Exploiting diverse observation perspectives to get insights on the malware landscape.

Corrado Leita
Symantec Research Labs
Sophia Antipolis, France
corrado_leita@symantec.com

Ulrich Bayer
Technical University Vienna
Vienna, Austria
ulli@seclab.tuwien.ac.at

Engin Kirda
Institute Eurecom
Sophia Antipolis, France
kirda@eurecom.fr

Abstract

We are witnessing an increasing complexity in the malware analysis scenario. The usage of polymorphic techniques generates a new challenge: it is often difficult to discern the instance of a known polymorphic malware from that of a newly encountered malware family, and to evaluate the impact of patching and code sharing among malware writers in order to prioritize analysis efforts.

This paper offers an empirical study on the value of exploiting the complementarity of different information sources in studying malware relationships. By leveraging real-world data generated by a distributed honeypot deployment, we combine clustering techniques based on static and behavioral characteristics of the samples, and we show how this combination helps in detecting clustering anomalies. We also show how the different characteristics of the approaches can help, once combined, to underline relationships among different code variants. Finally, we highlight the importance of contextual information on malware propagation for getting a deeper understanding of the evolution and the "economy" of the different threats.

1. Introduction

Malware plays a major role in today's Internet threat scenarios. Breaking into vulnerable systems and installing malicious code to remotely control user PCs puts an attacker into the position to undertake a number of illicit activities ranging from data confidentiality breach (e.g., through keylogging activities), denial of service attacks, to the generation of unsolicited traffic.

Several indicators suggest an exponential explosion of the number of newly generated malicious samples per day. For instance, according to [7], the amount of samples submitted to VirusTotal [25], a popular virus scanning platform, is in the order of millions of samples per month. Such

numbers translate into a daily load of approximately 30,000 samples per day. This load can be partially explained by the easiness with which malware writers can generate new code by personalizing existing code bases, or by re-packing the binaries using code obfuscation tools [21,22,27]. In addition, malware sample counts are biased by the increasing usage of polymorphic techniques [5]. For instance, worms such as Allaple [10] take advantage of these techniques to mutate the binary content of the executable file at each propagation attempt.

Such an explosion in the number of samples increases the task of the security analyst. On the one hand, a complete picture on the complexity of the malware landscape is possible only by discerning polymorphic instances from new variants. On the other hand, to get a full understanding of the malware landscape we need to go further, and get quantitative insights on the interrelations among the different families, and on the extent to which malware writers share code and produce patches to known variants.

This paper presents an empirical study based on one year of data collected by a distributed honeypot deployment, SGNET [17]. By taking advantage of protocol learning techniques, SGNET is able to emulate code injection attacks in a protocol agnostic way, and without prior knowledge of the vulnerabilities being exploited. Because of its properties and its deployment in a large number of different network domains, the SGNET dataset is representative of the current malware landscape for a specific class of malware samples, namely samples that propagate by means of server-side code injection attacks.

Many different tools and techniques have been developed in order to study the malware landscape. Previous work proposed different approaches to *cluster* malware samples according to either their static structural characteristics (such as in [26]) or their dynamic behavior [4]. Despite the general consensus in the research community that static approaches are not suitable to cluster and classify malware, we show in this work that most polymorphic engines in the wild have a low level of sophistication and

that simple static techniques can still be useful for clustering malware. More specifically, we show the effectiveness of a very simple pattern generation technique in classifying the propagation strategy and the structural characteristics of the malware samples observed by the SGNET dataset.

We show how, by combining clustering techniques based on either static or behavioral characteristics of the malware samples, we are able to detect clustering anomalies generated by different environmental causes. Moreover, we show how the combination of the two feature sets offers insights on patching and code sharing practices in the observed malware samples that would be invisible to any clustering technique based on a single feature type.

Many malware collection techniques solely focus on the collection of malware binaries. We underline, through practical examples, the usefulness of combining the knowledge generated by malware classification approaches with contextual information on malware propagation generated by the SGNET deployment. We show the importance of such information to generate rich, structured knowledge that helps the security analyst to obtain a better understanding of the "economy" of the different threats and on the modus operandi of the individuals at their root cause.

2. Related work

Previous malware clustering approaches that statically analyze the samples can be grouped into systems that disassemble the binary and systems that do not. The clustering system presented by Ghorghescu in [11] requires a prior disassembling step and then performs a clustering of the binaries by comparing their basic blocks. In contrast, the authors of [14] determine the similarity of two binaries by comparing a hex dump of their code segments. Recently, Wicherski presented peHash, a system for hashing PE binaries in such a way that polymorphic binaries receive the same hash value [26]. peHash is able to statically classify a large number of samples by grouping them according to the portions of the PE header that are not mutated by polymorphic packers.

The first attempts to cluster malware according to its behavior (i.e., requiring a dynamic analysis of the binaries) were based on system call traces. For example, the system for classifying malicious binaries described in [15] determines the similarity of two binaries by comparing system call traces of their execution. However, the performance of the system is poor because system call traces are usually large in size and, thus, result in long comparison times. Bailey et al. were the first to build a clustering system [3] that described a sample's behavior in more abstract terms. The authors evaluated their prototype implementation with several hundreds of samples. Although the clustering results were correct, the system suffers from the simplicity of
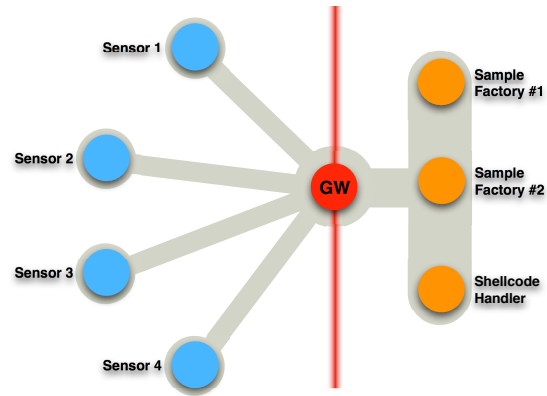

Figure 1. SGNET architecture

the behavioral descriptions that are not able to precisely express the behavior. Moreover, the presented system is not scalable because it requires the computation of $O(n^2)$ distances. Holz was proposing a classification system in [12] for malware samples that uses the analysis results of the CWSandbox sandbox system for computing the similarity of two samples. The approach, based on supervised machine learning techniques, relies on commercial antivirus labels to build the initial training set. Different works [3, 7] have pointed out in the past the limitations in the usage of such labels for malware classification purposes.

In this paper, we make use of the behavior-based clustering system provided publicly as part of the dynamic analysis platform called Anubis [1, 6] (i.e., see Section 3.3).

3. SGNET and EPM clustering

The analysis carried out in this paper is based on a freely accessible honeypot dataset, SGNET [17]. In contrast to other malware collections, SGNET focuses on the collection of detailed information on code injection attacks and on the sources responsible for these attacks. Each malware sample collected by the deployment is, therefore, enriched by contextual information on the attacks, the evolution of the attack in time, and on the structure of the code injection itself. The contextual information provided by the SGNET dataset is generated by an information enrichment approach [18] that aggregates data generated by different analysis tools such as VirusTotal [25] and Anubis [1, 6].

3.1. The SGNET dataset

SGNET is a distributed honeypot deployment focusing on the study of code injection attacks. SGNET was originally presented in [17], and then presented with further de-

tails in [16]. SGNET takes advantage of protocol learning techniques to address a trade-off between the need to retrieve rich information about the observed activities by prolonging as much as possible the conversation with clients and the need to reduce the resource and maintenance costs inherent in a distributed deployment. By using ScriptGen [19,20], SGNET honeypots are able to model protocol conversation through a Finite State Machine (FSM) model and use such models to respond to clients for well-known activities. Whenever a new/unknown activity is encountered, SGNET honeypots are able to dynamically proxy the conversations to a honeyfarm, and take advantage of the real service implementation to handle the interaction.

Figure 1 shows the main components of the SGNET deployment. SGNET is composed of multiple low-cost sensors whose FSM model is kept in sync by a central entity, the gateway. Whenever a new activity is encountered, SGNET honeypots require the instantiation of a new *sample factory* to the central gateway. The *sample factory*, based on Argos [23], acts as an oracle and provides to the sensors the required protocol interaction and, through memory tainting, detects and provides information on successful code injection attacks. Such information is used by the gateway to apply the ScriptGen algorithm and refine the FSM knowledge. After having seen a sufficient number of samples of the same type of interaction, SGNET sensors are, therefore, able to handle autonomously future instances of the same activity leveraging the newly built FSM refinement.

The memory tainting information generated by Argos, combined with simple heuristics, allows SGNET honeypots to identify injected shellcodes. SGNET takes advantage of part of the Nepenthes [2] modules to understand the intended behavior of the observed shellcodes and emulate the network actions associated to it.

All the information collected during the interaction of the different SGNET entities is stored in a database, and fed into an information enrichment component [18] that is in charge of adding additional metadata on the attacking sources, and on the collected malware. Among the different information sources, the most relevant to this work are the behavioral information generated by Anubis [6], and the AV detection statistics generated by VirusTotal [25]. Every malware sample collected by the SGNET infrastructure is, in fact, automatically analyzed by to these two services, and the resulting analysis reports are stored in the SGNET dataset to enrich the knowledge about the injection event.

3.2. EPM clustering

The SGNET dataset provides information on the different phases of each observed code injection attack. In [17], we show that SGNET is structured upon an epsilon-gamma-pi-mu (EGPM) model, an extension of the model initially proposed in [9]. The EGPM model structures each injection attack into four distinct phases:

- **Exploit** (ϵ). The set of network bytes being mapped onto data which is used for conditional control flow decisions. This consists in the set of client requests that the attacker needs to perform in order to lead the vulnerable service to the failure point.
- **Bogus control data** (γ). The set of network bytes being mapped into control data which hijacks the control flow trace and redirects it to somewhere else.
- **Payload** (π). The set of network bytes to which the attacker redirects the control flow through the usage of ϵ and γ.
- **Malware** (μ). The binary content uploaded to the victim through the execution of π, and that allows the attacker to run more sophisticated operations that would be impossible in the limited space normally available to the payload π.

In order to exploit the SGNET information, it is necessary to cope with the degree of variation introduced by the different propagation strategies. An example of this variation is the polymorphic techniques used to mutate the content of the payload and of the malware. However, we can have even simpler cases, such as the usage of a random filename in the FTP request used to download the malware sample to the victim.

Section 2 discussed several existing techniques for the quick classification of polymorphic samples starting from static features. Because of the unique characteristics of the dataset at our disposal, we have chosen not to reuse these techniques, and to introduce a new, simple pattern discovery technique, sufficiently generic to be applied independently to exploit, shellcode and malware features. We call this technique EPM clustering. The EPM clustering technique is a simplification of the multidimensional clustering technique described by Julisch in [13] and used in the context of IDS alerts. This technique is intentionally simple, and could be easily evaded in the future by more sophisticated polymorphic engines. Nevertheless, although it is simple, in practice, it proved to be sufficient to deal with the current sophistication level of the malware samples observed in the SGNET dataset. EPM clustering is based on the assumption that any randomization performed by the attacker has a limited *scope*. We assume that any polymorphic or randomization technique aims at introducing variability only in certain *features* of the code injection attacks. Since the randomization of each feature has a cost for the attacker, we can assume that it will always be possible to take into account a sufficiently large amount of features to detect a certain amount of *invariants* useful for recognizing a certain class of attacks.

Dim.	Feature	# invariants
Epsilon	FSM path identifier	50
	Destination port	3
Pi	Download protocol (FTP/HTTP/...)	6
	Filename in protocol interaction	22
	Port involved in protocol interaction	4
	Interaction type (PUSH/PULL/central)	5
Mu	File MD5	57
	File size in bytes	95
	File type according to libmagic signatures	7
	(PE) Machine type	1
	(PE) Number of sections	8
	(PE) Number of imported DLLs	7
	(PE) OS version	1
	(PE) Linker version	7
	(PE) Names of the sections	43
	(PE) Imported DLLs	11
	(PE) Referenced Kernel32.dll symbols	15

Table 1. Selected features

For instance, we observed that polymorphic techniques such as that of the Allaple worm [10] obfuscate and randomize the data and code sections of the executable files, but do not normally perform more expensive operations such as relinking, or leaving invariants when looking at the Portable Executable (PE) headers.

EPM clustering is composed by 4 different phases, namely feature definition, invariant discovery, pattern discovery and pattern-based classification. The technique is applied independently to the three distinct dimensions of the EGPM model: ϵ, π and μ.[1]

Phase 1: feature definition

The feature definition phase consists of defining a set of features that have proven to be useful in our experiments to characterize a certain activity class. Table 1 shows the list of features that we have taken into consideration to characterize the code injections in each dimension of the Epsilon-Pi-Mu space.

Most of the exploit classification is based on the information provided by the interaction of the attacker with the ScriptGen FSM models. Because of the way FSM models are built [20], a given FSM path incorporates together protocol features *and* specificities of a certain implementation. If all the network interactions take advantage of the same username, or the same NetBios connection identifiers, such parameters will be part of the generated model, and will be differentiated from other exploit implementations.

The Nepenthes shellcode analyzer provides information on the intended behavior of each shellcode collected by the SGNET deployment. Such information includes the type of protocol involved in the interaction (e.g., FTP, HTTP, as well as some Nepenthes-specific protocols), the filename

---

[1]We do not consider γ in the classification due to lack of host-based information in the SGNET dataset.

requested in the interaction (when available) and the involved "server" port. Depending on the involved protocol and on the entity interacting with the victim, we distinguish also between three types of interaction: 1) PUSH-based, in which the attacker actively connects to the victim and pushes the sample. 2) PULL-based (also known as phonehome), in which the victim is forced to connect back to the attacker and download the sample. 3) Based on a central repository, in the case in which the PULL-based download interacts with a third party different from the attacker itself.

The malware dimension, because of the extensive use of polymorphic techniques, requires a higher number of attributes in order to correctly classify samples. We have taken into consideration simple file properties, such as its size, as well as Portable Executable information extracted from the samples taking advantage of the PEfile [8] library. In order to identify non-polymorphic samples, the MD5 of the sample was also considered as a possible invariant. Looking at the level of sophistication of the SGNET malware collection, PE header characteristics seem to be more difficult to mutate over different polymorphic instances. These characterictics, thus, are good discriminants to distinguish different variants, since a change in their value is likely to be associated to a modification or recompilation of the existing codebase.

Clearly, all of the features taken into account for the classification could be easily randomized by the malware writer in order to evade our static clustering approach. More complex (and costly) polymorphic approaches might appear in the future leading to the need to generate more sophisticated techniques. This justifies the interest in continuously carrying on the collection of data on the threat landscape and on the study of its future evolution.

Phase 2: invariant discovery

For each of the features defined in the previous phase, the algorithm searches for all the *invariant* values. An invariant value is a value that is not specific to a certain attack instance (it is witnessed in multiple code injection attacks in the dataset), that is not specific to a certain attacker (the same value is used by multiple attackers in different attack instances) and is not specific to a certain destination (multiple honeypot IPs witness the same value in different instances). An invariant value is, therefore, a "good" value that can be used to characterize a certain event type.

The definition of invariant value is threshold-based: Throughout this work, we consider a value as invariant if it was seen in at least 10 different attack instances, and if it was used by at least three different attackers and witnessed on at least three honeypot IPs. Table 1 shows the amount of invariant values discovered for each dimension.

MD5	Size	NSect	DLLs
51ae2cf41b666137c0f3f63c08bd4f42	59904	3	kernel32.dll
09461eb9bb930abf66f9c3358aeabd6a	59904	3	kernel32.dll
3262d7ec241c75891471e7309436254f	59904	3	kernel32.dll
18777613abafd7d76ba8a7bfe7260b8a	**65402**	**4**	**kernel32.dll**
18777613abafd7d76ba8a7bfe7260b8a	**65402**	**4**	**kernel32.dll**

*	59904	3	kernel32.dll
18777613abafd7d76ba8a7bfe7260b8a	65402	4	kernel32.dll

Figure 2. Pattern discovery starting from a limited number of invariants (in bold)

Phase 3: pattern discovery

Once the invariant values are defined on each attack feature, we look at the way by which the different features have been composed together, generating patterns of features. As exemplified in Figure 2, we define as pattern a tuple $T = v_1, v_2, ..., v_n$ where n is the number of features for the specific EPM dimension and v_i is either an invariant value for that dimension or is a "do not care" value.

The pattern discovery phase looks at all the code injection attacks observed in the SGNET dataset for a certain period, and looks at all the possible combinations of discriminant values for the different features.

Phase 4: pattern-based classification

During this phase, the previously discovered patterns are used to classify all the attack instances present in the SGNET dataset and group them into *clusters*. Multiple patterns could match the same instance: For example, the instance $1, 2, 3$ would be matched by both the pattern $*, 2, 3$ and the pattern $*, *, 3$. Each instance is always associated with the most specific pattern matching its feature values. All the instances associated to the same pattern are said to belong to the same EPM cluster.

3.3. Clustering using static and behavioral features

The EPM classification technique described in Section 3.2 provides a fast and simple tool to explore the interrelationships between exploits, injected payloads and the resulting malware. By looking independently at exploit, shellcode and malware features, the EPM classification allows us to group attack events into clusters. For the sake of clarity, we will refer in the rest of the paper to *E-clusters*, *P-clusters* and *M-clusters* to identify all the groups sharing the same classification pattern over the exploit (E), payload (P) and malware (M) dimension respectively.

In parallel to the EPM classification of the malware and of its propagation strategies, we take into consideration the behavior-based malware clustering provided by Anubis [1, 6].

The clustering mechanism of Anubis has been described in detail in [4]. It is a behavior-based clustering system that makes use of the Anubis dynamic analysis system for capturing a sample's behavior. More concretely, the system works by comparing two samples based on their behavioral profile. The behavioral profile is an abstract representation of a program's behavior that is obtained with the help of state-of-the-art analysis techniques such as data tainting and the tracking of sensitive compare operations. The clustering system is scalable by avoiding the computation of all $O(n^2)$ distances.

Clusters of events associated to the same behavior according to Anubis behavior-based clustering will be referred to as *B-clusters* in the rest of this paper.

In the next session, we explore the information at our disposal in the SGNET dataset by means of EPM relations. We show how the combination of approaches based on static or behavioral characteristics of malware samples can be of great help in obtaining a better understanding of the different threats. Finally, we underline the value of the contextual information on malware propagation to add semantics to the different malware groups and acquire insights about the modus operandi of the malware writers.

4. Results

For this work, we analyzed all information that was collected by our SGNET deployment in the period from January 2008 to May 2009. During this period, the deployment collected a total of 6353 malware samples, 5165 of which could be correctly executed in the Anubis sandbox. This is consistent with the information reported in [7]. Note that due to failures in Nepenthes download modules, some of the collected samples, unfortunately, are truncated or corrupted, and, as a consequence, cannot be analyzed by a dynamic analysis system.

4.1. The big picture

By running the EPM classification technique described in Section 3.2, we have discovered a total of 39 E-clusters, 27 P-clusters and 260 M-clusters corresponding to the same number of groups. The analysis of the behavioral characteristics of the 5165 samples led to the generation of 972 B-clusters.

Figure 3 graphically represents the relationships between E-, P-, M- and B-clusters. Starting from top to bottom, the first layer of groups corresponds to the exploits, the second to the payloads, the third to the malware grouped according
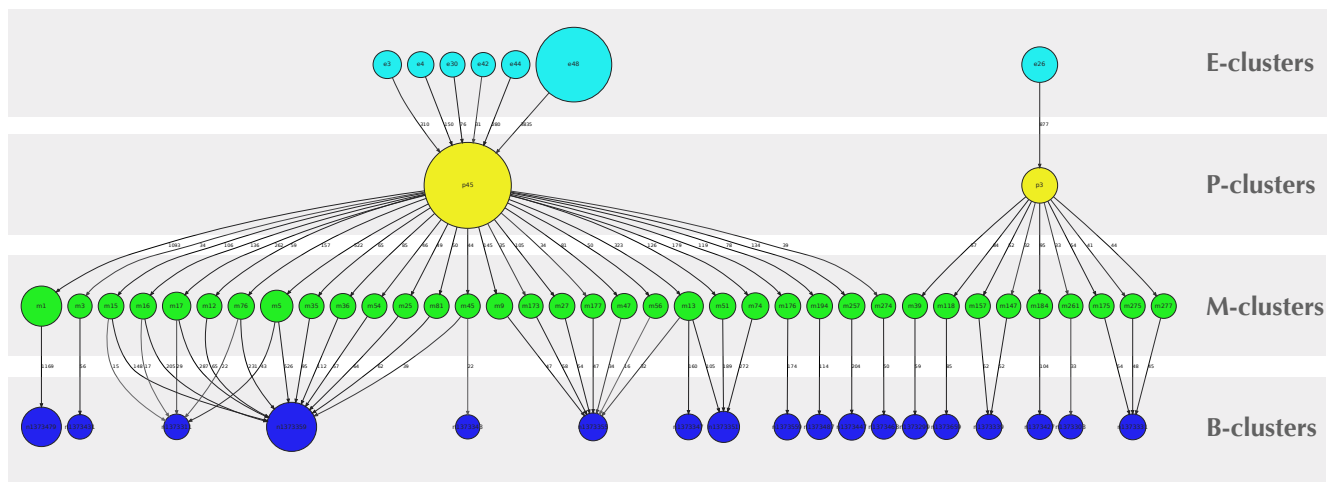
Figure 3. EPM relationships in the SGNET dataset and their comparison to B-clusters

to static information, and the last one to malware grouped according to its behavior. Because of space limitations, we have represented here only the E-, P-, M-, and B- clusters grouping together at least 30 attack events. Figure 3 is therefore a simplified, yet, representative view of the reality. We can identify a set of interesting facts:

- The number of exploit/payload combinations is low with respect to the number of different M-clusters. Most malware variants seem to be sharing few distinct exploitation routines for their propagation.
- The same payload (P-cluster) can be associated to multiple exploits (E-clusters).
- The number of B-clusters is lower than the number of M-clusters. Some M-clusters are likely to correspond to variations of the same codebase, and, thus, maintain very similar behavior.

In the following section, we dig deeper into some of the identified relations to underline the value of considering different standpoints.

4.2. Clustering anomalies

Clustering techniques relying on behavioral features are known to be subject to misclassification due to anomalies in the extraction of these features. On the one hand, the amount of variability in the behavioral profiles of some samples and their interaction with clustering thresholds may lead to clustering artifacts. On the other hand, the sample behavior may be affected by external conditions (e.g., availability of C&C servers) that affect its profile and can lead to misclassifications.

These problems are potentially very difficult to identify. In this section, we show how the combination of clustering approaches based on both static and dynamic information can be helpful in systematically identifying and filtering out clustering anomalies.

When going into the details of the relationships between M-clusters and B-clusters (details that do not appear in Figure 3 because of the filtering choice), we discovered a large number of B-clusters composed of a single malware sample. 860 B-clusters out of 972 are composed of a single malware sample and are associated to a single attack instance in the SGNET dataset.

Comparing these size-1 B-clusters with the associated M-clusters, we can identify a small number of size-1 B-cluster having a 1-1 association with a static M-cluster. These cases are likely to be associated to infrequent malware samples whose infection was observed by the SGNET deployment a very limited number of times.

In most of the other cases, instead, the size-1 B-clusters are associated to larger M-clusters, mostly associated to at least another, larger B-cluster. By manually looking at the behavioral profiles of the samples affected by this anomaly, we could not discern substantial differences from the behavior of the larger B-cluster. We believe these clusters to have been incorrectly clustered, a bias in clustering likely to be associated to the employment of supervised clustering techniques (single linkage hierarchical clustering) in Anubis clustering [4].

This is corroborated by other information available in the SGNET dataset: Figure 4 shows information on the names assigned to these misclassified samples by a popular AV vendor (top), and on the propagation strategy observed in the SGNET dataset in terms of EP coordinates (bottom). Most of the samples are classified as being different vari-
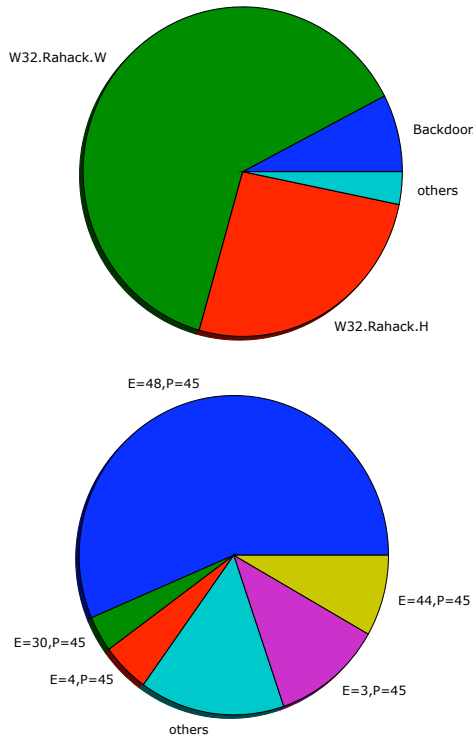
Figure 4. Characteristics of the size-1 clusters. On the top, AV names associated to the malware samples by a popular AV vendor. On the bottom, propagation strategy used by the samples in terms of combination of E and P clusters.

ants of the Rahack worm (also known as Allaple), and have been all pushed with a very specific P-pattern, P-pattern 45. This P-pattern is characterized by a PUSH-based download, in which the attacker forces the victim to receive the sample on a specific port, TCP port 9988. For all these reasons, all these samples seem to be highly related.

This anomalous condition would have been impossible to detect by looking solely at B-clusters: it would not have been possible to discern behaviors uniquely associated to a specific malware sample from this type of misclassification.

We have also detected more subtle clustering anomalies generated by changes in the context of the malware execution. For instance, M-cluster 13 is a polymorphic malware associated to several different B-clusters. The cluster is characterized by a very specific size, number of sections, and even by specific names for the different sections as it is possible to see by looking at the pattern invariant features:

```
{
    MD5='*', size=59904,
    type='MS-DOS executable PE for
         MS Windows (GUI) Intel
         80386 32-bit',
    machinetype=332,
    nsections=3,
    ndlls=1,
    osversion=64,
    linkerversion=92,
    sectionnames='.text\x00\x00\x00,
               rdata\x00\x00\x00,
               .data\x00\x00\x00'
    importeddll='KERNEL32.dll',
    kernel32symbols='GetProcAddress,
                    LoadLibraryA'
}
```

Seeing the rather large number of invariant features (only the MD5 hash is associated to a "do not care" field), we can, therefore, say that the cluster is rather specific.

This cluster is peculiar for the amount of similarities, and yet differences, with the behavior normally associated with the Allaple worm. Looking at the EPM classification, this M-cluster shares the same propagation vector as the samples classified by AV vendors as Allaple/Rahack. The associated pattern in the M dimension is slightly different though: different size, linker version and differences in the Kernel32 symbols. In both cases, the samples are polymorphic: the MD5 is not an invariant feature of the cluster. Interestingly, the polymorphic pattern is different: according to the SGNET data, Allaple mutates its content at each attack instance, while malware samples belonging to the M-cluster 13 seem to mutate their content according to the IP address of the attacker. The same MD5 hash appears multiple times in multiple attack instances generated by the same attacking source towards multiple honeypots, but because of the relevance constraints defined in Section 3.2, the hash is not chosen as a relevant feature.

Looking manually at the different behavioral reports for some of the behavioral clusters, we see that a number of samples exhibit different behavior because of environmental conditions. Upon execution, they try to resolve the name "iliketay.cn", and download from that domain additional components to be executed. Among the different behavioral clusters, one cluster is characterized by the download and execution of two separate components from the malware distribution site, while another cluster is characterized by the successful download of a single component. In both cases, the activity leads to the connection to an IRC server, that provides commands instructing the process to download additional code from other web servers. In a different behavioral cluster, the DNS resolution for the name "iliketay.cn" is unsuccessful: the entry was probably removed

from the DNS database[2].

This real-world example shows the usefulness of clustering based on static features in pinpointing clustering aberrations produced by the analysis of behavioral profiles. Samples that are apparently unrelated when purely looking at their behavioral profiles, are discovered to be related when analyzing their static characteristics. Once detected, these problems can be fixed by, for instance, re-running the misconfigured samples multiple times. We have experimentally seen on a limited number of samples that re-execution is indeed very effective in eliminating these anomalies. While the generation of multiple behavioral profiles for all the samples would be too expensive and probably unnecessary in most of the cases, the comparison with static analysis techniques allows to detect small groups of samples for which it would be useful to repeat the behavioral analysis to improve consistency.

4.3. The value of the propagation context

When looking at Figure 3, we can identify a considerable amount of cases in which a B-cluster is split into several different M-clusters. In order to get a better understanding on the motivations underlying these splits, we have selected two of the biggest B-clusters associated to multiple M-clusters.

The behavioral profiles generated by Anubis are obviously limited in time. At the time of writing, each behavioral profile corresponds to an execution time of four minutes. While such a behavioral profile includes information on the generated network traffic, this information is not sufficient to understand the dynamics of the malware propagation. This is especially true for bots, whose behavior depends on external commands generated by a bot-herder according to his will. Honeypots are instead helpful in better understanding and studying these dynamics.

Figure 5 shows contextual information for the two B-clusters under consideration. The X axis splits the B-cluster into the different M-clusters associated with it, while the different graphs show, from top to bottom, the distribution of the infected hosts over the IP space, the number of weeks of activity of the different classes, and the timeline of activity.

In the first case, we see that the different malware variants are associated with different population sizes in terms of infected hosts scanning the Internet. The different population sizes, combined with the small coverage of the SGNET deployment in terms of the number of monitored IPs (at the time of writing, 150 IPs are monitored by the deployment in 30 different network locations), makes the smaller groups account for only a few hits in the dataset.

---

[2]As of today, any resolution attempt for the above name fails, and the name appears in many popular blacklists

Nevertheless, the distribution of the infected hosts over the IP space, as shown in Figure 5, is very similar and generally widespread over most of the IP space. The specific case taken into consideration is indeed a cluster composed of multiple Allaple variants. Allaple is a self-propagating worm exploiting the Microsoft ASN.1 vulnerability (MS04-007) to propagate to its victims, infect HTML files present on the local system and carry on simple DoS attacks against specific targets. The analysis of the Allaple infection revealed the existance of several modifications and improvements to the original code [24], but due to the lack of a Command & Control channel, the worm lacks self-updating capability. Therefore, hosts infected by the different versions and patches of the original codebase coexist on the Internet. While all these different modifications lead to the generation of two different B-clusters, they generate almost 100 different static clusters. One of the main differentiation factors among the different classes is the file size: while the polymorphic routine used by Allaple modifies the binary content without modifying the size, we can detect in the SGNET dataset a variety of M-clusters, all linked to the same B-clusters, but characterized by different binary sizes. In some cases, the different variants also have different linker versions, suggesting recompilations with different versions of the compiler.

Looking at the right side of Figure 5, we can immediately identify important differences in the size of the infected populations, and on their distribution over the IP space. In this case, we face rather small populations in terms of numbers of attackers, with most of the static clusters associated to very specific networks of the IP space. All the activities are bursty, and in some cases, the temporal evolution exposes complex behaviors that we consider likely to be associated to bot-like coordinated behavior. For instance, looking in depth at the temporal evolution of one of the M-clusters, we obtain the following sequence:

- 15/7 - 16/7: observed hitting network location A
- 18/7: observed hitting network location B
- 26/7: observed hitting network location B
- 2/8-4/8: observed hitting network location A
- 27/9: observed hitting network location B

Such coordinated behavior suggests the presence of a Command&Control channel.

We have, therefore, looked closer into the behavioral profiles for these samples, and have tried to gather evidence of the presence of an IRC Command & Control server. While not all the samples were executed by Anubis during the activity period of the C&C server, we have been able to associate some of the M-clusters taken into consideration to the corresponding IRC server. Table 2 shows how, in most cases, each M-cluster is characterized by connections to a specific IRC server. In the minority of cases in which
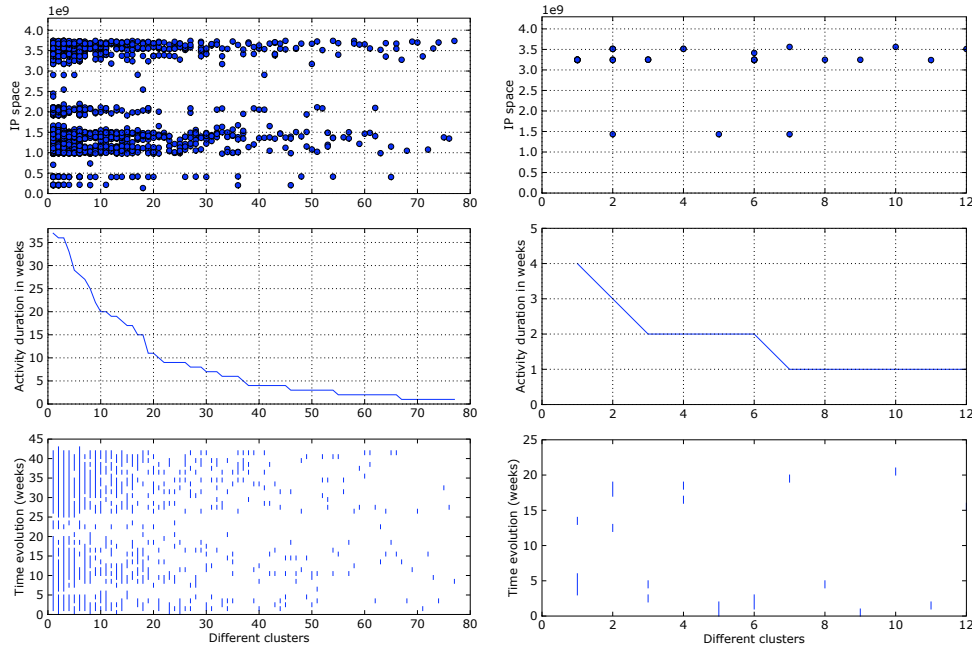
Figure 5. Propagation context information for two behavioral clusters.

distinct M-clusters operate on the same IRC channel and receive commands from the same room name, they are likely to be associated to different code variants or "patches" applied to the very same botnet. But even when looking at M-clusters operating on different IRC channels, the characteristics of the IRC channels reveal very strong similarities: many IRC servers are hosted in the same /24 subnet, and send commands to the bots from rooms with recurring names or name patterns. This suggests the operation of a specific bot-herder or organization that is maintaining multiple separate botnets.

The combination of malware clustering techniques (i.e.,

Server address	Room name	M-clusters
67.43.226.242	#las6	282, 70
67.43.232.34	#kok8	263
67.43.232.35	#kok6	23, 277
67.43.232.36	#kham	170
67.43.232.36	#kok2	37
67.43.232.36	#kok6	195, 275
67.43.232.36	#ns	234
72.10.172.211	#las6	266
72.10.172.218	#siwa	140
83.68.16.6	#ns	112

Table 2. IRC servers associated to some of the M-clusters

based on both static and dynamic features) with long term contextual information on their evolution is helpful in practice. That is, such techniques allow us to understand better how malware is developed and propagated. Our work shows the importance of leveraging different information sources for studying the threat evolution, and the "economy" of its driving forces.

5. Conclusion

In this paper, we evaluate and combine different clustering techniques in order to improve our effectiveness in building intelligence on the threats economy.

We take advantage of a freely accessible honeypot dataset, SGNET, and propose a pattern generation technique to explore the relationships between exploits, shell-codes and malware classes while being resistant to the current level of sophistication of polymorphic engines. Despite the simplicity of the approach and the easiness with which it could be evaded by malware writers, we show that the current level of sophistication of polymorphic techniques is very low and that simple clustering techniques based on static features often work well in practice. Furthermore, we show the importance of these techniques in detecting and "healing" known problems in dynamic analysis, such as the dependence of the execution behavior on external conditions (e.g., such as the availability of a command and control server).

Moreover, we offer insights into the relationships between different malware classes, their propagation strategies and their behavioral profiles. We show with practical examples how different techniques offer different perspectives over the ground truth. The propagation context allows to build semantics on top of the malware clusters, providing information that is not easily available through standard dynamic analysis techniques. We show how the propagation vector information can be used to study code-sharing taking place among malware writers, and how temporal information and source distribution can be effectively used to provide semantics on the threat behavior to the security analyst.

Acknowledgments

References

[1] ANUBIS: A platform for the dynamic analysis of malware. http://anubis.iseclab.org, 2009.

[2] P. Baecher, M. Koetter, T. Holz, M. Dornseif, and F. Freiling. The Nepenthes Platform: An Efficient Approach to Collect Malware. In *9th International Symposium on Recent Advances in Intrusion Detection (RAID)*, September 2006.

[3] M. Bailey, J. Oberheide, J. Andersen, Z. M. Mao, F. Jahanian, and J. Nazario. Automated classification and analysis of internet malware. In *Proceedings of the 10th International Symposium on Recent Advances in Intrusion Detection (RAID'07)*, September 2007.

[4] U. Bayer, P. M. Comparetti, C. Hlauschek, C. Kruegel, and E. Kirda. Scalable, Behavior-Based Malware Clustering. In *16th Annual Network& Distributed System Security Symposium*, 2009.

[5] U. Bayer, I. Habibi, D. Balzarotti, E. Kirda, and C. Kruegel. Insights into current malware behavior. In *LEET'09: 2nd USENIX Workshop on Large-Scale Exploits and Emergent Threats, April 21, 2009, Boston, MA, USA*, Apr 2009.

[6] U. Bayer, C. Kruegel, and E. Kirda. TTAnalyze: A Tool for Analyzing Malware. In *15th European Institute for Computer Antivirus Research (EICAR 2006) Annual Conference*, April 2006.

[7] J. Canto, M. Dacier, E. Kirda, and C. Leita. Large scale malware collection: Lessons learned. In *IEEE SRDS Workshop on Sharing Field Data and Experiment Measurements on Resilience of Distributed Computing Systems*, 2008.

[8] E. Carrera. Pefile, http://code.google.com/p/pefile/.

[9] J. R. Crandall, Z. Su, and S. F. Wu. On deriving unknown vulnerabilities from zero-day polymorphic and metamorphic worm exploits. In *12th ACM conference on Computer and Communications Security*, pages 235–248. ACM Press New York, NY, USA, 2005.

[10] F-Secure. Malware information pages: Allaple.a, http://www.f-secure.com/v-descs/allaplea.shtml, December 2006.

[11] M. Gheorghescu. An Automated Virus Classification System. In *Virus Bulletin conference*, 2005.

[12] T. Holz, C. Willems, K. Rieck, P. Duessel, and P. Laskov. Learning and Classification of Malware Behavior. In *Fifth Conference on Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA 08)*, June 2008.

[13] K. Julisch. Clustering intrusion detection alarms to support root cause analysis. *ACM Transactions on Information and System Security (TISSEC)*, 6(4):443–471, 2003.

[14] J. Z. Kolter and M. A. Maloof. Learning to detect and classify malicious executables in the wild. *J. Mach. Learn. Res.*, 7:2721–2744, 2006.

[15] T. Lee and J. J. Mody. Behavioral Classification. In *EICAR Conference*, 2006.

[16] C. Leita. *Automated protocol learning for the observation of malicious threats*. PhD thesis, Université de Nice-Sophia Antipolis, December 2008.

[17] C. Leita and M. Dacier. SGNET: a worldwide deployable framework to support the analysis of malware threat models. In *7th European Dependable Computing Conference (EDCC 2008)*, May 2008.

[18] C. Leita and M. Dacier. SGNET: Implementation Insights. In *IEEE/IFIP Network Operations and Management Symposium*, April 2008.

[19] C. Leita, M. Dacier, and F. Massicotte. Automatic handling of protocol dependencies and reaction to 0-day attacks with ScriptGen based honeypots. In *9th International Symposium on Recent Advances in Intrusion Detection (RAID)*, Sep 2006.

[20] C. Leita, K. Mermoud, and M. Dacier. Scriptgen: an automated script generation tool for honeyd. In *21st Annual Computer Security Applications Conference*, December 2005.

[21] C. Linn and S. Debray. Obfuscation of executable code to improve resistance to static disassembly. In *10th ACM conference on Computer and communications security*, pages 290–299. ACM New York, NY, USA, 2003.

[22] A. Moser, C. Kruegel, and E. Kirda. Limits of static analysis for malware detection. In *23rd Annual Computer Security Applications Conference (ACSAC)*, 2007.

[23] G. Portokalidis, A. Slowinska, and H. Bos. Argos: an emulator for fingerprinting zero-day attacks. In *ACM Sigops EuroSys*, 2006.

[24] D. Smith. Allaple worm (ISC diary), http://isc.sans.org/diary.html?storyid=2451.

[25] VirusTotal. www.virustotal.com, 2007.

[26] G. Wicherski. pehash: A novel approach to fast malware clustering. In *2nd USENIX Workshop on Large-Scale Exploits and Emergent Threats*, April 2009.

[27] T. Yetiser. Polymorphic viruses - implementation, detection, and protection , http://vx.netlux.org/lib/ayt01.html.