ask if it should block the malware completely because it behaves suspiciously,
- Just certain actions are asked for, because they look suspicious or finally,
- There may be no questions at all and the security software silently takes the decision and does the job. Alternatively, it may take the decision but inform the user of what action it took.

Too many messages from the security software can be unpleasant, intrusive and confusing, but messages are, of course, not a bad thing in themselves. Still, it is useful to test how many messages a user will be presented to and end-user when malicious software is being blocked. Often, it is enough just to tell him that malware has been blocked because of suspicious

actions. There is often no need to give a separate message for each of these actions, maybe even asking whether it should be allowed or not. The average user tends to get used to these messages and learns to click them away, without really noticing them. Hence it would be best to only present messages when needed or wanted by the user. These issues are being tested as noise tests.

Another related issue is False Positive testing. While too many messages about blocked malware can be considered a user interface issue, the incorrect detection of clean software as being malicious is far more critical. It is possible to distinguish between two different kinds of false positive detection, depending on how the security software works. Either the legitimate software is blocked from executing or reported as performing suspicious actions, or else only the actions that are actually considered to be suspicious are blocked or reported. Neither of these should happen and it is therefore very important to test for false positives. When a false positive occurs, the software should at least give the option of allowing the application to execute.

The tests can be performed by installing and using standard software known to be clean, while the security software is active and monitors the behavior.

The basic test set should include widespread standard applications as Microsoft Office or OpenOffice, Firefox, Adobe Reader, Messenger Software, an assortment of different legitimate Toolbars, game demos and media players. It is important to go through the whole installation process, from downloading the setups, installing the products, to updating them and, of course, using and verifying the basic functionality. Not only the most recent versions but also older ones should be used, to reflect the real world situation. Apart from all these widely known applications, rarely used and highly-specialized applications should be reviewed as well. These may be localized versions, or software tools used in certain corporate environments.

3.4 Performance

The last kind of test which can be performed is performance testing. Because monitoring the behavior of other software can be quite a challenge to system resources the impact of security software should be determined.
A wide variety of benchmarking tools can be used, as for example Winstone, WorldBench or SysMark. These utilize a wide range of standard applications, which makes it possible to resemble real usage of the system. Of course it is also possible to use other approaches, as long as they simulate real world usage of the system. This could include boot-up time as well as shutdown time, or the time it takes to launch certain applications.
Tests should then be performed without security software installed first. This will give the reference values. In the next step, the performance testing will be done with the security software installed and the results recorded can be compared to the reference. The smaller the impact in terms of slow down is observed, the better the performance of the security product under test.

4 Obstacles to "ideal" testing strategies and solutions

The "ideal" test strategies presented in section 3.1 are unfortunately not suitable for all kind of tests, and not all relevant details have been explained. A range of obstacles and problems may have to be solved. There are many, many variations in hardware and software, different versions and patch levels as well as the behavior of users. All these points influence the behavior of malware. As shown in section 2, it is not necessarily the goal to imitate reality fully and always to take every one of these points fully into account, but rather to create tests, which gain realistic results, while decreasing the complexity of the test performance.

Here we refer to the "ideal" testing points introduced in table 1 in chapter 3.1 and go through them point by point. First of all, the proposed base approach is explained and subsequently, several problems and options will be outlined.
There are of course always other and non-standard options or requirements possible, and they can not all be reviewed and mentioned here. But when following the rough outline of the testing procedure which is presented here, it should be possible to incorporate these requirements into the testing process and still create valid results.

4.1 Hardware and Operating System

In general, all tests should be performed on real hardware. But virtualization solutions such as VMware shouldn't be fully neglected, since these products are actually used in working environments and therefore also in danger of malware infection. While this is certainly only a small risk, it is a risk that exists nonetheless.

It is obviously necessary to use virtualization solutions when a comparison between a real machine and a virtual machine is wanted. Also, virtualization can be a big help when working with a large sample set, as one real system can hold several virtual machines, to reduce the time needed for the test. However, the downside of this solution always needs to be kept in mind and it should usually be avoided. This is mainly because some malware tries to determine whether it is running inside a virtual machine and changes its behavior accordingly. This usually differs from what the malware does in real life on real systems, and compromises the validity of the test. A lot of care is therefore required, even when virtual machines have to be used, because of the need to check every sample to see whether it behaves different on real machines than it does in a virtual environment. Additionally, the software product under test, might behave different on virtual machines as well. While this shouldn't happen, it is still a possibility and should be taken into account. Thus, when performing tests this way, it has to be clearly stated virtual machines have been used and the possible errors in the results have to be mentioned.

While everyone is likely to agree that Microsoft Windows is the most important operating system to review, it is not so clear which actual Windows versions should be used to test with. The options still range from Windows ME (or even 98) to XP SP1 and XP SP2 as well as Vista or Windows 2003 Server. (In principle, any version of Windows might still be in use somewhere, and therefore a legitimate testing target.) While the focus at the moment might still be on XP SP2, Vista will become soon equally important, and different threats might affect these operating systems in different ways. Depending on the scope of the test, different operating environments will have to be chosen, but at the moment Windows XP SP2 and Windows Vista SP0 are an appropriate base.

Next, we must consider the actual patch level. With automatic updates, as included in modern Windows versions, it is likely on many sites that the majority of the computers will be reasonably recently patched, but not necessarily to the very latest patch level. Hence, it is debatable how relevant this point is for a test environment, since not the patch level of the operating system should be tested, but the behavior blocking mechanisms of the anti-malware product. Therefore, either assume the vulnerability has been closed and disregard the testing of malware using that vulnerability, or assume that all vulnerabilities which are relevant for malware of the last few months remain open. The best compromise might be to use the latest service pack, while applying hotfixes remains optional.

The language of the operating system is also something to consider, since targeted attacks are becoming more and more common and might check the language of the system to decide their behavior. This will have to be decided from test to test, meeting the given criteria and depending on who commissions the test. In general, the language should be US-English.

Finally, the settings of the system have to be agreed on. Usually default settings are the way to go, since this is what most computer systems are like. However, included security solutions such as the Windows Defender should normally be disabled, so that they can't interfere with the tested security product. As far as the Windows Firewall is concerned, the tested security software should usually take care of disabling it, when it incorporates its own firewall. Again, a customer's special needs have to be considered as requested. For example, when testing for a company which uses special configuration settings on their systems, these should be set accordingly for the test. The same is true for additional software that can be installed on the systems used for testing to resemble the wanted environment.

Base approach:

- Real hardware
- Desired operating system with latest service pack but without hotfixes (at the moment Windows XP SP2 or Windows Vista SP0)
- US-English as language
- Default settings of the operating system should be used
- Included security solutions such as Windows Defender should be disabled

Options:

- Virtual machines as alternative or comparison
- Other operating systems depending on the customers needs, with the desired service packs and hotfixes
- Language of choice
- Special settings of the operating system as desired

4.2 Configuration of the security software products

The configuration of the tested security products should usually be left in default state since this is what the user gets. It is very possible that some users will change these settings to their preferred states, but that doesn't reflect what the majority of home users will have: however, the situation in corporates may be very different, since products may be distributed or installed with settings far more paranoid than the out-of-the-box default. Also, none of the different detection features should be turned off.

Products should be updated after their installation to make sure recent versions are used. This is important when testing boxed versions, which may contain setups which are several months old. Also, not all vendors update their downloadable setup files but rather expect to update the software after its installation. Since most products use automated update mechanisms it is rare to meet very outdated versions in the real world.

Signature-based detection has its downsides, but it still shows a pretty good performance. This is a point that becomes important when choosing samples. Think about a test that involves many different products and many different samples of different malware categories. It is often virtually impossible to collect samples which remain undetected by all products, but are still malware and belong to the categories needed. When only testing the behavior-based mechanisms, there are certain options when choosing samples and updating the products, to prevent the signature-based detection.

The easiest solution would be to simply turn off signature-based detection, when there is an option to do so in the application. However there are too many downsides in this solution. Of course, it would be necessary to make sure that this option really does what it says and that no vendor tries to cheat. Also, it is necessary to make sure that this doesn't affect the behavior-based detection components. It is also undesirable to turn off parts of the security software because that wouldn't reflect real life conditions. So other approaches should be chosen when possible. One possible option is to freeze signature updates and wait a while (up to several weeks) until the test is carried out. This however means, no updates to the software (for example, an engine upgrade) will be applied, and the test results could easily become outdated and invalidated when new features have been introduced during the interim. It is also possible that behavior-based rules are updated regularly, and these updates would be missed as well.

A second option is to roll back signatures manually, which could affect other parts of the software and might accidentally disable some features. This doesn't reflect real life conditions either, since products are brought into an artificial state that would never occur in reality. Also, it may be not be feasible to apply this approach to all products.

The last option is to use different samples for different products, which of course wouldn't be fair either, since one sample might be harder to detect than another one. This is simply because different samples might behave differently. As one of the most important requirements of comparison tests is comparability, this could have a major biasing impact on the test results.

After having taken all that care regarding the updates, it's also necessary to make sure that the products won't update themselves during the test when a real internet connection is used. This can either be done either by disabling the update mechanisms, when the software offers such

options, or by redirecting DNS requests, or simply by blocking known update server URL's and IP's. To be sure the software really didn't update, the files can be verified using hashes.

As the basis, when there is no option to turn off signature-based detection, the first choice should be used, since this guarantees real life conditions and the only downside is the needed waiting time and related effects, which are usually insignificant. Of course, all tested products have to be frozen at the same time. When waiting is not an option, the other two choices have to be considered. However, as these can have major effects on the reliability of the test results, they should be avoided when possible.

Base approach:

- Test the security products with default settings
- Update products to their latest version
- Turn off signature-based detection or freeze signatures for a certain amount of time until choosing samples and starting the test
- Make sure, products can't update themselves during the test

Options:

- Change settings to other states (e.g. best possible settings)
- Use products off the shelf without any updates
- Instead of freezing signatures, roll them back manually
- Use different freeze points

4.3 Implementation of the Tests

This part of the paper contains the most controversial points. However, when keeping section 2 in mind, it should be possible to understand why some approaches will be preferred over others. Also, other approaches won't be ruled out, instead they will be used as optional extensions to the basic test approach. This will then increase the complexity of the test, but may also increase the quality of the results, depending on the pre-agreed, predetermined criteria.

The first issue is the choice of samples used for the testing. Signature-based detection won't be an issue here, either because this kind of detection is allowed, or because it has been disabled with one of the approaches described in part 4.2.

The basic requirement to samples is relevance. It is not easy to define relevance in terms of malware, as it changes fast and can be locally different. Also, depending on the system configuration and software used, some threats are simply irrelevant for many people. But these threats could still affect a lot of other users and therefore have great relevance for a global audience. Clearly the criteria will change from test to test, but basic guidelines will always apply.

A statistically significant number of recent and new samples has to be used for results to be meaningful, as it is not possible to generate an answer out of only three or five samples which in the worst case are also outdated. This also means the test set should show wide variety, covering many different malware categories and consist of samples actually found in the wild.

However, there is only the WildList and the vendor-specific lists and resources which list threats in the wild. This might become a problem when trying to choose relevant samples, because these lists cover only a subset of the actual In-the-Wild malware. When testing special malware, it has to be ensured that only the desired categories are included.

Deciding whether to test while connected to the internet or to use a simulated network is another choice which has to be made. Both options have their good and bad sides.

When using the internet, real life conditions are guaranteed and no special effort has to be made to set up a simulated network. However, there are downsides from two perspectives. The first is that we are dealing with malware. It is therefore critical to let a compromised system run freely on the internet, maybe taking part in Denial-of-Service attacks when a bot is tested or infecting another computer system in the case of a worm. This is a scenario which must not be allowed, for both ethical and legal reasons, so precautions have to be taken and the network traffic generated by the malware has to be filtered and limited to prevent damage. This could be done with a proxy which only allows certain protocols and limits outgoing traffic.

It is also critical from a tester's point of view. Often up to ten or more products take part in one comparative test. Also, the data in the internet is constantly changing, which means that some malware will download a file from some server and the next hour this server is gone or the data on it changed completely. This means, each product might have to cope with different behavior of the malware so that results cannot be compared meaningfully.

It is therefore necessary to ensure that all products are tested in a similar timeframe. When doing this, the next problem occurs. It is likely that servers which distribute malware will check IP addresses and won't send any data or different data after a certain number of requests, mindful of the possibility that they're being monitored by anti-malware agencies. So either you get lucky when testing and always receive the same data, or else several lines with different IP addresses have to be used for the tests. This can sometimes be difficult, when there is no way to get more than three or four lines from the telecommunication companies. The other option is to test the products one after another on a few lines and reconnect them for every test, when possible, for example no dedicated lines are used, to change IP addresses. However this re-introduces the problem of the timeframe again, by the end of the test, the malware might behave totally differently to how it did at the beginning.

Some of the problems may be solved when limiting the connectivity of the test systems and using only certain ports and protocols for incoming and outgoing traffic. Also, a proxy could be used here too to communicate with the malware. But this would also impose a limit on the malware and change its behavior yet again. Additionally, different connectivity limits might be necessary for every malware sample, which could be a bit unhandy when testing a lot of samples and many products.

Clearly, given all these issues, a simulated network environment should be considered as well. This is not an easy thing to do and it has serious downsides too. But all in all it is secure and could ease testing significantly. This is so because it can be guaranteed that, as far as network communication is concerned, all tested products will have to work under the same conditions and malware behavior won't be affected by the network, irrespectively of the different products currently under test. Basically, all protocols that are possibly used by malware would need to be emulated, these include but are not limited to:

- DNS and ICMP queries
- HTTP and FTP connections
- SMTP and IRC connections
- SMB and NetBIOS connections
- Instant Messaging and Peer to Peer connections

A typical example of the network use might be a DNS query or ping using ICMP queries to determine whether the system is online. If it is, HTTP or FTP can be used to download additional malware components which will then connect to an IRC network to act as an IRC bot or use Instant Messaging or Peer to Peer protocols to propagate further.
To guarantee this behavior in a simulated network, proper responses have to be created. Of course, some limitations apply. Malware can only download a faked file, and not the real file it would download in the real internet. This limitation can be avoided, when utilizing a proxy which downloads the file from the internet and then forwards it to the malware in the simulated network. This can be prepared in an analysis phase before carrying out the tests with the different products. Simply execute the malware, record its actions and store the files it downloads and the network traffic it generates, so as to be able to deliver them later in the simulated network.
Similar statements are true for IRC connections. For example, even when the malware is fooled into “thinking” it is connected to an IRC server, it will still not receive commands from a controller, which would influence its behavior. This could be simulated in a similar way as described above for file downloads: Execute the malware once to record command sequences, and use the recorded data afterwards in the simulated network. But once again, the malware should not be allowed to run freely in the internet: instead, all the generated traffic has to be controlled and restricted when necessary.

With this in mind it is also possible that specific malware tests for the existence of emulated networks or restricted connections, just as some malware types test for the presence of sandboxes or virtual machines. Several simple tests could be performed, starting with a test for whether a protocol is implemented or not. Malware might also test whether a response is meaningful. For example, pinging a non existent IP address or querying for a domain name that doesn’t exist. When the results are unexpected, such as when the non existent IP address replies or the query for the non-existent domain name succeeds, the malware knows that something is wrong and can change its behavior. Many more tests are possible, and it will not be possible to catch all of them.

This demonstrates the main problem of the simulated network. It can become arbitrarily complex, and it will be impossible to simulate every single option that is out there. In conclusion, combining restricted internet access through a proxy with a basic simulation of the blocked protocols and connections might be the best choice at the moment. Again, it is possible to use different layers of complexity here.

The last point to make in this section concerns the actual introduction vector and execution type of the malware samples. Malware can arrive on many different ways on a system. Some of these are listed below:

- Direct download with a Browser or FTP client
- As an E-Mail attachment
- Through Peer to Peer network downloads
- Through Instant Messaging
- By exploiting a vulnerability
- MP3 player, USB sticks, CD/DVD, and other removable media

This introduces two problems. Firstly, it is hard to tell how a sample would actually arrive on a user's system. The second point is that this doesn't tell us how the user actually executes the malicious program and whether it was supposed to arrive that way. Even if it arrives as e-mail attachment, that doesn't mean the user will start the file from the mail application, he might as well save it somewhere and execute it later. Also, even if it is known that a sample arrived on a certain way, that doesn't mean, this is the only way the sample could arrive. And each arrival and execution type could yield a different behavior.

It is therefore difficult to limit the execution type to just one possibility. Rather several execution types should be considered. There may be products which take this into account and there may be others which don't. Hence, the basic test procedure should contain the most simple execution type, which means just double clicking the file on hard disk, but also contain other execution types which will be used as a heuristic by some behavior-based mechanisms. E-mail attachments would be an example, as described above. With this it is possible to compare results, between different but relevant infection vectors.
Optionally, more infection vectors could be tested, or, when it is known how a sample arrived, the test procedure for this sample can be limited to that respective infection vector. However, some of the limitations apply which have been mentioned above.

There may be disagreement on whether the simple double-clicking method should be used to start malware or not. There are reasons why it should be used. Malware is always trying to avoid detection somehow, so it will learn to adapt to behavior-based mechanisms. Of course, this is just theoretical at the moment, as behavior-based solutions are still new to the market. However, knowing how fast virus writers react to the anti-malware industry, it is a point that has to be kept in mind. It makes sense for behavior-based systems to rate files that sit on the hard disk and are executed by the user via a double click as a lower risk (since this is what most legitimate applications look like), than files which get executed through an exploit (obvious) or by the user within an e-mail application, or as download, for example. When ruling out the infection vector in the test, this would certainly mean a loss of information to some anti-malware software (not all products take the infection vector into consideration). But, obviously, the infection vector is just one of many possible clues which help to decide whether a file is malicious or not. The infection vector only has a meaning at all in conjunction with many other factors. There should still be a reasonable detection result by the security software, even if no suspicious infection vector has been used, as long as the malware acts malicious.

Additionally, there are two possible (again, still theoretical) ways for malware to bypass the infection vector issue. The first option is social engineering: the successes of phishing attacks show that this is a technique which works very well. The malware author will just "tell" the user what to do and some may just do as they were told. For instance they may save the file on the hard disk, and only then start it with a double click. On the other hand, there are tools to simulate user input, which might soon be used to fool behavior-based blocking mechanisms when it is known that malware executed via a double-click right off the hard disk gets a lower risk rating.

Of course, these kinds of tricks are not yet used and the focus should remain on current malware and their current behavior. But it would be very desirable for anti-malware products to adapt to them before malware starts to make use of them. Otherwise there will be yet another gap to close. Because of this, testers should make use of both the simple execution type with double clicking and of typical infection vectors.

Base approach:

- Choose samples, consider constraints regarding signature detection
- The number of samples should be statistically significant and a wide variety of recent samples should be used to cover all important malware types
- Pay attention to other relevance criteria of the samples, depending on test requirements
- Set up a simulated network or use restricted internet, emulating/allowing the most important protocols (but prevent harm)
- Execute the malware samples in at least two different ways (direct execution from hard disk and execution with an typical infection vector)

Options:

- Limit the test to special malware categories
- Use samples of different age over a big time span
- Consider different relevance criteria (localized attacks)
- Experiment with simulated network settings and network restrictions (always deliver clean or always infected files on download requests of the malware, respond good or bad to checks whether it's a simulated network or not, restrict different protocols)
- Execute the malware either many more different ways or if known in one special way (as it appears in the wild)

5 Concluded Test Setup

Taking all the above into account, it is quite complex to implement the "ideal" test setup. There will always be a trade-off between fairness of the test across all the different products, and a full simulation of the real behavior of malware, which could be different on every single test. Additionally, constraints regarding time and money will always have to be considered. Of course, when special setups and options are required, these can be included in the tests. The following setup will provide a framework which is a good compromise between the aforementioned issues and will help to provide meaningful results, which can be compared across different products. The base approaches as described above will be the main framework. All the other possible options mentioned will be listed under the corresponding point. When applying these or other options not mentioned here, that will certainly increase complexity of the tests, but might also increase realism or quality of the results, meeting special criteria that have to be considered throughout the test.

1. Real hardware is used. Not brand new, not outdated
    a. Option: Use virtual machines, accept the risk of different behavior of the malware
    b. O.: Use virtual machines to compare results between real and virtual machines

2. The base system is either Windows XP SP2 or Windows Vista SP0 in US-English
    a. O.: Other operating systems depending on the customers needs
    b. O.: Not only latest service packs but also latest hotfixes applied
    c. O.: Other language/localization, as desired

3. OS-Included security solutions such as Windows Defender are disabled, everything else is in default state
    a. O.: Turn on Windows Defender to test for combined results
    b. O.: Special settings of the operating system as required

4. Products will be tested with default settings
    a. O.: Change settings to other states (e.g. most secure possible)

5. Update products to their latest version
    a. O.: Use products off the shelf without any updates

6. Signature updates should be frozen some time before collecting samples, to avoid signature-based detection
    a. O.: Instead of freezing, roll back signatures manually
    b. O.: Use different freeze points

7. Choose samples, consider constraints regarding signature-based detection and the given relevance criteria

8. The number of samples should be statistically significant and a wide variety of recent samples should be used
    a. O.: Limit samples to special malware categories
    b. O.: Use samples of different age over a wide time span

9. A simulated or restricted network will be used which reacts to the needs of the malware
    a. O.: Combine a limited internet connection with a simulated network
    b. O.: Experiment with settings of the simulated network

10. The same samples are used for all products

11. The same execution types for all samples and products are used (1. Direct execution from hard disk, 2. Other execution type e.g. from an e-mail application)
    a. O.: Execute the malware in the way it is known to spread in the wild (when this information is available)
    b. O.: Execute the malware in many other different ways

12. Monitor the malware and the security software and check what is being detected, reported and blocked

13. Perform false positive, noise and performance tests

Table 3: Concluded test setup

6 References

[1] Gordon, S. What is wild? Proceedings of the 20th National Information Systems Security Conference. NISSC, 1997. *http://csrc.nist.gov/nissc/1997/proceedings/177.pdf*

[2] Gordon, S.; Ford, R. Real world anti-virus product reviews and evaluations – the current state of affairs. Proceedings of the 19th National Information Systems Security Conference. NISSC, 1996. *http://csrc.nist.gov/nissc/1996/papers/NISSC96/paper019/final.PDF*

[3] Williamson, M.; Gorelik, V. Towards new standards for real-time evaluation of anti-virus products. Proceedings of the 17th Virus Bulletin International Conference.

[4] Marx, A. A Guideline to Anti-Malware-Software testing. Proceedings of the 9th Annual European Institute for Computer Antivirus Research Conference. EICAR, 2000. *http://www.av-test.org/down/papers/2000-02_eicar_2000.zip*

7 Acknowledgements

We would like to thank the following persons (and their whole teams), companies and organizations for their support in creating this paper. They provided valuable feedback and shared their opinions and knowledge in numerous discussions which helped improving many parts of this document.

- Alex Eckelberry, Sunbelt Software
- Alisa Shevchenko, Kaspersky Lab
- Andrew Lee, Eset
- Christian Mairoll, Emsi Software
- David Harley, Eset
- Gerhard Eschelbeck, Webroot
- Holly Stewart, IBM ISS
- Ilya Rabinovich, SoftSphere Technologies
- Jim Meem, PC Tools
- John Hawes, Virus Bulletin
- Mark Kennedy, Symantec
- Mark Obrecht, Symantec
- Matt Williamson, Sana Security
- Mika Stahlberg, F-Secure
- Pedro Bustamante, Panda Security
- Raimund Genes, Trend Micro
- Roel Schouwenberg, Kaspersky Lab
- Sarah Gordon, Symantec
- Stuart Taylor, Sophos