

ZERO-DAY MALWARE

Dr Igor Muttik
McAfee Labs, UK

Email mig@mcafee.com

ABSTRACT

The term ‘zero-day’ came from vulnerability research, but it is now widely used for malware, too. *Wikipedia* defines ‘zero-day virus’ as ‘a previously unknown computer virus or other malware for which specific anti-virus software signatures are not yet available’. Of course this is just silly – nearly all contemporary malware is zero-day according to this definition!

It is easy for any malware writer to obtain a security product and test that his or her creation is not going to be detected. There are many underground web portals offering cross-scanning services – they even include email notifications whenever detections are implemented by any of the anti-virus (AV) products. Thus, only a very lazy or careless malware writer would not be able to build a zero-day piece of malware.

Fortunately, streaming updates and cloud-based security protection redefine the ‘zero-dayness’ for malware. With a global security cloud, even a truly novel piece of malware may have the chance to hit only a handful of targets before global protection is rolled out. At that point, all other users would be safe. So, bad guys can no longer rely on the absence of detection for very long. This is the area where the agility of AV solutions is way ahead of contemporary vulnerability patching. We will argue that cloud-based security is blurring the line between reactive and proactive protection, making the term ‘zero-day’ rather meaningless. We suggest a replacement concept – an ‘exposure function’.

We will present a mathematical model showing that the impact of vulnerability exploitations and malware attacks can be measured based on the timing and intensity of attacks and the availability of protection. We will attempt to answer a perennial question – how can we determine which security software is best? We shall give recommendations to testers of security products.

INTRODUCTION

The purpose of security software is to minimize the cost of computer ownership. Ideally, security should protect a computer from the effects of any malicious attack while staying completely invisible. Detecting an attack instantaneously (or even before it happens) would be the ultimate protection, equivalent to thwarting a zero-day attack.

Unfortunately, security products are not always successful in doing this. The reason is simple – traditional desktop security relies on periodic updates ‘pulled’ onto the desktops by the OS and by AV software. Because the attackers have easy access to these security updates, they are able to test their malware against currently available protection; and, by modifying the malware, it is relatively easy to avoid detection. As a result, attackers will frequently enjoy a reasonably long window before protection is

deployed. The same approach to avoid detection works with multiple AV products (or, at least, those products which attackers care about), with just a little more effort. This is frequently referred to as a zero-day malware attack.

The situation started to change recently in favour of the AV guys due to the dramatic increase in the frequency of updates (down to minutes) and the use of cloud-based protection databases (down to seconds and globally). As a result, malware authors have a shrinking window of exposure.

ZERO-DAY DEFINITIONS

What prompted us to look at the problem of bettering security metrics were cloud-based security products. They use a remote security database of threats (usually accessed via the Internet). The endpoint security software contacts this database as/when necessary – for example, when a new object is created on any local storage device (or when one is clicked or executed). Such cloud-based products can be exceptionally agile in rolling out protection. It is a common scenario when, after just a handful of threat sightings (and the cloud clients themselves act as telemetry sources), the response is generated (the cloud database is updated to recognize the new threat based on telemetry) and from that point in time all other users are protected (globally!). The speed of cloud reaction can be within just seconds of the first sighting of a threat. Such a rapid reaction will protect the majority of users (sometimes even all of them – e.g. when the initial sightings come from honeypots), quickly mitigating zero-day malware attacks.

Because contemporary tests do not track security response over time, cloud-based solutions do not score as well as they should. In essence, modern agile security software technologies may get penalized in comparison with easier-to-test products.

Let us look closely at the definition of a zero-day attack (note – above we already mentioned the definition of zero-day malware but this is a little different). According to *Wikipedia*, ‘a zero-day attack or threat is a computer threat that tries to exploit computer application vulnerabilities that are unknown to others or undisclosed to the software developer’. There is also a notion of a *vulnerability window* which is the time between the first exploitation of a vulnerability and when software developers start to develop a countermeasure to that threat.

Essentially, these definitions compare two time points – that of the attack release and the moment when the very first mitigation is available.

There are at least three significant limitations to this simplistic view, especially if we want to use ‘zero-dayness’ as an objective security metric:

- the attack intensity (or even establishing if there has been any real attack at all) is disregarded
- the time span is ignored
- there is an implicit assumption that mitigation takes a long time.

The intensity of the attack is clearly relevant to measuring users’ exposure. Also, the same intensity of attack continuing for a longer period (greater time span) clearly poses more danger.

These factors are important and by including them in our security metrics we should be able to make them more informative than a simple zero-day ‘yes’ or ‘no’.

Unfortunately, additional confusion around the term ‘zero-day’ was introduced because many contemporary attacks which use vulnerabilities subsequently deploy malware. This further clouded the term zero-day because the same term can sometimes be applied to two different stages of the same attack.

CONTEMPORARY MALWARE ATTACKS

Contemporary malware distribution occurs in waves. The bad guys are now largely driven by monetary incentives. Once their returns from a piece of malware start to diminish (mainly due to security response from AV vendors and changes in the environment – e.g. monthly vulnerability patch roll-outs), they launch a new attack. Moreover, before the attackers observe an actual drop in return on investment (ROI) there is no need for them to launch another attack. (Incidentally, this is most likely to be the reason we are seeing so many new malware samples – increased frequency and quality of security updates has forced our adversaries to pump out more and more new malware samples to sustain their ROI.)

As an example, Figure 1 shows the count of field sightings for a threat called Spy-Agent.bw collected via *McAfee Artemis* telemetry (time is in hours, the dashed portion corresponds to the period when sightings decreased due to protection provided through regular DAT updating):

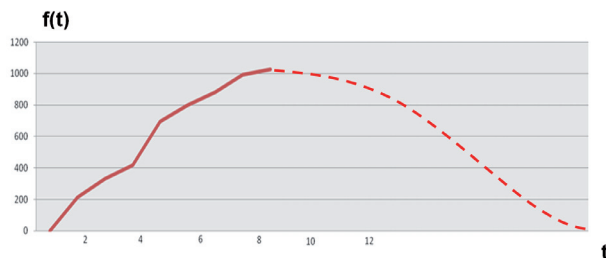


Figure 1: A graph of field sightings vs. time – $f(t)$.

Recording attack intensity over time and plotting the function of field sightings is easy when one has enough data (that depends on the number of honeypots/sensors/reports/etc. in the field). Depending on the type of the threat and scale of the attack (global or targeted, for example), the graph would look very different though:

- For self-replicating threats, the attacks may subside very slowly. This depends on changes in the level of malware reproduction in its ecosystem.
- For mass-spammed malware, the initial uptake may be very rapid.
- For non-replicating malware, attacks would probably be shorter and would normally be fairly quickly replaced by different attacks.
- Targeted attacks may have a single spike and affect very few targets.

For most modern attacks first there is the initial seeding (spamming of a URL linking to a malware sample, spamming of the malware itself or deployment of a drive-by attack kit on several websites). After a while the attack slows down due to the deployment of countermeasures (AV updates, security patches, blacklisting of bad URLs, shutting down bad websites, anti-spam rules, etc.). At some point the field sightings will disappear completely. For self-replicating threats it can take years (for example, the W32/Netsky worm, initially released in 2004, is still around). But regardless of the type of malware (common or rare, replicating or not) we can plot field sightings as a function of time declining in the end.

Malware infections are in a way analogous with real-life diseases: imagine, for example, injecting a virus into a guinea pig and monitoring the animal’s health. In the short term, the virus is likely to replicate a few times, which causes the immune reaction and production of antibodies. They find and kill the viral copies so our guinea pig is healthy again. However, there is an immune system reaction to the virus – a short learning process that occurs before a reliable response is deployed.

In their reactive mode security products operate similarly: they produce a response to the new attack and deploy it. They can observe a piece of malware just once or twice and protect many millions of users after that point. As we already mentioned, the response becomes especially quick with cloud-based security, where global online threat intelligence delivers data about new attacks almost instantly and the deployment of the response is also global and immediate. In this scenario, even a 100% reactive solution could be extremely effective in protecting users globally. For these protected users the ‘reactive’ security product provided proactive protection! Thus, reactive protection becomes proactive because it is generated and deployed quickly. We can see that there is no way to say at any given moment whether a product’s protection is reactive or proactive – it is reactive for some users and proactive for others. And it is the reaction time which converts one into another (in other words moves users from the unprotected state into the protected one).

CURRENT STATE OF TESTING FOR ZERO-DAY MALWARE ATTACKS

Apart from ‘zero-dayness’ being binary (yes/no), the measurement of successes and failures of security products has been based on binary logic as well. Detection tests check if a computer is either protected or unprotected because a corresponding malware sample was either detected or missed. The majority of tests that compare AV software are based simply on counting the misses over a set of files.

In the field of AV products the test of zero-day protection is usually performed by using so-called proactive testing methodology (also known as *retrospective testing*). This involves ‘freezing’ a product (creating a snapshot and subsequently denying the product the ability to receive updates) and then testing detection over attacks which appeared after the freeze point. Of course, in this scenario the frozen product will only face unknown threats and therefore all the reactive capabilities will be excluded from the test.

There are some problems with this approach, though:

1. The duration of this freeze can have a dramatic effect on the results. It is easy to imagine a security product that very quickly reacts to threats in the field and updates its heuristics and generics. This is not a theoretical speculation – apart from the previously mentioned cloud-based security products, there are now some that employ anti-spam rules, URL and IP blacklisting, etc. These rules are frequently updated and they block malware, too. A security product can proactively catch a newly spammed piece of malware just a few minutes after receiving a fresh anti-spam rule. And, of course, any retrospective test that froze the product before that rule was received would register the malware sample as not proactively detected. Yet in the real world it would have been.
2. If protection is not delivered incrementally to the client but is either constantly streamed, delivered via P2P communication or is cloud based then a product cannot be tested with the frozen definitions (stream/cloud/P2P are external and not under the tester’s control). To address this problem the AMTSSO guidelines on cloud-based security products [1] recommend collecting field data over time and averaging the results.
3. When a proactive detection is recorded, there is an implicit assumption that the product will keep detecting the threat later, when the attack really happens. Moreover, there is also an assumption that the product will keep detecting reliably (meaning that it will always detect). In reality there can be situations when detections ‘drop off’ completely, change or disappear temporarily.
4. There is an implicit assumption that a product which missed the first sample would fail on the second (and subsequent) samples. But imagine a product which always misses the first sample but always detects the second (and subsequent) one. Such a product would score zero in proactive protection, but overall it would actually provide very good protection to most of the users.

The time point of the ‘proactive’ test is shown by a dotted line on Figure 2 – it is before the attack starts. This is, of course, not the best time for a test.

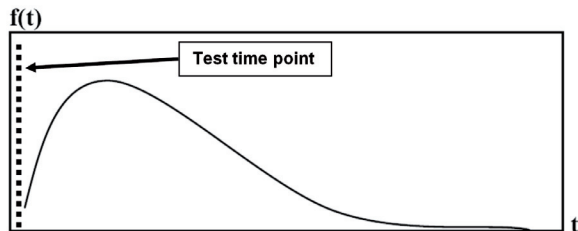


Figure 2: A ‘proactive/retrospective’ test timed ‘before’ the attack.

Apart from proactive tests, other tests try to isolate and measure the ‘reactive’ capabilities of scanners (typically on-demand scans over relatively large sample sets). These show much higher detection scores – for example, check ‘Retrospective/Proactive’ detection rates vs. ‘On-Demand Comparative’ [2].

There are problems with reactive testing too (Figure 3):

1. The test runs after the attack and there is an implicit assumption that detections were present when the attack was happening.
2. It is very easy to manipulate the results. There could be a product that failed miserably in the field but scored really well in a test – if the detections were added immediately before the test started.

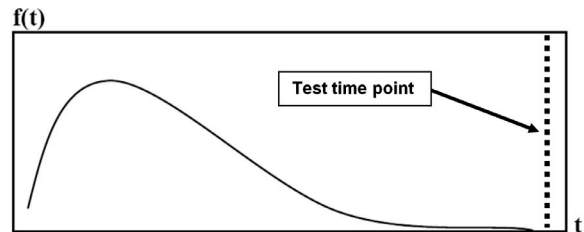


Figure 3: A ‘reactive’ test run after the attack.

The time point when the ‘reactive’ test is executed is shown by a dotted line in Figure 3 – it is after the attack is over. In reality, such tests include many samples and the gap after the attack varies for every sample.

Because of their intrinsic limitations, the testing methodologies, perhaps unsurprisingly, produce very different results. Figure 4 gives the average detection rate for six AV scanners tested by AV Comparatives from 2004–2010. The graph shows the ‘industry average’ performance in reactive and proactive tests. You can see that the gap between these two kinds of test is really wide.

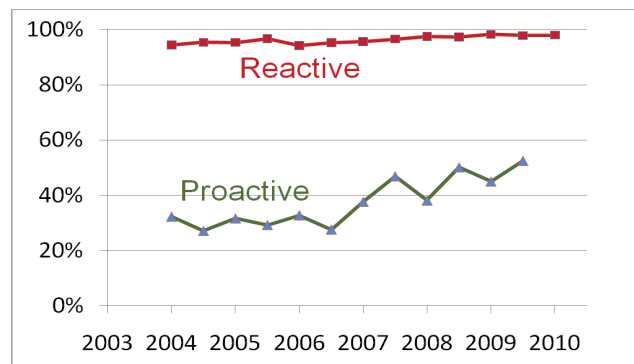


Figure 4: The ‘industry average’ efficacy in proactive and reactive tests

How useful is a statement that real protection probability is between, say, 50% and 99% boundaries? Additionally, even separating pure reactive and proactive scores is pretty much impossible so the boundaries are rather fuzzy.

Some tests attempt to mix reactive and proactive results together [3] by separating samples into four ‘weekly buckets’ (three reactive and one proactive).

Yet another testing metric of AV products’ quality is tracking the time between the first sighting of a threat and the moment when protection is made available [4, 5]. This ‘reaction time’ testing was born when there were global outbreaks (~1998–2004).

Global outbreaks are now a thing of the past and so the popularity of this kind of testing has declined. The ‘reaction time’ approach, however, demonstrates the importance of timing in evaluating security products.

All these approaches underline the fact that the agility and timeliness of the security updates are very important in measuring the quality of protection. So, how can we track security reaction over time?

TRACKING SECURITY REACTION OVER TIME

Expanding existing testing methods and tracking the entire history of security responses over time would provide deeper visibility into the agility of a product and, therefore, will yield a metric better than existing proactive and reactive tests can provide. Instead of testing at only two time points (as we have shown above these time points are rather unfortunately placed) we shall have the entire protection history.

A natural quandary related to ‘attack tracking’ is that before an attack starts one cannot be sure if field sightings even belong to the same attack. To resolve this problem we can collect raw data about field sightings (as well as the raw security reaction of the product under test) and sort these events later, when we have enough knowledge to determine their nature (malicious or not) and classification (which specific attack they belong to). We shall be able to check the validity of the security reactions (and sort them into false negatives, false positives, true negatives and true positives). Moreover, full historical records will allow re-processing to correct classification errors.

Separating different attacks is easy to do for static malware (one can simply compare samples or their cryptographic hashes) but can be hard in the case of polymorphic code (regardless of whether it is self-morphing or server-side polymorphic). It is important to realize that at this stage we cannot rely on security products to classify or group attacks because the protection may not be available yet. Even after the attack is over one should avoid relying on security products to classify the attacks because this introduces a certain bias (it is not a good idea to use any product under test to affect the classifications and test set). Later on we shall discuss how to avoid this ‘attack-splitting’ step.

An additional benefit of tracking attacks over time is that products may actually temporarily lose detections during attacks (and both reactive testing and retrospective testing are very likely to miss this fact altogether). Any product may have unreliable detection (either due to a buggy update or due to randomness of security responses – e.g. caused by uninitialized variables, a memory footprint issue, or something similar), and the overall product efficacy observed at a single time point may vary considerably from the real one observed over a period of time.

As usually happens, understanding a problem in detail logically leads you to a solution. To improve zero-day and detection rate metrics we need to perform continuous testing over time and accumulate security responses. Then we’ll use this recorded data to compute the probability of protection by a given product – regardless of whether that protection is reactive or proactive.

SUGGESTED METRIC: THE PROBABILITY OF SUCCESSFUL PROTECTION

Let us first look at tracking an individual attack. Now that we have tracked the security reaction of a product during an attack, we can plot both the attack intensity and security reaction together on the same graph. (We shall discuss later how to combine the results obtained for individual attacks into an overall score.)

The grey area on the graph (Figure 5) indicates that a product had protection (correct security reaction) $r(t)=1$:

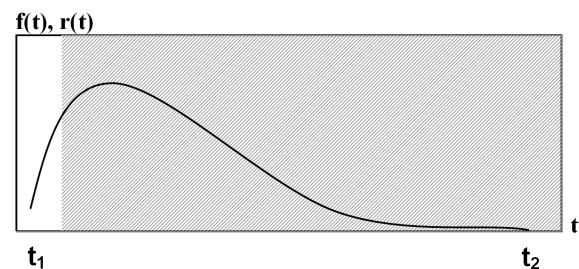


Figure 5: Field sightings $f(t)$ and security reaction $r(t)$ in grey.

Or, for example, if a product had an unreliable security reaction then we may have a graph like this (Figure 6):

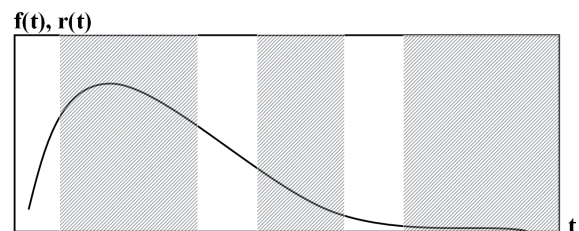


Figure 6: Unreliable security reaction $r(t)$.

Several simple metrics can describe the quality of the protection presented in these graphs: the delay in providing the first reaction (see [5]), the reliability of protection (‘unreliable’ if the detection was ever lost after first being introduced), etc.

Let us have a look at two examples (Figures 7 and 8).

The first case (Figure 7) provides no proactive protection but, later, the product successfully covers the great majority of users.

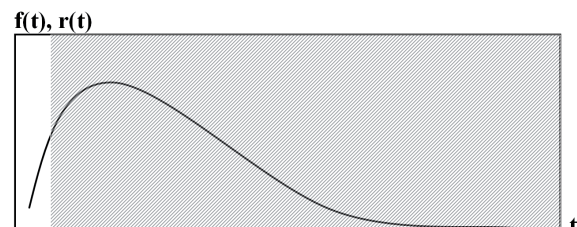


Figure 7: Security reaction $r(t)$ missing attack at start.

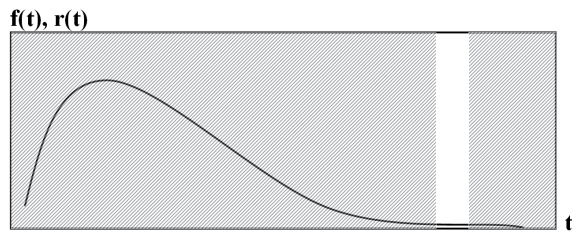


Figure 8: Security reaction $r(t)$ missing attack at the end.

The second product (in Figure 8) has ‘unreliable’ detection but the temporary failure at the end of the malware attack would affect a smaller number of users compared to Figure 7. Clearly, neither is perfect but the product from Figure 8 performed better (because there are fewer users affected during the ‘tail’ of the attack). On the other hand, a ‘protection gap’ (lack of detection) of the same duration during the attack peak would have been even worse!

So, how can we make sure these factors are taken into account in our metric? A correct metric should satisfy the following three conditions:

1. It should be 100% for perfect protection of all users over the entire lifetime of the attack
2. It should be 0% if protection is not available at any period of time during the attack
3. It should be affected more when there are more field sightings (i.e. when $f(t)$ is larger).

We want our metric to reflect the probability of successful protection over the user base and over the time of the attack. By definition, such probability, p , is the total number of protected users over time divided by the total number of attacked users. To compute these totals we integrate respective functions over time. Thus, a formula to calculate the protection probability (quality of protection) for an attack would look like this (Equation 1):

$$p = \frac{\int_{t_1}^{t_2} f(t) \cdot r(t) dt}{\int_{t_1}^{t_2} f(t) dt}$$

Equation 1: Probability of protection for an attack.

In our formula $f(t)$ is the attack intensity and $r(t)$ is the security reaction (both are functions of time). We integrate from the beginning of the attack, t_1 , to the very end of it, t_2 .

In mathematical terms the bottom integral is the area under the whole field sightings (attack) function – it represents all user hits over the entire lifetime of the attack. The top integral represents the area under the protection function – it is the total number of sightings when users were protected (the bigger this

area is, the better; its maximum is reached for perfect protection $r(t)=1$ and then both integrals are the same).

We can see that our conditions are satisfied because when $r(t)=1$ (protection always available) we have a trivial case of dividing equal integrals and the result is $p=1$ (protection during the entire attack was always perfect, or 100%). Similarly, for $r(t)=0$ (protection never available) we will get $p=0$. You can also see that our approach gives more weight to security failures when the attack is most intensive (a drop in $r(t)$ near the peak of $f(t)$ will result in a larger area of loss compared to, say, at the tail of the attack when $f(t)$ is much lower) and affects most users – exactly as it should be.

It is a trivial fact that the probability of successful protection, p , also defines the probability of a successful attack which is simply $(1-p)$. In other words, our metric can be applied to the calculation of the return-on-investment (ROI) values for both defenders and attackers (although the latter, of course, is an unfortunate side effect!). For example, if p (the protection probability) is very low then the attackers have a high ROI. If a security vendor has growing p (growing quality of protection) then the R&D investments do actually result in increasing protection for their users.

There is an interesting corollary in our model:

Corollary 1. We should verify the protection function $r(t)$ when and only when (if and only if – aka iff) there are field sightings of malware. While the attack is in progress any temporary glitch in protection should affect the quality metric. At the same time, it would be wrong to check protection when there are no field sightings because protection failures at these ‘quiet’ times would have no effect on users in the field. (We assume, of course, that the traps/honey pots/reports provide adequate global field visibility.)

EXPOSURE FUNCTION AND PATCHING VULNERABILITIES

If we have a security reaction function then in its simplest form it is either $r(t)=0$ or $r(t)=1$. A complementary function will be simply $1-r(t)$ – it will be zero when protection is available (i.e. when there is no exposure) and it will be equal to 1 otherwise.

Now, if we multiply this function (which is either 0 or 1) and the attack frequency, we get a function which represents the ‘population exposure’. Population exposure function $f(t) \cdot (1-r(t))$ is a generalization of *zero day* and of *window of exposure*. You can see in Figure 9 that both of these terms can be derived from the exposure function. Apart from timing the initial protection the population exposure function carries information about the number of field attacks and the reliability of patching.

So, if we want to reduce the number of users affected by malware (or by some vulnerability) then our goal should be not just to avoid zero-day exposure but also to minimize the entire exposure function.

Exposure functions will also apply both to malware attacks and to exploiting software vulnerabilities. The biggest difference compared to cloud-based malware protection (at least at the moment) is that vulnerability patches use a traditional software

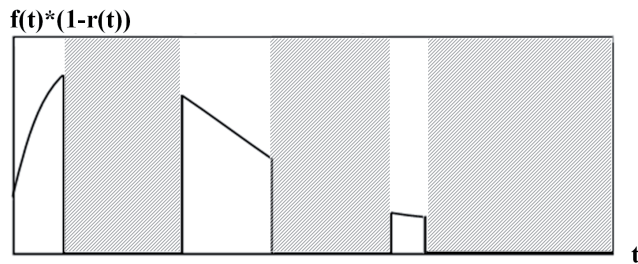


Figure 9: Exposure function – $f(t)*(1-r(t))$.

distribution model and deployment is slower. (Patches also usually come from one source while malware protection is a highly competitive area and there are multiple protection offerings due to that.)

We described above unreliable malware detection. One can also imagine unreliable patches for software vulnerabilities. They may result in several exposure windows – for example, if the first patch closes a vulnerability but, later, another patch opens it back up. (Figure 9 would represent a vulnerability being opened three times and patched thrice too. A curious question here would be if second/third exposures count as *zero-day* according to the existing definition or not!)

MULTIPLE ATTACKS

If we deal with many attacks, then we need to satisfy the same three conditions as for a single attack. There is one extra condition though – a larger attack (affecting more users) should have more weight. This happens automatically if we sum up the protection functions and the exposure functions separately and then divide one by another as shown in Equation 2:

$$p = \frac{\sum_{i=1..N} \left(\int f_i(t) * r_i(t) dt \right)}{\sum_{i=1..N} \left(\int f_i(t) dt \right)}$$

Equation 2: Probability of protection for multiple attacks.

In this case N equals the number of attacks.

Our extra condition is automatically satisfied because the relative scale of the attacks is automatically taken into account via $f_i(t)$ being higher for more prevalent attacks (we assume here that the field data for all attacks is coming from the same network of sensors). It is also possible to add additional weights to ‘more dangerous’ attacks (for example, those stealing user data) through introducing an individual attack multiplier (this is left as an exercise for the reader).

Unfortunately, classifying samples and separating attacks is not an easy job. We can significantly simplify the logistics of interpreting data if we remove the step of separating the attacks. Is it possible? Yes, but only if we treat all field sightings of malware as equal (that is, if we do not assign individual attack multipliers).

If we do that then the computations in Equation 2 are reduced to the same formula as in Equation 1 which we used for a single attack. Essentially, all field sightings from multiple attacks are treated as **one** attack with no internal structure. Such an aggregated attack will have no ‘start’ or ‘finish’ because malware attacks occur constantly. Therefore the calculation of our protection probability metric will need to be an ongoing (continuous) process. The benefit of tracking all attacks as one is that with more data the results will become more statistically significant.

Applying this improved metric should hopefully bring us closer to answering the question which everybody wants to know the answer to – which security software is the best? Our resulting metric can reflect a global view of the attacks over time (can be ‘a moving average’ in the case of continuous testing) – this would be superior to current testing methods and should inch us closer to the answer.

PRACTICAL IMPLEMENTATION

The theory above is nice, but we must eliminate several obstacles before it can be used in practice:

1. Not having enough field sensors/honeypots/traps/reports may not allow decent tracking of the attack-intensity function $f(t)$. Unfortunately, there is no substitute for real field data, so the only proper way to solve this problem is to get more field feeds. There is an industry effort underway for sharing sample metadata in IEEE XML format [6]. This effort assists in boosting the volume of field data. About ten security companies are already participating in this exchange (at the time of writing this paper, in June 2010) and the number is constantly growing.
2. There is no clear definition of security reaction $r(t)$. Whether a security product is successful in providing protection may be debatable because some products (typically behaviour-based products) may, for example, react after executing malware. Some malicious actions may have occurred at this point; thus evaluating whether blocking is successful (no stolen user information? no data egress? no persistent changes to the system?) could be controversial. All in all, defining $r(t)$ is not a trivial task and may justify writing a separate paper. (Watch <http://www.amtso.org/documents.html> for AMTSo’s expected guidelines on this.)
3. We assumed that $r(t)=0$ or $r(t)=1$ while for polymorphic malware, for example, it may be something in between to represent unreliability of detection at a given moment in time (if detection of a specific polymorphic virus is not reliable, that is). Replicating polymorphic viruses in order to track changes in $r(t)$ may be desirable but is certainly not trivial.
4. Yet another assumption is that $r(t)$ is global and does not vary across the user base. The speed of the updates’ deployment may have some effect here (with the cloud-based security products being potentially easier to handle due to negligible delays.)

5. We have to deal with discrete summing (instead of integrals) because the real $f(t)$ and $r(t)$ are not going to be mathematical functions but most likely timed records in a database (representing times and counts of attacks). This is trivial to do (Equation 3):

$$p = \sum_{i=1..N} (f(t_i) * r(t_i)) / \sum_{i=1..N} f(t_i)$$

Equation 3: Practical calculation – discrete sums instead of integrals.

Here, N is the total number of time points t_i .

There is an obvious optimization – we can exclude time points when there were no updates to security software (which means $r(t)$ is the same as it was before). This optimization would not work with cloud-based security products however.

Additional considerations:

1. If security software uses different update cycles for different components (for example, one for reactive protection and another for proactive), then the calculation of the final probability needs to cover periods longer than the update cycle to perform meaningful integration and averaging.
2. The generation of local knowledge (learning through artificial intelligence, which is capable of creating a local security response).
3. For cloud-based security there could easily be a dependency on the location of the client and the connectivity conditions. The frequency of updating is also unknown, so all field sightings would require a check.

CONCLUSIONS

1. Zero-day malware is a poor term and the very concept of ‘zero-dayness’ provides too simplistic a view on users’ exposure to an attack. That can be improved by looking at the exposure function which can be constructed from the data on the intensity of an attack and security response over time.

We recommend that anti-virus professionals avoid using the term *zero-day malware* as it overlaps with vulnerability research and can be confusing. When we face new attack vectors (like worms spreading through open shares or USB drives), viruses spreading via new objects (like HLP, PIF, CHM or LNK files) and malware for new platforms (e.g. PSP3 and iPhones) we would be better off calling them what they are instead of piggybacking on terminology from adjacent security branches.

2. Existing reactive and proactive tests are not timed properly (they evaluate protection before/after the attacks). Better tests would take into account the timing of protection (tracked continuously and processed historically) and the number of affected users.

3. We introduced an ‘exposure function’ which is a generalization of a zero-day concept. Our way of calculating the probability of successful protection (based on field sightings and security reaction functions over time) is a more sensible metric than the ‘binary’ approach provided by stating whether an attack is ‘zero-day’ or not.

Our metric can be used to evaluate and compare different kinds of security products, including even very hard-to-test, cloud-based security solutions. It can also be applied to anti-spam and vulnerability patching as well as to software updates in general.

REFERENCES

- [1] AMTSO Best Practices for Testing in the Cloud Security. <http://www.amtso.org/documents.html>.
- [2] AV Comparatives Retrospective/ProActive Tests for 2004–2010. <http://www.av-comparatives.org/comparativesreviews/main-tests> and http://www.av-comparatives.org/index.php?option=com_content&view=article&id=127&Itemid=162.
- [3] Hawes, J. VB RAP Testing. <http://www.virusbtn.com/vb100/vb200902-RAP-tests> and http://www.virusbtn.com/news/2009/10_09.xml?rss.
- [4] Marx, A. Outbreak Response Times: Putting AV to the Test. http://www.av-test.org/download/papers/2004-02_vb_outbreak.pdf.
- [5] Marx, A. Anti-virus outbreak response testing and impact. <http://www.virusbtn.com/conference/vb2004/abstracts/amarx.xml>.
- [6] IEEE Meta-Data Sharing XML Schema. <http://grouper.ieee.org/groups/malware/malwg/Schema1.1/>.