



## Best Practices for Dynamic Testing

### Introduction

This document describes best practices for the dynamic testing of host-based anti-malware products, where dynamic testing means tests where a PC is exposed to a live threat (for example, by attempting to execute the malware) as part of the test. This type of test is a more realistic test of product efficacy than standard static tests (for example, on-demand scanning), as it directly mimics malware executing on a victim's machine. While dynamic testing is the only way to test some anti-malware technologies, it is appropriate as a test methodology for all types of anti-malware products.

Unfortunately, this type of test is more complex to operate than static tests, with many more issues to be considered. This document seeks to summarize best practices and provide guidance to help testers address those issues. The following sections cover reproducibility, product selection, sample selection, testing environment, logging and auditing, measurements of success, and the handling of popups and events requiring user interaction. Two examples of test methodology are given at the end of the document.

This document is an extension to the AMTSO "The Fundamental Principles of Testing" document available at <http://www.amtso.org>.

### Reproducibility

Unlike static tests, dynamic tests are inherently difficult to reproduce, meaning that the same test run at different times may produce different results. This variability can be caused by changes in vendors' products (for example as signature detections are updated), and also by changes to the malware's environment. Some malware will only operate properly if certain external resources (such as NTP servers, banking website pages, drop sites) are available. While it is possible to mimic some of these factors in a test environment, it is difficult to reproduce them with complete accuracy, and the test environment can become increasingly artificial. This variability means that it is difficult to draw strong conclusions from any single test run.

There are two defenses for the tester against this variability. The first is to collect enough logs and information to verify what happened in the test. To show, for example, that malware x tested on day y against product z did indeed take actions. The second is to use sufficient samples, and repeat tests over time (with different sample sets) so that inconsistencies in malware behavior are less likely to skew the

results. For example, fresh malware could be tested on the day that it was obtained over the period of a month, and the daily detection rates or trends in performance compared.

## Product selection

Sometimes anti-malware protection is incorporated in suites of products, while others are found in standalone products. The tester should be aware of these differences and choose products carefully to allow reasonably appropriate comparative tests. One way to discover these differences would be to use vendor claims as a basis for choosing products, e.g. if vendors claim that their products handle x, then it's reasonable to test multiple products that cover x.

## Sample Selection

In any test, sample selection is important. However, for dynamic tests the sample set should be evaluated according to the following criteria. In general the quality of the samples is more important than the quantity.

1. **Functionality.** To be a good test of protection, the malware sample needs to be viable. It is important to choose samples that "work" (corrupted or truncated samples, for instance, are not valid test material) and to verify that the samples did actually do something malicious in the test environment.
2. **Diversity.** Often dynamic testing will be carried out on a smaller sample set than other tests. It is thus more important that the sample set is diverse. Diversity in this sense means both in the variety of malware families tested (e.g. 50 variants of RBot would not be diverse), and in the underlying behavior of the malware. For example, it doesn't make sense to test for general efficacy using a test set that consists of 50 dialers.
3. **Relevance.** The prevalence of malware in the field is important to take into account in creating a relevant sample set.
4. **Freshness.** An important aspect of any technology is zero-day protection. This is best evaluated using fresh and currently relevant threats. Thus the age of samples needs to be considered.

## False Positive testing

To provide a balanced test of user experience, tests should include looking for false positives by testing against non malicious programs. These programs should cover the set of common operations that users undertake on their machines, e.g. installing applications, updating applications, running applications, applying operating system patches and installing and using browser plugins. Installed applications should be run to ensure that they function correctly.

## Testing environment

In dynamic tests, the performance of products is strongly determined by the behavior of the malware, and malware behavior is itself strongly determined by the environment in which it runs. It is thus important to create a reasonable runtime environment in order to get good test results. Environment in this case means for instance the operating system of the machine, whether it is a real or virtual machine, the network connectivity, how the malware is launched, etc. There is no "right" environment,

so testers should be aware of the tradeoffs in selecting a particular environment and how that might (inadvertently) bias test results. The following covers some of the major aspects:

## **Use of virtual machines**

Many malware will not exhibit their full range of behavior in virtual environments (e.g. VMware or Virtual PC). In addition, the use of some products is not supported in virtual environments. The alternative to using virtual machines is using real machines, which is more technically complex and awkward to automate. In spite of these difficulties, using real machines is recommended for dynamic tests. To help with the technical difficulty, AMTSSO will encourage participating members to make useful testing tools openly available to all other members. Testers should always disclose what types of machines were used in tests.

## **Network**

Many forms of malware and some products require network connectivity in order to run with full functionality. Thus, testing requires allowing access to the Internet from test machines. However, this is dangerous as the malware might be able to spread to other machines or cause other damage. There are two common approaches to dealing with this issue. The first is to allow network connectivity, but restrict the protocols allowed. For example, a common arrangement is to allow http (web) traffic access to the Internet, but block all other protocols. Normally this is accomplished at the gateway rather than on the test machine. An alternative is to create a virtual Internet (also known as a Truman box) where fake replies are sent to all network requests. Another alternative is to use a slow network link (e.g. ISDN or modem access instead of DSL).

There is no “right” setup, so testers should make an informed choice and document the setup in the test methodology.

## **Launching the malware**

Products may take into account how the malware infects a machine and be tuned to be sensitive to common infection vectors, for example drive by downloads. How the malware is started can thus affect product performance. A guiding principle here is that the malware should be executed as it would have been delivered were it a real infection. For example, if a specific piece of malware originates as an HTTP download, the test should present the malware via its native download protocol rather than executing locally. This can be difficult to implement, so a weaker alternative is to manually introduce the malware and vary the ways that it is launched.

## **Logging/Auditing**

Since in dynamic tests, the behavior of the malware is crucial to how the products perform, it is very important for the tester to have adequate logging and auditing of how the test proceeds. At the very least this should cover

1. The actions the malware takes on the infected/compromised machine
2. Modifications made to files, registry, and system areas

### 3. Traces of network activity

#### Measure of success

Because dynamic tests require the execution of malware, and malware can have many different effects on a system (such as installed files, configuration changes, information leakage, and so on), it is important that the definition of success is carefully considered. There are a variety of ways to measure success, some of which may be more relevant to a particular test than others. These measures include:

- **Detection.** Did the product detect (report or log) anything?
- **Removal.** Were there any files or configuration changes (e.g. registry modifications) remaining after the product has handled the malware? Ways in which this could be applied range from a strict interpretation (flagging any and all changes), to ignoring (for example) junk files and data files created or added, configuration settings that have no actual function, and so on. Some approaches may disable malware, rendering it incapable of doing further damage, rather than removing it, so this approach should be considered in judging success or failure.
- **Persistence (Activity/Survival after reboot).** Was the malware active (running in memory), or configured to survive reboot after the product handled the malware? This is a weaker, but more attainable form of the previous point above.
- **Damage.** Did the malware successfully compromise the machine? This is particularly relevant for information stealing malware, where some personal information may have been altered, removed or leaked from the machine even if the malware was successfully blocked. While this can be a good measurement of success, it is often difficult to measure in practice.

#### Popups and user interactions

Many products use a popup or other window to ask the user for direction on how to proceed. This can cause confusion in test results (for example, if the product asks for permission to block a threat, and the tester always says no, then the product's performance will appear very different from if the tester always clicks to block). The most important guidelines for handling popups and user actions are

1. **Policy.** Testers should decide a policy on how to handle user interactions. For example, they could choose to answer popups in the most favorable way to the product, or in the least favorable way. The policy should be explicitly described in the test report.
2. **Consistency.** Once a policy has been decided, they should apply that policy consistently across all tests (for example, applied to both false positive and detection tests)
3. **Reporting.** The tester should measure and report how many interactions with the user the products require. This will allow consumers of the report to determine what sort of product it is (e.g. is it effective, but very chatty; effective but less intrusive etc.). There are a number of broad classes of popup, which should be reported separately. For instance:
  - Forgiveness – the product takes an action without requiring interaction, then reports to the user that it has been done
  - Permission – the product prompts for permission or a decision before taking an action
  - Notification – notifications in general, perhaps differentiating between those that require user interaction and those that do not.

## Dynamic test styles

This section describes two common testing styles for dynamic testing. This is not an exhaustive list, and should be used as a source of inspiration for testers.

The first style is the “one at a time” approach. Here machines are set up with a single vendor’s product installed, a single sample of malware is introduced, and after the product has had a chance to detect and remove it, the state of the machine is analyzed to ascertain whether the malware was detected and removed successfully. After the test the machine is reverted to its pristine state and the test repeated for the next malware. This approach is precise but can be very time consuming. The analysis should at the very least measure any changes to the file system (files added, removed or modified) and to the registry configuration as a consequence of the malware being run and being handled by the product.

The second style would be “many at a time”, having the same general approach but running multiple pieces of malware at a time. This is less precise, but more efficient in terms of analysis. An interesting variation on this is to obtain the malware by causing the machine to visit a large number of suspicious web sites, in the hope of its being infected, for example by using a script to launch the browser repeatedly. After the sites have all been visited, the machine can be analyzed to see how successfully the vendor’s product protected the machine from infection. This test is a good simulation of a common infection vector (drive by downloads). One caveat of this approach is that, partly due to the efforts of malware researchers, malicious servers may not serve the same (or any malicious code) if multiple requests are made. In this test it is hard to guarantee that each vendor’s product will be exposed to the same threats so it is important to both use a diverse list of suspicious sites, and to repeat the test a number of times to see trends in performance.