DUET: Integration of Dynamic and Static Analyses for Malware Clustering with Cluster Ensembles

Xin Hu
IBM T.J. Watson Research Labs
huxin@us.ibm.com

Kang G. Shin
University of Michigan, Ann Arbor
kgshin@eecs.umich.edu

ABSTRACT

Automatic malware clustering plays a vital role in combating the rapidly growing number of malware variants. Most existing malware clustering algorithms operate on either static instruction features or dynamic behavior features to partition malware into families. However, these two distinct approaches have their own strengths and weaknesses in handling different types of malware. Moreover, different clustering algorithms and even multiple runs of the same algorithms may produce inconsistent or even contradictory results. To remedy this heterogeneity and lack of robustness of a single clustering algorithm, we propose a novel system called DUET by exploiting the complementary nature of static and dynamic clustering algorithms and optimally integrating their results. By using the concept of *clustering ensemble*, DUET combines partitions from individual clustering algorithms into a single consensus partition with better quality and robustness. DUET improves existing ensemble algorithms by incorporating *cluster-quality measures* to effectively reconcile differences and/or contradictions between base malware clusterings. Using real-world malware samples, we compare the performance of DUET (in terms of clustering precision, recall and coverage) with individual state-of-the-art static and dynamic clustering component. The comprehensive experiments demonstrate DUET's capability of improving the coverage of malware samples by 20–40% while keeping the precision near the optimum achievable by any individual clustering algorithm.

1. INTRODUCTION

The growing popularity of automatic malware-creation toolkits, which allow even marginally skilled attackers to create and customize malware programs, has produced a plethora of malware variants. Clustering plays a key role in processing new incoming malware samples. Automatic and accurate partitioning of these samples into families will (1) allow analysts to prioritize allocation of precious human resources for more important, distinct malware programs; (2) enable automatic classification of incoming malware samples by associating them with existing clusters; and (3) generalize existing mitigation techniques for new variants. However, different clustering algorithms often generate inconsistent results. Even multiple runs of the same algorithm may produce distinct results due to different parameter settings, random initialization or stochastic learning process [32]. Furthermore, different approaches have their respective strengths and limitations that often depend on the characteristics of input data, such as data distribution, pre-processing procedures, anti-analysis techniques used by malware programs, etc. Thus, no single algorithm can perform optimally across various data sets, and a wide range of clustering algorithms have been proposed [4, 6, 20, 23, 27, 31].

Most existing approaches cluster malware using their *dynamic behavior* or *static features*. The main benefit of dynamic behavior based clustering is its resilience to low-level mutation techniques, such as packers or binary obfuscation that do not affect runtime behavior. However, behavior-based approaches also suffer from several weaknesses. First, they may have only limited coverage of an application's behavior, failing to reveal the entire capabilities of a given malware program. This is because a dynamic analysis system can only capture API or system-call traces corresponding to the code path taken during a particular execution. Unfortunately, depending on the program's internal logics and/or external environments, different code paths may be taken during different executions. For instance, many malware exhibit interesting behavior only when certain conditions are met, such as specific date/time or receipt of network commands (e.g. botnet). In contrast, static clustering algorithms analyze features extracted from a program's binary or disassembled instructions. They are capable of covering all possible code paths of an application, including parts of the program that normally do not execute, thereby yielding more accurate characterization of the program's functionalities. However, their performance may suffer when facing binary-level packing, anti-reversing and anti-disassembly techniques. Because of these limitations, it is often very challenging, if not impossible, to develop a single, effective clustering algorithm for all sets of malware programs.

In this paper, instead of focusing on developing a single clustering algorithm that works only for a narrow range of datasets, we design a *unified* clustering framework, called DUET, exploiting the complementary features of static and dynamic clustering approaches. DUET systematically combines different clustering algorithms based on the concept of *cluster ensemble*. Specifically, given a set of (dynamic or static) clusterings $\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_m$, cluster ensemble attempts to derive a single clustering $\mathcal{C}$ that, according to certain criteria, is in as much agreement as possible with the original m clusterings. By exploiting the consensus among different clustering algorithms, cluster ensemble combines strengths of individual clusterings and yields better-quality clusters

ACSAC '13 Dec. 9-13, 2013, New Orleans, Louisiana USA

ACM 978-1-4503-2015-3/13/12 ...$15.00
http://dx.doi.org/10.1145/2523649.2523677.

with higher coverage. For instance, a dynamic approach may mistakenly put all malware programs that stop execution after detecting a virtual environment into the same cluster, owing to the similar system calls invoked by the malware before they exit. Such coverage gaps can often be resolved by using a static approach to grouping them according to their binary traits. One challenge is how to reconcile the conflicting results generated by different clustering algorithms. If static and dynamic approaches are assigned an equal weight when reconciling their results, the ensemble algorithm can, at best, make random choices. We address this problem via a *Clustering-Quality* metric, which gauages how strongly connected/related the data points are within each cluster. By assigning each cluster a quality score, high-quality clusters, where member malware programs share stronger connections, are assigned heavier weights when combining the conflicting results, thus improving the overall clustering accuracy. Our evaluation of DUET with real-world malware samples demonstrates its ability to improve the malware coverage by 20–40% while keeping the accuracy near the optimum achievable by any individual clustering algorithm.

The main contribution of DUET is to employ clustering ensemble as a principled method to build a framework that *systematically* unifies static and dynamic clustering results, achieving much better results than any single approach alone. Although the ideas of exploiting diverse perspectives of static and dynamic features have been discussed before, DUET is unique in that it systematically addresses this problem and demonstrates the effectiveness of a relatively large-scale working prototype. DUET thus offers additional insights into the complementary nature of different analysis approaches and efficient ways of integrating them in malware clustering.

The rest of this paper is organized as follows. Section 2 discusses related work. Section 3 provides a brief overview of DUET architecture. Section 4 describes static and dynamic clustering algorithms. Sections 5 and 6 detail the proposed cluster ensemble algorithms. Section 7 presents the evaluation results and Section 8 discuss the limitations. Finally, Section 9 concludes the paper.

2. RELATED WORK

Static feature-based methods are among the first few approaches proposed for malware clustering due mainly to their simplicity and fast speed. Static features like PE headers [31], code patterns [7], instruction sequences [16] or binary sequences [15] are extracted and used to train various learning algorithms [19] for malware categorization and detection. Unfortunately, the popularity of static approaches encouraged malware writers to develop obfuscation techniques and thwart the static analysis. As a result, *dynamic* approaches have recently received significant attention. For instance, Rieck *et al.* [27] trained a SVM classifier using the malware's run-time behavior, such as copy files and create processes. Bailey *et al.* [4] employed a hierarchical clustering algorithm to group similarly-behaving malware samples based on non-transient state changes. More recently, Bayer *et al.* [6] adopted locality-sensitive hashing (LSH) and Rieck *et al.* [20] developed a prototype-based clustering algorithm to improve the scalability of behavior-based clustering. Unfortunately, dynamic approaches also have their own weaknesses, e.g., an incomplete view of program's behavior. Therefore, a systematic approach to combining the strengths of both static and dynamic analysis algorithms is highly desirable. Although the ideas of combining static and dynamic clusterings have been mentioned in both industry and academia [3, 24], very few have addressed their systematic integration. The work most relevant to ours are those of Leita *et al.* [24] and Anderson *et al.* [3]. In their work in [24], Leita *et al.* focused on gaining insights into the relationship among different code variants and presented several scenarios where the combination of static and dynamic sources can help detect clustering anomalies. However, these integrations of static and dynamic clusterings are mostly done manually and studied on a case-by-case basis. Anderson *et al.* [3] applied a multi-kernel learning method to combine similarity metrics for different data sources including static and dynamic features of malware programs. They trained a multi-kernel SVM (Support Vector Machine) classifier on features extracted from 780 malicious and 776 benign programs, and demonstrate benefits of combining different data sources in classifying malicious programs.

Cluster ensemble is a process of obtaining a single consensus and better clustering results from a number of different clusterings [2]. Strehl and Ghosh [2] considered cluster ensemble as a knowledge-reuse framework for combining different clusterings. Hong *et al.* [14] implemented an approach that combines the relabeling and voting to achieve the best agreement between the labels of partitions. Fed [10] and Jain [11] constructed the consensus function as a co-association matrix which represents the association or connectivity between each pair of data samples. These work all showed that ensemble is an effective way to improve the robustness, stability and accuracy of clustering. In this paper, we employed the connectivity matrix-based ensemble approaches.

3. SYSTEM OVERVIEW

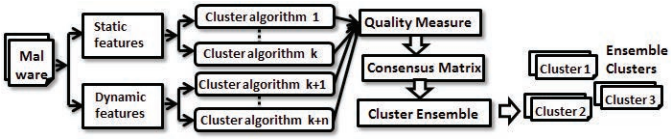

Figure 1: An overview of DUET

Fig. 1 depicts the system architecture of DUET. Given a set of malware programs, DUET first uses two different feature extractors to derive (1) static-instruction features and (2) dynamic-behavior features. Second, the results of individual clusterings are generated by applying different clustering algorithms with different parameters settings. Then, the quality of each individual clustering is evaluated and used to construct the connectivity-based consensus matrix, where ensemble methods are applied to derive the final partitions. We will elaborate each component of DUET in the following sections.

4. MALWARE CLUSTERING

This section introduces the base static and dynamic clustering algorithms used by DUET for ensembles. We adopt the state-of-the-art malware clustering algorithm proposed in [20] because of its effectiveness and scalability. Note, however, that DUET does not rely on specific algorithms and can easily accommodate other clustering algorithms to increase the diversity of input clusterings.

4.1 Clustering Using Static Features

DUET's static clustering component, henceforth referred to as DUET-S, exploits the common observation that a large portion of today's malware are file-level variations of a small number of families and tend to share similar *instruction sequences*. DUET-S consists of the following steps.

Unpacking: Packing is arguably the most popular techniques used by malware writers to circumvent anti-virus detection. A typical packer (e.g UPX) works by creating a new binary containing the compressed version of the original binary followed by the unpacker code. When the packed program runs, the unpacker will first be executed. The unpacker de-compresses the original program codes, write them into some memory location and jump to the first instruction of the restored codes to start execution. Packing allows malware to hide malicious instructions and evade anti-virus detection, while keeping the original functionalities intact. Since DUET-S , likely any other static analysis approach, relies on features extracted from original instructions, it is imperative to correctly unpack input malware programs.

While there exist unpacking tools such as UnPECompact, ASProtect Deprotector, etc., they are often targeted specifically at one or a few packers. In addition, they often have to perform expensive processing to ensure that the unpacked program can be successfully executed (e.g., the file headers and imported tables must be correctly reconstructed), making them too slow for large scale processing. In DUET-S, we tailored a generic unpacking algorithm proposed in [13] for efficient malware clustering by exploiting the observation that, for static analysis, DUET-S need not guarantee the executability of unpacked programs as long as the original instructions can be inspected and features extracted. The basic idea is to utilize the inherent property of the unpacking procedure, i.e., a packed binary has to write the original code into some memory space and execute them. Hence by continuously monitoring memory access patterns of write and execute, one can detect the occurrence of some form of unpacking, self-modification or on-the-fly code generation.

To achieve this, DUET-S takes advantages of the physical non-execution (NX) support in modern x86 CPUs. Specifically, given a packed program, DUET-S loads it into memory, marks all the memory pages as *executable* but *non-writable*. Then during the execution, when the unpacker attempts to write original codes into some memory page (which has been marked as non-writable), a write exception will occur. DUET-S captures such an exception, marks the page as dirty and changes the permission to *writable* but *non-executable*. When the unpacker jumps to the newly-generated code for execution (e.g., after finishing unpacking), the absence of executable permission on these pages triggers an execution exception. DUET-S intercepts the exception and records the memory page's address (these modified-then-executed memory pages likely contain the original machine instructions and thus are targets for feature extraction). Then DUET-S removes the write permission from these memory pages again, grants execution privilege and continues the program's execution. Finally, DUET-S dumps the process memory image either at the end of program execution or after certain time threshold (e.g., 2 minute). The rationale is that after the program has been running for a sufficient amount of time , it is fairly safe to assume that the program has finished unpacking and the original codes are present in the memory. Notice that this generic unpacking process can be seamlessly integrated into dynamic analysis, thereby incurring little additional overhead for DUET.

Table 1: Unpacking effectiveness (IC: Instruction Count; NG Dist: N-gram Difference)

Packer	Diff in IC (%)	NG Dist	Packer	Diff in IC (%)	NG Dist
PECompact	0.88%	0.068	ASProtect	6.70%	0.133
EXECryptor	3.20%	0.176	UPX	0.88%	0.068
EXEStealth	0.88%	0.071	NSPack	0.87%	0.069
VMprotect	2.50%	0.10	Armadillo	–	–

To better understand the effectiveness and limitation of the generic unpacking algorithm, we use 8 popular packing tools to pack malware programs and then compare DUET-S unpacked files with their original version. Ideally, the unpacked binary should be byte- to-byte identical to the original file. However, this is neither possible (DUET-S does not reconstruct the import table, and the unpacker code is also dumped from the memory), nor necessary for the purpose of malware clustering. As a result, we use following two metrics are directly related to clustering accuracy: (i) the difference in their instruction count (IC), and (ii) the distance between their N-gram feature vectors (NG, details are elaborated later in this section). Table 1 summarizes the results. For most packers, DUET-S successfully recovered original binaries with only a 1–6% increase of ICs which is often due to the inclusion of unpacker routines in the dumped memory. Besides, the feature vectors of unpacked binaries are very similar to those of the original binary with most normalized distance measurements below 0.1, where 0 means identical and 1 means completely different. However, DUET-S also failed on some packers e.g., Armadillo. A further investigation showed that Armadillo works by unpacking an intermediate executable on disk and creating another process to run this executable. Therefore, memory dump of an Armadillo-packed file does not contain original instructions. After running DUET-S on the large data set, we have also observed other causes of unsuccessful unpacking, such as malware samples refuse to run in a virtual machine or the time required for unpacking is longer than the threshold. Hence despite the effectiveness of DUET-S against popular packers, these failed cases demonstrates the necessity and importance of complementing DUET-S with dynamic analysis. DUET-S discard binaries that cannot be unpacked to avoid extracting bogus features from unpcker codes. Thanks to DUET's ensemble approach, most of these un-packable binaries can still be clustered by their dynamic features.

Feature Extraction: Since malware programs often undergo changes due to polymorphism and obfuscation, exact comparison between instructions will not tolerate any syntax variation. At the other extreme, if all forms of variation are tolerated, correctness is compromised. To strike a balance, DUET-S exploits the x86 instruction format[1] and uses the opcode as a succinct representation of instruction semantics. Using opcodes—instead of widely-used features such as binary sequences [15] or mnemonics i.e. 'mov' and 'add'—offers several benefits. First, by ignoring operands, opcode allows DUET-S to tolerate low-level mutations and obfuscation among malware variants. Second, comparing with

[1]In x86, an instruction (e.g., add eax, 4Fh whose binary form is 83C04F) consists of *opcode* which specifies the operation to be performed and *operants* on which the operation is performed. In the above example, the opcode is '83', the operation is 'add' and the operants are 'eax' and '4F'.

Table 2: Opcodes provide fine-grained representations of instruction semantics

Binary	Instruction	Opcode	Mnem
83 C0 4F	add eax, 4Fh	83	ADD
B8 4F 00 00 00	mov eax, 4Fh	B8	MOV
0F 22 C0	mov CR0, ecx	0F 22	MOV

mnemonics, the opcode sequence offers more precise representation of instruction semantics. Mnemonics sometimes *overly* generalize underlying CPU operations, leading to accidental similarity between different code sequences. For instance, in Table 2, although both mov instructions share the same mnemonic, the underlying semantic is drastically different.

DUET-S represents each malware program as a sequence of opcodes and applies the standard N-gram analysis [20] to construct a feature vector V. Each value in V represents the number of *occurrence* of a particular opcode N-gram. The similarity between two programs can then be calculated geometrically as the Euclidean distance between their feature vectors in the vector space.

Clustering with N-gram Feature Vectors: DUET-S employs the prototype-based linear-time clustering algorithm (henceforth referred to as *ProtoCluster*) developed in [20]. The main advantage of ProtoCluster is its significantly faster clustering speed achieved by performing most computation on a relatively small set of prototypes and avoiding expensive pair-wise data point comparisons required by many classical clustering algorithms (e.g., K-mean or hierarchical clustering). The first step of ProtoCluster is to extract a set of prototypes—data points that are typical for a group of homogeneous data samples. Because selection of an optimal set of prototypes is known to be NP-hard, ProtoCluster uses a greedy algorithm. It starts with choosing a random sample as the first prototype and then iteratively selects the next prototype by searching for the sample that has the largest distance to its nearest prototype. This selection process is terminated when the distance from all the data points to their nearest prototype is smaller than a predefined threshold P_{max} (i.e., all the input data points are located within a certain radius from some prototype). Then, ProtoCluster performs standard agglomerative hierarchical clustering only on the prototypes (rather than the original input data). First, each prototype form a single-node cluster. The algorithm recursively merges two closest prototype clusters until the distance between the closest clusters is larger than a predefined threshold Min_d. Prototypes within the same cluster are assigned the same label and the label is subsequently propagated to their associated data points (i.e., those within the P_{max} distance to the prototype). The authors of [20] showed that the runtime complexity of ProtoCluster is $O(k^2 log\ k + kn)$ where k is the number of prototypes and n the total number of malware samples. Since k depends only on the distribution of the data (i.e., k is proportional to the number of malware families), with proper choice of P_{max}, the algorithm is linear in the number of input data n. Applying ProtoCluster on the extracted static features with different parameters P_{max} and Min_d, DUET-S produces multiple clusterings of input malware samples, which serve as an input to the ensemble algorithm.

4.2 Malware Clustering Using Run-time Traces

This section briefly describes the dynamic-behavior-based component of DUET, referred to as DUET-D. DUET-D adopts Malheur [20], a state-of-the-art tool for automatic clustering of malware behavior collected from sandbox environments such as CWSandbox or Anubis. In DUET-D , for better scalability and privacy, instead of using existing web-based malware analysis services as those used in Malheur, we built our own system call monitor to intercept and record all API/system calls of malware programs in VMWare virtual machines. This allows DUET-D to take advantage of the VMWare VIX API to automate and parallelize the running of malware programs. Each malware program is monitored for 2 minutes, and the virtual machine is reset to the clean-state snapshot, preventing interference between malware programs.

Table 3: Examples of encoding system/API calls

	Category	Operation	System Call
File System	0x03	0x01	CreateFile
	0x03	0x02	CopyFile
Process Handling	0x0A	0x02	CreateProcess
	0x0A	0x02	CreateProcessInternal
	0x0A	0x02	CreateProcessAsUserr
	0x0A	0x02	NtCreateProcess

The monitor produces a textual format of API/system call traces. According to [20], we encode each call into a tuple '(category, operation)', where a category represents a group of calls that operate on a similar type of objects (e.g., registry, file systems, DLLs or processes), and an operation specifies a particular function. For example, in Table 3, the tuple '(0x03, 0x01)' indicates that the call belongs to the 'File System' category and is performing the 'CreateFile' operation. Furthermore, we assign the same operation value to those calls that can achieve identical results. For instance, 4 system calls that can be used to create a new process are encoded with the same tuple, '(0x0A, 0x02)'. This canonicalization of function variations enables a more generalized representation of call traces and also ensures locality of calls with similar functionalities in the encoded space. In total, DUET-D intercepts and encodes 211 windows API/system calls into 21 categories, covering most frequently used functionalities including file system, registry, network, mutex, process, thread, virtual memory, etc.

Because typical malware behavior patterns, such as modifying the registry keys and file systems, can be reflected in the system call sequences, DUET-D apply the N-gram analysis as similar to DUET-S , embedding the encoded system call sequences into a fixed-length feature vector whose distance represents the similarity between malware behaviors. However, one important difference is that: unlike in DUET-S where the feature vector elements represent how often a specific n-gram of *instructions* occurs, DUET-D uses *binary features* (i.e., 0 or 1) where each feature vector element represents the absence or presence of a specific N-gram of the call traces. Using binary features help DUET-D reduce the influence of many external factors, such as the length of traces, the repetition of behavior, etc. For instance, depending on the monitoring period and system condition, the number of system calls in a call trace can vary significantly, even for identical malware programs. Some system calls may be executed in a loop and produce thousands of repetitions, which considerably skews the values in the feature vector. To compensate for this bias, DUET-D uses binary features and normalizes the feature vector to ensure that the difference between feature vectors depends only on the presence/absence

of features. After feature extraction and vector encoding, DUET-D applies the ProtoCluster algorithm with different parameter settings to generate clusterings as the inputs to the ensemble algorithm.

5. CLUSTER ENSEMBLE

The goal of using cluster ensemble is to improve the quality and the robustness of clustering results by exploiting the diversity of multiple clustering algorithms.

Table 4: Success rate of different approaches (DUET-S and DUET-D) in extracting malware features

	# of success (%)	# of failure (%)
Dynamic approach	4943 (87.5%)	704 (12.4%)
Static approach	4799 (84.9%)	944 (16.7%)
Both approaches	5575 (98.72%)	72 (1.28%)

Motivating Examples The respective limitations of dynamic and static approaches, when used alone, can render them ineffective for certain types of malware. To illustrate this, we ran both approaches on a set of 5,647 real-world malware samples (Table 5 in Section 7 lists the details) to collect N-gram features on their code instructions and dynamic system-call traces. Table 4 summarizes the number of malware samples whose features can be successfully extracted by dynamic, static and both analyses. It demonstrates the capabilities and shortcomings of these two approaches as well as their complementary nature. Table 4 shows that the with a 12 % and 17% failure rate[2], neither static nor dynamic approach alone is able to analyze all the samples. Detailed investigation revealed that among 12 % samples that failed dynamic analysis, 645 cannot be executed, and 25 only made a single system call, i.e., TerminateProcess, due to detection of the virtual machine environment[3]. On the other hand, the disassembler failed to extract any static instructions from 655 binaries (particularly for FakeAV and Gammima) due to the use of sophisticated obfuscation and packing techniques. By contrast, a combination of dynamic and static analyses yielded much better results: malware samples that can be successfully analyzed increased to 98.72%, with only 72 malware samples being able to evade both approaches. This preliminary experiment demonstrates that aggregating different analysis approaches has the potential for achieving more robust and complete clustering.

5.1 Formulation of Cluster Ensemble

Consider a set of n malware programs, $X = x_1, x_2, \ldots, x_n$, and a set of T clusterings of X, $\mathcal{C} = \{\mathcal{C}^1, \mathcal{C}^2, \ldots, \mathcal{C}^T\}$. Each clustering, $\mathcal{C}^t$, $t = 1, 2, \ldots, T$, is a partition of X into k disjoint clusters, i.e., $\mathcal{C}^t = \{C_1^t, C_2^t, \ldots, C_k^t\}$ where $\bigcup_{i=1}^{k} C_i^t = X$ and $C_i^t \cap C_j^t = \phi$, $\forall i \neq j$. Let $L^t(x)$ denote the label of the cluster to which the malware program, x, belongs, i.e., $L^t(x) = j$ if and only if $x \in C_j^t$. With these T clusterings, the cluster ensemble is defined as a consensus function Γ [2] that maps a set of clusters to an integrated clustering: $\Gamma : \{\mathcal{C}^t | t \in \{1, 2, \ldots, T\}\} \rightarrow C$.

If the relative importance of each individual clustering is not known *a priori*, a natural goal of cluster ensemble is to find the final clustering, $\mathcal{C}$, that shares the most commonality with the constituent clusterings [12].[4] To measure the similarity between clusterings, we define a connectivity matrix, $M(\mathcal{C}^t)$, for each clustering $\mathcal{C}^t$. The connectivity matrix is an $n \times n$ pair-wise matrix defined for all malware programs and represents the structural information of a particular clustering. More specifically,

$$M_{ij}(\mathcal{C}^t) = \begin{cases} 1 & \text{if } x_i \text{ and } x_j \text{ belong to the same cluster} \\ 0 & \text{Otherwise.} \end{cases}$$

Then, the difference between two clusterings, $\mathcal{C}^a$ and $\mathcal{C}^b$, can be defined as the number of malware pairs for which the two clusterings disagree [12]:

$$\begin{aligned} d(\mathcal{C}^a, \mathcal{C}^b) &= \sum_{i,j=1}^{n} d_{i,j}(\mathcal{C}^a, \mathcal{C}^b) = \sum_{i,j=1}^{n} |M_{ij}(\mathcal{C}^a) - M_{ij}(\mathcal{C}^b)| \\ &= \sum_{i,j=1}^{n} (M_{ij}(\mathcal{C}^a) - M_{ij}(\mathcal{C}^b))^2. \end{aligned} \quad (1)$$

Cluster ensemble strives to find a consensus clustering, $\hat{\mathcal{C}}$, that is closest to all of the given clusterings, i.e., that minimizes the average distance between $\hat{\mathcal{C}}$ and $\{\mathcal{C}^t | t \in \{1, 2, \ldots, T\}$:

$$\mathcal{C}^{opt} = \arg\min_{\hat{\mathcal{C}}} \sum_{t=1}^{T} d(\mathcal{C}^t, \hat{\mathcal{C}}). \quad (2)$$

Since $\sum_{t=1}^{T} d(\mathcal{C}^t, \hat{\mathcal{C}})$ is a convex function, minimization results in an optimal matrix $M_{i,j}^{opt}(\hat{\mathcal{C}}) = \frac{1}{T} \sum_{t=1}^{T} M_{ij}(\mathcal{C}^t)$ [32]. Here M^{opt} represents connectivity relationship between data samples in the final ensemble clusters, and the values of $M_{i,j}^{opt}$ ranges between 0 and 1, where 1 (0) means all (none) of the clusterings agree that x_i and x_j belong to the same cluster. Given $M_{i,j}^{opt}$, our goal is to derive the final ensemble clusters $\mathcal{C}^{opt} = \{C_1^{opt}, \ldots, C_k^{opt}\}$.

5.2 Clustering Based on Connectivity Matrix

Several methods have been proposed to generate a final clustering from the optimal connectivity matrix[2, 12]. A naive approach is to set a threshold such that two samples are assigned to the same cluster if their connectivity is greater than the threshold and if these samples belong to different clusters, these clusters are merged. However, the main drawback of such an approach is that it tends to overmerge unrelated clusters which have some samples accidentally close to each other, leading to low clustering accuracy. To address this, in DUET we employ the following algorithms.

The ball algorithm iteratively finds a set of samples that are close to each other (i.e. within a ball) and far from others, removes them from the data set and then continues clustering with the remaining samples. Because finding the globally optimal sequence of clusters is NP-complete, a greedy algorithm is used to create a bounded approximation. Viewing the connectivity matrix, M^{opt}, as a graph's adjacency matrix, where each $M_{i,j}^{opt}$ represents the edge's weight connecting x_i and x_j, the algorithm sorts the samples in decreasing order of their edges' total weights. At each step, the algorithm chooses the first unclustered sample, x_u, and finds a set of samples, $V = x_{v1}, x_{v1}, \ldots, x_{vk}$ whose connectivity to x_u is greater than the threshold, β. Then, their union $V \cup x_u$ forms a cluster.

[2]Failure rate is in accordance with observations from previous work, e.g., [24] reported about 18.7% of their malware set cannot be analyzed by Anubis

[3]Bredolab is one of the top malware families that produce only one system call. According to [28], it is a botnet program that terminates if it detects being executed in a virtual environment.

[4]The algorithm can be easily generalized to the case where some clusterings may carry more weights than others.

The agglomerative algorithm starts by placing all samples as singleton clusters. Then, it recursively merges the two clusters with the smallest distance until the distance between any pair of existing clusters is larger than a given threshold, h. If h is set to 1/2, the algorithm is guaranteed to create clusters where the at least half of the original clusterings are in agreement.

Hypergraph partition algorithm: Essentially, the cluster ensemble re-partitions the original dataset based on constituent clusterings' indication of strong connections. Therefore, it can also be formulated as a hypergraph partition problem, where each sample is a vertex in the hypergraph, and the hyperedge between vertices is weighted based on M^{opt}. In this case, the goal is to cut the minimum set of edges such that the remaining subgraphs consist of connected components corresponding to new clusters.

6. IMPROVING ENSEMBLE WITH CLUSTER QUALITY MEASURES

Standard cluster ensemble approaches weight all clusters equally, i.e., M_{ij} is set to 1 if two data points, x_i and x_j, are in the same cluster, regardless of cluster quality. However, clustering is essentially an unsupervised learning process. Without prior knowledge of the underlying data distribution, every clustering algorithm implicitly assumes certain data models. When these assumptions are not supported by the input data, the algorithms may produce erroneous or meaningless clusters. In other words, because of the exploratory nature, most clustering algorithms will create clusters even for data points that have little or no correlation, eventually degrading the quality of ensemble clusters.

DUET overcomes this limitation and improves existing ensemble algorithms by differentiating high-quality clusters that have non-random structures from those that are created due to the artifact of the clustering algorithms. Ideally, when grouping certain type of malware, we should give higher weights to the clustering methods that are known to better at handling that particular type. For instance, we want to rely more on dynamic analysis when clustering heavily-obfuscated malware programs. Unfortunately, such information is not always readily available. Instead, we use cluster-quality as an indirect measure to enable the ensemble method to bias towards high-quality clusters. Our objective is to define quality measures that evaluate the "goodness" of clustering by looking at the intra- and inter-cluster data correlations, and incorporate them in the ensemble algorithms. Intuitively, high-quality clusters should be compact and well separated from other clusters, i.e., data points share a strong bond. We measure cluster quality with following metrics:

- **Cluster Cohesion (C_o)** determines how closely samples in a cluster are related. For a cluster C_i^t, cohesion is calculated as the average distance function between two members: $C_o(C_i^t) = \frac{2}{|C_i^t|*(|C_i^t|-1)} \sum_{x,y \in C_i^t; x<y} d(x,y)$. Note that, smaller C_o indicates better cluster cohesion.
- **Cluster Separation (C_s)** measures how well a cluster is separated from others. The separation between two clusters, C_i^t and C_j^t, is defined as $C_s(C_i^t, C_j^t) = \frac{1}{|C_i^t|*|C_j^t|} \sum_{x \in C_i^t; y \in C_j^t} d(x,y)$. Then, the separation of the cluster C_i^t is defined as the average separation from all other clusters: $C_s(C_i^t) = \frac{1}{k-1} \sum_{j=1; j \neq i}^{k} C_s(C_i^t, C_j^t)$ where k is the number of clusters.

Table 5: Malware families of reference data set

Family	#	Family	#	Family	#
Pilleuz	500	Bredolab	362	Mabezat	129
Virut	500	Vundo	334	Qakbot	44
Silly	500	Almanahe	327	Waledac	41
Fakeav	500	Tidserv	242	Ackantta	36
Koobface	496	Sasfis	219	Mebroot	26
Banker	489	Gammima	206	Hotbar	21
Zbot	486	Graybird	189		

DUET exploits this "goodness" information in the ensemble process to preserve high-quality clusters (i.e., those with small C_o and high C_s) and reduce influence from those with weak connections among their members. To achieve this goal, we define a weighted boost score $\beta = \omega_o(1-C_o)+\omega_s C_s$ ($\omega_o+\omega_s = 1$), with a higher value indicating a better clustering. We use β to augment the connectivity matrix of each member clustering and increase M_{ij} if x_i and x_j are in a high-quality cluster. Specifically, the new boosted connectivity matrix is computed as: $M^B_{i,j}(\mathcal{C}^t) = M_{i,j}(\mathcal{C}^t) \times (1+\beta^t)$ and the cluster ensemble algorithm is applied on the boosted connectivity matrix to derive the final clustering results.

7. EVALUATION

In this section, we first compare the experimental results of DUET-S and DUET-D using real-world malware, verifying their complementary nature. Then, we evaluate the cluster ensemble algorithms and demonstrate their advantages over the single clustering algorithm. By comparing DUET with individual state-of-the-art clustering algorithms used in DUET-S and DUET-D[5], we are able to demonstrate the main point of DUET, i.e. benefits of integrating static and dynamic analysis.

7.1 Malware data set

The malware dataset used in our experiment was offered by a large AV company containing 5,647 malware files, each of which was manually labeled with a malware family name by experts in the company. Table 5 lists the name of each family and number of variants. In order to ensure a meaningful representation of malware's semantics, DUET imposes a threshold constraint, discarding malware programs that have less than 10 extracted N-grams. This is to prevent the use of overly generic features in clustering. Further, when a malware program has too few extracted N-grams, it often implies that the feature extraction process (disassembling or runtime monitoring) failed, making it unsafe to include these malware in clustering. Table 6 lists the number of malware programs that passed the threshold and again demonstrates the benefits of ensemble methods: the combined approaches significantly increases the percentage of *analyzable* malware programs from below 80% to almost 97%.

7.2 Results of Individual Clusterings

Next, we present the results of DUET-S and DUET-D components. Assume each clustering algorithm creates a set of clusters $C = C_1, C_2, \ldots, C_c$. The performance is assessed with three metrics: *precision*, *recall*, and *coverage*. Precision and recall are two standard metrics widely-used for

[5]For instance, DUET-D is built upon the state-of-the-art malware analysis tool Malheur[27]. Also, since ProtoCluster is essentially an scalable version of hierarchical clustering algorithm, DUET-D's perform should be similar as previous work such as [4, 6] and DUET-S is similar to [15].

Table 6: Number of samples with over and under 10 N-grams for Behavior (B) and Static (S) analysis

	# of malware with > 10 N-grams	# of malware with < 10 N-grams
B: 3 gram	4026 (71.29%)	1621 (28.71%)
B: 4 gram	4038 (71.51%)	1609 (28.49%)
S: 3 gram	4622 (81.85%)	1025 (18.15%)
S: 4 gram	4605 (81.55%)	1042 (18.45%)
B+S	5454 (96.58%)	193 (3.41%)

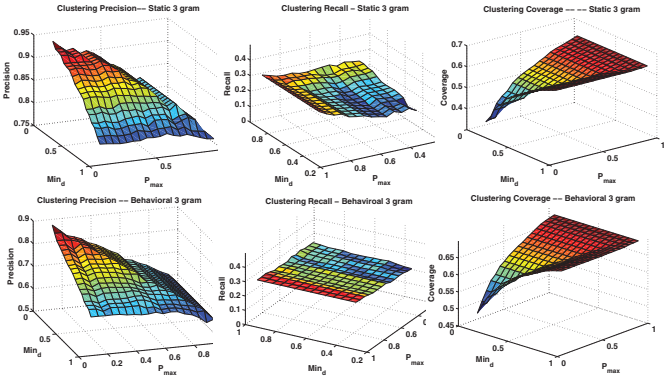

Figure 2: Precision, Recall and Coverage of DUET-S (top) and DUET-D (bottom)

measuring clustering results [15, 20] . Precision assesses the accuracy of clustering in terms of how well the individual clusters agree with the original malware classes. More formally, assume the malware programs are grouped into a set of clusters, $\mathbf{O} = \{O_1, O_2, \ldots, O_o\}$ according to manually created family labels. Then, *precision*, P, is defined as: $P = \frac{1}{n}\sum_{i=1}^{c} max(|C_i \cap O_1|, |C_i \cap O_2|, \ldots, |C_i \cap O_o|)$. On the other hand, *recall* R measures the degree of the malware classes' scattering across the clusters and is defined as: $R = \frac{1}{n}\sum_{j=1}^{o} max(|O_j \cap C_1|, |O_j \cap C_2|, \ldots, |O_i \cap C_c|)$. Precision is 1 if all the samples in every cluster C_i are from the same family and recall is 1 if all malware samples from the same family fall into a single cluster (but not necessarily the only family in this cluster). The third metric, *coverage*, measures the percentage of malware programs that can be successfully clustered after excluding single-member clusters. Coverage is an important metric measuring how well clustering algorithms yield useful clusters.

Fig. 2 plots the performance of DUET-S and DUET-D with two varying parameters of ProtoCluster algorithms—P_{max}, Min_d. From these figures, one can observe that with proper selection of parameters, both algorithms are able to cluster malware samples with precision ranging from 70 to 90%, and static-feature-based clustering generally outperforms behavior-based clustering. We have also observed that the recall values of both DUET-S and DUET-D are around 0.3, which appears to be low. However, this low value of recall is mainly because the malware family labels in our experiments are very generic and within the same family, significant diversity exists across variants. Taking Vundo as an example, the size of the largest Vundo variant in our training data is 9.6M bytes while the smallest variant size is only 13K and the standard deviation is 2.2M bytes. Because of this enormous diversity, clustering algorithms tend to break the original family into several sub-families, e.g., DUET-S and DUET-D often create 100–380 clusters for the reference dataset, resulting in a low recall value. In fact, this result is in line with previous research[6]. Note also that the recall score is much more stable than precision and coverage with little fluctuation across different parameter values, which, to some degree, supports our conjecture that the low recall is caused by input data rather than clustering algorithms per se.

Finally, the coverage results in Fig. 2 show that both clusterings can create useful clusters for only 50–70% of samples. In addition, there exists a natural trade-off between precision and coverage; an increase in precision is accompanied by a decrease in coverage, and vice versa. The reason is that, in order to achieve a higher precision, clustering algorithms must avoid merging unrelated malware programs into the same cluster. While this produces high-quality clusters for some samples, many more remain uncovered in singular clusters. For example, as Min_d and P_{max} both decrease toward 0.1, precision improves because a small P_{max} dictates that each prototype includes only extremely close data points within its range and a small Min_d terminates the merging process early to avoid combining unrelated classes. Together, they ensure that only very similar samples are grouped together, excluding a large portion of samples with moderate resemblance. This leads to a sharp drop in coverage from 75 to 45% for DUET-D and below 30% for DUET-S .

7.3 Evaluation of Cluster Ensemble

we evaluate DUET's cluster ensemble under two scenarios. In each scenario, input to DUET are 8 base clusterings—two clusterings for each combination of 3-gram/4-gram and static/dynamic with different P_{max} and Min_d values. In the first (*best-case*) scenario, P_{max} and Min_d are selected to make each base clustering optimized for precision (thus with low coverage) or coverage (thus with low precision). Note that this is an *ideal* but not *realistic* scenario, since in reality, we would not be able to determine which values of P_{max} and Min_d would achieve optimal precision or coverage. As a result, the best-case scenario is used to evaluate the optimal performance of cluster ensemble under an ideal condition. Our second (*random*) scenario offers a more realistic evaluation by randomly choosing P_{max} and Min_d for the 8 input clusters. Tables 7 lists the parameters of these two scenarios. From the tables, we can see that the precision for the best-case scenario's clusterings ranges from 0.69 to 0.88, while the random scenario has a slightly lower range of 0.57 to 0.83. Most recall values are surrounding 0.3 with a couple of outliers (0.2 and 0.49). The coverage is usually 50–68% for dynamic clusterings and 54–74% for static clusterings. Next, we show that DUET can leverage the diverse perspectives of input clustering to improve final results.

Cluster ensemble based on the ball algorithm: Fig. 3 plots the precision and coverage[7] of this ensemble approach with different threshold β (Section 5.2) between 0.2 to 0.9. From the figure, one can see that the ball algorithm achieves a precision value consistently higher than 0.8 and coverage close to 80%. Using a threshold of 0.5, the precision for the best-case and random scenarios are 0.85 and 0.8, respectively—both very close to the maximum of individual clusterings. Furthermore, the coverages in these two cases

[6]the clustering algorithms in [15] created 200 clusters for 3,935 samples, and the size of each cluster is comparable to DUET.

[7]Due to space limitation and small variation of recall values across different parameters, we will plot only precision and coverage, and present recall using its average and standard deviation.

Table 7: Parameters of best and random scenario

		Min_d	P_{max}	Precision	Coverage	recall
Best Precision	B: 3 gram	0.15	0.1	0.87	51.5%	0.26
	B: 4 gram	0.1	0.15	0.88	51.1%	0.24
	S: 3 gram	0.65	0.2	0.86	57.4%	0.21
	S: 4 gram	0.3	0.7	0.85	56.7%	0.20
Best Coverage	B: 3 gram	0.85	0.45	0.69	67.9%	0.27
	B: 4 gram	0.4	0.95	0.68	68.1%	0.30
	S: 3 gram	1.3	0.3	0.7	74.4%	0.39
	S: 4 gram	1.3	0.75	0.7	71.0%	0.37

	Min_d	P_{max}	Precision	Coverage	recall
B: 3 gram	0.20	0.60	0.71	64.7%	0.27
B: 3 gram	0.30	0.20	0.80	57.9%	0.28
B: 4 gram	0.50	0.20	0.77	62.2%	0.27
B: 4 gram	0.75	0.10	0.71	65.7%	0.27
S: 3 gram	0.60	0.70	0.83	59.5%	0.23
S: 3 gram	1.10	1.25	0.57	72.3%	0.49
S: 4 gram	0.30	1.10	0.74	64.3%	0.33
S: 4 gram	0.85	1.15	0.69	65.7%	0.4

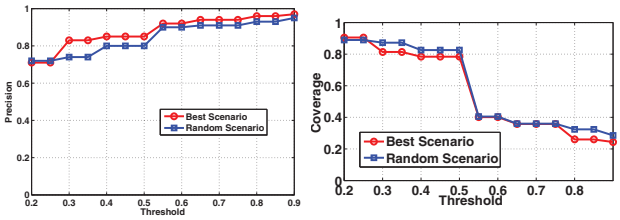

Figure 3: Results of ball algorithm ensemble (Average recall: 0.22, standard deviation: 0.02)

are 0.78 and 0.82, which are 5% and 10% higher than the best coverage for individual clusterings. Also, the ensemble results of random and best-case scenarios are close to each other, indicating that the ensemble's effectiveness is not very sensitive to the choice of its member clusterings. This is a salient property, as it is not always possible to select *a priori* the best individual clusterings.

Cluster ensemble using the agglomerative algorithm The benefit of the agglomerative algorithm is that it starts with the most similar samples and continues with the "best" pair of clusters. It also allows fine-grained control in halting the merging process, such that remaining clusters can be far enough from each other to ensure a clear separation. Fig. 4 plots the results and shows that single linkage is the worst of all in terms of precision, suffering from the same over-merging problem as the single-threshold approach. Average linkage is slightly better than complete linkage, resulting in 0.84 precision and 81.85% coverage for the random scenario, and 0.87 precision and 77.6% coverage for the best-case scenario, all better than the base clustering.

Cluster ensemble based on hypergraph partition We employ the HMETIS [17], a widely-used hypergraph partition algorithm, to find minimum cut in the connectivity matrix. With this approach, DUET achieves as high as 91.2% of coverage, but with very low precision of only 0.72. Detailed investigation reveals that the low precision is attributed to the standard constraint in the hypergraph partitioning algorithm—attempting to avoid trivial partitions by making clusters comparably sized. However, in practice, the size of malware families are very unbalanced (Table 5). As the hypergraph partitioning balances the size of resulting

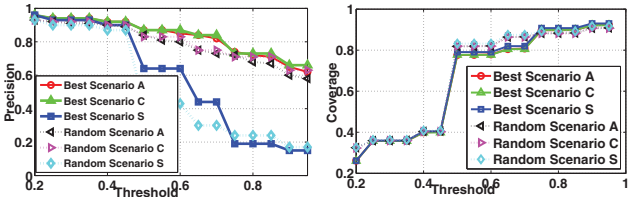

Figure 4: Results of agglomerative algorithm ensemble. A, C, S represent Average, Complete, and Single linkage distance metrics in agglomerative algorithm, i.e.,. Avg recall: 0.29, stdev: 0.07)

clusters, it creates groups containing samples from *multiple* small families, resulting in a lower precision value.

Summary In sum, individual clusterings often have to make a trade-off between precision and coverage, achieving a high value for the one at the expense of the other. By contrast, the cluster ensemble is able to leverage information from multiple clusterings and improve both metrics simultaneously. The "Avg. Improvement" column in Table 8 lists the average improvements of cluster ensemble over all individual clusterings. The table shows that, except for the hypergraph-based approach, incorporating ensemble algorithms allows DUET to improve precision by 5–10% and coverage by 20–40% over individual clusterings.

Table 8: Summary of cluster ensemble's results and improvements over individual clusterings

	Best Scenario			
Ensemble Approach	Precision	Avg. Improvement	Coverage	Avg. Improvement
Ball	0.85	9.24%	78.38%	25.88%
Agglomerative	0.87	11.81%	77.60%	24.63%
Hypergraph	0.72	-7.47%	91.20%	46.48%
Non-Ensemble	0.78	N/A	62.26%	N/A
	Random Scnario			
Ensemble Approach	Precision	Avg. Improvement	Coverage	Avg. Improvement
Ball	0.8	9.97%	82.56%	28.94%
Agglomerative	0.84	15.46%	81.85%	27.83%
Hypergraph	0.71	-2.41%	89.90%	40.40%
Non-Ensemble	0.7275	N/A	64.03%	N/A

7.4 Improvement with Cluster-Quality Metrics

We first examine whether cohesion and separation are effective measures of cluster quality. We take four best-coverage cases, one from each clustering (see Table 7), and compute cohesion and separation for each constituent cluster. We separate "clean" clusters (i.e., those consisting of malware from only one family) from "mixed" clusters (i.e., those consisting of malware samples from multiple families), and plot their CDFs in Figs. 5. These figures show that C_o and C_s work fairly well in distinguishing between good and bad clusters, with clean clusters often having a smaller cohesion[8] and a higher separation than mixed clusters.

Next, we integrate the cluster-quality measures into the DUET according to Section 6 and apply same ensemble algorithms on the augmented connectivity matrices for both best-case and random scenarios (Table 7). Fig. 6 compare the precision and coverage results of the agglomerative ensemble algorithm with and without quality measures. The results for other algorithms share the same trends and are

[8]a smaller cohesion means better (Section 6) cluster quality.

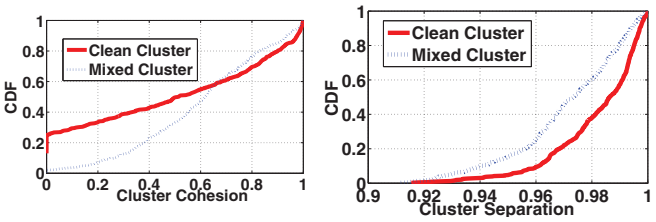

Figure 5: CDF for Cohesion C_o and Separation C_s

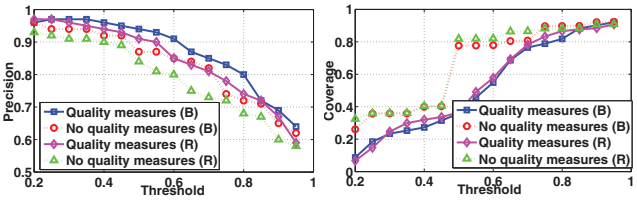

Figure 6: Agglomerative ensemble algorithm with quality measures. (B) and (R) represent the best and random scenario. Avg recall: 0.27; stdev: 0.07

omitted due to space limit. From the figure, we can observe that ensemble algorithms with quality-measures outperform their original counterpart, by 5–10% in terms of precision. However, Fig. 6 also shows that the improvement of precision does not come without cost. We observe a decrease in the malware coverage after incorporating the quality-measures.[9] This is because employing the quality-measures weakens connectivity between samples in low-quality clusters, making them more likely to be excluded from final clusters. In all the plots, the coverage is shown to be reduced by 3–30%, with the biggest drop often occurring when the threshold is around 0.5. This is due to the composition of member clusterings (i.e., a half from dynamic approaches and a half from static approaches). Without quality-measures, a threshold of 0.5 (i.e., majority voting) can result in a false consensus among approaches of the same type. Fortunately, by using the quality-measures, the reduced connectivity between weakly-related samples drop below the threshold, improving precision but lowering the sample coverage (Fig. 6). However, the decrease in coverage is not necessarily a disadvantage, as the remaining clusters are often of better quality, allowing higher confidence when using these clusters. Incorporating diversified sets of clustering techniques could potentially mitigate this problem.

7.5 Run Time Performance of DUET

DUET consists of two main components: trace collector and clustering system. The most time-consuming step is the trace collection. For all 5,647 malware programs, it took approximately 12 hours to extract static features and 7 days to collect dynamic traces on a single machine with an Intel Core i7 3.0G CPU and 16 GB of RAM. The trace-collector step, albeit expensive, (1) is required for any malware analysis tasks and has already been efficiently conducted as a daily process in AV industry, (2) only needs to be done once and is also amenable to parallelization thanks to the independence between malware samples. For instance, Anubis, an online service for executing and dynamically analyzing

[9]Astute readers may notice that the same trade-off can be obtained by changing the parameters, e.g., P_{max} and Min_d. However, the trade-off made here provides additional benefits of enhancing the cluster qualities.

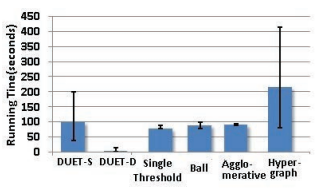

Figure 7: Running times of different components

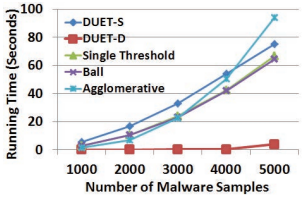

Figure 8: Scalability of DUET's components

malware, can process more than 1 million samples on the fly, as they were submitted during one week period of time [21]. Hence, scalability is becoming a less concern for dynamic malware analysis, making its integration with static analysis very practical and beneficial.

The second component of DUET is the clustering system including individual static/dynamic clusterings and cluster ensemble. Their running times on the entire data set is summarized in Fig. 7. We observe that the clustering runs much faster for dynamic features than static features. This is because the space of static features is orders-of-magnitude larger than that of dynamic features. As a result, DUET-S takes longer to compare similarity between longer static feature vectors (which also help DUET-S achieve better precision than DUET-D as shown in Section 7.2). The running times of different ensemble algorithms are usually in the range of 70–90 seconds except for the hypergraph partitioning being the most complicated and time-consuming (about 200s). To better understand the scalability, we measure the running time of DUET with different number of input malware samples, as shown in Fig. 8. The figure confirms the runtime complexity of DUET's ensemble component is $O(n^2)$ where n is the number of malware samples. Because of this quadratic complexity, DUET's performance may suffer when the number of malware samples grows significantly. A potential solution is to employ the principle of the *ProtoCluster*. The ensemble algorithms can first be applied to a relatively small set of prototypes and then propagate to associated samples, avoiding expensive computation over entire dataset.

8. DISCUSSIONS

Here we discuss several limitations of the current DUET prototype, and possible improvements to alleviate them. DUET's performance hinges on the successful extraction of useful features from malware binaries and run-time behaviors. First, like any other static-analysis approach, DUET-S is vulnerable to binary/instruction-level obfuscation and advanced run-time packers. For instance, anti-disassembly techniques, such as mixture of code and data, and indirect control flow, can be used to confuse disassemblers, thus preventing DUET-S from extracting features. Instruction-level polymorphism can be used to create syntactically distinct but semantically similar variants, bypassing DUET's similarity comparison, Although the current DUET prototype does not handle these types of obfuscation for simplicity, advanced de-obfuscation and normalization [22, 30] can be used to mitigate the problems. For the packers than cannot be handled by generic unpacking algorithm e.g. in section 4.1, specialized unpacking tools such as Armadillo Killer [1] can be employed.

Similarly, DUET-D as a dynamic analysis system can be circumvented by specifically crafted evasion techniques. First, since any dynamic analysis system typically can only afford

to execute a malware program for a small period of time, it can be circumvented by inserting stalling codes before the real malicious codes. Systems such as HASEN [18] have been proposed to detect and automatically skip such stalling code. Another limitation of DUET-D is its reliance on virtual machines to provide a controlled environment, for malware execution. Unfortunately, malware programs can check for virtual environments [25] and behave differently from what they do when running in a real system. Countermeasures include use of more transparent environments [8, 9], detecting split identities of malware samples [5] and forcing multiple path exploration [26]. Finally, both DUET-S and DUET-D cannot handle file infectors or parasitic malware which injects itself into host binaries (e.g. Sality virus), because majority of extracted features may belong to the host binaries. In practice, techniques for detecting parasitic malware [29] should be used to pre-filter these samples.

Malware analysis has always been covering an active battle field between adversaries and defenders. None of the aforementioned countermeasures are perfect or long-lasting. Nevertheless, new emerging techniques can be leveraged to raise the bar and handle common malware types, which, in practice, prove to be beneficial. Furthermore, the respective limitations of static and dynamic analysis again strengthen the necessity and importance for integrating these two approaches to deal with advanced malware samples.

9. CONCLUDING REMARKS

In this paper, we design, develop and evaluate an automatic malware-clustering system, called DUET, which exploits cluster ensemble as a principled method to effectively integrate static and dynamic analyses. Using a number of real-life malware samples, we have evaluated the performance of ensemble methods for both best-case and random scenarios, demonstrating that DUET can improve coverage by 20–40% while achieving nearly the highest precision of the individual clustering algorithms. We have also made further improvements of existing cluster ensemble algorithms by leveraging cluster-quality measures. Overall, the evaluation results have shown DUET's ability of combining multiple malware clustering techniques to create more effective and accurate clusters.

10. REFERENCES

[1] Unpackers. http://www.exetools.com/unpackers.htm.

[2] C. C. Alexander Strehl, Joydeep Ghosh. Cluster ensembles - a knowledge reuse framework for combining multiple partitions. *Journal of Machine Learning Research*, 2002.

[3] B. Anderson, C. Storlie, and T. Lane. Improving malware classification: bridging the static/dynamic gap. In *Proceedings of the 5th ACM workshop on Security and artificial intelligence*, AISec '12, 2012.

[4] M. Bailey, J. Andersen, Z. M. mao, and F. Jahanian. Automated classification and analysis of internet malware. In *In Proceedings of RAID*, 2007.

[5] D. Balzarotti, M. Cova, C. Karlberger, C. Kruegel, E. Kirda, and G. Vigna. Efficient detection of split personalities in malware. In *NDSS*, 2010.

[6] U. Bayer, P. Comparetti, C. Hlauschek, C. Kruegel, and E. Kirda. Scalable, behavior-based malware clustering. In *Proc. of the 16th NDSS*, 2009.

[7] M. Christodorescu and S. Jha. Static analysis of executables to detect malicious patterns. In *Proc. of the 12th USENIX Security Symposium*, 2003.

[8] A. Dinaburg, P. Royal, M. Sharif, and W. Lee. Ether: malware analysis via hardware virtualization extensions. In *Proceedings of CCS'08*, 2008.

[9] A. Fattori, R. Paleari, L. Martignoni, and M. Monga. Dynamic and transparent analysis of commodity production systems. In *Proceedings of ASE'10*, 2010.

[10] A. Fred. Finding consistent clusters in data partitions. In *Proc. 3d Int. Workshop on Multiple Classifier*, 2001.

[11] A. L. N. Fred and A. K. Jain. Data clustering using evidence accumulation. In *Proceedings of the 16 th International Conf. on Pattern Recognition*, 2002.

[12] A. Gionis, H. Mannila, and P. Tsaparas. Clustering aggregation. *ACM Trans. Knowl. Discov. Data*, 1, March 2007.

[13] F. Guo, P. Ferrie, and T.-C. Chiueh. A study of the packer problem and its solutions. In *RAID '08*, 2008.

[14] Y. Hong, S. Kwong, Y. Chang, and Q. Ren. Unsupervised feature selection using clustering ensembles and population based incremental learning algorithm. *Pattern Recognition*, 41(9):2742–2756, 2008.

[15] J. Jang, D. Brumley, and S. Venkataraman. Bitshred: feature hashing malware for scalable triage and semantic analysis. In *Proceedings of CCS'11*, 2011.

[16] M. E. Karim, A. Walenstein, A. Lakhotia, and L. Parida. Malware phylogeny generation using permutations of code. *J. in Computer Virology*, 2005.

[17] Karypis Lab. Hypergraph partitioning software. http://glaros.dtc.umn.edu/gkhome/views/metis, 2010.

[18] C. Kolbitsch, E. Kirda, and C. Kruegel. The power of procrastination: detection and mitigation of execution-stalling malicious code. CCS '11, 2011.

[19] J. Z. Kolter and M. A. Maloof. Learning to detect and classify malicious executables in the wild. *Journal of Machine Learning Research*, 7:2006, 2006.

[20] C. W. Konrad Rieck, Philipp Trinius and T. Holz. Automatic analysis of malware behavior using machine learning. Technical report, 2011.

[21] C. Kruegel, E. Kirda, U. Bayer, D. Balzarotti, and I. Habibi. A view on current malware behaviors. In *2nd USENIX Workshop on LEET*, 2009.

[22] C. Kruegel, W. Robertson, F. Valeur, and G. Vigna. Static disassembly of obfuscated binaries. In *USENIX Security'04*, 2004.

[23] T. Lee and J. J.Mody. Behavioral classification. In *Proceedings of EICAR Conference*, 2006.

[24] C. Leita, U. Bayer, and E. Kirda. Exploiting diverse observation perspectives to get insights on the malware landscape. In *DSN*, 1 2010.

[25] T. Liston. On the cutting edge: Thwarting virtual machine detection http://handlers.sans.org. /tliston/ThwartingVMDetection_Liston_Skoudis.pdf.

[26] A. Moser, C. Kruegel, and E. Kirda. Exploring multiple execution paths for malware analysis. In *Proceedings of Oakland'07*, 2007.

[27] K. Rieck, T. Holz, C. Willems, P. Düssel, and P. Laskov. Learning and classification of malware behavior. In *Proceedings of DIMVA'08*, 2008.

[28] P. Security. Bredolab.aw. http://www.pandasecurity.com/homeusers/security-info/220087/Bredolab.AW.

[29] A. Srivastava and J. Giffin. Raid. In *Recent Advances in Intrusion Detection*. 2010.

[30] S. K. Udupa, S. K. Debray, and M. Madou. Deobfuscation: Reverse engineering obfuscated code. *Reverse Engineering, Working Conference on*, 2005.

[31] G. Wicherski. pehash: A novel approach to fast malware clustering. In *LEET'2009*.

[32] Y. Ye, T. Li, Y. Chen, and Q. Jiang. Automatic malware categorization using cluster ensemble. In *Proceedings of the 16th ACM SIGKDD*, 2010.