A multiple layered approach to malware identification and classification problem

Tonimir Kišasondi[1], Domagoj Klasić[2], Željko Hutinski[1]

Faculty of Organization and Informatics[1], Department of the national CERT[2]

University of Zagreb[1], Croatian academic and research network – CARNET[2]

Pavlinska 2, 42000 Varaždin, Croatia[1], Josipa Marohnica 5, 10000 Zagreb, Croatia[2]

tonimir.kisasondi@foi.hr, domagoj.klasic@carnet.hr[2] zeljko.hutinski@foi.hr[1]

Abstract. *The increasing threat of malware is a constant problem for information system security. Current detection methods are showing lack in sufficiency and are bulky, with a slow response to high traffic needs and for new samples. In this work we will present a method for in-depth malware identification and classification. We will show a concept of a multi layered approach where we can detect and classify malware mixed with legit data samples based on speed or precision trade-offs. We will employ a classification and risk based method with various detection criteria that can identify various hazardous aspects of various malware instances. The classifiers will be organized in layers which will help us in building various high speed or high precision detectors based on the protection needs and requirements.*

Keywords. malware identification, multilayer classification, malware, botnets, static analysis, runtime analysis

1. Introduction

Today, malware is a growing concern for computer security. From 2003 till today, there have been significant incidents involving malware, such as Titan Rain and GhostNet attacks that have been perpetrated with the help of malware. Most current methods of malware detection are based on pattern (signature) matching or behavior analysis. Signature matching is quick, easy and very reliable if you can match the correct signature which is usually stored in a signature base which is updated every couple of hours. The problem with signature detection is when we want to match a malicious sample for which we don't have the signature. Anti-malware houses have to analyze the malware and then issue a signature for it's anti-malware software so the malware can be detected.

Unfortunately, there is a gap between the following events:

1. A new malware type begins its infection run (outbreak)
2. Anti-malware houses receive a sample of the malware
3. Anti-malware houses issue updates for their software that can detect and remove the sampled malware
4. Users of the anti-malware software begin removing the malware
5. The infection rates for the specified malware type drop because of the detection mechanisms

Currently, there are about 40 anti-malware applications that are issued by their respective anti-malware houses or companies. About 50% of these houses are predominantly using signature detection, where the other half uses signature detection mixed with heuristics, behavior analysis or other proprietary methods. The downside on the behavior based methods is the time that is needed to detect a sample. On the other side, signature detection by that works only after the stage 3, when we can detect the sample with the signatures. We would like to propose a new model that should be able to work from stage 1, because it wouldn't be confide in only signature detection, but also would be more flexible to detect a wider variety of malicious code.

2. Prior research

Much research has been done in the field of malware analysis and prevention. One of the few formal results has shown us that it is not possible to create an algorithm that can perfectly detect all possible viruses (malware software) [3, page 1]. Although this result is discouraging, researchers have found a way to increase the detection rate for malware samples. Some researchers have tried to approach the problem using strict formalization. In [9] authors have proposed a malware formalization based on process algebra. Authors have also stressed that using such a

formalization can improve detection and prevention of malware samples. There have also been attempts at classification of metamorphic and polymorphic malware samples using formalism. In [10] authors use value set analysis to detect different instances of the same metamorphic malware family. Authors in [2] have demonstrated the usage of Strong Token-Pair (STP) signatures scheme for detecting polymorphic worms. These papers provide a valuable advancement in the field of malware detection, but they are focusing only on a few classes of malware.

Other researchers have focused on analysis of different behavioral aspects of malware. In [8] researches propose a system for detecting malware infection that monitors network communication between internal and external entities. Authors in [14] have created Pandora, a system for detecting malware that focuses on processing behavior and information access. It works by detecting which information malware accesses and checks if this access breaches user privacy.

3. Methods used by malware authors

In order to evade detection as long as possible, malicious software uses many anti-reversing techniques. These techniques significantly prolong the time required for successful analysis of malware, and thereby reduce the detection rate of modern anti-virus software.

Most malware does not use direct code obfuscation but instead rely on packer to hide from analysis. Packers are programs designed to compress or encrypt another executable program in place and therefore be completely invisible to the end-user [5, page 287]. Although it is very difficult to say exactly how many malware is packed, some studies estimate that over 80% of them use some form of packing [7, page 1]. The most widely used packer is UPX, while behind him we can find ASPack, FSG and UPack [7, page 2.]. There are also malware samples that use more advanced packer programs which represent the pinnacle of the packing technology, while some of them use packing methods not currently seen in the wild.

Malicious software also uses more direct anti-reversing techniques for which malware authors write code themselves. For example, Conficker approaches the problem of anti-reversing using multiple layers. On one layer it's packed with UPX, second layer is packed with a custom made packer and it also uses cryptography to hide its network traffic [4].

There is a large number of anti-reversing techniques, but most of them can be categorized to one of the following categories (as presented in [5]):

- Code Encryption
- Anti-debugger techniques
- Anti-Dissasembler techniques
- Code obfuscation
- Control flow transformation
- Data transformation

No matter what anti-reversing technique malicious software uses in the end it's functionality doesn't change. And based on that, we think that a multiple layered approach to identifying malware would be more suited, because we don't want to invest a large amount of time in bypassing anti-reversing techniques.

4. Analyzing detection and response rates

As a part of our preliminary research we wanted to experiment with the detection rates of popular anti-malware software. Since [13] offers almost all current and up to date scanners and forwards the sample to anti-malware houses for analysis, we used it to assess some samples. In the course of the authors work, we managed to collect a few thousand unique malware samples, and used a small subset of those for testing. In this work, we will list only some most interesting conclusions from our experiments:

The main weakness of signature matching is that even basic alterations to the core of the malware will be undetectable to the scanner. We can show this problem easily with the following experiment. We took a number random malware samples out of our batch, and employed a simple packing method. For example, here we list 5 very popular malware samples and the detection success with the regard to 39 popular anti-malware scanners provided by an online service [13] at the time of our analysis. Our results are presented in table 1:

Table 1: Detection rates for selected malware samples

Malware sample:	Detection rate:
alpha	37/39 (94.87%)
beta	38/39 (97.44%)
gamma	37/39 (94.87%)
delta	33/39 (84.62%)
eta	28/39 (71.8%)

After that, we took a simple packer [12] and ran it each of those samples trough the same scanners. It is important to note that the demo version is free to download, and anyone can download and run the packer and pack any common malware type, which doesn't require any special malware writing skills. Our results follow in table 2:

Table 2: Detection rates for packed samples

Malware sample:	Detection rate:	Difference:
alpha-vmp	18/39 (46.16%)	48.71%
beta-vmp	18/39 (46.16%)	51.28%
gamma-vmp	16/39 (41.03%)	53.84%
delta-vmp	15/39 (38.47%)	46.15%
eta-vmp	17/39 (43.59%)	28.21%

According to this, even simple trivial obfuscation mechanisms like packing can defeat a lot of signature based scanners. The anti-malware applications that detected the sample were either using behavior or some other in-depth analysis, so they could unpack the malware and test it. It is important to note that behavior based detection takes longer than signature matching, and it is logical to assume that even lower detection rates could be achieved with a better or custom built method. Since [13] automatically sends the sample to anti-malware houses, we wanted to assess the speed in which the anti-malware houses would recognize our packed sample as malicious and issue signatures to detect our packed variant. For that experiment we used the evaluation VMProtect packer, picked an extremely popular and simple malware sample (which had 100% detection rate by all anti-malware tools), we packed the sample and uploaded it and scanned it with [13] in 24 hour intervals for 10 days. That way we can see how fast can anti-malware houses deploy signatures for new malware variants. Our results are in table 3:

Table 3: Daily detection rates for a selected sample

Day	Detection rate
1	16/40 (40.0%)
2	19/40 (47.5%)
3	21/40 (52.50%)
4	23/41 (56.1%)
5	27/42 (64.29%)
6	27/40 (67.5%)
7	27/41 (65.86%)
8	27/41 (65.86%)
9	28/41 (68.3%)
10	29/41 (70.74%)

As we can see, it took the 40 anti-malware houses about 10 days to improve their detection rate for about 20%. It is important to note that this is a simple single stage obfuscation mechanism and better results could be had by coding a custom packer. Also, the results could be skewed because our sample was just forwarded as a malicious one to the anti-malware houses and was not released into the wild. We believe that if our malware sample was a real live high risk malware, the anti-malware houses hopefully would have a faster response. Unfortunately, there is no representative, ethical and legal way to assess this response.

5. Classifier elements

Our classifier uses a multiple layered approach to malware identification. Each layer of identification performs one test on the malware subject. With each test performed, our classifier engine computes the total score of malware risk. This score indicates if the subject in question is legit malware sample or not.

Our test's can roughly be divided in two groups.

- Static tests
- Behavioral analysis tests

Static check include test that are performed without executing the malware sample and do not pose any risk to the machine running the tests. On other hand, behavioral analysis test observe the patterns that malware is exhibiting after it's been executed, preferably in a virtual machine.

Here are some static tests that we recommend:

1. Entropy analysis:

An average standard PE executable or other non compressed files have low to medium Shannon entropy, if additional compression, encryption or packing is used the entropy will rapidly approach to 8 bits. The samples that we analyzed gave the same conclusion that high entropy samples use encryption or packers. This is a low complexity, fast check that can help us to decide will we try to scan the sample with known malware patterns or try to unpack the sample and maybe continue with further analysis.

2. Packer analysis and unpacking

If the entropy value is high, we can try and detect packers. We can use some publicly known patterns like [packpatterns], where we can use PeID [11] or other methods like Ero Carrera's python pefile package [6], which we used to implement a custom packer detector which we compared to PeID. If we detect a common packer like UPX, we can try and unpack it. Failure to unpack a standard packer should raise a red flag that would mean that some kinds of anti-debugging/disassembly or obsfucation methods are used. Some custom packers are custom built for malware or are made to obfuscate the file to thwart reversing. The use of custom malware centric packers or usage of standard packers with anti-reversing methods should automatically raise a high risk warning.

3. Identify crypto signatures

Cryptographic algorithms can be detected by their implementation in code. S-Boxes, P-Boxes, standard initialization vectors, library calls, machine code patterns or other elements are static in code, and can be detected. Encrypted chunks are not necessary a high risk case, but are a good indicator that we need to submit the sample to behavior testing.

4. Known code patterns

Malware authors, especially the ones that created multiple malicious code variants share the common trait as each programmer: They reuse their code. Similar code chunks like update processes for malware, fast flux techniques, stealth or polymorphism segments, payloads, exploits/shellcode, rootkit or other elements are reused between the malware created by the same authors or by the same group. If detected, such malicious code patterns are a high risk threat.

Complementing those static tests we can run behavior or dynamic tests:

1. Self removal test

A large percentage of malware samples will remove itself from current working directory to some obscure location on the system in hope of avoiding user detection. Most legit software doesn't exhibit this characteristic. This classifier should monitor the executable that's being analyzed and check if it has deleted itself from the current working directory. Although this is not a definite indication of malware behavior it will raise suspicion.

2. Persistency test

Most malware samples (excluding only memory resident malware) will try to use some known auto-start method in order to ensure the survival of system shutdown or reboot. Although legit software also uses this method, there is a big difference - malware will not ask user for permission and it will try to insert itself in less known locations that are not reserved for user software but for operating system use. If we cross-reference this test with self removal test, we can get a strong indication of malware behavior.

3. Third-party injection

Malware inject their code or dll in another process using mechanism provided by operating system. Using this techniques malware can go undetected by executing its code in trusted applications process. Legit software rarely has any need to create threads in remote processes or load rouge dll's. Our classifier should monitor the subject for any such attempt and logs it adding it to the global score of the samples malware behavior.

4. System call tracing

Most AV solutions employ some kind of heuristics based on examining which system calls an executable is using, also, the order of system calls is of significance. However, most AV solutions have problems with packed or encrypted malware. Our classifier employs a novel approach at tracing system calls that can't be circumvented by regular user-mode malware. Using this technique we can apply advanced heuristic on which system calls is application using and in which order.

5. Hooking techniques (IAT, SSDT, SSDT Shadow, IDT)

In order to control the infected computer and steal users information malware tries to insert itself into normal execution path. One of the most common techniques for achieving this is hooking. Our classifier should monitor the system for presence of such hooks, if any of them is found this will be a strong indication of malware behavior.

Tests don't have equal significance, and therefore carry different weights that are used to compute the final score of for the malware being tested.

6. Classifier design

Our classifier should be created in phases, where we group similar activities to achieve better performance. The two most costly operations are disk reads and behavior testing. Considering we separated the tests in two distinct groups, we consider the following:

Phase 1: Single pass testing

The first phase tests are Entropy testing, packer detection, cryptographic signature detection and known code pattern scanning. The optimization is that all those measures require one read/scan per sample. By implementing those measures in parallel, we can achieve better performance. The most significant weight would be if a sample has common malicious code elements found in other malware samples, followed by packing information if a malware centric packer was used.

Our suggestion is, if a malware specific packer is used, to automatically flag the sample as high risk or malicious. If a packer is used, the sample can be unpacked and rescanned for code samples or other data.

Other interesting information can be obtained if a packing attempt fails, or if we can see anti-debugging

methods in the file. Crypto signatures can add a low risk score.

All above mentioned operations can be multi-threaded, so a multiple threaded implementation would have a speed gain. Also, it is important to note that the searching algorithms are slower as we have more samples to search. Therefore, we can say it is better to search for a smaller concrete subset of patterns then a large malware pattern base

After all those methods, we can parallelize scanning for known malware patterns, which is costly by time. We suggest ordering the patterns where newer malware samples are at the beginning of the list, where they can be matched faster and other methods can use a quick hash function lookup, where we calculate a hash function from a sample and try to check if the sample is widely known as malicious.

Phase 2: Simple runtime tests

There are two simple runtime tests, Self removal and persistence. If a sample removes itself from its current runtime location, then this is a medium risk flag. Adding itself to a automatic run location is another medium risk flag. If both are present at a same time, then it is a very high risk flag as this behavior is not exhibited in installing applications, but only malware.

Phase 3: Advanced runtime tests

Advanced tests are runtime injection, syscall tracing and hooking detection. Those tests are highly reliable, but as phase 2 they require executing the sample. Therefore, it would be highly recommended to run the samples in a bare-bone virtual machine that can be scrubbed after testing. Unfortunately, the most reliable tests are the slowest and should be used as a last resort measure.

7. Future research

Our future research will be concentrated on benchmarking the classifier speeds with various methods and parameters. Also, we have collected a wide malware sample base, and we want to tune and benchmark the model against other commercial and open source providers to obtain the optimum speed and classification precision considering each classifiers importance and reliability. One of our research areas will cover other methods that are even faster than the ones covered here or the importance of some methods that we have shown here.

8. Conclusion

In this work, we have shown a concept for malware detection that does not rely solely on signature matching or behavior detection. We have presented a concept that is both faster and We believe while an increasing number of malware samples is presented each day the methods of malware detection at the gateways and routers to our networks are not improving as fast as they should. Complex malwares like Conficker or custom malware like the ones used in GhostNet and Aurora attacks are gaining popularity. We can only conclude that additional research needs to be done in this area.

7. References

[1] http://abysssec.com/AbyssDB/Database.TXT

[2] BAYOGLU B., SOGUKPINAR I., *"Polymorphic worm detection using strong token-pair signatures"*, Gebze, Koaceli-Turkey

[3] Chess D., White S., *"An Undetectable Computer Virus"*, Hawthorne, New York, USA

[4] http://mtc.sri.com/Conficker/, Acc. 29.3.2010

[5] Eliam E., *"Reversing: Secrets of Reverse Engineering"*, Wiley Publishing Inc., 2005

[6] http://code.google.com/p/pefile , Acc. 29.3.2010

[7] Guo F., Ferrie P. , Tzu-cker C., *„A study of the Packer Problem and Its Solutions"*, Symantec Research Labaratories, 2008.

[8] Guofei G., Porras P., Yegneswaran V., Fong M., Lee W., *"BotHunter: Detecting Malware Infection Through IDS-Driver Dialog Correlation"*, 16th USENIX Security Symposium, p. 167-182

[9] Jacob G., Filiol E., Debar H., *"Formalization of malware through process calculi"*, eprint arXiv:0902.0469, 2009

[10] Leder F., Steinbock B., Martini P., "*Classification and Detection of Metamorphic Malware using Value Set Analysis"*, Institute of Computer Science, Bonn, Germany

[11] http://www.peid.info/, Acc. 29.3.2010

[12] http://www.vmprotect.ru/, Acc. 29.3.2010

[13] www.virustotal.com, Acc. 29.3.2010

[14] Yin H., Song D., Egele M., Kruegel C., Kirda E., *"Panorama: capturing system-wide information flow for malware detection and analysis"*, Conference on Computer and Communications Security