Execution Context in Anti-Malware Testing

David Harley BA CISSP FBCS CITP
ESET

About Author

David Harley BA CISSP FBCS CITP is Director of Malware Intelligence at ESET. He previously worked for Imperial Cancer Research Fund as security analyst, and later managed the Threat Assessment Centre for the UK's National Health Service. His consultancy Small Blue-Green World provides security and publishing services, and he is Chief Operations Officer of AVIEN. His books include "Viruses Revealed"(Osborne) and "The AVIEN Malware Defense Guide for the Enterprise"(Syngress), and he has contributed to many other books and publications on security, programming, and education. His research interests include all aspects of malware, security software testing, Macintosh security, and the psychosocial aspects of security.

Contact Details: c/o Research Group, ESET, 610 West Ash Street, Suite 1900, San Diego, CA92101, USA, phone +1-619-204-6461, e-mail dharley@eset.com

Keywords

anti-malware testing, dynamic analysis, static analysis, on-demand, on-access, command-line, multiscanner, sample validation, statistical anomaly, false positives

Execution Context in Anti-Malware Testing

Abstract

Anti-malware testing methodology remains a contentious area because many testers are insufficiently aware of the complexities of malware and anti-malware technology. This results in the frequent publication of comparative test results that are misleading and often totally invalid because they don't accurately reflect the detection capability of the products under test. Because many tests are based purely on static testing, where products are tested by using them to scan presumed infected objects passively, those products that use more proactive techniques such as active heuristics, emulation and sandboxing are frequently disadvantaged in such tests, even assuming that sample sets are correctly validated.

Recent examples of misleading published statistical data include the ranking of anti-malware products according to reports returned by multi-scanner sample submission sites, even though the better examples of such sites are clear that this is not an appropriate use of their services, and the use of similar reports to generate other statistical data such as the assumed prevalence of specific malware. These problems, especially when combined with other testing problem areas such as accurate sample validation and classification, introduce major statistical anomalies.

In this paper, it is proposed to review the most common mainstream anti-malware detection techniques (search strings and simple signatures, generic signatures, passive heuristics, active heuristics and behaviour analysis) in the context of anti-malware testing for purposes of single product testing, comparative detection testing, and generation of prevalence and global detection data. Specifically, issues around static and dynamic testing will be examined. Issues with additional impact, such as sample classification and false positives, will be considered - not only false identification of innocent applications as malware, but also contentious classification issues such as (1) the trapping of samples, especially corrupted or truncated honeypot and honeynet samples intended maliciously but unable to pose a direct threat to target systems (2) use of such criteria as packing and obfuscation status as a primary heuristic for the identification of malware.

Introduction

It’s common to think of product evaluation purely in terms of detection performance: after all, detection and removal (or blocking before infection) are usually considered to be the most important functionalities of an anti-malware package. However, in real life, effective evaluation can and should include a far wider range of criteria. Indeed, corporate evaluation is sometimes based on an assumption that there is comparatively little variation in overall detection rates between products, and that detection, as assessed by testing, is of less importance than other factors (Lee & Harley, 2007a) such as:

- Ergonomics and usability
- Configurability
- Documentation
- Support
- Functional range
- Performance (speed of operation, impact on system resources, and so on)

However, the deciding factor is often cost, which may even be the only factor given serious consideration, especially in a procurement process where knowledge of the technological issues is not part of the procurement remit (Harley, Slade & Gattiker, 2001).

As a matter of fact, this realignment of priorities could be defended quite vigorously, given the difficulties of detection performance evaluation in the current threat landscape, where 100% detection of known and unknown malware has become virtually impossible, except by the application of generic countermeasures often too draconian to be accommodated by organizations and businesses attempting to live in an interconnected “Web 2.0” world. How has this situation arisen?

Testing and Over-Abundance

Sheer sample glut has had a devastating impact on the 1980s/early ‘90s model of one signature to each malicious program. When anti-malware labs are processing unique samples running into six figures on a daily basis, it is no longer sensible or practicable to attempt to produce a unique signature for each and every short-lived incarnation of a malicious program that flares briefly and then is never seen again. This doesn’t, of course, mean that the base code changes so frequently: rather that the use of packers and obfuscators to change the “wrapper” around the base code changes the “appearance” of the program so that simple signatures are either invalidated, or can only work in combination with other techniques such as de-obfuscation.

This doesn’t mean that there is no longer a use for traditional known-malware detection (“signatures”), but that approach is incapable of approaching anything like the detection levels that were expected in the 1990s, when malware was more specialized, individual malicious programs and variants were much rarer, and distribution mechanisms were less effective. In the 21st century, such offensive techniques as server-side polymorphism, where each instantiation of the basic code is regularly replaced by a repacked version, are far less susceptible to the primarily static detection techniques (including passive heuristics) that were largely successful back then. (The assumption here is that 100% detection is, however desirable, not usually a realistic definition of success for an anti-virus product.)

In fact, if we were to count each detection of a malicious program as a separate “signature”, we would see a staggering increase compared to the daily detections released in the 1990s. However, these are now generally consolidated into generic signatures, heuristic detections and so on, so this escalated detection ability is far from obvious to the everyday computer user. What is obvious, however, is that anti-malware software doesn’t meet the expectation of customers (and many testers) that it will achieve close to 100% detection. Of course it never did, but, paradoxically, while the technology has become infinitely more sophisticated and capable, scanners have declined in effectiveness. However, we can’t realistically compare the number of detections to the absolute totality of malware, because no-one (tester or vendor) has every malicious sample. We cannot overstate the difficulties of compiling a test set that represents that totality with reasonable statistical validity and a minimum of vendor bias (Harley, 2008).

The Execution Context Problem

Even if we assume that the “best” tests (Harley & Lee, 2007b) use a large enough (and correctly validated) sample set to keep the margin for error acceptably small, there is widespread concern among vendors and testers that a wide range of tests generate seriously misleading results because the testers are not aware of the importance of execution context in detection performance evaluation (AMTSO, 2009).

This issue reflects growing awareness that proactive technologies based on some form of dynamic analysis of malicious code and incorporated into modern antimalware solutions require testing methods that are better able to capture the detection capabilities of these approaches (Morgenstern & Marx, 2007).

Because many tests are based purely on static testing, where products are tested by using them to scan presumed infected objects passively, those products that use more proactive techniques are frequently disadvantaged in such tests. (Note, however, that static testing is not strictly synonymous with passive scanning, which is not normally thought of as allowing the execution of scanned code: we will address this anomaly in the section below on Testing Methodologies.)

Our intention in the next section is, therefore, to clarify the terminology most commonly used in this context in relation to the technology it describes. We will also highlight some specific problems that arise when the issue of execution context is ignored.

Discussion

Evaluation strategies focused entirely on a product’s detection technology limit the practical value of the evaluation, since the scope of a real-world implementation is dependent not only on the inherent functionalities of a product, but other factors such as corporate infrastructure and the nature of the systems to be protected.

Evaluation Based on Detection Performance

Even where detection performance is a (or *the*) major evaluation criterion, the prospective customer (the tester’s audience) cannot be supplied with a full appreciation of a product’s capabilities unless the evaluation process uses the results of a wide range of detection performance tests, irrespective of whether they come from a single source or many sources. Even where the customer has a limited and clearly-defined set of functional requirements, data relating to other functions may be informative in the present and relevant to possible future developments.

Even within the context of detection-specific, testing-based evaluation, the evaluator should be aware that testing is often highly specialized. For example:

- On-demand testing, using a tested scanner to check a set of samples passively, rather than by opening and/or executing each scanned object.
- On-access testing, using the realtime functionality of the scanner to check for malicious code when an object is opened or executed.
- Time-to-Update (TtU) testing, monitoring the time it takes for a company to produce a signature for a specific threat. This approach to testing has declined, as testers have acknowledged the difficulties of implementing it accurately in the current threatscape, especially since most companies now make use of proactive detection technologies that reduce the need for signatures. However, the practice survives in the misuse of multi-scanner sites like VirusTotal and Jotti to monitor the perceived ability of one or a range of products to detect a specific sample. This practice is, however, based on incorrect assumptions about scanning technology that are explored further below.
- WildList or In-the-Wild (ItW) testing, based on the WildCore collection of replicative malware maintained by the WildList Organization (http://www.wildlist.org). This type of test is commonly used by organizations that award certification for detection performance, such as ICSAlabs, West Coast Labs, and Virus Bulletin. The scope and currency of the sample set is limited: it represents only a small subset of the totality of malware extant at any one time (i.e. viruses), and samples in the test set will no longer be "fresh" by the time they are used in testing. However, it has the advantage for the tester that it consists (or should consist) of valid, pre-screened malware. In addition, it provides a "level playing field" in that all mainstream vendors represented in the WildList Organization should have equal access to WildCore, so the risk of geographical or other bias favouring some vendors over others is reduced.

  Unfortunately, the term "In-the-Wild" testing (or some variation on the term) is often used without reference to its association with and use by the WildList Organization (Gordon, 1997). While it's not unreasonable to use the term in a general sense according to a definition like this – "... for a trojan to be considered "In the Wild", it must be found on the computers of unsuspecting users, in the course of normal day-to-day operations." (WildList, 2001), it's often used misleadingly to refer to unvalidated samples drawn from honeypots and honeynets, the testers mailbox (Harley, 2008) and so on. This is problematical, in that some researchers believe that 30% or more of such samples are likely to prove to be corrupted, and many vendors differentiate between such damaged samples and active or "live" malware. Tests that don't take this factor into account therefore favour samples that don't differentiate, but often neither the tester nor the tester's audience is aware that a bias has been introduced.
- Conceptually, testing signature detection is fairly easy, requiring only some valid samples of known malware. Testing a product's capacity for the detection of unknown malware is another matter (Jacob, Filiol & Debar, 2008). Some testers do this by creating new malware or by modifying known malware to create unknown variants. The anti-malware industry, however, being notoriously averse to the unnecessary creation of malware, prefers to advocate proactive testing, where a product is prevented from downloading updates for a

fixed period, then tested with samples that have been gathered during the time when updates were frozen: this can serve as a fairly accurate indicator of how successful a product is at detecting unknown malware (i.e. malware for which it has no malware-specific detection), depending on implementation.

- Detection of a single class of malware, for example rootkits, spyware, even the EICAR test file. You could regard WildList testing as a special case of this type of test, since the test set consists at present entirely of replicative malware, though it normally includes a wide range of sub-classes of such malware.
- Vulnerability detection, where a *vulnerability* in a scanned object is flagged, rather than the presence of malicious code that *exploits* that vulnerability. This is a legitimate test target. However, a review based on performance in such a test may be misleading if the tester/reviewer, or his audience, is unaware that some products may not flag that vulnerability if no exploit is present, but will detect even an unknown exploit of that vulnerability. As with other test types addressed in this paper, the validity of the test is in part dependent on the understanding of the tester of both the technology and the design philosophy of the vendor. If the tester assumes that only one design philosophy is legitimate, this is likely to introduce a bias in favour of products that conform with that assumption.
- Detection capability using “Out-of-the-box” configuration.
- Detection using the most paranoid settings (sometimes this *is* “Out-of-the-box” configuration, but often this isn’t the case, since many vendors prioritise speed over detection performance by default – whether this is appropriate is a discussion for another time).

Single/Multi-Function Detection Testing versus Whole Product Evaluation

Why is it important to maintain these distinctions? Because understanding the distinctions helps us to understand the differences between:

- Single-function detection testing (that is, testing a subset of a tested product’s detection capabilities: for example, its ability to detect fake antimalware, or its ability to detect malware in the course of a scheduled on-demand scan).
- Testing a product’s more general detection capabilities. It may be too much to expect one test-based comparative review to address the full detection functionality of every product under test. However, it’s not unreasonable to expect a test to give an accurate assessment of general detection capabilities unless it’s made clear to the audience that the test is highly specific.
- Testing the whole product. An evaluation/procurement process is often based on shortening an initial list by discarding those that don’t meet particular criteria. This may be perfectly appropriate. However, it’s not uncommon for an unwary tester to assume that a product that doesn’t work in the way he expects is not working as it should. In fact, it’s possible that the product will meet the needs of his audience as well as (or better than) a more “conventional” product. For example, the results of a scanner test based purely on static testing of static detection can penalize products that make use of dynamic analysis such as some forms of behavior analysis. In order to accommodate these complications, it’s necessary to understand that “detection” is a blanket term that includes many approaches to scanning. In consequence, detection performance can only be evaluated realistically by taking into account execution context.

Defining Execution Context

Execution context actually has two dimensions: one dimension maps to the execution status of the security software under test, the other to the execution status of the possibly infected object.

The most common differentiation applied to the execution context of an anti-malware scanner is the core differentiation between on-demand and on-access scanners or scanner components.

On-Demand versus On-Access

Mainstream anti-malware products have two main functional scanning modes, conventionally categorized as on-demand and on-access (or real-time). The essential distinction between them is that on-access scanning involves checking an object for infection or malicious content when it accessed. On-demand scanners check individual files, folders, or mounted disks, or as specified by the end user or system administrator. This may be literally "on demand", or according to some form of automated scheduling. Scheduling functionality may be built into the scanner itself, or the scanner may be called by another program such as a shell script or a scheduling utility built into the operating system.

On-access scanning does not invariably involve execution of programmatic content, and on-demand scanning does not necessarily require a scanned object to be inactive (i.e. not executing). However, sound testing practice requires that the tester be aware that on-demand scanning will not necessarily make use of the full detection capability of the scanning software. Depending on product, platform and other contextual parameters, the scanner may or may not be capable of a full dynamic analysis of the suspect code, whether scanning on-demand or on-access.

Not all products have a command-line interface (CLI) nowadays: where a product does have a CLI, it interposes an extra layer of complexity. The term command-line scanner usually suggests an on-demand scanning module: after all, one of the uses for a CLI is to launch a scheduled scan. However, depending on platform and context, it's not impossible or unknown for a real-time scanning module to be launched from a command-line prompt. For testing purposes, the same caveats apply to CLI on-demand scanners: the tester should not assume that the scanner will be able to execute a full dynamic analysis of the scanned object.

Execution context is, clearly, at least partly responsible for defining a program's execution status.

Execution Status

Execution status of the (possibly) infected object may include a number of possible states:

- The object is unable to execute because of the operating environment in which it finds itself (for instance, a Macintosh binary on a Windows PC)
- The object is unable to execute because it is prevented by other factors (behavior blocking software, system resource constraints)
- The object is capable of execution but is not currently active.
- The object is currently executing normally
- The object is currently executing but is modifying its own behaviour in response to some aspect of the execution context in which it finds itself. For instance, some malware behaves differently if it detects that it's running in a virtualized environment. This can happen in a normal business environment, of course: virtualization is a commonly-used systems tool in day-to-day office work, for instance in a multi-server context or in the context of emulating

an operating system (OS) on a system that doesn't support the "real" OS. However, many types of security software make use of some form of virtualization in order to carry out some form of dynamic and/or behaviour analysis. Self-modifying malcode may behave in a number of ways in these circumstances:

- It may simply terminate, or remain in memory but inactive
- It may continue to execute, but exhibit "innocent" behaviour
- It may attempt to make active use of any vulnerabilities in the virtualized environment, for instance to effect some malicious action on the host machine.

Basic Detection Algorithms

While exhaustive discussion of detection technologies is beyond the scope of this document, it is necessary at this point to define some basic approaches to detection as implemented in mainstream antimalware, in order to avoid confusion and ambiguity.

(Near-)Exact Identification

Exact identification has been defined as "Recognition of a virus when every section of the non-modifiable parts of the virus body is uniquely identified" or as calculating "a checksum of all constant bits of the virus body" (Szor, 2005), while near-exact identification checksums a single range of constant bytes (Szor, 2005).

Signature

Exact and near-exact identification have become inextricably confused with the much-misused term signature: however, the term signature is used (when unavoidable) in this document to refer to *any* detection algorithm that implements known-malware detection. A generic signature is a signature that detects more than one member of a malware family. Sometimes a generic signature will detect a new member of such a family, so the line of demarcation between a generic signature and a heuristic detection can be somewhat fuzzy.

Playing First: Proactive Analysis

Heuristic analysis uses a variety of approaches to detecting new malware proactively, as well as new variants that closely resemble known variants. Static or passive heuristics use code inspection without execution, but a number of closely-related technologies (emulation, sandboxing) use code execution in a virtualized environment to analyse behaviour dynamically. Where this analysis uses advanced heuristic analysis, the term active heuristics is sometimes used. Proactive analysis techniques provide an effective method of addressing the common cybercriminal practice of using packers and obfuscators to keep repackaging binaries so as to evade passive scanning techniques (especially malware-specific scanning). Unfortunately, it is less effective where they expend development resources on modifying instantiations of a malicious binary with the express intention of evading the latest versions of targeted scanners using the latest released updates: however, that particular problem is beyond the scope of this document.

Forensic Analysis and Execution Context

To understand the importance of execution context, it helps to know a little about malware forensics. In principle, there are two main approaches to the analysis of malware: static analysis and dynamic analysis.

Static analysis is based on the review of suspected code. While this is often most effective when performed manually, it can be facilitated and automated using scripting to incorporate an appropriate toolset. Conventional anti-malware can be seen as taking a similar but further-developed approach, in that it is almost completely automated and consolidates the analytical tools into the base package. It may search for known malware using a simple search-string or more sophisticated algorithm, or it may check heuristically for malicious characteristics: either way, the code is inspected passively, as opposed to by execution. This approach to detection is sometimes referred to as passive scanning.

Static analysis which is restricted to identifying known malware is often associated with exact identification or nearly (or near-) exact identification. However, static analysis is not restricted to (near-)exact identification, signatures (variant-specific or generic), or other malware-specific algorithms.

Dynamic analysis takes what is often a faster route to identifying malicious code, by observing its behaviour when it is executed. For this reason, it is often seen as synonymous with behaviour analysis, since if there is no execution, there is no behaviour to analyse. However, as previously noted, while the term passive heuristics can also be used to describe code inspection using heuristic analysis, it's also possible to apply the technique to the *predicted* behaviour of unexecuted code when actually executed: in other words, behaviour analysis does not necessarily involve either active heuristics or dynamic analysis, since the behaviour of a program can, in principle, be predicted by analysing its code without executing it.

Testing Methodologies

There are two primary categories of testing methodology: static testing and dynamic testing. (Three, if you consider hybrid tests that use elements of both static and dynamic methodologies to be a separate and primary category.)

Static Testing

One (so far unratified) definition of static testing can be summarized as the testing of products in a context in which scanned code is not at any time in control of the target machine, while they are being scanned. In principle, static testing according to this definition includes contexts where code is executed but has no control over host machine, as happens where a scanner uses emulation or a similar technology to allow the suspected malware to execute in a (hopefully) safe environment with no direct communication with the real processor. Whether this definition can be strictly applied during a particular test is a question specific to the implementation of both the product under test and the details of the methodology, since these have a direct bearing on the execution status of both the scanner and the scanned program. Even if a scanner is able to execute the scanned object in isolation from the real processor, products using a more dynamic approach (in a formal sense) to analysis may be disadvantaged in a comparative test.

Static testing has significant advantages for the tester, depending on implementation. On-demand scanning, which is usually regarded as static, is comparatively easy to set up and automate, even with large test sets, compared to dynamic testing. However, if the factors discussed in this paper are not taken into account, it can be wildly inaccurate in its assessment of some kinds of product.

Dynamic Testing

Dynamic testing is defined by the Anti-Malware Testing Standards Organization as tests where a PC is exposed to a live threat (for example, by attempting to execute the malware) as part of the test.” (AMTSO, 2008) By this definition, dynamic testing is considered to be a more effective test of “product efficacy” as it “directly mimics malware executing on a victim’s machine”.

Unfortunately, this definition does not constitute a direct antithesis to the definition of static testing above, since that definition also allows for execution of the scanned program, though only in a restricted or virtualized environment. We must also question the assumption that testing by using on-demand scanning is automatically definable as static testing, since that really depends on whether on the unclear relationship between on-demand scanning and static/dynamic analysis. If static testing includes restricted execution of scanned code, is it really the opposite to dynamic analysis?

At the time of writing, AMTSO has not ratified a guidelines document on static testing or a formal definitions document: when it does so, we must hope that the organization finds a satisfactory way to resolve this anomaly.

The same source (AMTSO, 2008a) argues that “dynamic testing is the only way to test some anti-malware technologies”: this is true, for instance of tests that require a connection to the internet (or a simulation thereof) in order to analyse a program’s behaviour on execution. It may be possible to argue that it is “appropriate as a test methodology for all types of anti-malware products.” However, dynamic testing is generally complex and therefore time- and resource-intensive to implement. It requires skill and technical knowledge to implement not only accurately, but safely. This difficulty is, perhaps, reflected in the fact that dynamic testing is only just starting to gain traction, 33 years after Cohen pointed out that it’s possible to predict the viral nature of a program by its behavior (Jacob, Debar & Filiol, 2008). Of course, predicting malice (viral or *non*-replicative) is a little harder.

Behavioural Testing

This term is not synonymous with dynamic testing, although dynamic testing is usually implemented as a means of analysing behaviour. However, as previously discussed, it is often possible and practical to predict behaviour without dynamic analysis, that is, by passive code review.

Some Specific Problems

There seems to be no end to the diverse and inventive approaches used by aspirant testers to evade good practice as the anti-malware would like to see it put into effect. Here are a few of the most commonly found.

VirusTotal is the best known example of a site that many people find very useful as a shortcut to checking a possibly malicious file (as well as submitting it to multiple vendors), but it’s often misused as (a) a means of monitoring vendor awareness of a specific threat (b) a quick and easy substitute for a comparative detection performance test. VirusTotal passes files submitted by a visitor to the site to a battery of command-line scanners. This gives the visitor a good chance of identifying a known malicious program, but the fact that no scanner identifies a file as malware does not mean it isn’t malicious, obviously. However, if a file is identified as malicious by one group of scanners but not another, it doesn’t necessarily mean that the second group is less competent at detection, either. Scanners that use sophisticated behaviour analysis, active heuristics and so on can be disadvantaged by misuse of such facilities for a purpose for which they were never designed or intended. Some = sites use VirusTotal submissions as a substitute for hands-on comparative testing, while security researchers outside the antimalware research community commonly use it as a tool to track the ability of the industry as a whole to detect a specific threat. The assumption that VT reports should, over time, get nearer to 100% vendor detection on one specific sample is based on a 1990s view of anti-malware as being primarily signature-based. There is, in fact, no absolute reason why a product that has effective heuristic or behavioural detection (which sites like VirusTotal doesn't necessarily measure) should "update" it to malware-specific detection when a sample is available. Such an update may happen, for example as a sop to the popular belief that detection has to be based on the principle that each instantiation of malware requires not only a unique detection but a unique name (Harley & Bureau, 2008): whether or not it does is not, however, a fair assessment of product capability. It's actually rather similar to "Time to Update" testing, which has declined as testers have realized that it penalizes products that use proactive detection techniques.

Virus Total is not really suitable for use for comparative testing,either: it is not a test site, and uses command-line scanners. As Hispasec Sistemas, the company who developed and maintain Virus Total, have suggest (Quintero, 2007), the use of the site for comparative analysis is based on at least two major misconceptions.

- The assumption that command-line scanners will work in the same way as desktop versions on a platform where fully GUI antimalware is the norm. Command-line scanners are more likely to be server-hosted nowadays, which makes servers a natural home for a range of test-related functions such as automated scanning of large, multiple test sets. Generally, command-line scanners inspect the code passively, rather than running it in a safe environment to see what it does in practice, so products that are heavily dependent on static analysis in the form of signature detection may seem to do better than products that make more use of proactive technologies than painstaking generation of signatures. In the real world, however, where on-access scanning is the first line of defense for most people, the advantage may swing the other way.
- The assumption that desktop and perimeter solutions, both of which are used by VirusTotal, can be accurately assessed for detection performance using the same testing environment. In fact, according to the product, platform and operating environment, both detection mechanisms and default settings may vary widely. For instance, perimeter products are likelier to make use of aggressive heuristics and be more tolerant of false positives. Some products use detections that are so generic that they amount to the detection of a class of executable rather than a class of malware, using such classifications as “suspicious”. These are not necessarily false positives in any technical sense: for example, they are often based on the fact that a file has been compressed, packed or otherwise modified using a run-time packer, or packaged in some obfuscated form.

Malware distributors often use passworded archives and run-time packers to obfuscate known malicious code in order to slip it past security software that would recognize the unobfuscated code. However, legitimate programs may also be packed, for a variety of reasons (compression, Digital Rights Management and so on), and it’s not unknown for legitimate programs to use packers associated with malware packing. Some products use the detected presence of packers and obfuscators as a heuristic indicator in its own right, an approach analogous to the common practice of blocking all executable files when encountered as an email attachment. It's a perfectly *rational* approach, as long as the potential customer (often the tester's main audience) is aware that they're making a trade-off: they'll be protected from all malware that has a blacklisted characteristic, but they'll lose access to those innocent files that have the same characteristic. Many modern scanners include detection algorithms based on combinations of packer detection, behavior analysis and known malware detection. However, such detections are not necessarily triggered by command-line scanning, depending on product and execution context, among other factors.

Old malware (known malcode for which the base code hasn't changed) using a new packer combination may not be detected until it actually attempts to execute: once it's unpacked (in an emulated or real environment) an on-access scanner can recognize it where the on-demand component of the same product on a different system may not.

Cross-platform Testing

It's important to a potential corporate customer in a mixed environment (where Apple, Linux and/or Unix, Netware and Windows desktop and server systems may all be use) to know how well the whole computing environment is protected by a product or product suite.

Heterogeneous Malware Transmission (formerly Heterogeneous Virus Transmission) describes the transmission of malware via an environment (operating environment, execution context) in which it is not capable of self-replicating (or executing at all), but can be transmitted passively. For example, an application which is only capable of executing in a Windows environment can be transmitted via another environment in which it can't execute (or can only execute using some form of Windows emulation). The original term was coined (Radatti, 1996) with reference to PC viruses transmitted via UNIX systems, and later extended to refer to PC malware transmitted by Mac systems).

There is a wide variation in the way that anti-malware products handle cross-platform threats, not only between competing products, but between different components of a single vendor's product range. For example, some Windows-specific desktop products do not detect Macintosh-specific malware and vice versa, whereas gateway products are more likely to range threats associated with a range of platforms, not just those specific to the gateway host environment. Of course, it's entirely reasonable for a reviewer to draw attention to these details of implementation, but the review audience may be misled if a tester is unaware of differences in architectural and infrastructural design, and doesn't take them into account when establishing a test methodology.

Execution context is highly relevant here: a Windows scanner may well employ one of the techniques previously described to execute PC-specific malware in a safe, virtualized environment, but it can't necessarily extend that technique to other system architectures. Testing meant to meet the needs of an audience used to a multi-platform environment has to cover the differing design parameters applied at different loci in a complex environment.

Security Suite Testing

It's becoming ever rarer to see mainstream protective software that focuses on a single range of malware (even viruses), or a single defensive layer (malware blacklisting). Increasingly, antimalware vendors, aware that antivirus software alone is insufficient protection in these dangerous times, are making available security suites including a range of tools such as anti-spam tools, Host Intrusion Prevention Systems (HIPS) and personal firewalls, in order to enable consumers to take advantage of defensive multi-layering.

This phenomenon has attracted the attention of testers whose experience is in security fields other than anti-malware, who may not take into account the fact that vendors whose origins lie in antivirus do not always implement these complementary technologies in the same way as more “traditional” vendors. While there are many instances of such a collision of mindsets, we’ll focus here on a single representative issue drawn from firewall testing.

There is a wide range of ways in which a firewall’s ability to block both external and internal attacks. The latter issue is sometimes addressed using a technique called leak testing, which assesses the firewall’s ability to prevent applications from transferring potentially sensitive data to an attacker. Firewall testers use specialized tools to simulate this behavior, but this is in sharp contrast to traditional practice in the AV industry, where some products have declined to treat simulated malware in the same way as real malware by detecting and blocking it (Gordon, 1995). Even the EICAR test file, which *is* generally prevented from executing by anti-malware, is not usually flagged as if it was real malware. However, in at least one instance, flagging leak test tools in the same way as the EICAR file is flagged would not meet the case, since detection of the application rather than the behavior (in this case the simulated “leak”) is likely to be regarded as gaming the system.

In this case, the main issue is a conflict between the expectations of testers from *outside* the antimalware industry, and the impact of historical and ethical considerations *within* the antimalware community on design decisions. However, execution context still plays a part, as does the *intent* behind the pseudo-malicious program. Similar (but by no means identical) considerations may apply with regard to other forms of simulated testing (Wismer, 2006).

Conclusion

While to some extent this paper focuses on the technical aspects of anti-malware testing and execution context, the problems touched on here go much further than a simple matter of getting the definitions right. Most of us in and around the antimalware industry live in a world where myth and misunderstanding inform the perceptions of most people of what good testing practice is or should be. In the past, the antimalware industry has not helped itself by maintaining that “if you don’t know what good practice in testing is, you aren’t qualified to test”, without actually helping either aspiring testers or their audiences to understand why so much past (Solomon, 1993) and current testing (Harley, 2008) is inadequate.

Hopefully, the attempts by AMTSO (AMTSO 2008b) to begin to establish testing standards, and the anticipated parallel initiatives from EICAR (Hayter, 2008) will start to break down the psychosocial barriers to popular acceptance of the need for more rigorous testing practices, with a better understanding of the principles of testing and the specialized technology behind both malware and antimalware.

References

Lee, A., & Harley, D. (2007a). Antimalware Evaluation and Testing. In D. Harley (Ed.), AVIEN Malware Defense Guide for the Enterprise (pp. 441-498): Syngress.

Harley, D., Slade, R., & Gattiker, U. (2001). Viruses Revealed: Osborne.

Harley, D. (2008). Untangling the Wheat from the Chaff in Comparative Anti-Virus Reviews. Retrieved 20th January 2009, from http://www.smallblue-greenworld.co.uk/AV_comparative_guide.pdf

Harley, D. & Lee, A. (2007b). Testing, testing: Anti-Malware Evaluation for the Enterprise. In AVAR 2007 Conference Proceedings: AVAR.

AMTSO (2009). Testing & Execution Context: Accurate Testing of Anti-Malware Detection. In preparation.

Morgenstern, M., & Marx, A. (2007). Testing of "Dynamic Detection". In AVAR 2007 Conference Proceedings: AVAR

Gordon, S. (1997). What is Wild?" Retrieved 19th January, 2009, from http://csrc.nist.gov/nissc/1997/proceedings/177.pdf

WildList (2001). Retrieved 19th January, 2009, from http://www.wildlist.org/faq.htm

Jacob, G., Filiol, E., and Debar H. (2008). Functional Polymorphic Engines: Formalisation, Implementation and Use Cases. In 17th EICAR Annual Conference Proceedings: EICAR.

Szor, P. (2005). The Art of Computer Virus Research and Defense: Addison-Wesley.

AMTSO (2008a). Best Practices for Dynamic Testing. Retrieved 19th January, 2009, from http://www.amtso.org/documents/doc_download/7-amtso-best-practices-for-dynamic-testing.html

Harley, D., and Bureau, P. (2008). A Dose by any other Name. In Virus Bulletin 2008 Conference Proceedings: Virus Bulletin.

Quintero, B. (2007). AV Comparative Analyses, Marketing, and VirusTotal: A Bad Combination. Retrieved 19th January, 2009, from http://blog.hispasec.com/virustotal/22

Radatti, P. (1996). Heterogeneous Computer Viruses in a Networked UNIX Environment. Retrieved 19th January, 2009, from http://www.radatti.com/published_work/details.php?id=32

Harley, D. (1997). Macs and Macros: the State of the Macintosh Nation. In Virus Bulletin 1997 Conference Proceedings: Virus Bulletin.

Gordon, S. (1995). Are Good Virus Simulators Still a Bad Idea? Retrieved 19th January, 2009, from http://www.research.ibm.com/antivirus/SciPapers/Gordon/Simulators.html

Wismer, K. (2006). Are good spyware simulators still bad. Retrieved 19th January, 2009, from http://anti-virus-rants.blogspot.com/2006/03/are-good-spyware-simulators-still-bad.html)

AMTSO Principles

AMTSO Glossary [in preparation]

Solomon, A. 1993). A reader's guide to reviews. Retrieved 19th January, 2009, from http://www.softpanorama.org/Malware/Reprints/virus_reviews.html.

AMTSO (2008b). The Fundamental Principles of Testing. Retrieved 19th January, 2009, from http://www.amtso.org/documents/cat_view/13-amtso-principles-and-guidelines.html

Hayter, A. (2008). Report from the 17th EICAR Annual Conference May 4-6, 2008 Laval, France. Retrieved 19th January, 2009, from http://www.aavar.org/'08%20eicar%20report%202.html

Jacob, G., Debar, H., & Filiol E. Behavioral detection of malware: from a survey towards an established taxonomy. In Journal of Computer Virology (2008) 4:251-266.