Spearphishing Malware: Do we really know the unknown?

Yanko Baychev[1] and Leyla Bilge[2]

[1] Airbus CyberSecurity, Taufkirchen, Germany, yanko.baychev@airbus.com
[2] Symantec Research Labs, Sophia Antipolis, France, leyla_bilge@symantec.com

Abstract. Targeted attacks pose a great threat to governments and commercial entities. Every year, an increasing number of targeted attacks are being discovered and exposed by various cyber security organizations. The key characteristics of these attacks are that they are conducted by well-funded and skilled actors who persistently target specific entities employing sophisticated tools and tactics to obtain a long-time presence in the breached environments. Malware plays a crucial role in a targeted attack for various tasks. Because of its stealthy nature, malware used in targeted attacks is expected to act differently compared to the traditional malware. However, to our knowledge, there is no previous study that performed an empirical validation to this assumption.
In this paper, we perform a study to understand whether malware used in targeted attacks is any different than traditional malware. To this end, we dynamically analysed a set of targeted and traditional malware to extract more than 700 features to be able to measure their discriminative power. These features are calculated from the network, host and memory behavior of malware. The rigorous experimentation we performed with several machine learning algorithms suggest that targeted malware indeed behaves differently and even with raw features extracted from the dynamic analysis reports, fairly good classification accuracy could be achieved to distinguish them from traditional malware.

Keywords: targeted attacks, malware, dynamic analysis

1 Introduction

Since 2010, we witnessed a dramatic change in the cyber threat landscape. Before that, the main motivation of cyber attacks was financial gain and therefore, the goal was to infect as many computers as possible. While this kind of attacks still exist and constitute the majority of the cyber attacks seen in the wild [25], a new class of attacks has emerged and became the top priority risk for both governmental and commercial organizations: targeted attacks. In targeted attacks, well-organized operators target specific entities persistently with high motivation, evade security defenses in place, employ advanced tools and tactics, maintain long-time presence in target environment and operate slow and stealthy to avoid detection [26].

Malware plays a vital role in the success of a targeted attack and is employed almost in every phase of the attack lifecycle until the operator's goal is achieved. It is used to perform a wide range of tasks including compromising systems, escalating privileges, maintaining presence, exfiltrating data, communicating with the operators over command and control servers, carrying out commands, etc. Even though these tasks are not peculiar to targeted malware per se, targeted malware is expected to act differently compared to traditional malware and in this paper, we seek to understand whether this common belief is aligned with the reality or not.

As the sophistication level of targeted attacks rises gradually in time, increasing number of targeted attacks are being discovered using malware with advanced stealth techniques or even non-persistent malware that only resides in memory [27]. Recently cyber security professionals came up with a new term for targeted attacks employing non-persistent in-memory malware and named them as Advanced Volatile Threats [36], because there is no easy way to detect them other than analyzing the volatile memory. In order to detect fileless malware and also malware that hides its presence on the system by using advanced malware stealth techniques such as hooking, injection, hollowing, etc., memory analysis is a must and needed to be conducted along with dynamic analysis. In this paper, we used popular open-source memory forensics tool called The Volatility Framework [34], which can be integrated with the Cuckoo Sandbox [11] and offers wide range of plugins for various memory related analysis tasks to be able to detect fileless malware that remains solely in memory. Cuckoo Sandbox was configured to dump full memory of the guest analysis machine just before the execution finishes and run Volatility with selected plugins on the memory dump. We additionally utilized YARA tool [32] to supplement behavioral characterization of malware and included YARA rules that match certain behavioral patterns during the execution.

Past evidence suggests that targeted attacks are significantly more impactful compared to traditional malware attacks for the victims [33]. Therefore, they deserve more attention and there is an immediate need for methodologies that can distinguish them from traditional malware. However, before that we need to first find out whether they are different at all such that depending on the result more effort could be put to devise new detection methodologies. In this paper, we aim at identifying the differences between targeted and traditional malware. Unfortunately, obtaining a large-scale dataset that is representative for the real targeted malware that is used in advanced persistent threats is difficult. One one hand such attacks are more rare to see compared to the volume of other cyber attacks. One the other hand, as this kind of attacks might include actions done by a more skilled attacker manually, it might be even hard to identify them. For this reason, in this paper, we employ a set of malware that is used in spearphishing attacks. We believe that the general spearphishng malware will share some behavioral similarities to more advanced targeted malware as they are both carefully choosing their victims and also some of the advanced persistent threats use spearphishing as means of the infection vector. Therefore, it is possible that

in our data we actually include some examples for more sophisticated targeted malware. Our assumption here is that by analysing this dataset, it is possible to achieve an approximation to how general targeted attacks behave differently than the traditional malware.

We investigate the behavioral differences between targeted and traditional malware by running them in a controlled environment. During the analysis period, we extract a wide range of features from both the dynamic analysis and memory analysis traces. Afterwards, we experiment with a number of machine learning algorithms to find the most discriminative features and test the feasibility of distinguishing targeted malware. The result of the extensive evaluation made on spearphishingmalware used in the wild indicates that targeted malware behaves differently and can be distinguished from the traditional malware. Note that our goal with this paper is not to propose a detection method for targeted malware but to obtain insights about their differences. However, the classification accuracy achieved using the raw features extracted from the dynamic analysis reports is good enough for many real-world threat intelligence applications. For example, the methodology explained in this paper could be used as a filtering step during targeted attack detection that involves in more sophisticated analysis.

In summary, this paper makes the following contributions:

- We present the first study on understanding the differences of targeted malware from traditional malware.
- We extract an extensive list of dynamic analysis features from a broad perspective including behavioral patterns when fed to a machine learning classifier are capable of distinguishing targeted malware.
- We present a novel set of memory features extracted from memory analysis that could be incorporated to targeted malware detection systems.
- We provide a comprehensive evaluation of the features using different machine learning algorithms and provide a detailed discussion about the most discriminative features and their reasoning behind them.

2 Analyzing the Malware

Figure 1 illustrates the architecture of the system we devised to obtain insights about malware that is used in spearphishing attacks. Before digging into the details about which features we focus on and why, we will briefly summarize each step in the process of analyzing the malware samples. These malware samples were provided to us by Symantec and each of them were categorized by human analysts whose main goal is to mine large amounts of e-mail data to identify spearphishing attacks. This process does not only consist of data mining but also reverse-engineering the binary that is attached to the mail. While we do not know the details about the methodology adopted to identify the samples provided to us, we are confident that the false positives in the data is fairly low as each of them were carefully analyzed by these human analysts. The traditional

malware data, on the other hand, was build by randomly choosing malware that is observed by Symantec during the targeted malware collection period. Unfortunately, we could not acquire the binaries for these samples and therefore, we needed to search for them ourselves.

To obtain the binaries of the malware samples, we download them from Virus Total [31] and store them in our malware repository. Easy but also very efficient way to reveal malware's behavior once it infects a system is to run it in a controlled environment and capture all the changes on the host and the network during the analysis process [10]. Despite its obvious limitations, dynamic analysis of malware in a sandbox environment is extensively used in malware analysis field due to its convenience on analyzing malware in fast and automated manner. For our work, we utilize a modified version of Cuckoo Sandbox [24]. Cuckoo Sandbox is a very powerful and highly customizable open-source dynamic analysis sandbox backed by a large community of researchers and developers. It leverages memory analysis framework for conducting memory analysis. Each malware is analyzed in the cuckoo sandbox and their analysis reports that contain both information about their dynamic analysis behavior and the memory analysis are stored in our database. Then, the reports are processed to extract a large number of features that can capture the behavioral characteristics of malware. Finally, as a last step, we employ a number of machine learning classifiers to test whether targeted malware could be distinguished using the features we will explain in the following section.

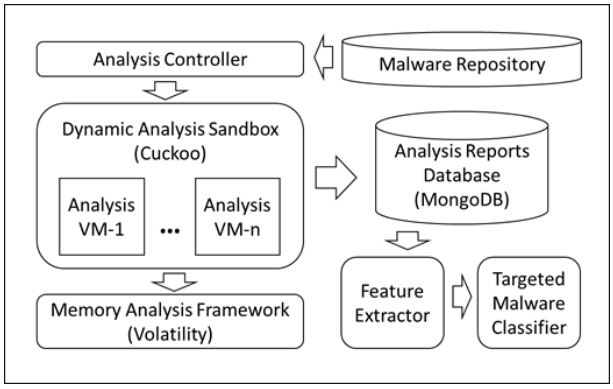

Fig. 1: System Architecture.

2.1 Features

A wide range of system-level artifacts are captured during the whole analysis process of each malware sample. We examined the analysis reports in detail to identify useful features for our purpose. Instead of utilizing high granularity

Table 1: Network features.

#	**Feature Name**
1	Number of UDP connections
2	Number of distinct IP addresses
3	Number of distinct UDP destination ports
4	Number of IRC connections
5	Number of HTTP connections
6	Number of SMTP connections
7	Number of TCP connections
8	Number of ICMP connections
9	Number of hosts
10	Number of hosts with reverse DNS
11	Number of DNS requests
12 - 17	DNS request type frequency
18	Number of domains
19 - 22	Domain level frequency
23 - 27	Domain name length frequency
28	Is top websites visited

features that represent low level atomic operations [5, 22], we decided to utilize low granularity and mostly quantitative features that could represent malware behavior from a wider aspect [18]. We identified a total of 754 features either by defining new dynamic and memory features or adopting the existing ones from state-of-the-art in malware classification [2,13,18,29]. These features are divided into six categories: network features (28), file system features (174), registry features (4), system call features(289), memory features (71) and miscellaneous features (208). At total we have extracted 754 features.

Network Features During the execution of malware, we captured all incoming and outgoing network traffic. From the network traffic we extracted a total of 28 network features such as number of DNS requests, TCP/IRC/SMPT/HTTP connections, IPs contacted, etc. Most of these features are single features and represent counts, therefore the name of each feature in Table 1 provides sufficient information about the feature itself.

In the course of the analysis, we encountered only the following DNS requests types; A, AAAA, MX, SRV, TXT, PTR. For each of these DNS record types, we counted the number of DNS requests and kept them as features. For each second, third, forth level domain name and the fully qualified domain names we observed in the network, we counted number of queries as well. Finally, we extracted some features to express the length of the domain names used by malware. We calculated the number of domain names whose lengths fall into the following ranges: 0-10. 11-16, 17-20, 20-32 and 32-∞.

File System Features Cuckoo Sandbox employs several injection and hooking techniques while executing the malware in order to detect and capture the file system operations including, access, read, create, modify and delete operations. From these file system artifacts we extracted a total of 174 file system features (see Table 2). We came across a wide range of file extensions of the created, modified or deleted files throughout the analysis process. In order to characterize this behavior, we defined a feature vector, which we call the top extension frequency, composed of top 34 extensions varying from "dll", "exe", "jpg" to "tmp", "txt" each corresponding to a specific extension and representing its count. During the analysis of malware, it creates files under specific paths in the file systems and typically these are good features to identify malware. These paths are usually associated with environment variables and named as common folder or known folder in Windows operating system [16]. We counted number of times file operations were done on specific file paths and used them as features in our analysis. For example, the following paths are some of the well-known paths where malware often copies itself;

- C:\Windows\System32\
- C:\DOCUME~1\<user name>\LOCALS~1\Temp\
- C:\Documents and Settings\All Users\

Another observation we had during the analysis phase is that files are created in varying sizes from bytes to megabytes. After examining the file sizes in detail, we decided not to utilize the size as a whole but in ranges and we picked the following four size ranges in bytes; 0-64, 65-4096, 4097-262144 and 262144-∞. We have also identified five types of files that malware often interacts with. These file types are the files, the directories, the drivers, the pipes and the alternate stream types. For each of these file types, we counted number of reads, accesses and modify operations.

Malware samples in general read and access an extensive number of files which are mostly duplicates or a noise because of the nested path traversal. To obtain the unique list of such operations, we pre-processed and filtered out these duplicates from the data. While five types of file artifacts are captured during the analysis, in terms of access, read, modify, create and delete operations, we did not build the frequency vectors for all of the file operations, but only for the ones that are more characterizing for malware. For instance, the known path frequency vector is constructed only for the file create, modify or delete operations but not for the read or access operations.

Registry Features We extracted 4 features from the registry operations conducted by malware. Similar to the file operations, malware samples read and access an extensive number of registry keys. After we pre-processed them to exclude the duplicates and noise, we counted the number of registry keys deleted, modified, read and accessed.

System Call Features We have also included some features that we extracted from the system call traces malware produces during its execution. To decrease

Table 2: File system features.

#	**Feature Name**
1	Number of files deleted
2	Number of files modified
3	Number of files deleted in distinct paths
4	Number of files deleted with distinct extensions
5-38	Top extension frequency of deleted files
39-52	Known path frequency of deleted files
53	Number of files modified in distinct paths
54	Number of files modified with distinct extensions
55-88	Top extension frequency of modified files
89-102	Known path frequency of modified files
103-107	File type frequency of modified files
108	Number of files created
109	Number of files created with distinct extension
110	Number of files created in distinct path
111-114	File size frequency
115-148	Top extension frequency of created files
149-162	Known path frequency of created files
163	Number of files read
164-168	File type frequency of read files
169	Number of files accessed
170-174	File type frequency of accessed files

the number of features, we performed an aggregation over the system call types (network, threading, hooking etc.) and counted number of system calls that fall into one of the 16 categories we have identified.

Although tracking counts of system calls in individual categories can provide good insights about the malware, some categories have disproportionally high counts of system calls. To capture this behavior at a higher level, we calculated proportion of number of system calls in each of the 16 categories over the total number of system calls as well.

As system calls are generally not directly accessed by programs but via APIs, we have also extracted the API calls made by the binaries. In our data, we have observed 256 different API functions called by malware samples and for each of these API calls we included a feature which represents the number of times the particular API call was made by malware.

Miscellaneous Features In addition to the features explained above, Cuckoo sandbox generates analysis reports enriched with several other malware related static or dynamic artifacts including API functions resolved, mutexes created, services started, commands executed, etc. In order to incorporate these rich artifacts into our analysis, we extracted a total of 208 miscellaneous features that include the total number of resolved API functions and API files, the number of times each of these API functions are resolved, the total number of commands

executed, number of services/mutexes created and started, number of YARA signatures matched and for each signature how many times the matches were identified. Although majority of the features in this category are based on the dynamic analysis reports, we were also able to include few simple static analysis features such as resolved API files and functions, strings included in the binaries.

We also leveraged the YARA tool [32] that comes with the Cuckoo Sandbox. YARA is capable of identifying particular malicious behavior or generic techniques adopted by malware. These signatures can provide valuable insights into malware behavior when utilized as a feature. Note that we only include the yara signatures that are related to behavioral characteristics. The malware specific yara signatures are excluded from our analysis so that they are not used by the classifier to distinguish the malware family from the targeted malware rather than the general traditional malware. For the sake of brevity, we do not list the whole list of YARA signatures that are matched, however, in the evaluation section we demonstrate the ones that are the most useful during the classification process.

Memory Features We leveraged the infamous Volatility Framework [34] to extract a large number of features from the memory introspection we performed during our analysis. The targeted attacks typically perform obfuscation on the malware they use and for that reason, without memory analysis it would be very hard to fully observe the malicious behavior. Getting motivated from this insight, we investigated memory indicators that could identify a malicious activity and selected 13 Volatility plugins. The reason for this is that some plugins can take very long time to complete and are not feasible to perform on thousands of samples. From outputs of these plugins, we identified and extracted a total of 51 features. In Table 3, all extracted memory features are listed along with the name of the Volatility plugin used for the feature. These features are a novel set of features that have not been used in any other works before to our knowledge.

3 Experiments and Results

3.1 Experimental Setup

Dynamic analysis of the malware samples and experiments were performed on a server with two quad-core Intel Xeon E5440 2.83 GHz processors, 16 GB DDR2 RAM, three 146 GB 10,000 RPM SAS hard drives and two Gigabit network adapters. The server was running Ubuntu 12.04 LTS as host operating system along with headless-mode VirtualBox (version 4.2.22) as the hypervisor.

We utilized a heavily modified and improved [20] version of Cuckoo Sandbox (Cuckoo 1.3-dev by Brad Spengler) for dynamically analyzing the malware samples and Volatility Framework (version 2.4) for analyzing the memory dumps acquired at the end of execution. Malware was allowed to access the Internet during the analysis. Note that we did not allow the malware to do large scale attacks such as denial of service attacks. With the help of Pafish tool [21], we assessed

Table 3: Memory features.

#	**Feature Name**	**Vol. Plugin**
1	Number of mutexes	Mutantscan
2-5	Mutex length frequency	Mutantscan
6	Number of processes exited	Pslist
7	Number of processes running	Psxview
8-14	Process list frequency	Psxview
15-21	Module in common Windows process freq.	Dlllist
22-28	Avg. mod. in common Windows proc. freq.	Dlllist
29-31	Injection VAD tag frequency	Malfind
32	Number of injections	Malfind
33	Average number of injections per process	Malfind
34	Number of processes with privileges	Privs
35	Number of proc. with Administrator SID	Getsids
36-38	Hidden DLL type frequency	Ldrmodules
39	Number of services	Svcscan
40	Number of driver names	Devicetree
41	Found duplicate driver name (0 or 1)	Devicetree
42	Number of device offsets	Devicetree
43	Number of device names	Devicetree
44	Number of kernel drivers	Modscan
45	Number of timers	Timers
46	Number of distinct timer periods	Timers
47	Number of distinct timer modules	Timers
48	Number of callbacks	Callbacks
49	Number of distinct callback types	Callbacks
50	Number of distinct callback modules	Callbacks
51	Number of distinct callback details	Callbacks

the analysis environment against several anti-vm and anti-sandbox techniques and hardened the environment to avoid detection by fixing the identified issues. After the Cuckoo Sandbox was configured properly, all malware samples were executed in parallel, each for a period of five minutes, on two virtual analysis machines.

3.2 Data Set

Obtaining malware samples that are attached to targeted attacks is a challenging task. While there are several malware repositories such as VirusShare [30] that share a big volume of malware samples, information regarding their category is not provided for most cases. Some cyber security firms dealing with targeted attacks publish reports on attack campaigns and disclose hashes of files involved. Although this is useful, the number of malware hashes in those reports is too small for generalization.

To increase our knowledge about malware samples that are used in targeted attacks, we obtained a list of manually chosen malware that are used in so-

phisticated spearphishing attacks. As we mentioned before, by analyzing these spearphising malware we hope to achieve an approximation to how targeted attacks behave as a spearphishing attack is a form of targeted attack itself. The analyst in Symantec also provided an additional list of traditional malware samples for comparison. It is important to note that all of the targeted malware samples were vetted and labeled manually by an analyst, not by an automated process. We obtained 2032 targeted and 10K traditional malware hashes. Afterwards, we searched Virustotal [31] for hashes and managed to find 709 targeted malware samples in Virustotal. We believe not finding the majority of the targeted samples but the traditional samples in Virustotal is a good indicator for the targetedness property of these malware samples.

After submitting them to our deployed Cuckoo Sandbox for analysis, only 471 were executed properly on our guest virtual machines by producing reports containing system-wide artifacts. Dynamic analysis of the other 238 samples were failed because either the analysis task was terminated shortly after the start or no single process was created throughout the analysis period. It is possible that these malware samples employ techniques for detecting the sandbox environment or require newer operating system to execute. In order to have balanced sets of targeted and traditional malware samples, we searched for only randomly picked small fraction of the traditional malware hashes and downloaded 618 malware samples from Virustotal. 550 out of 618 traditional malware samples were executed in the sandbox without any problems. Compared to the scale of malware seen in the wild this is definitely a very small number, however, unfortunately the number of spearphishing malware we were able to obtain was very small and that was the reason for limiting ourselves to these numbers.

To be sure that our traditional malware set is not composed of a big cluster of samples from the same category, therefore, we are not distinguishing the behavior of a particular malware family from the targeted malware, we investigated the AV labels of those 550 malware samples. From the selected top keywords from all traditional malware samples, we constructed a global list of keywords and number of their occurrences. Our traditional malware set includes a balanced distribution over various malware types including some downloaders, backdoors, zeus, password-stealers, autorun, autoit, etc. We also investigated keywords of targeted malware samples to make sure that our targeted malware set is not comprised of malware mostly with same type or functionality, e.g. "dropper", "downloader", etc. The top 20 keywords we identified have at least 10 and average of 40 matches each.

3.3 Leveraging Machine Learning

To evaluate the proposed method and the discriminative power of the features extracted in terms of targeted malware classification, we conducted experiments using the following supervised learning algorithms: Support Vector Machine (SVM), Logistic Regression, k-Nearest-Neighbor (kNN), Decision Tree and Random Forest. In all of our experiments, we used 5-fold cross validation where the dataset is divided into five equally sized partitions with four partitions used to

train the classifier and the remaining partition used for validation. This process was repeated five times and resulting scores were averaged. Because we have extracted a wide-variety of features each lying in differing ranges of value, we employed standard feature scaling and centered the feature values around 0 with zero mean and unit variance to avoid biasing toward any feature resulting misclassification.

Table 4: Targeted malware classification results.

Algorithm	**Accuracy**	**Precision**	**Recall**	**F1**	**False-Pos.**	**AUC**
SVM RBF	89.27 %	93.51 %	80.90 %	86.75 %	4.31 %	92.50 %
SVM Linear	89.27 %	91.36 %	83.15 %	87.06 %	6.03 %	94.20 %
K-NN	88.29 %	84.95 %	88.76 %	86.81 %	12.07 %	88.35 %
Log. Regression	88.78 %	90.24 %	83.15 %	86.55 %	6.90 %	91.85 %
Decision Tree	87.80 %	84.78 %	87.64 %	86.18 %	9.73 %	87.41 %
Random Forest	89.17 %	91.33 %	82.92 %	86.92 %	6.03 %	93.88 %

The obtained experiment results in Table 4 show that supervised learning algorithms exhibited similar performance on accuracy measure and they all achieved above 87% accuracy rate. There is also not a single algorithm that outperformed others in most of the measures. Performance not being dependent on a single algorithm suggests that the identified set of features are comprehensive enough to characterize malware from different aspects and therefore ensure high level of accuracy independent of the algorithm used for classification.

While both SVM algorithms achieved best result on accuracy with 89.27%, SVM RBF yielded highest precision and lowest false-positive rate with 93.51% and 4.31%, respectively. SVM Linear yielded best f-1 measure with 87.06% and k-Nearest Neighbor algorithm yielded best recall rate. As a measure representing overall performance, computed AUC for SVM Linear algorithm was the highest followed by Random Forest algorithm with a small margin.

3.4 Why is spearphishing malware different?

The results of the machine learning experiments suggest that the targeted malware can be distinguished from the traditional malware using the set of features utilized in this study. In the following section, we will look into the list of most discriminative features to understand why targeted malware is different than the traditional malware and how this difference is captured by these top features.

In order to determine contribution of each feature to the prediction of targeted malware, we performed feature selection using the Recursive Feature Elimination (RFE) technique [1] which is commonly used for problems with small sample size and high dimensionality along with Support Vector Machines [8]. As its name suggests, RFE recursively eliminates features with smaller weights and constructs the model repeatedly to compute the model accuracy. We employed

Table 5: Top 20 Features.

Ind.	Cat.	Description
672	Misc	Matched signature frequency: infostealer_mail
553	Misc	Resolved top API function frequency: oleaut32.dll
680	Misc	Matched signature frequency: injection_runpe
666	Misc	Matched signature frequency: antidbg_windows
421	Call	API function frequency: SHGetFolderPathW
395	Call	API function frequency: FindWindowW
547	Misc	Resolved top API function frequency: urlmon.dll
267	Call	API function frequency: NtSetContextThread
468	Call	API function frequency: InternetConnectW
459	Call	API function frequency: HttpSendRequestW
411	Call	API function frequency: RtlDecompressBuffer
360	Call	API function frequency: HttpOpenRequestW
80	File	Number of files modified in distinct paths
52	File	Top extension frequency of deleted files: tmp
434	Call	API function frequency: CryptCreateHash
237	Call	System call category percentage frequency: com
586	Misc	Resolved top API function frequency: sxs.dll
723	Mem	Module in common Win. proc. freq: winlogon.exe
749	Mem	Number of distinct timer modules
668	Misc	Matched signature frequency: disables_uac

RFE with SVM Linear classifier which yielded best result in terms of area under ROC curve and identified most discriminative 20 features in classifying targeted malware. Top 20 features are presented in Table 5 in descending order starting with the most discriminative feature.

Most of the identified top features in Table 5 align well with the behavioural characteristics of modern malware [4]. In this section, we will provide a discussion about the most interesting discriminative features going over the plots of the empirical cumulative distribution functions (ECDF) for both targeted and traditional malware (in ascending order according to index of feature in feature vector).

Top extension frequency of deleted files - Tmp: This feature corresponds to the number of files deleted during the malware execution with extension "tmp". It is known that malware can download or drop temporary files which can be used as additional payload or configuration files. Because of this known characteristic, antivirus software can flag files having extension "tmp" as suspicious or malicious. Therefore, it is expected for a targeted malware to delete temporary files after they are consumed to avoid getting the attention of the antivirus software or the security analyst. This behavioral pattern is captured with this feature and it is revealed in Figure 2 that targeted malware tends to delete at least one temporary file almost twice as many times as the traditional malware. Because while only 20% of the traditional malware samples deleted at least one file with extension "tmp", it is 38% for the targeted malware samples.

System call category percentage frequency - Com: This feature corresponds to the proportion of number of system calls in communication category to the total number of system calls triggered by the malware. Difference between the targeted and traditional malware's empirical cumulative distribution functions suggests that percentage of system calls initiated by traditional malware related to the communication is almost 8 times more than the targeted malware. It can be deduced from this behavior that targeted malware avoids initiating too many communication attempts to avoid triggering any alarms which aligns well with its stealthy nature.

API function frequency - NtSetContextThread: This feature corresponds to the number of times NtSetContextThread function is called by the malware. As the "Nt" prefix implies, NtSetContextThread is an Windows Native API system call that is used for modifying an existing thread's context, e.g. CPU registers. Malware utilizes this function for performing various advanced actions including evading Microsoft's EMET to execute shellcode, injecting code into processes or anti-debugging. It is important to note that this particular function is also used in process hollowing technique [17] for changing a thread's EIP and EAX registers which is widely employed by the sophisticated and targeted malware, i.e. Duqu. Difference in the traditional and targeted malware behavior with regards to this feature is very clear in Figure 2 showing that while only 8% of the traditional malware samples called NtSetContextThread API function, 59% of the targeted malware samples called it at least once which reveals the sophisticated and state-of-the art nature of targeted malware.

API function frequency - HttpOpenRequestW: This feature corresponds to the number of times HttpOpenRequestW function called by the malware. HttpOpenRequestW is an Windows Internet (WinInet) API function that is used for creating an HTTP request. It is called by malware while accessing a web resource or downloading a file. It is also known that malware hooks this function to perform a wide variety of information theft attacks. Because WinInet is a higher level API and also used by browsers, it allows malware to hide itself in the regular network traffic.

API function frequency - FindWindowW: This feature corresponds to the number of times FindWindowW function is called by the malware. FindWindowW is an Windows API function that is used to get a handle to the window with the given name or class. It is usually called by malware to search for a window that belongs to a specific security tool used in malware analysis, mostly a debugger as an anti-debugging trick. It is also utilized in code injection in a very stealthy manner. Only 8% of the traditional malware samples called FindWindowW API function. On the other hand, almost half of the targeted malware samples called it at least once, mostly more than once. This difference in plots of ECDFs in Figure 2 indicates that targeted malware takes additional measures to avoid being detected and analyzed which matches well with the previously discussed characteristic of the targeted malware.

API function frequency - CryptCreateHash: CryptCreateHash is an Windows API function that is used to initiate hashing of a stream of data. It

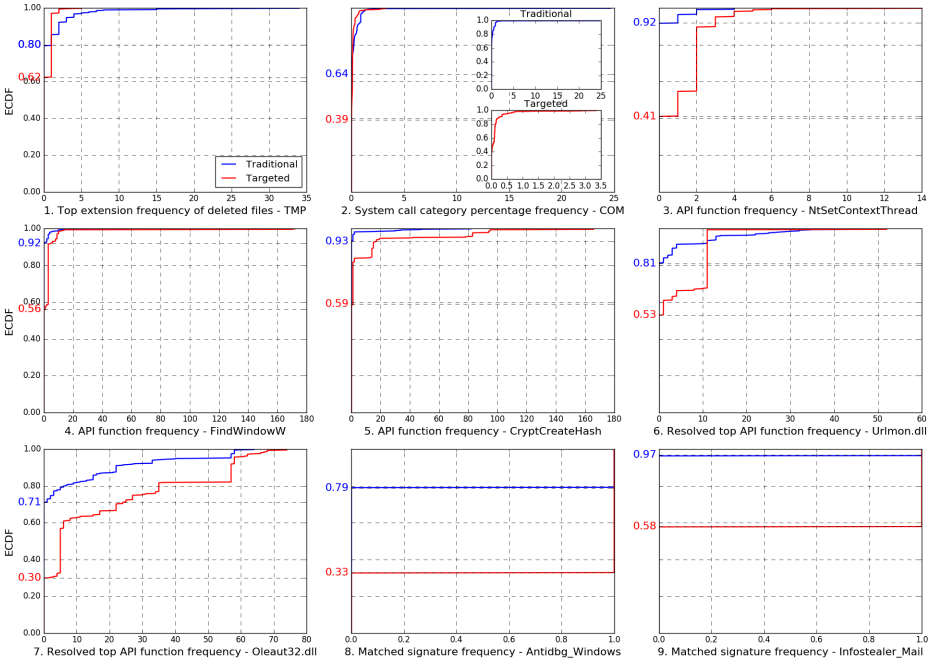

Fig. 2: ECDF of top features for targeted and traditional samples.

is called by malware to perform cryptographic hashing functions to generate encryption keys or obfuscating key internal data. In a report published about evasive malware [9], obfuscating internal data is found among the four most common evasive behaviours. Figure 2 shows that whereas only around 7% of the traditional malware samples called CryptCreateHash function, it was called by more than 40% of the targeted malware samples confirming the evasive nature of the targeted malware by making it difficult to identify its true behaviour.

Resolved top API function frequency - Urlmon.dll: URLmon.dll provides URL Monikers API to perform Internet communications and its URLDownloadToFile (or URLDownloadToCacheFile) function is very effective to download a resource to a specified filename via Internet Explorer with just one call [15]. Therefore, it is commonly used in shellcode exploiting a vulnerability and also in malicious scripts embedded in Office or PDF documents to drop malware. Spear phishing campaigns with malicious document attachments or zero-day exploits [25] are widely employed in targeted attacks and this characteristic is captured in Figure 2 showing that while only less than 20% of the traditional malware samples resolved the urlmon.dll file, it was resolved by almost half of the targeted malware samples.

Resolved top API function frequency - Oleaut32.dll: This feature corresponds to the number of functions resolved within oleaut32.dll file. Oleaut32.dll provides a mechanism, OLE Automation, to access and manipulate objects in

another application, e.g. ActiveX, Word macro, etc. It allows malware to execute VBScript macros in remote Word and Excel applications for malicious activities in a hidden way. In a report published in 2016 [25], it was found that Word and Excel documents containing malicious code accounted for almost half of the attachments in spear-phishing attacks used in targeted attacks. This difference is clearly revealed in plots of ECDFs in Figure 2 in respect to oleaut32.dll file where 70% of the targeted malware samples resolved this API file, while only less than 30% of the traditional malware samples used it. When this and the previous features are both taken into account, it gives a clear picture of the attackers techniques to target individuals or organisations via spear-phishing attacks.

Matched signature frequency - Antidbg_Windows: This feature corresponds to the occurrence of a match of antidbg_windows YARA signature. Antidbg_windows signature checks if the malware contains strings of windows names of popular debuggers and forensic tools which are indication of malware employing anti-debugging tricks. Figure 2 shows that whereas only around 20% of the traditional malware samples were discovered having anti-debugging tricks, almost 70% of the targeted malware samples contained strings that are used to determine the presence of debuggers or other security tools in the execution environment. This difference in plots of ECDFs, similar to FindWindowW feature, aligns well with the targeted malware characteristics of making it hard for the analysts or automated systems to detect and analyze it.

Matched signature frequency - Infostealer_Mail: This feature corresponds to the occurrence of a match of infostealer_mail YARA signature. Infostealer_mail signature checks if the malware attempts to collect credentials from various local email client programs. Whereas only 3% of the traditional malware samples were stealing information regarding the email, more than 40% of the targeted malware samples as shown in Figure 2 were found harvesting email credentials and addresses possibly to be used in spearphishing campaigns which are widely conducted in targeted attacks such as APTs. It was found out in a report published in 2016 that while the number of spearphishing campaings were increased by 55% in year 2015, they became more stealthier and number of recipients as well as the average duration of the campaigns fell by 39% and 33%, respectively [25].

Matched signature frequency - Injection_Runpe: This feature corresponds to the occurrence of a match of injection_runpe YARA signature. Injection_runpe signature checks if the malware launches a new process and injects code into it which can also include unpacking of code. The API functions filtered in this signature including NtUnmapViewOfSection, NtSetContextThread, NtResumeThread, etc. [12] are commonly used in process hollowing technique which allows stealthy code execution within another process' address space and is frequently used in targeted attacks. Only around 7% of the traditional malware samples were matched. On the other hand, more than 40% of the targeted malware samples were identified as employing such sophisticated injection techniques.

4 Related Work

Finding a suitable way to represent binary files lies at the heart of every method or system presented so far to detect or classify malware via machine learning. In this section a literature background about malware classification is provided with a focus on the use of static and dynamic features in malware classification.

4.1 Static Feature Extraction

Static analysis is the process of examining a binary file without actually executing it in order to determine if it is malicious or not. Extracting static features from a binary file to perform malware classification is carried out by static analysis tools or techniques.

[23] were among the first who introduced static features for malware detection by employing several different classifiers. They conducted experiments based on three different types of static features including Portable Executable (PE), strings and byte sequence n-grams. From PE header of a binary, they extracted dynamic link library (DLL) information and constructed three different feature vectors representing if a DLL was used, if a specific function inside a DLL was called and count of unique function calls inside each DLL. Encoded strings inside a binary was extracted and used as a feature. They also converted binary files into hexadecimal codes and used byte sequences of codes as n-gram features. Based on these three features, different classifiers are employed to classify new binaries as malicious or clean. While string features yielded the highest accuracy, the best detection rate was achieved by using byte sequence n-grams.

Their work inspired and encouraged others to try similar approaches for malware classification. [14] adopted and enhanced the byte sequence n-grams technique. They achieved better results in detecting malicious binaries via classifiers including Support Vector Machine, Decision Tree and boosted versions of them. [37] extracted API calls from PE header of a binary file similar to what [23] did in their work and used them as features for classifying a binary as malicious or not.

[28] disassembled the binary and then extracted the length and frequency of function names. Based on the function name length features, they perform malware classification between different malware families and their results suggested that function name length is significant as a feature in distinguishing malware families.

While static features have been widely used to detect or classify malware via machine learning, there exists some limitations. Authors assume that the malware is unpacked or not encrypted and static features can be extracted right away from the binary. However, it is very common for a malware to be packed or encrypted and in some cases it is not possible to fully unpack or decrypt malware. There also exists a wide variety of obfuscation techniques that can thwart the whole process [19].

4.2 Dynamic Feature Extraction

Dynamic analysis is the process of examining a binary file by executing it in a controlled environment and capturing its behavior in order to determine if it is malicious or not. Controlled environment where the analysis is conducted could be a specially designed sandbox [6,11,35] offering virtual, emulated or even bare-metal environment, but also a single computer equipped with dynamic analysis tools.

[3] constructed a high level behavior profile consisting of system change counts for process, file, registry and network categories for each binary sample after running it in a virtual environment and collecting system events. Using the behavioral profiles representing malware behavior as features and Normalized Compression Distance (NCD) as distance metric, they conducted hierarchical clustering on malware samples to cluster them into families. [5] extended the previous work [3] by constructing a low level behavior profile using a generalized form of system resources and system calls after running the sample in a dynamic analysis sandbox [6]. They achieved high run-time performance by performing hierarchical clustering using Locality Sensitive Hashing (LSH) and Jaccard index as distance metric. While both of the previous works used an unsupervised algorithm for the malware classification, [22] performed malware classification using Support Vector Machines (SVM) by extracting features from analysis report generated after running a malware sample in a dynamic analysis system [35]. Feature vectors in the works [5, 22] were representing malware behavior using a generalized form of the system calls and their parameters captured during the analysis.

[29] executed binaries in a virtual environment and captured system calls and their parameters with the help of an automated tool. A global list with strings representing captured system calls and parameters is compiled. Feature vector for each binary was constructed using this list and consisted of boolean values for each string in the global list specifying whether it was encountered during the analysis of the sample or not. They used supervised learning classifiers both for classifying a binary as malicious or not and for classifying malware into malware families. [13] extended the previous work by incorporating static features including printable strings and function length frequency into the dynamic system call features. Function length frequency feature vector consists of counts of functions for fixed length ranges. They conducted experiments for each feature separately and also for the integrated features using different supervised learning classifiers. All experimented classifiers achieved best results using the integrated features. [2] presented a method for combining features from six different sources including three static sources, two dynamic sources and one source containing statistics about the other sources. They achieved high accuracy classifying binaries as malicious or not using SVM classifier.

[18] proposed a system consisting of two components, one for analyzing binaries dynamically and one for classification and clustering. 65 features in three categories including file, registry and network were extracted from the artifacts captured during the dynamic analysis. Extracted features were in low granular-

ity, mostly counts, e.g. number of files created, unique number of extensions of the created files, file size, etc. Memory artifacts were also collected during the analysis phase but used only for signature matching to enhanced labeling, but not included in the feature vector. Experiments conducted using wide range of supervised learning classifiers in order to classify binaries as malicious or not and achieved high accuracy. They also performed clustering using several different distance metrics and parameters and presented results in terms of accuracy and performance.

There is also considerable amount of work that utilizes DNS features for classification. [7] identified malicious domains by monitoring the DNS traffic passively, extracted 15 features from the monitored traffic and achieved high detection rate. Types of features defined in terms of granularity are similar to the features presented in this paper.

5 Conclusion

Targeted attacks constitute one of the greatest risks for many organizations. As they are typically carefully prepared and designed particularly for the victims, they are harder to detect both during the infection and the further phases of the attacks. In this paper, we put an effort in understanding whether the malware that is used in targeted attacks could be distinguished from traditional malware. In big organizations, the security products in place produce many malware detection alerts. Unfortunately, the number of alerts in general is too high and therefore too overwhelming for the security analysts who are responsible for prioritizing the most risky attacks. Hence, a methodology to identify correctly targeted attacks from the haystack of attacks organizations everyday face with would be very beneficial for the organizations.

In this paper, we first aimed at understanding whether malware that is used in targeted attacks is different from traditional malware. To achieve this, we dynamically analyzed a substantial amount of targeted and traditional malware and compared their behavior. We profiled the behavior of malware and their execution characteristics by extracting over 700 features that could represent the network, host and memory behavior of the malware analyzed. We then, experimented with a number of machine learning algorithms and found out that while not perfectly, targeted malware could be distinguished from traditional malware. Furthermore, we have shown that using raw simplistic features that could be easily calculated from the dynamic analysis traces of executables, it is possible to achieve good classification accuracies. This is a very important finding as it opens the door for future works that could focus on improving the methodology we proposed in this paper.

References

1. Ambroise, C., McLachlan, G.J.: Selection bias in gene extraction on the basis of microarray gene-expression data. Proceedings of the national academy of sciences 99(10), 6562–6566 (2002)

2. Anderson, B., Storlie, C., Lane, T.: Improving malware classification: bridging the static/dynamic gap. In: Proceedings of the 5th ACM workshop on Security and artificial intelligence. pp. 3–14. ACM (2012)
3. Bailey, M., Oberheide, J., Andersen, J., Mao, Z.M., Jahanian, F., Nazario, J.: Automated classification and analysis of internet malware. In: International Workshop on Recent Advances in Intrusion Detection. pp. 178–197. Springer (2007)
4. Barbosa, G.N., Branco, R.R.: Prevalent characteristics in modern malware. Black Hat USA (2014)
5. Bayer, U., Comparetti, P.M., Hlauschek, C., Kruegel, C., Kirda, E.: Scalable, behavior-based malware clustering. In: NDSS. vol. 9, pp. 8–11 (2009)
6. Bayer, U., Kruegel, C., Kirda, E.: TTAnalyze: A tool for analyzing malware. na (2006)
7. Bilge, L., Kirda, E., Kruegel, C., Balduzzi, M.: Exposure: Finding malicious domains using passive dns analysis. In: Ndss (2011)
8. Chen, X.w., Jeong, J.C.: Enhanced recursive feature elimination. In: Machine Learning and Applications, 2007. ICMLA 2007. Sixth International Conference on. pp. 429–435. IEEE (2007)
9. Christopher, K.: Evasive malware exposed and deconstructed. RSA Conference (2015)
10. Egele, M., Scholte, T., Kirda, E., Kruegel, C.: A survey on automated dynamic malware-analysis techniques and tools. ACM Computing Surveys (CSUR) 44(2), 6 (2012)
11. Guarnieri, C., Tanasi, A., Bremer, J., Schloesser, M.: The cuckoo sandbox (2012)
12. Harrell, C.: Prefetch file meet process hollowing. https://journeyintoir.blogspot.be/2014/12/prefetch-file-meet-process-hollowing_17.html (2014)
13. Islam, R., Tian, R., Batten, L.M., Versteeg, S.: Classification of malware based on integrated static and dynamic features. Journal of Network and Computer Applications 36(2), 646–656 (2013)
14. Kolter, J.Z., Maloof, M.A.: Learning to detect malicious executables in the wild. In: Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining. pp. 470–478. ACM (2004)
15. M-Labs: Reversing malware command and control: From sockets to com. FireEye, https://www.fireeye.com/blog/threat-research/2010/08/reversing-malware-command-control-sockets.html (2010)
16. Microsoft: Common folder variables. https://www.microsoft.com/security/portal/mmpc/shared/variables.aspx (2015)
17. MITRE: Process hollowing. https://attack.mitre.org/wiki/Technique/T1093 (2016)
18. Mohaisen, A., Alrawi, O., Mohaisen, M.: Amal: High-fidelity, behavior-based automated malware analysis and classification. Computers & Security 52, 251–266 (2015)
19. Moser, A., Kruegel, C., Kirda, E.: Limits of static analysis for malware detection. In: Computer security applications conference, 2007. ACSAC 2007. Twenty-third annual. pp. 421–430. IEEE (2007)
20. Optiv: Improving reliability of sandbox results. https://www.optiv.com/blog/improving-reliability-of-sandbox-results (2014)
21. Ortega, A.: Pafish (paranoid fish). https://github.com/a0rtega/pafish/ (2012)
22. Rieck, K., Holz, T., Willems, C., Düssel, P., Laskov, P.: Learning and classification of malware behavior. In: International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment. pp. 108–125. Springer (2008)

23. Schultz, M.G., Eskin, E., Zadok, F., Stolfo, S.J.: Data mining methods for detection of new malicious executables. In: Security and Privacy, 2001. S&P 2001. Proceedings. 2001 IEEE Symposium on. pp. 38–49. IEEE (2001)
24. Spengler, B.: Modified edition of cuckoo. Github, https://github.com/brad-accuvant/cuckoo-modified (2013)
25. Symantec: Internet Security Threat Report Vol. 21. https://www.symantec.com/security-center/threat-report (April 2016)
26. Tankard, C.: Advanced persistent threats and how to monitor and deter them. Network security 2011(8), 16–19 (2011)
27. Teller, T., Hayon, A.: Enhancing automated malware analysis machines with memory analysis. USA: Black Hat (2014)
28. Tian, R., Batten, L.M., Versteeg, S.: Function length as a tool for malware classification. In: Malicious and Unwanted Software, 2008. MALWARE 2008. 3rd International Conference on. pp. 69–76. IEEE (2008)
29. Tian, R., Islam, R., Batten, L., Versteeg, S.: Differentiating malware from cleanware using behavioural analysis. In: Malicious and Unwanted Software (MALWARE), 2010 5th International Conference on. pp. 23–30. IEEE (2010)
30. VirusShare: Virusshare.com - because sharing is caring. https://virusshare.com/ (2017)
31. Virustotal: Virustotal - free online virus, malware and url scanner. https://www.virustotal.com/ (2012)
32. Virustotal: Yara - the pattern matching swiss knife for malware researchers. https://virustotal.github.io/yara/ (2014)
33. Virvilis, N., Gritzalis, D., Apostolopoulos, T.: Trusted computing vs. advanced persistent threats: Can a defender win this game? In: Ubiquitous Intelligence and Computing, 2013 IEEE 10th International Conference on and 10th International Conference on Autonomic and Trusted Computing (UIC/ATC). pp. 396–403. IEEE (2013)
34. Walters, A.: The volatility framework: Volatile memory artifact extraction utility framework (2007)
35. Willems, C., Holz, T., Freiling, F.: Cwsandbox: Towards automated dynamic binary analysis. IEEE Security and Privacy 5(2), 32–39 (2007)
36. Wilson, T.: Move over, apts – the ram-based advanced volatile threat is spinning up fast. DarkReading, http://www.darkreading.com/vulnerabilities---threats/move-over-apts----the-ram-based-advanced-volatile-threat-is-spinning-up-fast/d/d-id/1139211 (2013)
37. Ye, Y., Wang, D., Li, T., Ye, D., Jiang, Q.: An intelligent pe-malware detection system based on association mining. Journal in computer virology 4(4), 323–334 (2008)