IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received June 28, 2017, accepted August 17, 2017, date of publication September 7, 2017, date of current version September 27, 2017.

Digital Object Identifier 10.1109/ACCESS.2017.2749538

The Effects of Traditional Anti-Virus Labels on Malware Detection Using Dynamic Runtime Opcodes

DOMHNALL CARLIN[1], (Graduate Student Member, IEEE), ALEXANDRA COWAN[2], PHILIP O'KANE[1], AND SAKIR SEZER[1]

[1]Centre for Secure Information Technologies, Queen's University Belfast, Belfast BT7 1NN, U.K.
[2]Institute of Electronics, Communications and Information Technology, Queen's University Belfast, Belfast BT7 1NN, U.K.

Corresponding author: Domhnall Carlin (dcarlin05@qub.ac.uk)

This work was supported by EPSRC under Grant CSIT 2 EP/N508664/1.

ABSTRACT The arms race between the distributors of malware and those seeking to provide defenses has so far favored the former. Signature detection methods have been unable to cope with the onslaught of new binaries aided by rapidly developing obfuscation techniques. Recent research has focused on the analysis of low-level opcodes, both static and dynamic, as a way to detect malware. Although sometimes successful at detecting malware, static analysis still fails to unravel obfuscated code, whereas dynamic analysis can allow researchers to investigate the revealed code at runtime. Research in the field has been limited by the underpinning data sets; old and inadequately sampled malware can lessen the extrapolation potential of such data sets. The main contribution of this paper is the creation of a new parsed runtime trace data set of over 100 000 labeled samples, which will address these shortcomings, and we offer the data set itself for use by the wider research community. This data set underpins the examination of the run traces using classifiers on count-based and sequence-based data. We find that malware detection rates are lessened when samples are labeled with traditional anti-virus (AV) labels. Neither count-based nor sequence-based algorithms can sufficiently distinguish between AV label classes. Detection increases when malware is re-classed with labels yielded from unsupervised learning. With sequenced-based learning, detection exceeds that of labeling as simply ''malware'' alone. This approach may yield future work, where the triaging of malware can be more effective.

INDEX TERMS Network security, machine learning, computer security.

I. INTRODUCTION

From the first PC virus, titled Brain, in 1986 to recent highly obfuscated ransomware, the malware pandemic has demonstrated alarming expedition. By mid-2017, the number of *new* malware samples had breached the 30 million mark in Q1 alone; the total quantity of malware samples held by McAfee Labs has broken the 650 million mark [1]. The unending evolution and distribution of advanced malware has created a major challenge to end users, from industry to the private citizen. Previously seen as merely a bragging right for marginalized youths, malware has evolved into a major tool for the extortion of invididuals, businesses and the public sector.

The main aim of the on-going fight against malware is to be able to detect, prevent and mitigate such disruptive software effectively. With recent advances in malware sophistication, current signature-based and vulnerability-monitoring malware detection methods have been demonstrated to be increasingly ineffective in detecting malicious code [2].

Signature-detection methods are the most widely utilized approach within commercial anti-virus (AV) software [3]. These methods rely on statically analyzing an entire file for pre-known malware signatures stored in databases. As such, any new malware instance must be captured, analysed for a signature, stored, and any updates to anti-malware software deployed globally as counter-measures to the new threat. With the rapid development of malware, the production of relevant and up-to-date anti-malware measures in a timely fashion is an increasingly difficult task. Furthermore, with increasingly sophisticated obfuscation techniques, the same piece of malware can employ polymorphic and metamorphic strategies to generate new signatures on each run, thus evading detection. These weaknesses in signature-detection methods are particularly apparent with Zero-Day attacks.

Such attacks exploit unknown or unresolved vulnerabilities, which must be retrospectively patched. The time-lag between the release of the malware 'in the wild' (Zero Day) and the counter-measures being developed, allows a window for the malware to cause substantial damage. The effort against malware is therefore constantly playing catch-up against the creation of new forms of malicious code and the resulting damage caused in the interim period can be significant [4].

The motivation for the present research is to develop a strategy, immune to modern obfuscation methods, for the detection of malware. The approach proposed in the present research uses virtualized dynamic analysis to yield program run-time traces of both benign and malicious files. Machine learning techniques are applied to the execution traces, with the aim being to create a predictive model which can accurately and speedily discriminate between unseen samples of malicious and benign code.

In section II, related research is presented and the created dataset is placed in the context of other datasets. In section III, the rationale for the present research is explained and the scope is defined. In section IV, the methodology used to create the new dataset from beginning to end is outlined. Section V describes the dataset to date. Section VI presents the clustering used, with count-based and sequence-based classification. Section VII discusses the results with respect to the effects of AV labels on classification. Finally, section VIII summarizes the paper's contributions.

II. RELATED RESEARCH

Obfuscation has presented challenges for malware detection, both for anti-virus (AV) vendors and researchers. Malware research has previously been frustrated by analysis evasion strategies increasingly employed by malware authors. Several authors have researched the use of machine learning algorithms for classification of malware from unseen datasets. Reference [5] presented the first such study. Three classifiers were employed and compared to a signature-based commercial AV-scanner, with features such as *program header*, *string features* and *byte sequence features* examined. Two of the classifiers doubled the accuracy of the scanner, and all three were more accurate. In [6], boolean n-gram analysis was applied to the hex representation of malware executables. A boosted decision tree classifier gave excellent accuracy, with an area under receiver operating characteristic (ROC) curve of 0.996. Moskovitch *et al.* [7] collected data on 323 features per second across 5 unseen worm variants. Using only 20 features, the authors demonstrated an average detection rate of >90% accuracy, with specific worms exceeding 99%.

A. OPCODE ANALYSIS

Recent investigations into malware detection have focused on the run-time behaviors of malware and how these differ from the benign behavior of goodware. Operational codes (opcodes) are assembly language instructions which perform various CPU operations [8]. Analyzing the host environment's native run-time opcodes, allows the user to detect the presence of malware while circumventing obfuscation tactics [2].

Reference [9] examined the n-gram representations of opcode traces from statically-yielded datasets. With ROC-SVM(support vector machine), the authors achieved accuracy and an F-measure rate of >85%, though only bi-grams (n=2) were investigated.

Reference [10] applied graph algorithms to opcode traces from Windows Portable Executable (PE) files, extending [11] with metamorphic malware. Using similarity scores, the model detected metamorphic malware from both benignware and also metamorphic malware from different families within the same malware type.

The work of [2] created a process to investigate dynamically-yielded opcodes in the classification of malware using supervised machine learning techniques. Run-time opcode traces of both goodware and malware files were captured using a virtual machine (VM). The virtualization platform enabled the researchers to execute malicious files in a contained environment.

B. HIDDEN MARKOV MODELS IN MALWARE DETECTION

The research in [11] used dynamically-yielded runtime traces modeled by Markov chains and represented by weighted directed graphs, which showed the probabilities of one opcode being followed by another. The main research aim of the authors was to demonstrate that their method could perform better than n-gram representations and signature-based detection algorithms. The dataset was composed of 1615 malware samples and 615 benign samples, however no detail is given on the make-up of the dataset in terms of malware type, family, age, obfuscation or size, which severely limits the interpretation of the analyses provided. The Markov model out-performed the others with an accuracy of 96.41%, 47 FPs and 33 FNs. However, with the dataset skewed in the positive class direction, the authors note that the FN rating may be an artifact of the sampling; this underlines the data collection issue.

Reference [12] sought to find weaknesses in HMM-based detection tools which could be exploited by metamorphic malware. The authors developed a kit for metamorphic virus creation and were able to use the subsequently derived malware samples to evade detection by HMM algorithms. The authors note that high levels of metamorphism to render the malcode different to similar families of malware was not sufficient to evade detection. Rather, altering the code to resemble benignware by injection subroutines from normal files caused a rise in misdetection.

Reference [13] used Profile HMMs (PHMM) to investigate metamorphic malware which uses code slicing with jumps as an obfuscation tactic. A reduced instruction set of 36 opcodes plus a wild card for all other opcodes was used to detect malware created using three separate metamorphic kits. Sample sizes were low (10, 30 and 200) and a large error rate was found for the largest class, rendering the solution impractical.

Reference [3] compared traditional HMMs with PHMMs on the frequency of the 36 most common API calls, using both static and dynamic analysis. Dynamic analysis outperformed static on all but one of the 7 malware types examined, using both regular HMM and PHMM (which was only dynamically attained).

Reference [14] trained HMMs to identify which of three families to which 1500 malicious samples belonged. Their models were successful on 88.4%-90.86% of the samples, exceeding 2 major AV products and the work of [15]. However, this work did not attempt to detect malware from benignware, but merely confirmed the accepted family label. This family label was assigned by [16] and so the models of [14] rely on this for accuracy.

C. CLUSTERING IN MALWARE DETECTION

Reference [17] trained HMMs on statically-yielded opcode sequences for five compilers and two metamorphic generators. Each of 8119 malware samples was scored against each model and *K*-means clustering was applied to the 7-tuple. The authors found that clusters formed from the scores of HMMs untrained on the families under investigation, were able to accurately classify unseen malware samples into their correct families 84% of the time. Reference [18] used partitional clustering algorithms, including k-medoids and Density-Based Spatial Clustering of Applications with Noise (DBSCAN), to group malware instances with similar statically analyzed call graphs. Using this approach, the authors were able to correctly assign data sets of 194, 675 and 1,050 distinct malware samples to one of 24 malware families which had been manually designated by an AV company. Reference [19] performed large-scale dynamic analysis of 270,000 malware samples using 351 features. These were derived from three behavioral feature-types: *Passed API Calls, Failed API Calls, and Return Codes*, reduced to 32 features using feature extraction. The subsequent dataset was clustered using the unsupervised Self Organizing Map (SOM) algorithm and clusters were evaluated with respect to the number of AV-provided types and the correct classification rate. A 2x2 4-cluster SOM model provided the lowest level of misclassified instances at around 20%. The authors note, however, that classification on malware behavior should avoid using AV labels as the information provided does not match the actual behavior of the malware.

D. PREVIOUS DATASETS IN CONTEXT

The dataset of [2] is comprised of 300 benign and 350 malicious PE files. Although greater than previous research, for example [20], the size of that dataset is comparatively inadequate when other similar research is considered.

Reference [5] sampled 3265 malicious files and 1001 benign files in the first such study in the current branch of research. Reference [21] used a dataset of 2000 files, split evenly between classes. References [22] and [23] compared 7688 malicious files with 22735 benign, reported by the authors to be the largest dataset in the field at the time of writing. These datasets enable a greater range of comparisons to be made along a wider variety of variables; however there are limitations with the manner in which the data in [23] were collected. The authors report that the software could only actively disassemble 74 % of their source files. The attrition rate for malware was approximately 26% (2011 files) and 10%(2319 files) for benignware. No analysis is provided into the remaining 26%, except for a statement that the files excluded were either compressed or packed. The implication is that more malware was unable to be disassembled by the researchers than benign software. This, again, could well have introduced a bias into the dataset, as it may be that the analyses used by the authors could not be replicated on more sophisticated obfuscated malware.

The dataset used by [9] comprised 2000 files, due to unspecified technical limitations. The malicious files were randomly sampled from 17000 sample files archived on the VxHeaven website [24]. Critically, packed files were not used during the analysis, which was static in nature. The implication is that more sophisticated obfuscated malware was not examined by the process, a very real limitation of the study. Over 50% of the sample set was made up of three classes of malware: back-door, hack-tool and email worm, which may have artificially induced within-class imbalances.

Reference [25] used 6721 malware samples harvested from VxHeaven [24] and categorized into 3 types (backdoor, trojan, worm), with a total of 26 malware families and >100 variants per family. No controls were in place to ensure matched sampling across categories, resulting in 497 worm variants, 3048 backdoor variants and 3176 trojan variants. The dataset generated by [2] has the benefit of being dynamically generated from the source PE, yielding a statically analyzable raw file. The run-times were fixed, which controls for opcode quantity being passed to the machine learning component.

III. RATIONALE FOR THE CURRENT RESEARCH

It is an aim of the current project to create a dataset using the same dynamic approach as [2]. The dataset generation process is envisaged as open-ended as, with the proliferation of new malware, any detection models must include recent variant samples of zeitgeist threats. Reference [23] categorized their dataset into year of creation, from 2000-2007 to investigate the effects on a machine learning algorithm of training on historical samples, but testing on recent samples. The authors found a decline in performance in every test of a historically-trained model with a more recent test set, by as much as $>$ 30 % in as little as one year variation. Although prescience may not be entirely possible from a machine learning perspective, the development of a malware detection model which cannot detect new or emerging threats would, therefore, be redundant from the offset.

Previous research typically uses samples harvested from the VxHeaven website [24], which are old and outdated,

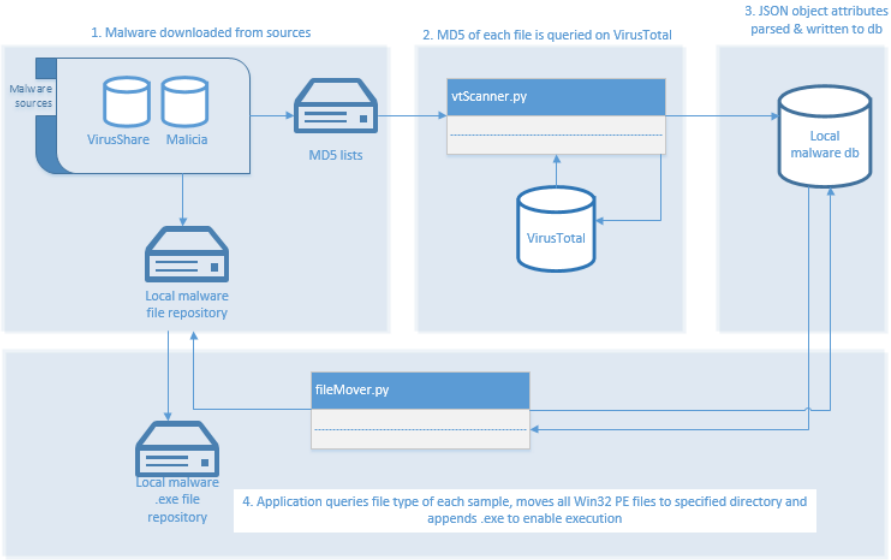

FIGURE 1. Processing malware samples to store their descriptive attributes.

having been last updated in 2010. The new dataset should be comprised of samples sourced from the most up-to-date repository possible, including any emerging threat families of malware. The new dataset should be capable of subdivision along a number of variables for example *type, family, variant, file size, runtime, payload, attack vector, creation, obfuscation-type*, while retaining sufficient quantities per sub-category. This will allow a large degree of sub-analyses in order to investigate methods to increase detection rates. An initial categorization-by-type approach to organizing the database will be necessary, although this will present challenges of its own, as discussed further below. This proposed dataset would address the limitations identified in previous literature as already stated.

From the body of research in this field, there are clear and notable limitations. Of the studies reviewed, the majority had datasets which were inadequately sized, sampled, and structured. Statically generated datasets failed to bypass any serious form of obfuscation, with some just disregarding any packed executable. Although dynamic analysis is not without flaw, it at least offers a snapshot of the malware at runtime. Few attempts were made to properly structure data with both static and dynamic analyses providing insufficient instances across categories. Malware types have different methods of action, which is what defines their type. Each has different instruction sets, yet little attempt has been made to investigate malware per type, particularly in a dynamic setting. Discrimination is tested with binary results, without regard for the differing make-up of each type of malware. We believe this will give greater scope to applying machine learning and data mining techniques by covering a more representative sample of malware. In this paper, we examine the effectiveness of traditional malware labels as provided by 54 AV scanners.

A. SCOPE

The main overall challenges to be addressed in this paper are:

- Configuration of a host environment, capable of automated dataset creation which captures the dynamic runtime of both benign and malicious software.
- Generation of a dataset sufficient in size and depth to allow sub-analyses along specific parameters, which will be made available
- Explore and improve on the data mining and machine learning techniques already in use to extract meaningful information from the dataset, in order to increase detection accuracy and minimize false-negative decisions.
- Explore malware types and the advantages of investigating per malware type.
- Explore unsupervised learning techniques to allow malware to self-organize along their opcode representations.

This paper will present the methodology used to create a dataset of sufficient quality to enable these aims, and the analyses of the effectiveness of classification per type.

IV. METHODOLOGY

A. SOURCE DATA

Many studies in the area have used malware repositories such as VxHeaven.org [24]. The malware collection in the VxHeaven repository is now old in malware terms, having not been updated since 2010. The repository at VirusShare.com [26] was selected for its size, modernity and facilities. For the present paper, the most recent 3 folders, at the time of writing, from the corpus of the repository were obtained, representing approximately 180,000 samples of malware. The dataset from [16] was also obtained, with 11,688 malware binaries collected in the wild. As all samples are named by MD5 hash with no other metadata, a bespoke file attribution system was engineered to create a database of the metadata of each sample. Fig. 1 depicts the

working system employed. An API key was obtained for the VirusTotal.com back-end, allowing rapid automated MD5 hash searches to be performed, causing the files to be checked against 54 AV scanners, along with domain scanners and file characterizers. Using the report from the analytics engine, files were categorized, creating a database of all salient attributes for each file. For the present research, PE files were extracted and processed, yielding almost 90,000 instances of executables.

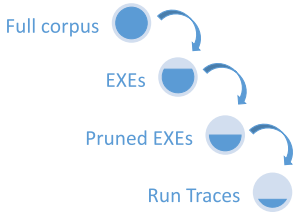

FIGURE 2. Depletion of sample quantities during work flow.

When building the database of attributed malware, a wide variation existed in the detection of each binary by AV scanners. If a sample of malware was detected by a few AV scanners, it could be either a false positive detection, or a sample of malware which can successfully evade detection by scanners. To select the malware files to be executed, a majority-rules algorithm was used. Malware detected by 50% or more of scanners was selected for execution. While this may be a point of discussion in our methodology, the establishment of a baseline corpus of malware is one main aim of this research; established instances of malware are weighted accordingly. The data flow, and sample depletion, is depicted figuratively in fig. 2.

In the next phase, the attribution system used the AV scanner reports to give an initial label to each malware sample. A majority-decision algorithm was employed, with the most common label being selected and applied to each sample; 15 AV classes were selected. Work carried out in parallel by [27] follows a very similar approach, though focused on malware family. Further, that work was published after our attribution work had completed, so their system was not available to us.

B. PROCESS AUTOMATION

The process for extracting runtime opcodes from dynamically executed software, as established in [2] and [8], is manually configured, implemented and operated, which limits the potential to create large scale datasets, due to the slow and laborious nature. The process developed for the present work, as also used in [28], is now described.

Consideration was given to the popular Cuckoo sandbox framework. While an asset to the malware research community as a whole, we wanted a system which provided exactly the desired output using an established procedure. Further, a long term goal of our research is to develop an opcode-focused malware detection framework. As such, our own analysis environment was engineered.

VirtualBox was employed as the virtualization platform for the present research. VirtualBox exposes a rich API [29], one of the features for which VirtualBox was selected over other virtualization platforms. The platform is freely available, has a large open-source community and provides regular updates. The VirtualBox environment is composed of multiple layers built upon the core hypervisor, which operates at the kernel level. It is responsible for the actual execution and control of the VMs, ensuring isolation from the host and any additional VMs. Next in the stack resides a layer of extra features, including resource monitors and RDP services. The API resides on the subsequent layer, exposing a full-feature implementation of the layers below. This API can be accessed through the application GUI, through the command line or programmatically. As the API which exposes the full feature set of the hypervisor is used in the standard public-facing GUI, it is well-documented and well-tested [30].

A clean VM image was created with Windows 10 64-bit, 2GB RAM, VT-x acceleration, using 2 cores of an Intel i5 5300 CPU, and the VBoxGuestAdditions v4.3.30 add-on installed. Windows 10 was employed, as the OS is reported as having a similar market share to Windows 7 in some sectors and is growing. VBoxGuestAdditions allows extra features to be used by the VM, including guest application execution permission from the host, read-only shared folders and time synchronization. Full internet access was granted to the guest and the packet traffic was captured for parallel research, but monitored as a further check for the liveness of the executed malware. Security within the guest OS was minimized to maximize the potential effects of the malware. To yield the runtime trace of the file-under-investigation(FUI), Ollydbg v2 was employed as a debugger as per [2]. Ollydbg is an open-architecture assembler-level debugger, capable of direct loading and debugging of PE and DLL files. The use of debugging tools can, however, be detected by the FUI and anti-debugging tactics deployed. This is true not just for malware, but for obfuscating legitimate code too. StrongOD v0.4.8.892 was used to mask the presence of the debugger, as per [2]. The guest OS was crafted to mimic a real system, with objects in the recycling bin, a browsing history, a recent document history, Flash, Java, .Net Framework etc. Although tactics to mitigate anti-virtualization techniques were originally investigated, they were not fully utilized in the execution stage. The presence of VBoxGuestAdditions is one such indicator of being in a virtual environment, but it is required to provide guest program execution and the automation process. Further mitigation attempts would still falter at this point. Furthermore, the aim of the dynamic investigation of malware is to experience the execution of the malware as a user would potentially experience it, including in virtualized instances. Lastly, the anti-virtualization tactics and strategies of malware can also provide useful features for detection. When examining executables at the opcode level, these can be observed.

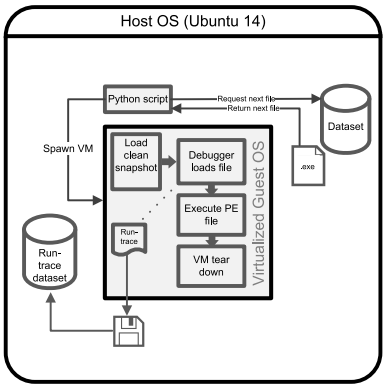

FIGURE 3. Automated data collection system model.

As shown in fig.3, the system automates processing of the executables and extraction of runtraces. The guest VM is spun-up from a pre-configured image set to the required state, in order to reset any impact from previously executed malware. The debugger performs a trace of the FUI, recording opcode instructions issued to the CPU. It is a key point to note that this is a major difference from statically analyzed code, as only the actions which are actually performed are recorded, rather than all possible functions. The runtime trace files are saved in a standard format and the VM is torn down and the trace file is archived on the host. The process then repeats, with a clean snapshot and the next FUI from the folder, until it has been exhausted. The runtime trace files yield a standard file as depicted in fig. 4.

```
main <ModuleEntryPoint>  PUSH EBP                          ESP=0019FF80
main 004024E2            MOV EBP,ESP                       EBP=0019FF80
main 004024E4            LEA ESP,[EBP-64]                  ESP=0019FF1C
main 004024E7            PUSH 0E                           ESP=0019FF18
main 004024E9            PUSH OFFSET 00405376              ESP=0019FF14
main 004024EE            PUSH OFFSET 00405365              ESP=0019FF10
main 004024F3            CALL DWORD PTR DS:[<&KERNEL32.GetLongPa
```

FIGURE 4. Sample lines from a runtime trace file.

A nine minute execution time was allowed for the present work, with a one minute for set-up/tear-down. This window was allocated to provide enough time for the samples to execute, but allowing coverage over a large number of samples with limited time and resources. A longer execution time may give better results when investigating malware which employs sleep functions in order to avoid detection. Other frameworks allow the stepping-over of such sleep functions, but are negated by analysis of time-stamps and mouse movements. Similarly to anti-virtualization issues, it would not be feasible to create a virtualised system mitigating all such variables. A fundamental aim of the present research is to detect malware within as short an execution time as possible, and the dataset generation is therefore concentrated on the initial stages of execution.

C. DATA POST-PROCESSING

The runtime trace files must be parsed for extraction of the individual opcode instructions, prior to any analysis being performed. While tools are available to yield opcodes from PE files, these are static in nature and do not examine traces from executed applications. A bespoke parser was implemented to store a running count for all opcodes from a pre-known list. The preconfigured list of opcodes was, for the first count phase, the 610 opcodes from the Intel x86/x64 architecture [31]. This approach is more fine-grained than [2], as prior work [20] and [25] has indicated that observing rarely occurring opcodes may increase malware detection. The parser exports this file as csv, ready for use in any external processing.

As the raw runtime trace files will vary in length depending on the FUI, it is necessary to standardize the instances into set run lengths as defined by the opcode quantities. A bespoke file slicer was employed to splice files into file sets of given runtime length, as discussed in [8]. This provides control over the quantity of opcodes passed to the machine learning algorithm, allowing direct comparison of files. The parser was then rerun over the truncated data, to form opcode count datasets of various run-lengths, where length is stated in quantity of observed opcodes.

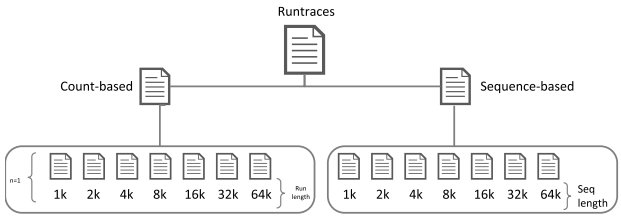

FIGURE 5. Derived datasets.

One benefit of dynamic analysis is the ordered and time-sequenced yielding of opcodes, which can be utilized by sequence-based algorithms. A second parser was implemented in order to translate the observed run traces into a format readable by sequence-based machine learning toolboxes. The stated opcode observed sequences are translated into a numerical representation and cut at a specified length, depending on the observation sequences desired. These sequences were compiled into a separate dataset. Fig. 5 enumerates the datasets which were derived from the overall corpus of run traces. For example, in 4, the count-based data would yield: `PUSH`: 4, `MOV`: 1, `LEA`: 1, `CALL`: 1. The sequence-based data would consider `PUSH MOV LEA PUSH PUSH PUSH CALL` as the ordered sequence.

V. DATASET COMPOSITION

The quantity of malware samples depleted at each stage of the process. Typically from *full corpus* to *executables-only*, and from *executables-only* to *executables with detection rate >50%*, there was 20% data loss. There was approximately a further 10% attrition rate from selected samples failing to provide a run trace due to a variety of reasons, including missing dlls, lock-outs and forced reboots.

At the point of commencing our experiments, the dataset stood at approximately 48,000 executed, traced, and labelled

samples. The categorization of malware into discrete labels is inexact and meant for a reference starting point only. Analysis of these labels will be presented in future work. The label of each individual piece of malware in our database differs from scanner to scanner, and even within scanners may be subject to change. Our corpus of total executables still contains a substantial quantity of unexecuted samples, which await processing. This is a continuous process and our dataset grows by up to 14,000 samples per week. We wish to present this parsed runtrace dataset and the further derived datasets in this paper for use by the wider research community.

VI. ANALYSES

For analysis purposes, we took an early version of the dataset presented here, while the tracing effort continued in parallel. Our analysis focussed on the following questions:

When examining opcode representations:

1) Can benignware be distinguished from malware (taken as a single descriptive entity)?
2) Does labeling malware using traditional AV labels (Mal_{AV}) improve classification against benignware?
3) Can each class of AV-labeled malware be distinguished from the others (Mal_{AV})?
4) Can benignware be distinguished from malware, which has been relabeled using unsupervised learning (Mal_{CL})?
5) Can each cluster of Mal_{CL} be distinguished from each other?
6) If clusters accurately describe new information, based on opcode analysis, is this consistent across both count-based and sequence-based learning?

A. DATA PRE-PROCESSING

Preliminary analysis of opcode sequences showed that the models were sensitive to differing sequence lengths, particularly shorter lengths. As the focus of this paper is on the effects of malware labeling on classification of differing types, we truncated the malware dataset to include sequenced instances of only 1000 opcodes with a 1% margin, i.e. 990-1000 opcodes in a sequence. For direct comparison of models, the datasets must be matched and we therefore used the same instances from the count-based dataset to match the two experimental datasets for this paper. The experimental datasets comprised 18,827 malicious samples and 764 benign samples. The sample sizes for each AV label class are depicted in table 2.

B. CLUSTERING

Creating a dataset of this size provided the opportunity to apply unsupervised learning in order to yield new information about malware types. During pilot work, comparisons were made between various clustering algorithms. The Expectation-Maximization (EM) algorithm provided clusters which, though performed well with subsequent inter-cluster classification, provided a numerically-shallow model for the data, with very large clusters eclipsing small clusters.

TABLE 1. Cluster sample sizes.

	Samples
Cluster 1	7555
Cluster 2	6149
Cluster 3	1485
Cluster 4	3638

TABLE 2. AV label sample sizes.

AV Label	*Samples*
Malicia	4383
Adware	178
Backdoor	269
Browser Modifier	687
Null	3468
PWS	882
Ransom	33
Rogue	233
Software Bundler	418
Trojan	1842
Trojan Downloader	1378
Trojan Dropper	579
Trojan Spy	79
VirTool	370
Virus	1958
Worm	2070
	18827

K-means clustering was also applied, though traditionally this algorithm requires the number of clusters, *k*, to be defined in advance. For this reason, we used *x*-means to determine the optimal *k* value. *X*-means [32] extends *k*-means by using a Bayesian Information Criterion (BIC) to find the optimal number of clusters. This search informed us that four clusters appeared optimal, which was reinforced by asking a Random Forest classifier to distinguish between the clusters, with an f-measure of 99.3% averaged across all clusters. However, the f-measure range was 81.1% - 100%. To improve this, we used the *k*-means algorithm with 4 clusters requested. These clusters were then validated in a similar manner, with an averaged f-score of 98.8% and a range of 96.8%-99.1%. Further, three of the four clusters achieved an AU-ROC of 100%, with the fourth at 99.9 %. These clusters were applied to the dataset as an extra set of labels, the quantities of which are described in table 1. The mapping of the 15 AV classes to the new clusters is visualized in fig. 6. Future work will examine the reasoning for each cluster, with analysis beyond the descriptives issued in the present paper.

C. COUNT-BASED CLASSIFICATION

The Random Forest (RF) [33] classifier was used, as implemented in WEKA 3.8, to classify all count-based data for each experiment.

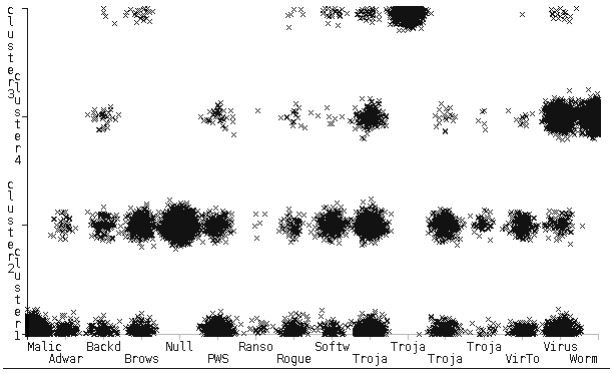

FIGURE 6. Traditional AV labels mapped to new clusters.

RF is an ensemble learner, combining the decisions of multiple smaller learners (decision trees). RF differs from traditional tree-based classifiers, in that the number of features used at each node in the tree to decide the parameter is randomized, which improves noise immunity and reduces overfitting. RF performs well on larger datasets, both in terms of instances and features. It parallelizes well [33], and is recommended with data with imbalanced classes [34]. In pilot experiments, RF provided the highest accuracy, despite imbalanced classes and shorter run-lengths, even with a large quantity of features. The RF classifier was trained with a holdout, with 2/3 of the dataset used for training and the remaining 1/3 for testing.

D. SEQUENCE-BASED CLASSIFICATION

Generative models can be used within the area of sequence based classification to model the probability distribution of sequences associated with a given class. The assumption by these models is that sequences associated with a given class are generated by an underlying model. Depending upon the model used, a specific set of assumptions is employed to define the model and a set of parameters defined to describe the probability distributions. A training process is used to determine an appropriate set of parameters from available data and then a classification, or recognition, process is used to assign a new sequence to the class associated with the highest likelihood [35].

1) HIDDEN MARKOV MODELS

Hidden Markov Models (HMMs) are an example of such generative models used to describe stochastic processes. HMMs are a popular choice for users wishing to model dynamic behaviors of underlying systems and are well known in the field of artificial intelligence and pattern recognition. HMMs are used to model time series data and have been applied successfully to a multitude of domains including speech recognition, computational biology and malware detection.

HMMs represent probability distributions over sequences of observed variables and can be easily visualized as a Bayesian network with a temporal dimension, more formally know as dynamic Bayesian networks [36], as seen in Figure 7.

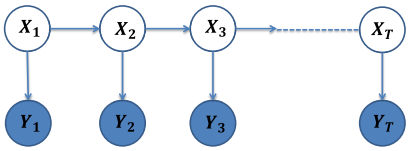

FIGURE 7. Example of hidden Markov model.

In fig. 7 a variable is represented by a node and the conditional dependencies between variables are represented by directed arcs. An observed variable at time t is denoted by Y_t for a given sequence of length $t = 1, \ldots, T$. These observed variables are sampled at equally spaced discrete time intervals. The discrete hidden state at time t is denoted by X_t and is a naming factor of HMMs, as it is assumed that at time t an observation, Y_t, is generated by a hidden process, X_t, which cannot be directly observed by the user. Another defining property of HMMs is that the hidden state is assumed to satisfy the Markov property, meaning that the current state X_t is independent of all states prior to $t - 1$ given X_{t-1}. The HMM observations also satisfy the Markov property with respect to the hidden states, whereby Y_t is independent of the hidden states and observations at all other time points. That is, the current time slice within a HMM encapsulates all that is required to predict the future process.

These properties allow a joint probability distribution over a sequence of hidden states and observations to be factored as follows;

$$P(X_{1:T}, Y_{1:T}) = P(X_1)P(Y_1|X_1)\prod_{t=2}^{T} P(X_t|X_{t-1})P(Y_t|X_t) \quad (1)$$

Given this factorization a number of components are required to define the joint probability distribution over a sequence of observed variables; a probability distribution over the initial state (prior), $\pi = P(X_1)$, a transition matrix associated with the hidden state, $A = P(X_t|X_{t-1})$ and an output model $B = P(Y_t|X_t)$ (which is defined as an observation matrix in the case of discrete observation symbols). A Hidden Markov Model can therefore be represented as a triple, $\lambda = \{A, B, \pi\}$ [37]. HMMs are traditionally assumed to be time invariant, meaning the transition and observation matrices are not dependent on time t.

2) HIDDEN MARKOV MODEL CLASSIFIERS

The main difference between utilizing Hidden Markov Models for the purpose of classification and traditional machine learning classifiers is that HMMs aim to classify instances based on temporal relations and can offer a different perspective on problem domains.

For the purpose of classification, or recognition, the aim is to classify new unseen sequences of observed symbols. To this end, a unique Hidden Markov Model is learnt per category using a training set compiled of sequences associated with a given class. For a problem domain consisting of N classes, the new observation sequence is assigned to the HMM that best describes the observation symbol sequence from the N available HMMs. That is, if λ_j describes the

parameter set used to define each of the N models, where $j = 1, \ldots, N$, and $\lambda_j = \{A_j, B_j, \pi_j\}$, when a new observation symbol sequence of unknown class is obtained $\mathbf{Y} = \{Y_1, \ldots, Y_T\}$, then $Pr(\lambda_j|\mathbf{Y})$ is calculated for each HMM $j = 1, \ldots N$ and the HMM described by λ_{N^*} is selected so that,

$$N^* = \underset{j}{\text{argmax}}(Pr(\lambda_j|\mathbf{Y})) \tag{2}$$

Simply put, we find the HMM that most probably generated the new sequence [38].

The task of classification therefore consists of two distinct problems; the user must first learn each HMM parameter set λ_j and then, given a new observation sequence, evaluate the log-likelihood of that sequence given each possible HMM. This log-likelihood defines the probability of a specific parameter set (HMM) given the observation sequence. During training a unique HMM must be learnt per class, whereby that HMM is most likely to have generated the observation symbol sequences for that class. This learning procedure consists of optimizing the model parameter set $\lambda = \{A, B, \pi\}$ to maximize the probability of the observation sequence (log-likelihood). To do so a special case of Expectation-Maximization is used known as the Baum-Welch algorithm [39] to find the maximum likelihood estimate of the HMM parameters. This consists of an iterative process, where an initial set of probability estimates are used to compute expectations of transitions/observations using the forward-backward algorithm and probability estimates are then maximized based on these expectations. The algorithm iterates between these two steps until convergence, which can be defined by a number of iterations or a difference in successive log-likelihood scores. The second task of calculating a score, or likelihood, of a new observation given each trained model can be computed using the forward algorithm [40]. The log-likelihood is calculated for each HMM and the most likely (greatest log-likelihood) is selected as the classification result.

3) IMPLEMENTATION OF HIDDEN MARKOV MODEL CLASSIFIERS IN OPCODE ANALYSIS

For the application of HMM classifiers to opcode analysis, the models are implemented using Kevin Murphy's HMM toolbox in Matlab [41]. A HMM is built per class for each individual experiment. For the binary classification task of determining benign or malicious sequences, a HMM is built to represent benign behavior and another to represent malicious behavior. In the case of the multi-label classification task of identifying malware families as determined by AV scanners, a HMM is built per family with the aim of representing the statistical properties of the malware family and so on. To describe a unique discrete HMM a number of initial parameters are required;

- T = length of the observation sequence.
- Q = Number of hidden states within the model
- M = Number of observation symbols

Given these parameters, the necessary triple can be defined for each HMM by training on available data to determine the prior π, the observation matrix B and the transition matrix A. For the purpose of this experiment the sequences used for both training and testing consisted of 1000 observations, $T = 1000$. Unlike the work presented in [42], who also apply HMMs to opcode sequences, a fixed set of opcodes are used as observable symbols, meaning M can be set appropriately. For this study 615 unique opcodes were incorporated, $M = 615$. To provide a usable data format for this analysis, each unique opcode was given a unique numeric value between 1 and 615, providing a final numeric sequence for training and testing purposes. Determining the number and representation of hidden states is an open problem [42] and one that has in the past been determined through empirical evaluation. To this end, a subsample of the dataset was utilized to determine the optimal number of hidden states, with $Q = 2$ providing the most promising preliminary results. A standard holdout procedure is used for experimental purposed, identical to that carried out during the implementation of count-based approaches. A 2/3 training to 1/3 testing (per class) was implemented i.e. the sequences found within the first 2/3 of the complete within class dataset are isolated for training and the final 1/3 are held out and compiled into a final testing set of all classes with the class label removed.

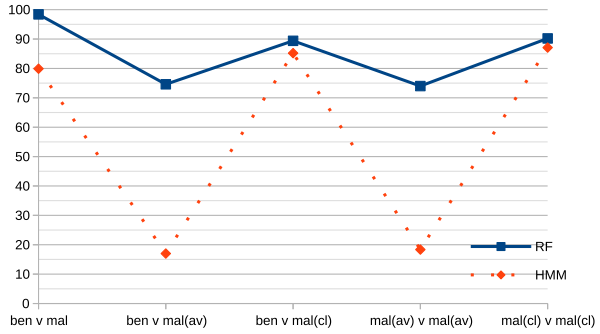

FIGURE 8. Effects of Mal_{AV} labels on accuracy for both algorithms.

VII. RESULTS

The use of AV labels can be shown in our research to be detrimental to malware classification when examining runtime behavior using opcode analysis. Fig. 8 shows two reductions in accuracy each time AV labels are used as class identifiers, which is markedly more severe for the sequence-based HMMs. Benignware can be distinguished from malware which has been represented as a homogeneous entity, even after only 1000 instructions. In this implementation, count-based classification offers superior accuracy to sequence-based classification in all comparisons. For the RF classifier, trying to discriminate between benign and mal_{AV} reduces overall detection from 98.4% to 74.6%, but reduces the detection of the malware category from 99% to 55% (see table 3). Similarly, trying to distinguish mal_{AV} classes from each other is only successful for 74% of the test set. When the malware is relabeled into one of four clusters, the discrimination between

TABLE 3. Performance of RF across 5 conditions.

Algorithm	Classes	Benign	Mal	Total
RF	Ben v Mal	87	99	98.4
	Ben v Mal_{AV}	86.2	55	74.6
	Ben v Mal_{CL}	83.6	85	89.4
	Mal_{AV} v Mal_{AV}			74
	Mal_{CL} v Mal_{CL}			90.2

TABLE 4. Performance of HMM across 5 conditions.

Algorithm	Classes	Benign	Mal	Total
HMM	Ben v Mal	11	98	79.9
	Ben v Mal_{AV}			17
	Ben v Mal_{CL}	12.9	73	85.2
	Mal v Mal_{AV}			18.35
	Mal_{CL} v Mal_{CL}			87.09

benign and mal_{CL} increases again to 89%. The mal_{CL} clusters can then be distinguished from each other to 90.2% accuracy. However, using the main dataset in preliminary work, this figure was as high as 98.4%, which could be in part due to sample size and class-imbalance problems. As illustrated in Table 4, this further indicates that there is a large degree of co-morbidity in the first 1000 opcodes between AV labels. This is exacerbated by the sequence-based learning, which suggests that the initial malware instructions are too similar or the AV labels do not sufficiently describe the runtime behavior of the malware. HMM had a poor performance on the benign test set, which may be due to the sample size and class-imbalance which occurred for both training and testing. However, the use of clusters notably enhances the detection of malware over that of the single-class representation.

VIII. SUMMARY AND CONTRIBUTIONS

Research to date using opcode analysis in both static and dynamic frameworks has failed to use adequately sampled datasets. Typically in the literature, the survey section of a paper will quote figures of the prevalence of malware, usually in the tens of millions. Yet in the experimental design section, as few as 70 samples are used in analysis. The make up of datasets mostly fails to address the percentage of malware in both training and testing sets, thus creating a machine learning model which may not be generalizable to a real world scenario. Further, the datasets are comprised of unmatched categories of malware, which introduces imbalances with the malicious set and between the malicious and benign sets.

This work presents a dataset of runtime trace opcodes, generated by a well-structured corpus of malware which has been attributed and initially labeled using traditional AV labels. This dataset of over 100,000 run traces, which grows continually, will be made freely available to the wider research community.

Our work, based on this new dataset, shows that malware can be detected within as few as 1000 instructions. The use of malware labels shows a negative impact across both algorithms when distinguishing between benign and malware, and shows poor classification between the AV labels themselves. We demonstrate that our clustering based on run-time opcodes provides new information in categorizing malware. We believe this approach will aid detection in a practical application, and will be of benefit to the continuing fight against malware.

REFERENCES

[1] C. Beek *et al.*, "McAfee labs threats report," McAfee, Santa Clara, CA, USA, Tech. Rep., Jun. 2017.

[2] P. O'Kane, S. Sezer, K. McLaughlin, and E. G. Im, "SVM training phase reduction using dataset feature filtering for malware detection," *IEEE Trans. Inf. Forensics Security*, vol. 8, no. 3, pp. 500–509, Mar. 2013.

[3] S. Vemparala, F. Di Troia, V. A. Corrado, T. H. Austin, and M. Stamo, "Malware detection using dynamic birthmarks," in *Proc. ACM Int. Workshop Secur. Privacy Anal. (IWSPA)*, 2016, pp. 41–46.

[4] P. O'Kane, S. Sezer, and K. McLaughlin, "Obfuscation: The hidden malware," *IEEE Secur. Privacy*, vol. 9, no. 5, pp. 41–47, Sep./Oct. 2011.

[5] M. G. Schultz, E. Eskin, E. Zadok, and S. J. Stolfo, "Data mining methods for detection of new malicious executables," in *Proc. IEEE Symp. Secur. Privacy (S&P)*, May 2001, pp. 38–49.

[6] J. Z. Kolter and M. A. Maloof, "Learning to detect malicious executables in the wild," in *Proc. 10th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2004, pp. 470–478.

[7] R. Moskovitch, Y. Elovici, and L. Rokach, "Detection of unknown computer worms based on behavioral classification of the host," *Comput. Statist. Data Anal.*, vol. 52, no. 9, pp. 4544–4566, 2008.

[8] P. O'Kane, S. Sezer, K. McLaughlin, and E. G. Im, "Malware detection: Program run length against detection rate," *IET Softw.*, vol. 8, no. 1, pp. 42–51, Feb. 2014.

[9] I. Santos, F. Brezo, B. Sanz, C. Laorden, and P. G. Bringas, "Using opcode sequences in single-class learning to detect unknown malware," *IET Inf. Secur.*, vol. 5, no. 4, pp. 220–227, 2011.

[10] N. Runwal, R. M. Low, and M. Stamp, "Opcode graph similarity and metamorphic detection," *J. Comput. Virol.*, vol. 8, nos. 1–2, pp. 37–52, 2012.

[11] B. Anderson, D. Quist, J. Neil, C. Storlie, and T. Lane, "Graph-based malware detection using dynamic analysis," *J. Comput. Virol.*, vol. 7, no. 4, pp. 247–258, 2011.

[12] D. Lin and M. Stamp, "Hunting for undetectable metamorphic viruses," *J. Comput. Virol.*, vol. 7, no. 3, pp. 201–214, 2011. [Online]. Available: http://dx.doi.org/10.1007/s11416-010-0148-y

[13] S. Attaluri, S. McGhee, and M. Stamp, "Profile hidden Markov models and metamorphic virus detection," *J. Comput. Virol.*, vol. 5, no. 2, pp. 151–169, 2009.

[14] S. P. Thunga and R. K. Neelisetti, "Identifying metamorphic virus using n-grams and hidden Markov model," in *Proc. IEEE Int. Conf. Adv. Comput., Commun. Inform. (ICACCI)*, Aug. 2015, pp. 2016–2022.

[15] T. H. Austin, E. Filiol, S. Josse, and M. Stamp, "Exploring hidden Markov models for virus analysis: A semantic approach," in *Proc. IEEE 46th Hawaii Int. Conf. Syst. Sci. (HICSS)*, Jan. 2013, pp. 5039–5048.

[16] A. Nappa, M. Z. Rafique, and J. Caballero, *Driving in the Cloud: An Analysis of Drive-by Download Operations and Abuse Reporting* (Detection of Intrusions and Malware, and Vulnerability Assessment). Berlin, Germany: Springer, 2013, pp. 1–20.

[17] C. Annachhatre, T. H. Austin, and M. Stamp, "Hidden Markov models for malware classification," *J. Comput. Virol. Hacking Techn.*, vol. 11, no. 2, pp. 59–73, 2015.

[18] J. Kinable and O. Kostakis, "Malware classification based on call graph clustering," *J. Comput. Virol.*, vol. 7, no. 4, pp. 233–245, 2011.

[19] R.-S. Pirscoveanu, M. Stevanovic, and J. M. Pedersen, "Clustering analysis of malware behavior using self organizing map," in *Proc. Int. Conf. Cyber Situational Awareness, Data Anal. Assessment (CyberSA)*, Jan. 2016, pp. 1–6.

[20] D. Bilar, "Opcodes as predictor for malware," *Int. J. Electron. Secur. Digit. Forensics*, vol. 1, no. 2, pp. 156–168, 2007.

[21] I. Santos, B. Sanz, C. Laorden, F. Brezo, and P. G. Bringas, *Opcode-Sequence-Based Semi-Supervised Unknown Malware Detection* (Computational Intelligence in Security for Information Systems). Berlin, Germany: Springer, 2011, pp. 50–57.

[22] R. Moskovitch *et al.*, *Unknown Malcode Detection Using OPCODE Representation* (Intelligence and Security Informatics). Berlin, Germany: Springer, 2008, pp. 204–215.
[23] A. Shabtai, R. Moskovitch, C. Feher, S. Dolev, and Y. Elovici, "Detecting unknown malicious code by applying classification techniques on opcode patterns," *Secur. Inform.*, vol. 1, no. 1, pp. 1–22, 2012.
[24] (2014). *VX Heaven*. [Online]. Available: http://vxheaven.org/vl.php
[25] B. Kang, K. S. Han, B. Kang, and E. G. Im, "Malware categorization using dynamic mnemonic frequency analysis with redundancy filtering," *Digit. Invest.*, vol. 11, no. 4, pp. 323–335, 2014.
[26] J.-M. Roberts. (2014). *VirusShare*. [Online]. Available: https://virusshare.com/
[27] M. Sebastián, R. Rivera, P. Kotzias, and J. Caballero, "AVclass: A tool for massive malware labeling," in *Proc. Int. Symp. Res. Attacks, Intrusions, Defenses*, 2016, pp. 230–253.
[28] D. Carlin, P. O'Kane, and S. Sezer, "Dynamic analysis of malware using run-time opcodes," in *Data Analytics and Decision Support for Cybersecurity: Trends, Methodologies and Applications*, 1st ed. Cham, Switzerland: Springer, 2017.
[29] *Oracle VM Virtualbox*, Oracle Corporation, Santa Clara, CA, USA, 2015.
[30] Oracle Corp., Santa Clara, CA, USA. (2015). *Oracle VM Virtualbox Programming Guide and Reference*. [Online]. Available: http://download.virtualbox.org/virtualbox/SDKRef.pdf
[31] K. Lejska. (Sep. 13, 2017). *X86 Opcode and Instruction Reference*. [Online]. Available: http://ref.x86asm.net/
[32] D. Pelleg and A. W. Moore, "X-means: Extending k-means with efficient estimation of the number of clusters," in *Proc. ICML*, vol. 1. 2000, pp. 727–734.
[33] L. Breiman, "Random forests," *Mach. Learn.*, vol. 45, no. 1, pp. 5–32, 2001.
[34] T. M. Khoshgoftaar, M. Golawala, and J. Van Hulse, "An empirical study of learning from imbalanced data using random forest," in *Proc. 19th IEEE Int. Conf. Tools Artif. Intell. (ICTAI)*, vol. 2. Oct. 2007, pp. 310–317.
[35] Z. Xing, J. Pei, and E. Keogh, "A brief survey on sequence classification," *ACM SIGKDD Explorations Newslett.*, vol. 12, no. 1, pp. 40–48, Jun. 2010.
[36] K. P. Murphy and S. Russell, "Dynamic Bayesian networks: Representation, inference and learning," Tech. Rep., 2002.
[37] Z. Ghahramani, "An introduction to hidden Markov models and Bayesian networks," *Int. J. Pattern Recognit. Artif. Intell.*, vol. 15, no. 1, pp. 9–42, 2001.
[38] J. Yamato, J. Ohya, and K. Ishii, "Recognizing human action in time-sequential images using hidden Markov model," in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 1992, pp. 379–385.
[39] L. E. Baum, T. Petrie, G. Soules, and N. Weiss, "A maximization technique occurring in the statistical analysis of probabilistic functions of Markov chains," *Ann. Math. Statist.*, vol. 41, no. 1, pp. 164–171, 1970.
[40] X. D. Huang, Y. Ariki, and M. A. Jack, *Hidden Markov Models for Speech Recognition*, vol. 2004. Edinburgh, U.K.: Edinburgh Univ. Press, 1990.
[41] K. P. Murphy. *Hidden Markov Model (HMM) Toolbox for MATLAB*. [Online]. Available: https://www.cs.ubc.ca/~murphyk/Software/HMM/hmm.html
[42] W. Wong, "Analysis and detection of metamorphic computer viruses," M.S. thesis, Dept. Comput. Sci., San Jose State Univ., San Jose, CA, USA, 2006.

DOMHNALL CARLIN received the M.Sc. degree in software development from Queen's University Belfast, Belfast, where he is currently pursuing the Ph.D. degree with the Center for Secure Information Technologies. His research interests include machine learning for the detection of malware.

ALEXANDRA COWAN received the B.Sc. degree in mathematics and statistics and operational research and the Ph.D. degree in statistics from Queen's University Belfast, Belfast. She is currently involved in machine learning and data mining solutions for biometric authentication and visual speech recognition technologies.

PHILIP O'KANE received the B.Eng. degree in electronic systems and the M.Sc. degree in informatics from the University of Ulster in 1994 and 2009, respectively, and the Ph.D. degree from Queen's University Belfast with a focus on the detection of obfuscated malware. His career started as an embedded Software Engineer in the Telecoms industry and later moved to application development for the Finance industry. He is currently a Lecturer with the Center for Secure Information Technologies, Queen's University Belfast. He has 20 years of software development in industry. His interests include the low-level analysis and detection of malware.

SAKIR SEZER received the Ph.D. degree in electronic engineering from Queen's University Belfast. He is currently a Professor and the Head of network and cyber-security research with the Center for Secure Information Technologies, Queen's University Belfast. He is also a Co-Founder and the CTO of Titan IC Systems and a member of various research and executive committees, including the IEEE International System-on-Chip Conference Executive Committee. His research is leading major advances in the field of high-performance content and security processing.