MalClassifier: Malware Family Classification Using Network Flow Sequence Behaviour

Bushra A. AlAhmadi and Ivan Martinovic
Department of Computer Science
University of Oxford, Oxford, United Kingdom
{bushra.alahmadi, ivan.martinovic}@cs.ox.ac.uk

***Abstract*—Anti-malware vendors receive daily thousands of potentially malicious binaries to analyse and categorise before deploying the appropriate defence measure. Considering the limitations of existing malware analysis and classification methods, we present *MalClassifier*, a novel privacy-preserving system for the automatic analysis and classification of malware using network flow sequence mining. *MalClassifier* allows identifying the malware family behind detected malicious network activity without requiring access to the infected host or malicious executable reducing overall response time. *MalClassifier* abstracts the malware families' network flow sequence order and semantics behaviour as an n-flow. By mining and extracting the distinctive n-flows for each malware family, it automatically generates network flow sequence behaviour profiles. These profiles are used as features to build supervised machine learning classifiers (K-Nearest Neighbour and Random Forest) for malware family classification. We compute the degree of similarity between a flow sequence and the extracted profiles using a novel fuzzy similarity measure that computes the similarity between flows attributes and the similarity between the order of the flow sequences. For classifier performance evaluation, we use network traffic datasets of ransomware and botnets obtaining 96% F-measure for family classification. *MalClassifier* is resilient to malware evasion through flow sequence manipulation, maintaining the classifier's high accuracy. Our results demonstrate that this type of network flow-level sequence analysis is highly effective in malware family classification, providing insights on reoccurring malware network flow patterns.**

I. Introduction

Most cyber-attacks leverage malware to perpetrate attacks that may lead to financial, privacy, and even human life loss. More than 317 million new variants of malware were observed in 2014, an estimate of 1 million unique malware each day, increasing the total number of malware to approximately 1.7 billion [35]. Perhaps the main challenge for anti-malware vendors is the growing stream of incoming malware samples due to the use of polymorphic and metamorphic techniques, allowing the same malware to be modified avoiding detection. Although the binary classification of an unknown executable to either malicious or benign (*malware detection*) is important, accurately determining which family the malicious executable belongs to is also a much sought after application [33], [18].

Malware classification determines whether a malicious binary is a member of a family seen previously, or whether it is a novel, previously unseen sample requiring further analysis to understand its behaviour. The number of unique profiles for malware variants of a family is huge, however, their malicious behaviour is similar and consistent within the malware family [34]. Grouping samples into families and extracting their distinctive behaviour patterns can help security analysts in understanding the evolution of the various families. Analysts can then assess the potential damage of a newly discovered sample, prioritise encountered threats, in turn enabling the development of effective mitigation mechanisms [2]. The incident response procedure for ransomware is different from a botnet, and being able to determine the family can speed up the remediation process.

Existing dynamic analysis tools for malware classification, such as [26], [29], use a sand-box approach that requires the existence of the malicious binary and running it in a controlled environment for classification. Today, more companies are outsourcing the security management and network monitoring to companies that offer Security Operations Centre (SOC) as a service to escape the significant investments and lack of cyber-security expertise [19], [1]. Although these companies monitor the client's systems and networks for signs of malicious activity, they may not have direct access to the client's hosts due to privacy or policy reasons. When malware-infected hosts are the source of the detected malicious traffic, the analysts need to negotiate access to the infected host with the client. Even if access is possible, the time required to negotiate this and running the executable in a sand-box for classification is not efficient, particularly in situations where rapid detection and response is critical. In addition, malware may produce different behaviours depending on the environment they are run in, or even may not act at all [18]. Therefore, analysts need *on-the-wire* malware classification systems capable of matching detected malicious network traffic to a malware family, thus not requiring to access the infected host.

Previous approaches such as [22], [23], [18] proposed systems for malware classification using network traffic. Perhaps the most related work to ours is [18] that observes the high-level network features of malware and applied n-gram document analysis for family classification. Our work improves on [18], by (1) using non-payload features, making our system privacy aware and robust against encryption; (2) adapting to malware behaviour changes; (3) not requiring the execution of the malware in a sand-box, thus performing classification on-the-wire. Similarly, previous approaches are either not content

978-1-5386-4922-0/18/$31.00

agnostic, relying on features from payload (e.g. [23], [22]) or protocol dependent (*e.g. cover only HTTP traffic*) such as [22]. We compare the related work to ours in detail in Section II.

In this paper, we propose *MalClassifier*, a novel system for the automatic extraction of network behavioural characteristics for malware family classification. As malware use some form of network communication to propagate or contact their command and control (C&C) servers [24], *MalClassifier* derives the *network* behavioural characteristics for a malware family, abstracting the malware family's network behaviour to a *flow sequence profile*. *MalClassifier* is effective in classifying malicious network flow sequences to a malware family *on-the-wire*, thus negating the need for access and sand-box execution of the malware binary. With the increasing use of encryption by malware for obfuscation and by hosts for privacy (*e.g.* HTTPS), malware classification based on non-content agnostic systems become challenging. Therefore, *MalClassifier* uses non-identifiable high-level network traffic for training the classifiers, making it resilient to encryption. This also makes the data required to train the classifiers accessible and easier to share than full PCAP traces due to privacy concerns. In designing *MalClassifier*, we make our approach IP-agnostic, as sophisticated malware, such as exploit kits, apply dynamic DNS and Domain Generation Algorithms (DGA) to change their communications destination [13].

Similar to [17], we apply n-grams to sort network flows into groups of n consecutive flows (i.e. n-flows). Such a representation provides granularity to the extracted malware behaviour. For example, a single failed SMTP flow might be benign, but multiple consecutive ones exhibit the behaviour of a bot sending spam. *MalClassifier* mines n-flows that are distinctive for each malware family. Such distinctive n-flows are used as features to train a supervised model capable of classifying unseen n-flows to the malware family. The models learn re-occurring network flow patterns that capture the characteristics of a malware family's network behaviour. The contributions of this paper are three-fold:

- We propose a novel fuzzy flow sequence similarity measure, that calculates the *value* similarity of two flow sequences.
- We propose an order sequence similarity measure robust against malware evasion through flow sequence manipulation.
- We develop a prototype of *MalClassifier*, and evaluate its performance using a dataset of malicious network traffic, achieving more than 95% F-measure for malware family classification.

II. Related Work

Malware is a constant cyber challenge for organisations, and the research literature is rich with contributions on its analysis and classification [9], [7]. We discuss the most relevant contributions that apply a sequencing analysis approach or malware network flows as features for malware family classification.

Host-based Malware Analysis and Family Classification. Early malware family classification contributions focused on classifying malware based on the malware binary content or sequence of bytes in binary files. For example, the authors in [14] performed malware family classification using n-grams of bytecode. However, such content-based approaches require dissembling the binary, which is a time-consuming process and vulnerable to malware obfuscation. Dynamic analysis was applied for malware classification in [20] to construct system calls behaviour graphs and control flow graph signatures [6]. Bayer et al. in [3] used dynamic analysis to generate malware behavioural profiles used as features in a clustering algorithm to group malware based on behaviour. Rieck et al. in [26] applied dynamic analysis to identify the unique behavioural features of malware samples and use these features to build Support Vector Machine (SVM) classifiers that map unknown samples to known malware families. Similarly, Rieck et al. in [28] proposed a framework for malware behavioural analysis, using clustering (to identify novel malware with similar behaviour) and classification (to assign unknown malware to these clusters). Although these approaches are effective in classifying malware by observing its host-level behaviour, they require access to the binary executable which might not be possible in certain scenarios. In contrast, *MalClassifier* classifies malware based on the network behaviour, requiring only the high-level network traffic.

Network-based Malware Family Classification. In this section, we focus on recent contributions in malware family classification using network traffic. Malware behavioural clustering approaches aim to group malware behaviour to reveal similarities between malware samples that may not be captured using system-level malware analysis. Perdisci et al. in [22], proposed a network-level clustering system to derive HTTP network behaviour similarity of HTTP-based malware for detection. FIRMA [23] used a similar approach to cluster and generate network signatures for malware network traffic.

The most directly related work to ours is *CHATTER* [18]. Such system considers the order of high-level network events as features and applies n-gram document analysis to produce the classifier achieving 90% accuracy. The malware network behaviour profile is represented as fine-grained events, thus each attribute of a single packet is considered an event. For example, an inbound packet using TCP on port 80 is represented as $A1\_A3\_A6$, where A1 refers to the inbound connection event, A3 refers to the TCP protocol usage event, and A6 refers to the usage of port number 80 event. Instead, *MalClassifier* uses a similar approach to [17], using coarse-grained groups to represent a network behaviour, i.e. $A1A3A6$ is considered a single event. Therefore, our system is able to capture behaviours such as SMTP flow followed by another SMTP flow that might indicate an email spam.

Moreover, the numeric features are mapped into one of the four quartiles that are pre-determined based on the training data. However, there is a risk of new malware variants falling out of the quartiles ranges leading to inaccurate family classifications. In addition, this abstraction may result in loss of underlying distinctive behaviour that could have resulted in more accurate classifications. Instead, *MalClassifier* uses

Cosine Similarity to compare the similarity of the numeric features, thus not requiring pre-determined ranges or thresholds.

The main limitation of previous work is their use of payload features, meaning they are vulnerable to malware obfuscation through encryption and that they are not privacy-preserving. In Table I, we compare the beforehand related work to *MalClassifier*. Specifically, we identify the data used to train the classifier, whether the approach is IP and content agnostic, uses n-grams, and if the approach requires running the sample in a sandboxed environment. We also identify whether related work addressed malware evasion through noise injection in their system design.

Malware Network Flow Detection. *MalClassifier* is a malware family classification system, meaning it classifies malware to a family based on its network behaviour. For completeness, we discuss malware detection systems that have applied a similar methodology or have applied network flows as features for detection. Mekky et al. in [17] used a similar approach to [18] by first isolating malware traffic from the benign traffic using Independent Component Analysis (ICA). However, their approach differs from [18] as they use coarse-grained groups of network events so that inbound, DNS, and port number are considered one network event. Similar to [18], they use a count-based approach for network flow identification.

Botzilla [27] monitored malware network traffic, specifically a bots' communication to the C&C server, in a sandboxed environment to find patterns and generate network signatures. Similarly, BotSniffer [12] is a network-based anomaly detection system, focused on detecting C&C network communication. BotFinder [36] monitors bots' network traffic in a controlled environment and generates malware detection models that can be deployed at network egress points. These models then detect individual, bot-infected hosts by monitoring their network traffic. BotMiner [10] detects botnet C&C communications by applying a clustering approach to detect correlated C&C communications, malicious activities, and it performs cross-cluster correlation to detect hosts that have similar malicious activity. Disclosure [5] extracts flow size-based features, client access patterns, and temporal behaviour features from netflow communication to detect botnet C&C communications. The main purpose of these systems is to detect botnet C&C communications. However, *MalClassifier* extracts malware network flow *sequence* behaviour during its various infection stages, and classifying unknown traffic to a malware family based on that sequence behaviour.

Sequential Pattern Mining. Research on extracting relevant patterns between malware samples where the values are in a sequence format for malware classification is well established. For example, Santos et al. in [30] used n-grams, which are sub-strings of a larger string with length n, to generate file signatures for malware detection. Wressnegger et al. in [38] studied the applicability of n-grams for anomaly detection and classification. Mekky et al. in [17] used n-gram to encode the order of sub-sequences of malware network communication events to build malware classification models. *MalClassifier* uses a similar approach in representing malware network communications as a sequence of flows of length n.

TABLE I: Comparison of related work on malware family classification and MalClassifier against our design goals.

	Data	Content-agnostic	IP-agnostic	Evasion Resiliency	Use n-grams	Use sand-box	Accuracy
Rieck et al. [26]	Behavioural reports	N	N	N	N	Y	80.7%
Rieck et al. [28]	Behavioural reports	N	N	N	N	Y	95%
Perdisci et al. [22]	HTTP Traffic	N	Y	N	N	N	N/A
FIRMA [23]	Network traffic	N	N	N	N	N	98.8%
CHATTER [18]	Network traffic	N	Y	N	Y	Y	90%
MalClassifier	Bro *conn* logs	Y	Y	Y	Y	N	96%

III. System Overview

Malware exhibit diverse and complex network traffic behaviour. Yet, malware variants of the same family have been known to have common behavioural patterns reflecting their origin and purpose [11], [26]. In our system, we aim to exploit these shared patterns, particularly their network behaviour, for malware family classification.

MalClassifier can be deployed by security analysts to understand and classify the behaviour of a malicious executable by observing its network flows when access to that executable itself is not possible. It maps malicious network flow sequences (n-flows) to a malware family by comparing these sequences to previously observed malware activity. In general, *MalClassifier* operates in three phases as illustrated in Figure 1.

1) *Pre-processing and Sub-Sequence Extraction:* Network traffic of malware variants of malware family y is input into Bro for network flow reassembly. The assembled flows are then encoded to a *textual sequence*. The sequence is then divided to sub-sequences (n-flows) of length n.
2) *Malware Family Profile Extraction:* The *Value Similarity* of each individual flow in the encoded sequence to all other flows is computed, in turn the sub-sequence similarity is determined. The most distinctive flow sub-sequences (n-flows) are selected as profiles for malware family y.
3) *Training and Building Models:* Using the profiles, the supervised machine learning model is trained for classifying unseen n-flows to a malware family.

A. Design Goals and Requirements

The main limitation of existing malware family classification approaches is the need to obtain the malware executable, run it in a sandbox to classify it to a malware family. Unfortunately, this timely process hinders its adoption in real-world application where access to the infected host is not possible or where a rapid analysis and response is crucial.

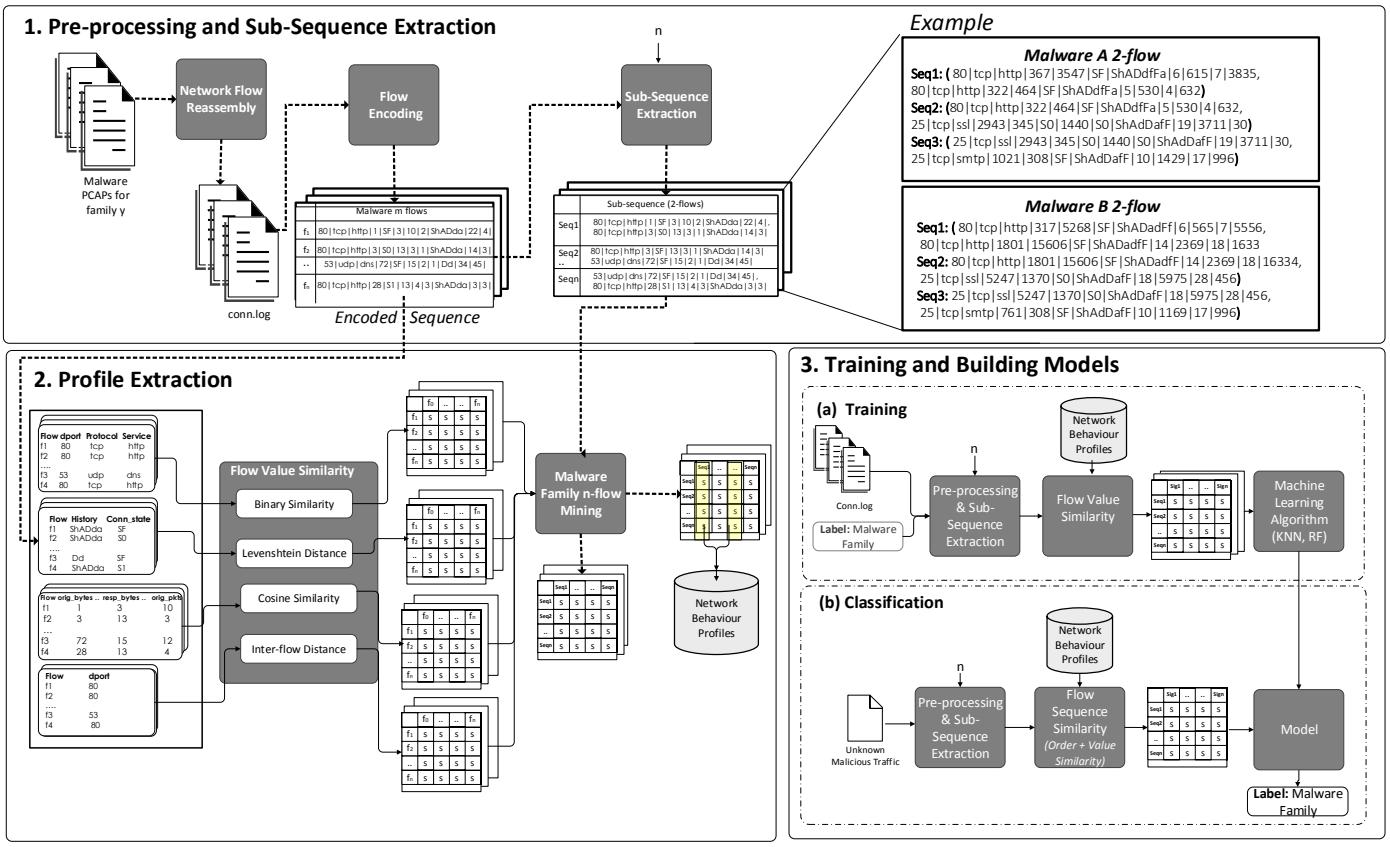

Fig. 1: MalClassifier System.

To foster its real-world use, we consider this limitation and the following requirements when designing *MalClassifier* to ensure that our system is resilient to malware evasion and classifies n-flows with a high accuracy. We compare the most related contributions to *MalClassifier*, and summarise their deployment of the identified design goals in Table I.

- **IP-agnostic.** The system must not use destination IP as a feature. Malware rapidly changes its (C&C) and deploy sophisticated domain generation algorithms and domain shadowing of legitimate domains to evade reputation filtering.
- **Non-privacy invasive.** SOC-as-a-service clients require privacy preserving monitoring and systems that observe the payload are privacy invasive. Therefore, the system must not rely on network traffic payload to extract features. In addition to reducing the storage space, this makes the system resilient to encryption which sophisticated malware and benign hosts (*e.g.* HTTPS) use. Moreover, the non-identifiable network data required for training the models are accessible, which is critical for supervised classifier training and for the potential adoption and acceptance of the system at scale.
- **Automatically identify distinctive malware network behaviour.** The system must be able to automatically identify and extract distinctive network behaviour (i.e., profiles) of each malware family.
- **On-the-wire classification.** Obtaining the malicious executable and running it in a sandbox should not be required. Instead, it should be able to classify malicious network traffic on-the-wire to a malware family.
- **Resilient to malware evasion attacks and adaptability to malware behaviour changes.** The system must be adaptive to malware evolution and behavioural changes. Meaning, if the malware network flow behaviour changes slightly (*e.g.* change protocol UDP to TCP or increase/decrease in size), then the system model should still be able to classify to the correct malware family. In addition, the system should be robust against malware evasion through flow field manipulation by using tamper-resistant features [4]. Malware may attempt to change the order sequence of the flows to avoid detection. Thus, the system should consider flow order deception, and be able to still classify manipulated sequences with high accuracy.
- **High classification accuracy.** The classifier must aim to provide acceptable classification accuracy using only sub-sequences of network traffic, thus not requiring a malware's full packet captures.

IV. *MalClassifier* Design

Considering the design goals in Section III-A, we discuss the design of each module of the *MalClassifier* system in detail in the following sections.

TABLE II: Description of the fields in conn.log generated by Bro network monitoring framework and used in *MalClassifier*.

Field	Description
resp_port	Destination port
proto	Transport layer protocol (TCP, UDP)
service	Application protocol being sent over the connection.
orig_bytes	Number of payload bytes the originator sent
resp_bytes	Number of payload bytes the responder sent
conn_state	State of the Connection. 13 different states (*e.g.* connection attempt rejected)
history	State history of connections as a string of letters.
orig_pkts	Number of packets that the originator sent
orig_ip_bytes	Number of IP level bytes that the originator sent
resp_pkts	Number of packets that the responder sent
resp_ip_bytes	Number of IP level bytes that the responder sent

A. Pre-processing

In order to convert the network flows into a format that can be applied to sequence mining methods, we first need to pre-process the data by reassembling and encoding the network flows.

1) Network Flow Reassembly: Organisations deploy network monitoring systems such as Bro [1], which generates statistical and behavioural logs about the network communications, the application level protocols, and exchanged payload of each network flow. Maintaining full network traces (PCAPs) requires a huge amount of storage, making it challenging for organisations to keep full network traces for more than a couple of days. On the other hand, to investigate security breaches, whose effects might show up much later, logs may be stored for a longer period and reduce storage costs. We envision *MalClassifier* to be deployed with network monitoring systems for on-the-wire malware classification.

We note that the *Network Flow Reassembly* module is only needed when converting network PCAP traces to Bro *conn* logs, and therefore is not needed if the Bro logs are available or when Bro is applied in the network for network monitoring.

To extract behavioural features from the malware PCAP traces, we use Bro to reassemble the network flows. A flow is a sequence of packets from a source host and port to a destination host/port that is part of a unique TCP/UDP session. Packets in a flow are either going to (or coming from) the same destination IP address and port. As input, Bro takes the captured malware PCAP network traces and generates a number of logs for each malware sample. These logs include information that is useful in understanding malware behaviour, such as C&C communication statistics, DNS queries and fast fluxing, unusual communications (*e.g.* unknown protocols) and port-host scanning. In *MalClassifier*, we leverage the Bro *conn.log* file that shows non-identifiable network flow header information of TCP/UDP/ICMP connections. Each row in the log represents an individual flow f and is described by 20 attributes representing the column fields. We use 11 of the attributes derived from the *conn.log* and described in Table II.

2) Flow Encoding: For each log $x \in X$, where X is the set of all Bro *conn.log* logs for samples of a malware family y, we encode the log x to a long *sequence* of network flows, $E(x) = f_1 \rightarrow f_2 \rightarrow .. \rightarrow f_i$, where f_i represents a single flow in the log x. Formally, we define a flow f_i as:

$f_i :=< resp\_port, proto, service, orig\_bytes, resp\_bytes, conn\_state, history, orig\_pkts, orig\_ip\_bytes, resp\_pkts, resp\_bytes >$

As we show in Figure 1, the result of the Flow Encoding module is a textual sequence representation of the flows in the *conn.log* of malware samples of a malware y.

3) Flow Sub-Sequence Extraction: We represent the behaviour of malware as a sub-sequence of flows. This provides higher granularity of the captured malware network behaviour. For example, a single $icmp$ flow does not map to a particular behaviour, but a sequence of multiple $icmp$ flows represent a malware performing an $icmp$ scan. To capture the malware flow-level behaviour, we consider *sub-sequences* of the malware network flows of length n, called n-flows as shown in Figure 1. Therefore, when $n = 1$, the 1-flow represents a single network flow, and when $n = 2$, the bi-flow represents two consecutive network flows, and so on. This results in a group of consecutive network flows of length n, that reflect flow-level behavioural patterns. Such an approach allows us to capture the network sequence behaviour and extract the unique sub-sequence (i.e n-flow) profiles.

B. Malware Family Profile Extraction

MalClassifier extracts the malware family's distinctive network flow sequence behaviour, abstracting that behaviour as a *sequence profile*. We discuss below how these profiles are generated for each malware family. To clarify the different steps involved in generating the behavioural profiles, we use a *running example*. Thus, we show in Figure 1 an example of the extracted sub-sequences or n-flows (when $n = 2$) of two malware samples (Malware A and Malware B).

1) Flow Sequence Similarity: Similarity measures are defined as 'functions that quantify the extent to which objects resemble each other, taking as an argument object pair and return numerical values that are higher as the objects are alike' [15]. Choosing the similarity measures relies on the data nature itself, whether binary, numerical or structured data (*e.g.* sequences, trees, graphs). Similarity measures for binary data are used to determine the presence or absence of characteristics in the object pair (*i.e.* pair of flows), thus take a value 1 if the flow possesses the characteristic, and 0 otherwise. For example, the authors in [18] applied a binary similarity approach, by counting the number of times the exact n-gram occurs. We argue that network flows are structured data, and thus binary similarity are not suitable as they do not capture the malware underlying network flow semantic.

Instead, *MalClassifier* applies a *fuzzy* Value Similarity measure. Thus, instead of determining *is a sub-sequence for malware A and a sub-sequence for malware B the same?*, fuzzy similarity determines how similar a sub-sequence in malware A to a sub-sequence in malware B by computing the degree of similarity. Value Similarity computes the similarity

[1]https://www.bro.org

of each flow f_{iA} in a sub-sequence for Malware A to its corresponding flow f_{iB} in the sub-sequence for Malware B.

A single flow f is composed of a number of attributes as we defined in Table II. Each attribute is semantically different in data type and value distribution, and thus when choosing a similarity measure the underlying differences between the attributes need to be considered. For example, $history$ is represented as a string of characters, where each character has a meaning and the order of the characters also has a meaning that should be considered. In contrast, numeric attributes such as (*orig_bytes, resp_bytes, orig_pkts, orig_ip_bytes, resp_pkts, resp_ip_bytes*), are represented as numeric vectors, thus numeric similarity measures such as Cosine Similarity should be applied.

Although $resp\_port$ is a numeric, the underlying meaning differs from other numeric fields. For example, although the values of $orig\_bytes = 100$ or $orig\_bytes = 99$ should be considered similar, a small difference in $resp\_port$ does not indicate a similarity (port 80 for HTTP could be considered related to port 443 for HTTPS, whilst port 23 being closer to port 80 is semantically different). Therefore, applying one similarity measure to all attributes is not sufficient.

We propose a hybrid value similarity measure to determine the similarity of two flow sequences. Our hybrid approach takes into account the semantic differences of the flow attributes and applies a similarity measure suitable to each attribute. In general, we apply four similarity measures, depending on the flow attributes.

We list in the following each similarity measure providing an example of how the similarity of Seq_1 of Malware A and Seq_2 of Malware B shown in Figure 1 is calculated.

- **Binary similarity** ($resp\_port$, $protocol$, $service$): The similarity is 1 if the attribute values are the same, otherwise 0. Thus, the Binary Similarity in our example of ($f_{0A} \in Seq_1 : 80|tcp|http, f_{0B} \in Seq_2 : 80|tcp|http$) and ($f_{1A} \in Seq_1 : 80|tcp|http, f_{1B} \in Seq_2 : 25|tcp|ssl$) is (1, 0.33) respectively, resulting in an average Binary Similarity of 0.665.
- **Levenshtein Distance** [16] ($history$,$conn\_state$): Levenshtein Distance is a fuzzy string similarity measure that measures the minimum number of modifications required (insertions, deletions, and substitutions) to change one string into the other, divided by the maximum length of the same two strings. It also takes into consideration the order of the characters in the string. Assuming the cost of insertion, deletion, and modification is the same ($= 1$), then the Levenshtein Distance of making $ShADdFa$ into $ShADadR$ is 3. The trivial implementation has a runtime and space complexity of $O(nm)$. The distance value ranges from [0,100], where 0 indicates a low distance thus higher similarity and 100 indicates a low similarity. We scale the distance value to be in the range [0,1] instead of [0,100].

  Using our example, the Levenshtein Distance of ($f_{0A} \in Seq_1 : SF|ShADdFa, f_{0B} \in Seq_2 : SF|ShADadfF$) and ($f_{1A} \in Seq_1 : SF|ShADdFa$, $f_{1B} \in Seq_2 : S0|ShAdDafF$) is (2, 4) respectively, resulting in an average Levenshtein Distance of 3, scaled to 0.03.
- **Cosine Similarity**: The numeric attributes of the two flows are represented as two vectors. Cosine Similarity measures the similarity of two non-zero vectors by calculating the cosine of the angle between them. Given the vectors x and y of length $n = 6$ (*number of numeric attributes*), the cosine similarity is represented as:

$$\cos(\mathbf{x}, \mathbf{y}) = \frac{\mathbf{x}\mathbf{y}}{\|\mathbf{x}\|\|\mathbf{y}\|} = \frac{\sum_{i=1}^{n} \mathbf{x}_i \mathbf{y}_i}{\sqrt{\sum_{i=1}^{n} (\mathbf{x}_i)^2} \sqrt{\sum_{i=1}^{n} (\mathbf{y}_i)^2}} \quad (1)$$

  The Cosine Similarity of $x = [367, 3547, 6, 615, 7, 3835] \in f_{0A}$, $y = [1801, 15606, 14, 2369, 18, 16334] \in f_{0B}$ is 0.00025. Similarly, the Cosine Similarity of $x = [322, 464, 5, 530, 4, 632] \in f_{1A}$, $y = [5274, 1370, 18, 5975, 28, 456] \in f_{1B}$ is 0.284. Thus, the average Cosine Similarity for Seq_1 and Seq_2 is 0.142.
- **Inter-flow Distance** ($resp\_port$): Inter-flow distance calculates the distance between the $resp\_port$ in every two consecutive flows of a sub-sequence. This helps identify malware network behavioural attributes such as performing a port scan (*e.g.* when the difference of $resp\_port$ of two consecutive flows is 1). To calculate the inter-flow similarity, we first calculate the distance of $resp\_port$ in each consecutive flows in a sub-sequence. For example, the $resp\_port$ distance between $f_{0A}, f_{1A} \in Seq_1$ is 0, as the $resp\_port$ in both flows is the same. However, the $resp\_port$ distance between $f_{0B}, f_{1B} \in Seq_1$ is 55, which is the difference between port 80 and port 25. The inter-flow similarity of Seq_1 and Seq_2 is the distance of (0,55), thus is 55.

  The Inter-flow distance is normalised to obtain a value in $[0, 1]$ to define a dissimilarity. The simplest and most common normalisation method uses a linear transformation, known as feature scaling where $\eta(d) = \frac{d - d\_min}{d\_max - d\_min}$. However, the drawback of applying this method is that it is sensitive to outliers [15]. Therefore, to normalise the distance value, we apply another normalisation method that overcomes this drawback by defining parameter $Z = (m, M)$, where m and M are user-defined values and are interpreted as a tolerance threshold, where $\eta_Z(d) = min(max(\frac{d-m}{M-m}, 0), 1)$ [15]. Any distance value d less than min means that there is zero dissimilarity, thus distinct points (that have a non-zero distance) could be determined identical. Consequently, distance values higher than max are considered totally dissimilar [15]. In our example, we set $Z = (10, 100)$, resulting in a dissimilarity of 0.5.

We define the hybrid similarity approach in Algorithm 1: *Value Similarity*. The Value Similarity function takes as input two flow sub-sequences, X and Y. For each flow f in sub-sequence X and Y, we extract the $port\_protocol\_service$, $history\_state$, and $numeric$ attributes, and calculate the Binary similarity, Levenshtein Distance, and Cosine Similarity, and Inter-flow Distance consequently. The similarity of the pair

Algorithm 1 Value Similarity

procedure VALUE SIMILARITY(X,Y)
  $X \leftarrow [x_1, x_2, ..., x_i]$, *where* $x_i = [f_0, f_2, ..., f_{10}]$
  $Y \leftarrow [y_1, y_2, ..., y_i]$, *where* $y_i = [f_0, f_2, ..., f_{10}]$
  attributes ← *[], attributes of flow in conn.log*
  Sim ← *[]*
  for i *in range* (0,n) **do**
    $x_i \leftarrow X[i]$
    $y_i \leftarrow Y[i]$
    $x_{i+1} \leftarrow X[i+1]$
    $y_{i+1} \leftarrow Y[i+1]$
    $port\_prot\_ser \leftarrow (x_i[0:2], y_i[0:2])$
    $history\_state \leftarrow (x_i[3:4], y_i[3:4])$
    $numeric \leftarrow (x_i[5:], y_i[5:])$
    if $port\_prot\_ser[0] == port\_prot\_ser[1]$ **then**
      $binary = 1$
    else
      $binary = 0$
    ld = 1−LEVENSTEINDISTANCE($history\_state$)
    $cosine$ = 1−COSINEDISTANCE($numeric$)
    $interflow = abs((x_i[0] - x_{i+1}[0]) - (y_i[0] - y_{i+1}[0]))$
    $s = w_0\ binary + w_1\ ld + w_2\ cosine + w_3\ interflow$
    $Sim.append(s)$
  return *Average(Sim)*

Algorithm 2 Order Similarity

procedure ORDERSIMILARITY(X,Y,n)
  $X \leftarrow [x_1, x_2, ..., x_i]$, *where* $x_i = [f_0, f_2, ..., f_{10}]$
  $Y \leftarrow [y_1, y_2, ..., y_i]$, *where* $y_i = [f_0, f_2, ..., f_{10}]$
  $n \leftarrow$ *sequence length*
  if $length(X) == 1$ **then**
    return VALUESIMILARITY(X,Y)$*\frac{1}{n}$
  else
    s_0 = VALUESIMILARITY(X,Y)$*\frac{length(X)}{n}$
    s_1 = ORDERSIMILARITY($X[1:]$,$Y[1:]$,n)
    s_2 = ORDERSIMILARITY($X[:-1]$,$Y[:-1]$,n)
    return Max(s_0,s_1,s_2)

of flows is the sum of the four similarities, each multiplied by a weight w. The weight w is defined as the number of attributes that were given to the similarity measure, *e.g.* $flow\_sim = 3\ binary + 2\ ld + 6\ cosine\_sim + 1\ inter\_flow$. The highest possible similarity score using this approach is 12. The weights w could also be used to give *importance* to a similarity measure over another.

Using our example, X would be Seq_1 and Y is Seq_2. The similarity of Seq_1 and Seq_2 is calculated as follows:
$sim = 3(0.665) + 2(1-0.03) + 6(1-0.142) + 1(1-.5) = 9.5$

As Levenshtein Distance, Cosine Similarity, and Inter-flow Distance are distance measures, we derive the similarity measure through decreasing functions as $S(x,y) = 1 - LevenshteinDistance(x,y)$ and $S(x,y) = 1 - CosineDistance(x,y)$. Thus the similarity measure is the complement to 1 of dissimilarity [15].

2) Malware Family n-flow Mining: To extract the network flow sequence behaviour for a malware family y, we derive the set S of n-flows of all the network flows in the log x that belong to the malware family y. We calculate the similarity $sim(i,s)$ of each n-flow i in a malware sample m_j to each n-flow s in the set S. To calculate the value $sim(i,s)$, we apply the value similarity measure defined in Section IV-B1.

To reduce memory and processing complexity, we pre-compute the similarity of each pair of flows in S and store as a key-value store. Consequently, when determining the similarity of two n-flows, the similarity of each flow is pre-computed and is fetched from the key-value store. Therefore, only unique flow pairs are stored and only the similarity of new flow pairs are computed.

We are interested in mining the malware family's network traffic for the highly frequent n-flows, i.e., flows that occur in all or the majority of the samples of a malware family. Therefore, we represent the n-flow i as a vector in a high-dimensional space, where the elements (*features*) in the n-flow vector corresponds to all n-flows of a malware family y.

3) Profile Selection: In order to extract the malware family profiles, we select the most relevant n-flows (profiles) that represent that malware family's network behaviour. For each malware family, we mine the network flows for all samples of that malware family, then select n-flows that are significant. We apply a modified version of class-wise feature selection approach as proposed in [25]. The class-wise document frequency is the number of network flows in a malware family y that contain an n-flow s. We assign each n-flow a class-dependent weight according to its coverage in the malware family network traffic (*tendency*). As we apply a fuzzy similarity approach, we multiply the average similarity of an n-flow s in a malware family y to its *tendency*. We then select the top k n-flows as profiles for that malware family.

C. Building Malware Family Classification Models

In situations where an analyst detects a malicious network behaviour and needs to determine its origin, the malware's full network traffic may not have been captured. Therefore, to avoid misclassifying malware binaries as a result of not having access to its full packet capture, we classify the network n-flows. This ensures that the malicious binary can be classified based on a sub-set of its network flow communication, thus not requiring the malware's whole network packet capture.

Training. We show in Figure 1 how the classifier is trained to produce the model used for classification of future unseen n-flows. To train the classifier, a collection of malicious n-flows of malware samples $M = m_1, m_2, ..., m_l$ and their corresponding malware families $Y = y_1, y_2, ..., y_k$ are required. Hence, we train a multi-class supervised classifier with the aim of determining if an n-flow i of a malware sample m belongs to a malware family y, where the selected profiles S represent the features in our classifier.

Classification. The trained model is then deployed to classify unseen n-flows to a family. One of the main design goals for *MalClassifier* is resiliency to malware evasion. Malware authors may try to evade the sequence detection by changing the order of the flows or injecting noise flows. Therefore, when deploying the trained model, in addition to calculating the *Value Similarity* we also introduce the *Order Similarity* defined as Algorithm 2. The Order Similarity algorithm computes the

highest Value Similarity score of all possible sub-sequence in X and Y.

Using our example, the possible sub-sequences for X are $(f_{0A}), (f_{0B}, f_{1B}), (f_{1A})$ and for Y are (f_{0B}), $(f_{0B}, f_{1B}), (f_{1B})$, where $length(subsequence) <= n$. The Order Similarity computes the Value Similarity of all possible sub-sequence orders, thus $S(f_{0A}, f_{0B})$, $S((f_{0B}, f_{1B}), (f_{0B}, f_{1B})), S(f_{1A}), (f_{1B}))$. Then, the Value Similarity is multiplied by $length(X)/n$. Thus, the similarity of the sub-sequence is at its highest when the length of sub-sequence for X $= n$, and at its lowest when the length of sub-sequence X $= 1$, thus is a single flow.

V. Evaluation

To evaluate our approach, we implement the system as a multi-threaded Python application. To accelerate the analysis, the application uses Python Multiprocessing with separate threads to calculate the similarity scores for each malware sample. The experiments were conducted on a 40-core processor with 126GB RAM, Centos OS.

A. Dataset

We experiment with a dataset of popular ransomware and botnets network traces, with a total of 11 malware family classes. We provide an overview of the malware families in our datasets in Table III.

Botnets are known to follow a certain infection life-cycle, sharing similarities in network flow characteristics in each infection phase [11]. Thus, we evaluate the effectiveness of our system in determining the unique network flow behaviour of each botnet family in our dataset, and its accuracy in classifying botnet network traffic to its family despite the botnets' network behaviour similarities. To train our classifier we used (1) the CTU-13 botnet traffic dataset [8] provided by the Malware Capture Facility project and (2) current botnets and ransomware (*Miuref, WannaCry, Conficker, Sality, Notpetya*) provided by Stratosphere IPS Project[2].

The CTU-13 dataset contains 13 scenarios of botnet network traffic. The botnet families represented in the dataset employed various protocols (*e.g. IRC, P2P, HTTP* and various techniques (*e.g. sending SPAM, DDoS attacks, Fast-Fluxing*). Our main motivation for using this dataset is that real botnet attacks network traces were captured, providing reliable datasets for model building. In addition to running all samples on the same network environment, precautions were taken to ensure that the full bot network flows were captured and outgoing traffic was not filtered or rate-limited. This ensures that the data is clear from artefacts that can affect the classifier performance, due to how the collection environment was set up. We applied network flows of scenarios $1 - 13$ to train and evaluate the classification models. We excluded scenario 7 as it did not contain a sufficient number of flows. We applied only the malicious flows (C&C and botnet flows) to train the classifier for malware family classification. For more information about the nature of the dataset scenarios and how it was generated see [8].

[2]https://www.stratosphereips.org/

TABLE III: Description of datasets.

Dataset	Family	# Samples	# Flows	# Unique Flows
CTU-13	Murlo	1	37019	422
	Rbot	4	46,184,716	1697
	Virut	2	358,378	4088
	Neris	3	839,077	24895
	Menti	1	291,677	160
	NSIS.ay	1	30,063	2591
Stratosphere IPS Project	Miuref	7	1867273	17257
	Sality	1	6,073,775	307,355
	WannaCry	7	291,677	330
	Conficker	2	323,238	6,300
	Notpetya	4	5,424	174
Total		33	56,302,317	365,289

B. Experiments

The aim of the experiments is to (1) determine how accurately we can classify an n-flow to its malware family using the extracted profiles as features; and (2) determine the classifiers robustness to malware evasion. In particular, we plan to explore the following:

Impact of Flow Sequence Similarity. We measure the effect of applying each similarity approach (*Levenshtein Distance, Cosine Similarity, Binary Similarity, and Inter-flow Distance*), on the classifier performance. This will help determine which similarity measure has the highest positive influence on the classification and thus will be assigned a higher weight w, as discussed in Section IV-B1.

Impact of n-flows. We evaluate the impact of using an n-flow approach for malware family classification. We separate the network flows in our dataset into n-flows of length 1 (single flow) to 7 consecutive flows. The aim is to determine if the malware family classification accuracy improves when a sequence of flows approach such as n-flows is applied as opposed to a single flow.

Robustness to Malware Evasion. Malware authors could attempt to evade detection by changing the malware network flow behaviour, either by injecting noise packets and changing the flow sequence order, affecting the numeric attributes in a flow *e.g.* sent/received packets. To account for this, we explore the robustness of the classifiers' to evasion by evaluating the classifier's performance in classifying n-flows when we randomly shuffle the order of the flows in a sequence.

C. Classifier Design

We build multi-class classifiers that map an unknown network n-flow input as belonging to one malware family.

Classifiers. We apply two supervised machine learning classifiers, K-Nearest Neighbour (KNN) and Random Forest (RF). These classifiers were chosen due to their popularity in text classification using a vectorial representation of features. KNN is a non-parametric lazy learning algorithm. It simply assumes that the classification of a sample is similar to other samples that are nearby in the vector space. Random Forest is an ensemble classifier that leverages multiple decision trees,

TABLE IV: Example of the selected 2-flows for each malware family in our dataset.

Family	Example 2-flow
Murlo	*135.0-tcp-0-0-0-S0-S-2.0-96.0-0.0-0.0* *135.0-tcp-0-0-0-S0-S-2.0-96.0-0.0-0.0*
Rbot	*14.0-icmp-0-0-0-OTH-0-0.0-0.0-0.0-0.0* *227.0-icmp-0-0-0-OTH-0-0.0-0.0-0.0-0.0*
Virut	*80.0-tcp-0-0-0-RSTO-^hR-2.0-80.0-1.0-44.0* *443.0-tcp-0-0-0-REJ-Sr-2.0-96.0-1.0-40.0*
Neris	*80.0-tcp-http-318-368-RSTO-ShADadR-10.0-1052.0-3.0-496.0* *25.0-tcp-0-0-0-S0-S-2.0-96.0-0.0-0.0*
Menti	*25.0-tcp-0-0-0-S0-S-4.0-192.0-0.0-0.0* *25.0-tcp-0-0-0-S0-S-2.0-96.0-0.0-0.0*
NSIS.ay	*32234.0-udp-0-103-596-SF-Dd-1.0-131.0-2.0-652.0* *31037.0-udp-0-103-596-SF-Dd-1.0-131.0-2.0-652.0*
Miuref	*5353.0-udp-dns-1599-0-S0-D-36.0-2607.0-0.0-0.0* *136.0-icmp-0-0-0-OTH-0-1.0-72.0-0.0-0.0*
Sality	*80.0-tcp-0-0-0-S0-S-2.0-96.0-0.0-0.0* *53.0-udp-dns-30-46-SF-Dd-1.0-58.0-1.0-74.0*
WannaCry	*445.0-tcp-0-0-0-REJ-Sr-1.0-52.0-1.0-40.0* *445.0-tcp-0-0-0-S0-S-1.0-52.0-0.0-0.0*
Conficker	*3824.0-tcp-0-0-0-RSTOS0-R-2.0-80.0-0.0-0.0* *3821.0-tcp-0-0-0-RSTOS0-R-2.0-80.0-0.0-0.0*
Notpetya	*130.0-icmp-0-0-0-OTH-0-1.0-72.0-0.0-0.0* *130.0-icmp-0-0-0-OTH-0-1.0-72.0-0.0-0.0*

that are trained using different subsets of the training set, thus overcoming over-fitting issues of an individual decision tree.

Classifier Features. The extracted flow sequence profiles for each malware family represent the features in our classifiers. We only use 20% of the n-flows for each malware family to generate and select ($k = 20$) behaviour profiles. This is to ensure that the profile selection does not bias the classifier estimates, known as *feature subset selection bias* or *selection bias* [31].

Dimensionality Reduction. Due to the sparsity of the dataset (20 profiles * 11 (Number of classes) = 220 features), we apply Principle Component Analysis (PCA) [37]. PCA reduces the dimensionality of the dataset while retaining the variation present in the dataset, up to the maximum extent. Thus, the 220 features are transformed to a new set of features, known as the principal components. We set $PCA = 10$, thus reducing the features space to 10 principle components. We evaluated different values for PCA, and the classifier performance when we apply $PCA = 10$ performs almost as well as when we use all features. The advantage of using PCA over using all features is the low memory complexity required to run the machine learning algorithms due to reducing the dimensions of the features space used by the classifier.

Classifier Performance Measures. To assess the performance of the classifiers, we apply 10-fold cross-validation. Cross-validation removes any bias in the data while maximizing the number of score computations from a given dataset. As the CTU-13 datasets consist of only a few PCAPs (executions) per family, when performing cross-validation we compare the different executions of the same family.

We employ evaluation measures such as *Precision*, *Recall* and *F-measure* to evaluate the classifiers' performance. Although these metrics are defined for a binary classifier, they can be extended for multi-class problems. Each class is represented by a binary classifier *(e.g. One-vs-rest approach*), and we average the binary metric across the set of classes. We also apply a macro-averaged Precision-Recall curve as an evaluation metric, which gives equal weight to the classification of each label compared to micro-averaging which gives equal weight to each per-family classification decision.

VI. Results

We discuss in the following our results from the experiments outlined in Section V-B and the impact of the various system configurations and approaches on the classification performance.

A. Malware Family Profile Extraction

Our results show that malware of a single family exhibit network flow sequence regularities that can be used for malware family classification. We select 20 n-flows for each malware family using the method described in Section IV-B3. We illustrate in Table IV an example of the selected 2-flows for each malware family. *Murlo* traffic contained sub-sequences of TCP flows to destination port 135 (*Messenger Services*). The flows have a connection state $S0$, meaning there was a connection attempt, but no reply. Therefore, the duration of the flow is 0 and there was no payload. Similarly, *Menti* performed a multiple flow connections to destination port 25 (SMTP) and port 21 (FTP). The connection state set to either RSTO, meaning the connection attempt was rejected by the destination or S0. However, there was some successful TCP flow with the exchanged payload. The high number of outgoing flows to an SMTP port means that the botnet is sending email spam, a behaviour linked to *Menti* that is known to send pharmaceutical and stock-based email spam. Although, *Neris* 2-flows show HTTP flows with connection state set to RST0. This means that the connection was established but the originator aborted. However, we do see other successful HTTP connections with connection state SF, meaning normal establishment and termination. *WannaCry* is a ransomware known for sending SMB flows to other victim hosts on the network. This is captured by the selected 2-flows of TCP flows with a destination port of 445. *Miuref's* $2-$flows sequences were mostly HTTPS flows (port = 443), followed by DSN requests or DSN requests followed by an ICMP flow.

B. Classifier Performance

Impact of the Value of n in n-flows. We illustrate in Figure 2 the models' performance for each value of n. Overall, the accuracy of the classifiers is at its best using 2-grams with the KNN classifier and 6-grams with RF classifier. The classifier accuracy was better when flows were represented as a sequence ($n > 1$) rather than individual flows ($n = 1$). This shows that malware behaviour is best captured when we look at sequences of flows. For example, an individual SMTP flow might not infer a malicious behaviour, or capture the behaviour of a particular malware class (*e.g. Menti*), whilst a sequence of rejected SMTP messages gives confidence of a maliciousness of the flow sequence.

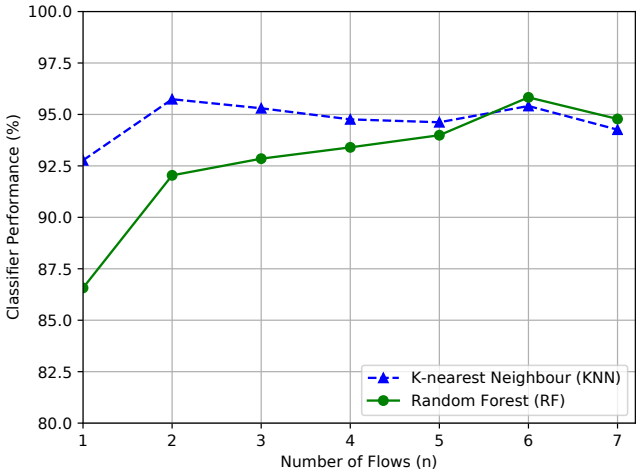

Fig. 2: Average F-measure for Random Forest and KNN classifiers, $n = 1 - 7$ for n-flows.

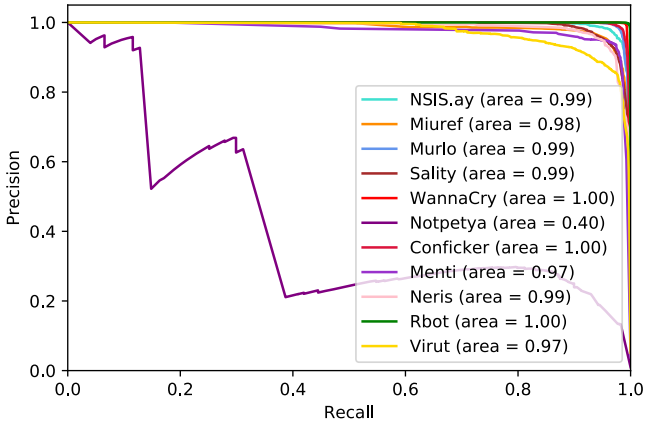

Fig. 3: Macro-averaged Precision Recall Curves for each malware family (*Random Forest classifier,* $n = 5$).

When choosing the value of n, the speed of classification needs to be considered. Although a longer sub-sequence (higher n) results in a higher classification accuracy, it delays the classification of the network behaviour to a malware family. For example, choosing 7-grams requires waiting for 7 consecutive malware network communications to classify and understand its behaviour, increasing the duration of the malware execution damage and delaying active remediation. In situations where the attack implications are severe such as in ransomware attacks, determining the malware class as soon as possible is critical for a quick reaction before it reaches the encryption stage.

Impact of Classification Algorithm. To determine the most accurate classifier, we measured the performance based on the machine learning algorithm used. KNN ($n = 2$) performed best (F-Measure = 95.74%) with shorter flow sequences, while RF classifier's accuracy increased as the number of flows in the sequence (n-flow) increases, reaching 95.83% when $n = 6$. To compare the precision and recall trade-off of the Random Forest classifier, we present the macro-average Precision-Recall (PR) curve when $n = 5$ in Figure 3. We employ macro-average that measures the overall precision and recall of all the classes to produce the macro-averaged ROC curves. We build multi-class models, therefore, averaging the evaluation measures can provide us with a view of the general classifier performance. The model has an AUC of 94%, indicating high recall and high precision. Overall, the classification of all malware families performs well. However, *Notpetya* had an AUC of 39%, with a high number of miss-classifications (42%). We will discuss the reasons for this miss-classifications and how to improve the classification accuracy in Section VII.

Impact of Flow Sequence Similarity. We measure the effect of the four similarity measures (*Binary Similarity, Cosine Similarity, Levenshtein Distance, and Inter-flow Distance*) and their associated flow attributes on the classifier accuracy. Specifically, we train four Random Forest (RF) classifiers, each using a similarity measure and a subset of the flow attributes. For example, we train a classifier using only $resp\_port, protocol, service$, applying only the Binary Similarity measure. We show the F-measure of each of the four classifiers ($n = 5$) per malware class in Figure 4. In the figure, a malware family with a classifier performance of 400% indicates that the Binary, Cosine, inter-flow, and Levenshtein classifiers each had an F-measure of 100%.

The highest performance resulted from the Cosine Similarity classifier, with a 90% F-measure over most malware classes. Binary Similarity classifier was best for representing *Murlo* behaviour with 97.46% F-measure, *Miuref* with 95.86%, and *Rbot* with 99.26%. However, the low performance (11.78%) of the Binary Similarity for *WannaCry* shows that the flows' attributes ($resp\_port, protocol, service$) that are measured using this similarity approach might not be a unique representative of its behaviour. However, *WannaCry*'s Inter-flow Distance classifier had the highest performance (93.59%). Thus, although the exact matching (*i.e. Binary Similarity*) of the res_port did not perform well, the average difference of the res_port between flows in the sequence (*i.e Inter-flow Distance*) was able to capture that malware family's flow sequence behaviour. Levenshtein Distance had the lowest performance overall, with only having high accuracy with Rbot 97.52%.

C. Robustness to Evasion

Malware authors can attempt to evade sequence detection by changing the order of the communication flows or even injecting noise flows. We evaluate the use of the Order Similarity, introduced in Section IV-B1. We randomly shuffle the order of the flows in the $5-flows$ of our malware families and test the model's classification performance on the shuffled $n-$flows. Change in the order of flows did not affect the classifier accuracy when applying Order Similarity, retaining an F-measure of 95.36% for Random Forest Classifier ($n = 5$).

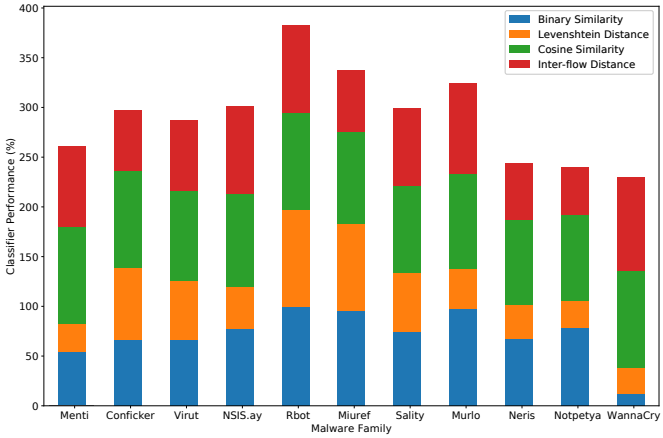

Fig. 4: The malware familys' classification F-measure of the four Random Forest Classifiers ($n = 5$), each using one of the four similarity measures.

VII. Discussion and Future Work

We discuss how *MalClassifier* meets our design goals and identify potential limitations, suggesting approaches to address them.

Meeting Design Goals. *MalClassifier* utilizes non-privacy invasive features to train and build its models, relying on packet header information and not requiring deep packet inspection (*i.e. content-agnostic*). In addition, all identifiable header fields such as IP addresses are removed in the network flow encoding module (*i.e. IP-agnostic*). As malware is known to change its behaviour in order to evade detection, *MalClassifier* applies a *fuzzy* approach to flow sequence similarity to ensure that slight deviations in flow attribute values are detected. The main challenge in malware classification is obtaining the required model training datasets. Therefore, *MalClassifier* uses only non-identifiable packet headers features making datasets needed for training and building the models accessible.

MalClassifier achieved a high accuracy for malware family classification (F-measure $\approx 95.5\%$), demonstrating the effectiveness of the system in identifying distinctive network n-flows for each malware family. It is worth noticing that despite the accuracy of our *MalClassifier* not improving significantly over the state-of-the-art, it still provides a high accuracy while preserving communications privacy and being robust against encryption. In addition, the classifier performance can be improved by modifying the profile selection module, as we will discuss in the next section. *MalClassifier* classifies n-flows to a malware family, meaning it only requires a subset of flows instead of obtaining the full packet trace of the malicious binary for classification.

Understanding Classifier Errors. We represent the confusion matrix for the Random Forest Classifier ($n = 5$) in Figure 5. Each row in the confusion matrix represents the instances of the actual class (*i.e. True Label*) while each column represents the instances of the predicted class (*i.e. Predicted Label*). The main observation is that 26% of *Notpetya* 5-flows where incorrectly classified to *Miuref*. We identified the miss-classified *Notpetya* flows to a sequence of 5 flows of *445-tcp-0-0-0-S0-S-4-192-0-0*. However, such a sequence was not selected as a profile for *Notpetya*, whilst a similar 5-flow *443-tcp-0-0-0-S0-S-1-48-0-0* was selected as a profile for *Miuref*. Therefore, the classifier was trained to assign such an n-flow to *Miuref*. To improve the classification, n-flows that are shared by more than one malware family should be identified and not selected beforehand. This could be done using clustering approaches, which we consider for future work.

Lessons Learned. The performance of the classifier relies on the quality of the profiles selected for each malware family. We introduced a novel method for profile selection that selects n-flows for each malware family using two metrics: (1) average similarity score for that sequence; (2) tendency, the number of times a sequence occurred. In our initial experiments, we noticed that selected flows for a malware family were all similar, as they all have a high score and similar flow attribute values except the destination port. Thus, the profile selection was not capturing the various distinctive behaviour. Accordingly, we amended the selection process to not include the destination port field, selecting a distinctive set of profiles for each malware family. This increased the accuracy of the classifier by 10%, as the profiles selected represented various stages of a malware family network behaviour.

We note that for extracting the sequence profiles we only looked at 20% of the traffic of each malware family. Using these profiles, we evaluated the classifier performance in classifying the other 80% of traffic. However, in application, the profile selection process should consider the various malware network behaviour stages, to ensure that the selected profiles capture the malware behaviour in each infection stage.

We measured the classifier performance using each similarity measure used in the *Value Similarity*. The effect of the similarity measure on each malware family differs. Although Cosine Similarity had a positive effect on the classification accuracy of each malware family, some families were also highly influenced by the Binary Similarity (*e.g. Murlo)*, and Inter-flow Distance (*e.g. WannaCry*). This provides insight on the similarity measures that have the most positive influence on the classifier performance, thus can be assigned a higher weight w as defined in Section IV-B1. Based on the results, we can assign Cosine Similarity the highest weight, followed by Inter-flow Distance, Binary Similarity, and finally Levenshtein Distance.

Malware Evasion. The main challenge in most behavioural-based malware analysis approaches is malware behaviour obfuscation and manipulation, known as *noise-injection* attacks [21]. Although malware evasion by altering the binary itself might be feasible due to available obfuscation tools, we believe that the network behaviour is more troublesome to tamper with. We determine the feasibility of such an evasion in respect of two associated costs: implementation complexity and effect on malware utility. The evasion complexity is based on the ease which the malware author can modify the code to include the evasion tactic which may result in affecting it's

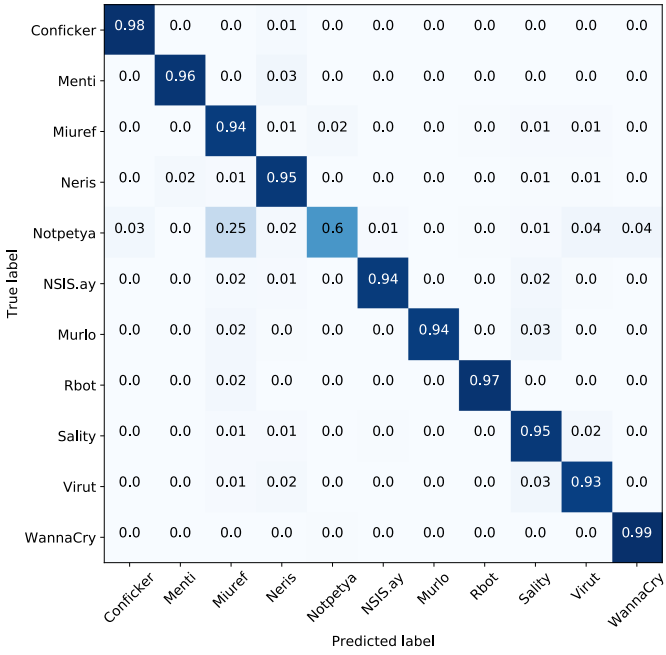

Fig. 5: Normalized confusion matrix showing actual classes vs. predicted classes for the Random Forest Classifier (n = 5).

utility [32]. Malware classification systems that apply supervised machine learning approach require continuous training of new malware variants to adapt to behavioural changes. However, applying a fuzzy similarity measure allows a degree of flexibility in malware behavioural change, thus only needs training on samples of a new malware family. In addition, *MalClassifier* adapts to flow sequence order manipulation by applying the Order Similarity approach.

Limitations and Future Work. We identified that the reason for the classifier misclassifications was due to the feature selection not considering similarities of n-flows between families. Although we discussed how we ensure the selection of distinctive flows for a malware family, these flows should also not be similar to selected flows for other families. For example, *Neris*, *Virut* and *Sality* are all bots that send email spam, and identifying network flow sequences distinctive for each family can avoid n-flow misclassifications. Thus, to improve the profile selection process, we plan to use K-means clustering to identify flows that are similar in more than one family, to avoid using these flows as profiles. Moreover, we plan on identifying the frequent n-flows in *benign* traffic, to reduce the false positives.

As a future work, we also aim to measure the evolution of malware network flow behaviour sequence and determine to what extent does change the sequence behaviour affect the classification accuracy. In particular, we will measure how the network behaviour of malware samples of a family has changed. Identifying behavioural changes of malware samples will assist in measuring how *MalClassifier* classifier performs against these changes. The capability of classifiers to adapt to network behavioural changes ensures classifier accuracy for long periods of time without the need for modifications or costly re-training.

VIII. Conclusion

We present a novel approach for analysing and classifying network traffic of malware variants based on their network flow sequence behaviour. Considering the limitations of existing approaches, we proposed a system that is privacy-preserving, time efficient, and resilient to malware evasion. We showed *MalClassifier*'s effectiveness in identifying frequent malware network n-flows and its robustness against malware evasion by flow order alteration. *MalClassifier* eliminates the need to have access to the malicious binary. This allows SOC analysts to classify malicious network flow sequences on-the-wire, reducing the time and effort required in other dynamic analysis approaches while maintaining a high classification accuracy.

IX. Acknowledgments

Bushra A. AlAhmadi is supported by the Ministry of Higher Education in the Kingdom of Saudi Arabia, the Saudi Arabian Cultural Bureau in London, and King Saud University.

References

[1] L. Axon, B. Alahmadi, J. Nurse, M. Goldsmith, and S. Creese, "Sonification in security operations centres: what do security practitioners think?" Internet Society, 2018.

[2] M. Bailey, J. Oberheide, J. Andersen, Z. M. Mao, F. Jahanian, and J. Nazario, "Automated classification and analysis of internet malware," in *International Workshop on Recent Advances in Intrusion Detection*. Springer, 2007, pp. 178–197.

[3] U. Bayer, P. M. Comparetti, C. Hlauschek, C. Kruegel, and E. Kirda, "Scalable, behavior-based malware clustering." in *NDSS*, vol. 9. Citeseer, 2009, pp. 8–11.

[4] Z. Berkay Celik, R. J. Walls, P. McDaniel, and A. Swami, "Malware traffic detection using tamper resistant features," in *Military Communications Conference, MILCOM*. IEEE, 2015, pp. 330–335.

[5] L. Bilge, D. Balzarotti, W. Robertson, E. Kirda, and C. Kruegel, "Disclosure: detecting botnet command and control servers through large-scale netflow analysis," in *Proceedings of the 28th Annual Computer Security Applications Conference*. ACM, 2012, pp. 129–138.

[6] S. Cesare and Y. Xiang, "Classification of malware using structured control flow," in *Proceedings of the Eighth Australasian Symposium on Parallel and Distributed Computing - Volume 107*, ser. AusPDC '10. Darlinghurst, Australia, Australia: Australian Computer Society, Inc., 2010, pp. 61–70.

[7] E. Gandotra, D. Bansal, and S. Sofat, "Malware analysis and classification: A survey," *Journal of Information Security*, vol. 2014, 2014.

[8] S. Garcia, M. Grill, J. Stiborek, and A. Zunino, "An empirical comparison of botnet detection methods," *computers & security*, vol. 45, pp. 100–123, 2014.

[9] S. García, A. Zunino, and M. Campo, "Survey on network-based botnet detection methods," *Security and Communication Networks*, vol. 7, no. 5, pp. 878–903, 2014.

[10] G. Gu, R. Perdisci, J. Zhang, W. Lee *et al.*, "Botminer: Clustering analysis of network traffic for protocol-and structure-independent botnet detection." in *USENIX Security Symposium*, vol. 5, no. 2, 2008, pp. 139–154.

[11] G. Gu, P. A. Porras, V. Yegneswaran, M. W. Fong, and W. Lee, "Bothunter: Detecting malware infection through ids-driven dialog correlation." in *Usenix Security*, vol. 7, 2007, pp. 1–16.

[12] G. Gu, J. Zhang, and W. Lee, "Botsniffer: Detecting botnet command and control channels in network traffic," 2008.

[13] F. Howard. (2015, jul) A closer look at the angler exploit kit.

[14] J. Z. Kolter and M. A. Maloof, "Learning to detect malicious executables in the wild," in *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2004, pp. 470–478.

[15] M. Lesot, M. Rifqi, and H. Benhadda, "Similarity measures for binary and numerical data: a survey," *Int. J. Knowl. Eng. Soft Data Paradigm.*, vol. 1, no. 1, pp. 63–84, Dec. 2009.

[16] V. I. Levenshtein, "Binary codes capable of correcting deletions, insertions and reversals," in *Soviet physics doklady*, vol. 10, 1966, p. 707.

[17] H. Mekky, A. Mohaisen, and Z.-L. Zhang, "Separation of benign and malicious network events for accurate malware family classification," in *Communications and Network Security (CNS), 2015 IEEE Conference on*. IEEE, 2015, pp. 125–133.

[18] A. Mohaisen, A. G. West, A. Mankin, and O. Alrawi, "Chatter: Classifying malware families using system event ordering," in *Communications and Network Security (CNS)*. IEEE, 2014, pp. 283–291.

[19] J. Oltsik, "Soc-as-a-service for midmarket and small enterprise organizations," The Enterprise Strategy Group, Tech. Rep., mar 2015.

[20] Y. Park, D. Reeves, V. Mulukutla, and B. Sundaravel, "Fast malware classification by automated behavioral graph matching," in *Proceedings of the Sixth Annual Workshop on Cyber Security and Information Intelligence Research*, ser. CSIIRW '10. New York, NY, USA: ACM, 2010, pp. 45:1–45:4.

[21] R. Perdisci, D. Dagon, W. Lee, P. Fogla, and M. Sharif, "Misleading worm signature generators using deliberate noise injection," in *2006 IEEE Symposium on Security and Privacy (S&P'06)*. IEEE, 2006, pp. 15–pp.

[22] R. Perdisci, W. Lee, and N. Feamster, "Behavioral clustering of http-based malware and signature generation using malicious network traces." in *NSDI*, 2010, pp. 391–404.

[23] M. Z. Rafique and J. Caballero, "Firma: Malware clustering and network signature generation with mixed network behaviors," in *International Workshop on Recent Advances in Intrusion Detection*. Springer, 2013, pp. 144–163.

[24] M. Z. Rafique, P. Chen, C. Huygens, and W. Joosen, "Evolutionary algorithms for classification of malware families through different network behaviors," in *Proceedings of the 2014 conference on Genetic and evolutionary computation*. ACM, 2014, pp. 1167–1174.

[25] D. K. S. Reddy and A. K. Pujari, "N-gram analysis for computer virus detection," *Journal in Computer Virology*, vol. 2, no. 3, pp. 231–239, 2006.

[26] K. Rieck, T. Holz, C. Willems, P. Düssel, and P. Laskov, *Detection of Intrusions and Malware, and Vulnerability Assessment: 5th International Conference, DIMVA 2008, Paris, France, July 10-11, 2008. Proceedings*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, ch. Learning and Classification of Malware Behavior, pp. 108–125.

[27] K. Rieck, G. Schwenk, T. Limmer, T. Holz, and P. Laskov, "Botzilla: Detecting the phoning home of malicious software," in *proceedings of the 2010 ACM Symposium on Applied Computing*. ACM, 2010, pp. 1978–1984.

[28] K. Rieck, P. Trinius, C. Willems, and T. Holz, "Automatic analysis of malware behavior using machine learning," *Journal of Computer Security*, vol. 19, no. 4, pp. 639–668, 2011.

[29] C. Rossow, C. J. Dietrich, H. Bos, L. Cavallaro, M. Van Steen, F. C. Freiling, and N. Pohlmann, "Sandnet: Network traffic analysis of malicious software," in *Proceedings of the First Workshop on Building Analysis Datasets and Gathering Experience Returns for Security*. ACM, 2011, pp. 78–88.

[30] I. Santos, Y. K. Penya, J. Devesa, and P. G. Bringas, "N-grams-based file signatures for malware detection." *ICEIS (2)*, vol. 9, pp. 317–320, 2009.

[31] S. K. Singhi and H. Liu, "Feature subset selection bias for classification learning," in *Proceedings of the 23rd international conference on Machine learning*. ACM, 2006, pp. 849–856.

[32] E. Stinson and J. C. Mitchell, "Towards systematic evaluation of the evadability of bot/botnet detection methods." *WOOT*, vol. 8, pp. 1–9, 2008.

[33] G. Suarez-Tangil, J. E. Tapiador, P. Peris-Lopez, and J. Blasco, "Dendroid: A text mining approach to analyzing and classifying code structures in android malware families," *Expert Systems with Applications*, vol. 41, no. 4, pp. 1104–1117, 2014.

[34] Symantec, "Adaptive Behavior-Based Malware Protection," Tech. Rep.

[35] ——. (2016) Internet security threat report.

[36] F. Tegeler, X. Fu, G. Vigna, and C. Kruegel, "Botfinder: Finding bots in network traffic without deep packet inspection," in *Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies*, ser. CoNEXT '12. New York, NY, USA: ACM, 2012, pp. 349–360.

[37] S. Wold, K. Esbensen, and P. Geladi, "Principal component analysis," *Chemometrics and intelligent laboratory systems*, vol. 2, no. 1-3, pp. 37–52, 1987.

[38] C. Wressnegger, G. Schwenk, D. Arp, and K. Rieck, "A close look on n-grams in intrusion detection: anomaly detection vs. classification," in *Proceedings of the 2013 ACM workshop on Artificial intelligence and security*. ACM, 2013, pp. 67–76.