Evolutionary Algorithms for Classification of Malware Families through Different Network Behaviors

M. Zubair Rafique, Ping Chen, Christophe Huygens, Wouter Joosen
iMinds-DistriNet, KU Leuven, 3001 Leuven, Belgium
{zubair.rafique,ping.chen,christophe.huygens,wouter.joosen}@cs.kuleuven.be

ABSTRACT

The staggering increase of malware families and their diversity poses a significant threat and creates a compelling need for automatic classification techniques. In this paper, we first analyze the role of network behavior as a powerful technique to automatically classify malware families and their polymorphic variants. Afterwards, we present a framework to efficiently classify malware families by modeling their different network behaviors (such as `HTTP`, `SMTP`, `UDP`, and `TCP`). We propose protocol-aware and state-space modeling schemes to extract features from malware network behaviors. We analyze the applicability of various evolutionary and non-evolutionary algorithms for our malware family classification framework. To evaluate our framework, we collected a real-world dataset of $6,000$ unique and active malware samples belonging to 20 different malware families. We provide a detailed analysis of network behaviors exhibited by these prevalent malware families. The results of our experiments shows that evolutionary algorithms, like sUpervised Classifier System (UCS), can effectively classify malware families through different network behaviors in real-time. To the best of our knowledge, the current work is the first malware classification framework based on evolutionary classifier that uses different network behaviors.

Categories and Subject Descriptors

D.4.6 [**Security and Protection**]: Invasive software (e.g., viruses, worms, Trojan horses); H.3.4 [**Information Storage and Retrieval**]: Systems and Software—*performance evaluation (efficiency and effectiveness).*

General Terms

Algorithms, Experimentation, Security

Keywords

Malware Classification, Network Behaviors, Machine Learning

1. INTRODUCTION

The classification of the diverse and massive amount of malware variants into families is one of the most challenging problem faced by the security community. A recent report by security company McAfee indicates a steep rise in distribution of malware variants, with the collection of more than 128 million samples in duration of one year [16]. Another report [10] demonstrates the same trend with the collection of $200,000$ malware samples each day by security analysts. This problem has persisted for many years and is only getting worse owing to easy availability of ready-made infection vectors (e.g., exploit kits [20]) and existence of disreputable malware crafting and dissemination services [12][7].

The traditional host-based defenses, like anti-virus softwares, lag behind to contend with this enormous increase of malware. This is largely due to extensive exploitation of evasive techniques (e.g., repacking, polymorphism, and obfuscation) in malware development [26]. Furthermore, the notable prevalence of malware families – `zeroaccess` with approximately 9 million infected PCs [31], `zbot/zeus` with anti-virus detection rate of only 39.35% [32], `ransomware` the number one threat of 2013/2014 [29], and `winwebsec` an infamous fake anti-virus fraud [20] – not only shows the limitations of anti-virus softwares, but also highlights the emerging need for tools to automatically classify unknown variants of known malware families. To this end, security analysts are profoundly inclined to use behavioral attributes for malware classification [26]. Behavioral attributes are effective, as they model specific behaviors that are persistent across the variants of a particular malware family.

One of the powerful behavioral attribute to classify ever-growing malware families and their polymorphic variants is the malware network behavior. The behavior is powerful because a vast majority of real-world malware relies on network communication to perform illegal malicious activities including identity theft, on line advertisement clicks, sending Spam, launching denial of service (DoS) attacks, downloading additional malware and collection of fraudulent ransoms. In addition, inferring knowledge from network behavior (of malware variants) can enable take-downs of malicious servers, identify compromised machines [21], and establish intelligence on the criminals running the flagrant operations [20]. Last but not least, detection of malware through network behavior is more feasible than host-based techniques, facilitating detection at the network edge rather than resource-consuming installations at every end host [24]. Despite the effectiveness of this powerful behavioral attribute, malware family classification through network behavior has received negligible attention in the rich paradigm of evolutionary classification algorithms.

In this paper we performed an empirical study to analyze the applicability of different evolutionary algorithms in classification of malware variants into families using network behaviors. Our study does not focus malware family classification by assuming a single malware family or a single protocol in the malware traffic [21, 19]. In contrast, we here propose a classification framework that extracts features from different network protocols (e.g., `HTTP`, `SMTP`, `UDP`, `TCP`) exploited by variants of multiple, recent and active malware families. The extraction of features from different protocols is important, as malware families can use various protocols for different purposes (e.g., an unknown protocol on `UDP` to communicate with command and control (C&C) and `HTTP` for click-fraud [31]). We therefore use a combination of protocol-aware and state-space modelings to extract features from different network behaviors exhibited by malware samples. For our study, we selected four evolutionary and four non-evolutionary classifiers from different machine learning paradigms. These classifiers are enumerated in Table 1.

We evaluate the performance of the selected classifiers on our collection of real-world active malware samples. Our dataset contains more than 6, 000 unique malware samples. These samples belong to 20 recent and active malware families. We collected the malware samples by using our customized malware-downloader tool and choosing active malware samples from [20]. We executed the collected malware samples in a controlled environment to obtain network behaviors for feature extraction and classification.

The results of our experiments show that our framework, using evolutionary classifiers (like UCS), achieves high accuracy in classifying malware samples of different families. Moreover, the processing overheads of our framework are low; as a result, it can be easily deployed for larger datasets. Finally, we make the implementation of our feature extraction tool publicly available to promote further research and allow future scholars to compare their results to ours.

The remainder of this paper is organized as follows. We present our classification framework in Section 2. We explain feature extraction schemes in Section 3. We discuss the classifiers in Section 4 and describe the evaluation strategy in Section 5. We report the results of our experiments in Section 6. Finally, we discuss the related work on malware family classification in Section 7 and then conclude the paper with an outlook to our future work in Section 8.

2. FRAMEWORK OVERVIEW

Our framework is designed to automatically learn different network behaviors of malware from the labeled binaries and extract discriminative features to classify unknown variants of known malware families. We set the following requirements for our framework: (1) it should correctly identify the unknown variants of known malware families (*malware family classification*), (2) the learning process must be robust to consolidate different network behaviors used by malware to communicate with its C&C (*multiple protocol incorporation*) and (3) the classification process should be efficient in terms of processing overheads (*real-world deployable*). Our framework is modular in nature, allows it to operate on different sets of attributes and with large variety of classification algorithms. We now discuss the different components of our system to classify malware families.

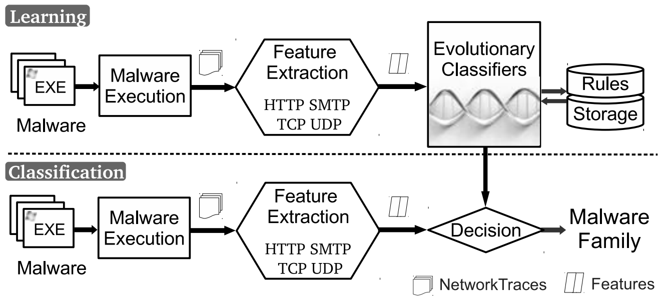

Figure 1: Architecture of malware family classification framework using different network behaviors.

Figure 1 shows the architecture of our framework. Our framework takes malicious executables as an input. These executables are marked with malware family label and can be obtained from different sources and honey-pots.

Next, the malware execution module runs each malware binary in a controlled environment. The fundamental objectives of running a malware in a controlled environment are to provoke the malicious executable to generate traffic, to gather different network behaviors such as `HTTP`, `SMTP`, `TCP` and `UDP`, and to avoid infections through network. To this end, we automatically execute malware binaries in different virtual machines (such as Virtual-box and Qemu), as provoking the malware to communicate through network usually requires executing the same malware sample numerous times in different environments. The traffic generated by executables is stored in Packet `CAP`ture (`PCAP`) format. Note that malware execution is a well studied research problem [12, 20, 21] and is not the main contribution of this paper.

As network traces are obtained, our feature extraction module models the information from different network behaviors of real-world malware. We designed our framework to analyze all kind of network requests generated by the malware. We use two separate modeling schemes, (1) *protocol-aware* modeling and (2) *state-space* modeling. In the first modeling we use protocol grammars to extract fine-grained information from application protocols (e.g., `HTTP`, `SMTP` etc.). The advantage of this modeling is that we can extract distinct information from different fields of known protocol messages. In the second modeling, we use state-space features to deal with unknown protocol requests. This modeling enables us to incorporate expert knowledge from malware network behaviors that do not follow any formal protocol specifications. In the interest of optimization, we discard all requests sent towards benign endpoints by malware. Further, we use gain ratio to select discriminative features for state-space modeling of unknown protocol requests.

The classification module takes the features, from both protocol-aware and state-space modeling, as input and solves a multi-class problem to distinguish instances of different malware families. This problem is solved with the help of rules (population) stored in the rules database. In this study, we analyze the performance of the malware detection framework on wide range of classification algorithms. These algorithms are described in Section 4. The decision made by the classification module can categorize unknown variants of known malware families based on their network behaviors. It can equally be used to identify if a network message belongs to a known malware family.

```
Port Protocol:Payload or Request Line
34354 TCP:e5aac031b4dc3e40315b74084d9b39c1beb4ffb8
53     UDP:4ee3750a8a9c9e981d396c39c7047a624eb8d8f1
16471 UDP:9dc8708a28948dabc9c0d199940c89e5
80    HTTP:GET /stat2.php?w=30461&amp;i=963e6170&amp;
      a=3291195
80    HTTP:GET /56995-B8EBEDA4/counter.img?
      theme=1&amp;digits=10&amp;siteId=31226369
80    HTTP:GET /count.php?id=31226369&amp;c=1&amp;
      d=7&amp;s=0
```

Figure 2: Real-world example of single malware family (Zeroaccess) exhibiting different network behaviors.

Our malware classification approach consists of two phases. In the learning phase, network traffic obtained from malware execution (labeled with family name) is passed to the malware family classification system. The classification module of system learns its rules or evolves its population based on extracted features from network behaviors (that correspond to different protocols). For effective classification, the system can be periodically updated to accommodate changes in the network behaviors of malware families. Once the learning phase of system is terminated, it starts operating on network behaviors of malicious binaries to classify malware family based on the knowledge acquired during learning.

3. FEATURE EXTRACTION

The key intuition behind incorporating different network behaviors, exhibited by real-world malware samples, in our feature extraction process is that the patterns in requests generated by malware (when communicating with a malicious server) tend to remain consistent and are not affected by code obfuscation techniques. This is likely due in part to the considerable amount of effort required to change C&C protocol, with code changes in both malware and malicious server. At the same time, incorporating network behaviors in feature extraction process is challenging as malware families often use multiple C&C protocols. For instance, a malware family can use both HTTP protocol and an unknown C&C protocol built on top of TCP to communicate with the malicious server, and additionally even a separate protocol to perform its fraudulent actions (like SMTP for sending Spam or IRC to send bot commands).

To further illustrate this fact, we provide example of Zeroaccess, the most prevalent and active malware family [31]. Figure 2 shows six different network behaviors of Zeroaccess. The malware family uses TCP-based protocol on port 34354 (to communicate with malicious servers), two separate unknown UDP-based protocols on ports 53 and 16471 respectively (for peer-to-peer communication), and three different application protocol (HTTP) requests (to communicate malware affiliate-id and to perform click-fraud [31]). Since malware families can use different protocols, it is important and relevant to extract features from different protocols to achieve better accuracy. Specifically, we argue that classification scheme based on malware network behaviors should accommodate both application and unknown protocols.

As described in Section 2, we use two separate modelings to extract features from malware network behaviors. These are (1) protocol-aware modeling (for application protocol requests like HTTP, SMTP etc.) and (2) state-space modeling (for unknown protocol requests built on transport protocols i.e., UDP and TCP). We now provide details on these schemes.

```
<proto name="http">
 <field name="http.req.method" value="GET"/>
 <field name="http.req.url" value="/stat2.php"/>
 <field name="http.req.ver" value="HTTP/1.1"/>
 <field name="http.req.user_agent" value="Opera 6"/>
 - - -
 - - -
</proto>
  ---------------Feature Embedding---------------
  attribute|http.req.method|http.req.url| - -
  value    |GET            |/stat2.php  | - -
  -----------------------------------------------
```

Figure 3: Protocol-aware feature extraction example.

Protocol-aware Modeling. The main advantage of using protocol-aware modeling is the possibility to extract fine-grained information from different fields of the application protocol messages. Malware families widely use different message fields of known protocols for malicious communication. For instance, in Figure 2 the siteId parameter in uri of second last HTTP request is used by malware to communicate its affiliate-id [31]. Moreover, a number of malware families craft custom protocol requests, which may include peculiar field values (like a customized value of User-Agent header in HTTP [24]). Therefore, it is pertinent to incorporate information from the application protocol fields in our feature extraction process.

We use Wireshark[1] dissectors to parse requests generated by malware into the field structures of known protocols. Wireshark dissectors can be used to parse a wide range of protocols, allowing us to extract information from variety of malware network behaviors. Figure 3 shows the dissection of an HTTP request, initiated by malware, into the structured format and the embedding of message fields into corresponding feature space. Here, we treated each field name as attribute and field value as attribute value. Any changes in the values of particular fields shows a different crafting of C&C message. This difference indicates a network behavior of a different malware family or a different network behavior of the same malware family. Our approach is not constrained to a specific protocol (e.g., only HTTP) or manual incorporation of limited protocol fields (like, http.req.method and http.req.url) as in [21].

One problem of using Wireshark is that it can not automatically parse common protocols on unusual ports. For instance, it cannot automatically parse HTTP on the unusual TCP port 1435, as it only uses the destination port to select the right protocol dissector. This unusual port communication is a common practice of malware families to avoid detection and bypass firewall rules. We therefore use protocol fingerprints for each TCP connection made by malware during execution. These fingerprints are designed to identify application layer traffic by checking the existence of protocol keywords in the starting bytes of a payload (e.g., GET in HTTP or EHLO in SMTP). Once the known protocol is identified, we use appropriate Wireshark dissector to parse the message into the field structure.

State-space Modeling. In compare to protocol-aware modeling, state-space modeling is more challenging as it deals with the network behaviors consisting of unknown protocol requests. For such requests we extract four attributes to transform a given payload into a feature vector. These attributes are: $\langle proto, size, dest\_port, [state\_space]\rangle$. Here,

[1] http://www.wireshark.com

proto represents transport protocol (i.e., `UDP` or `TCP`), *size* is the number of bytes in the payload, and *dest_port* represents the destination port to which an unknown message is sent by the malware. The [*state_space*] maps the payload information of unknown request into the feature vector.

The first two features (*proto* and *size*) are useful as it is less probable that different malware families use the same transport protocol with the same message size for an unknown protocol request. The third feature *dest_port* can be effective if a particular malware family utilize the specific set of ports for unknown protocol requests (e.g., ports 16471 or 34354 in Figure 2). Finally, the [*state_space*] attributes are valuable to extract the information from the byte sequences of unknown protocol requests.

In order to extract [*state_space*], we present a payload P, of an unknown protocol message, as order of bytes $P = \{b_0, b_1, b_2 \ldots b_l\}$, where l is the length of `UDP` or `TCP` payload. Without loss of generality, we can use n-`gram` (joint distribution of bytes) by computing frequency of consecutive n bytes in P to model the information from the payload of unknown protocol requests. However, the major problem with n-`gram` is that such feature-space contains redundant information and the selection of right value of n is not trivial (i.e., if n is small, the accuracy of the system decreases whereas large values of n significantly increase the processing overheads) [23].

We therefore use a first order state-space system to incorporate the relevant information from the payloads of unknown protocol requests. The state-space uses conditional distribution rather than joint distribution, which results in small and discriminative feature-space [23]. We build our system as a set of states $\Im = \{\Im_0, \Im_1, \Im_2 ...... \Im_k\}$, where $\Im_0$ is the initial system state. We apply transformation function φ that operates on `UDP` or `TCP` protocol to transform each byte of an unknown protocol request into a system state as: $(\varphi : b_i \mapsto \Im_i \in \Im)$. With each byte b_i in payload, such that $b_i \to \Im_i$, we compute the probability of byte b_{i+1} followed by byte b_i as a state transition $t_{\Im_i, \Im_{i+1}}$. Finally, the transitions probabilities of a given P between k states are detailed in a state transition matrix as:

$$T(\Im) = \begin{bmatrix} t_{\Im_0,\Im_0} & t_{\Im_0,\Im_1} & \cdots & t_{\Im_0,\Im_k} \\ t_{\Im_1,\Im_0} & t_{\Im_1,\Im_1} & \cdots & t_{\Im_1,\Im_1} \\ \vdots & \vdots & \ddots & \ddots \\ t_{\Im_k,\Im_0} & t_{\Im_F,\Im_1} & \cdots & t_{\Im_k,\Im_k} \end{bmatrix} \quad (1)$$

The total number of states are bounded by the length l of any given unknown protocol payload. The state transitions values represent the complete formation of the unknown protocol payload for a particular malware family. Any variation in the state transition values reflects a dissimilar payload (i.e., a different crafting of unknown protocol message). We use the transition values as [*state_space*] attributes in our defined feature vector for unknown protocol requests.

Optimization. We use two kind of optimizations to make our feature space more discriminative and computationally feasible. First, we filtered all the known protocol requests sent towards the benign top 200,000 Alexa domains. This is because malware often sends requests to benign sites in order to check Internet connectivity. Including these requests, in our feature space, has little impact on classification of malware families and is obviously an additional overhead. Second, we use the gain ratio feature selection scheme to choose the most discriminative transitions in the [*state_space*].

Table 1: Classification algorithms and machine learning paradigms (Evolutionary and Non-Evolutionary).

	Algorithm	Learning Paradigm
Evolutionary	GAssist-ADI	Pittsburgh-Style GBML
	SLAVE	Genetic Fuzzy
	UCS	Michigan-Style GMBL
	XCS	Michigan-Style GMBL
Non Evolutionary	C4.5	Decision Tree Induction
	C-SVM	Support Vector Machine
	KNN	Instance Based Learning
	Naïve Bayes	Statistical Modeling

4. CLASSIFICATION

Our research analyzed the utility of evolutionary algorithms in the classification of malware families through network behaviors. In addition, we also incorporated known non-evolutionary algorithms to gain knowledge about the classification potential of algorithms in our framework. We then evaluate the effectiveness of the selected algorithms based on their classification accuracy and processing overheads (their training and testing times). Our goal is to identify the best algorithm that can accurately classify malware families using different network behavior attributes.

We have used a diverse set of well-known evolutionary and non-evolutionary classification algorithms in our classification module. Our prime driver, for selecting various classification algorithms, was an attempt to cover the various paradigms of evolutionary computing and machine learning. The paradigms and corresponding classifiers used in our study are shown in Table 1. We now briefly describe each algorithm to make the manuscript self contained. Details on these algorithms can be found in the cited references.

4.1 Evolutionary Algorithms

eXtended Classifier System (XCS). XCS is a Michigan-style classifier that derives a set of rules based on accuracy [30]. In the training phase, the input data is used to generate the initial rules. Afterwards, these initial rules are evolved into new rules using a niche Genetic Algorithm (GA) reserved in the population. The fitness of the individual (or rule) is determined based on the accuracy of each rule. The predicted payoff values for all actions are saved in prediction array. Based on the criteria for rules, some of them are also deleted from the population. The evolution of accurate generalizations and accommodation of real values makes XCS suitable to solve various real-world problems.

sUpervised Classifier System (UCS). UCS is also a Michigan style classifier and is based on accuracy [5]. UCS inherits same principle and structure of XCS where an initial population of rules is derived from training data. However, it differs from XCS as (1) it is based on a supervised learning scheme that computes fitness instead of reinforcement, (2) the GA is applied to correct the rule-set for updating its population, and (3) it does not use a prediction array. The on-line learning and evolving abilities of UCS make it useful in efficiently solving different real-world problems.

Genetic clASSIfier SysTem (GAssist). GAssist [3] is a Pittsburgh style classifier based on the approach where each individual in the population depicts a complete solution to the classification problem. It uses GA to evolve the rulesets of the population and exploits a fitness function to get a good balance between complexity and accuracy of the generated rules. The incremental learning with alternating strata (ILAS) windowing approach is used to improve the generalization of rulesets. For this study we have used the adaptive discretization intervals (ADI) rule representation.

Structural Learning Algorithm in Vague Environment (SLAVE). SLAVE is a genetic learning algorithm based on the use of fuzzy logic concepts and the genetic iterative approach [11]. The algorithm elects and gains knowledge by only one fuzzy rule in each iteration. This results in pruning of the search space for potential solutions. Next, it chooses the most suitable prior for the class by fixing a class. Finally, a series of repeated iterations outputs a full rule-set for the classification of instances. Here, the fitness of rules is computed by using completeness and consistency of the rules. The algorithm has been effectively used in several real-world applications and shows promising results [25].

4.2 Non-Evolutionary Algorithms

C4.5 Algorithm. C4.5 generates a decision-tree to interpret the class and is based on the theory of information entropy [22]. Decision trees are mostly adopted to map information about an attribute for getting outcomes of the attribute's target value using some predictive models. To generate decision-tree from training data, C4.5 uses information gain of attributes by grading a set of attributes to an individual class. It then, recursively operates on the subsets of the attributes till each of the attribute is examined or no further information can be achieved by separating the rest of attributes. Finally, the attributes with the highest information gain are chosen to build the classification model.

Support Vector Machine (SVM). SVMs build hyper planes between elements of separate classes and are based on the concept of a decision surface to solve classification problems [13]. SVM converts diverse domain knowledge with overlapping inputs into non-overlapping parametric objects by modeling the input elements to feature space using mathematical functions called kernels. The commonly used kernels in SVM are Linear, Polynomial, Radial Basis Function (RBF) and Sigmoid kernels.

Naïve Bayes Algorithm (NB). Naïve Bayes is a simple probabilistic classifier based on Bayes' theorem [27]. The classifier assumes that the features used in model building are statistically independent. Mathematically, given a feature vector $F = \{f_1, f_2, ...f_n\}$ and class C, the probability of F belongs to C is computed as $P(F/C) = \prod_{i=1}^{|F|} P(F_i/C)$. The disjoint of class conditional feature distributions implies that each distribution can be independently computed as one dimensional distribution. Naïve Bayes is notably suited for a high-dimensional feature space and has been empirically proven to be effective in various real-world applications.

K Nearest Neighbors (KNN). KNN algorithm is a non-parametric way of categorizing instances into classes based on the k closest training instances in the feature space [8]. In KNN, there is no explicit training phase and it only stores features and class information. However, the testing phase is costly (both in time and space) because an instance is classified by a majority vote of its neighbors through computing the least distance from its k nearest neighbors. The majority class of nearest instances advocates the classification of the instance under test.

Table 2: Network behavior analysis of malware families.

Family (Samples)	Reqs.	HTTP	SMTP	TCP	UDP
1.Cleaman (32)	49	100%	0%	0%	0%
2.Coinminer (3)	8	100%	0%	0%	0%
3.Cridex (72)	1,175	100%	0%	0%	0%
4.Cutwail (2)	747	40.0%	56.7%	3.3%	0%
5.Drstwex.A (45)	51	0%	0%	100%	0%
6.Foreign (6)	6	100%	0%	0%	0%
7.Malagent(8)	15	100%	0%	0%	0%
8.Onescan (9)	85	100%	0%	0%	0%
9.Qakbot-AE (11)	65	100%	0%	0%	0%
10.Ramnit (5)	22	0%	0%	100%	0%
11.Simda (20)	60	100%	0%	0%	0%
12.Spybot.bfr (1)	151	6.0%	94.0%	0%	0%
13.Spyeye (7)	27	59.3%	0%	40.7%	0%
14.Suspectcrc (10)	192	100%	0%	0%	0%
15.Waledace.C (14)	14	100%	0%	0%	0%
16.Waledace.R (29)	29	100%	0%	0%	0%
17.Webprotection (3)	17	100%	0%	0%	0%
18.Winwebsec (2541)	3,395	100%	0%	0%	0%
19.Zbot (2095)	53,568	20.0%	0%	3.6%	76.5%
20.Zeroaccess (1087)	251,874	1.3%	0%	0.3%	98.4%
Total: (6000)	311,550	6.2%	0.05%	1%	92.3%

5. EVALUATION

In this section, we present our methodology for assessing and comparing the selected machine learning algorithms. Our leading objective was to investigate the classification potential of various algorithms to categorize real-world malware samples into families. Therefore, we first present our strategy to acquire recent and active malware samples. We then provide detailed analysis on different network behaviors of the real-world malware families. Finally, we explain our experimental setup for the classification of malware samples into families using network behavior attributes.

Dataset. Our dataset contains $6,000$ binaries from 20 different recent and active malware families, shown in Table 2. The dataset contains a broad range of recent malware classes such as fake anti-viruses, malware-kits, and peer-to-peer bots. All the malware families contain a high range of diversity in network behaviors with their traffic exhibiting common obfuscation techniques such as encryption and polymorphism (in IPs, domains, and payloads). The majority of malware binaries in our dataset are obtained from the publicly available `MALICIA` dataset [20]. These malware binaries are collected from different exploit kit servers from March 2012 to February 2013 (we only choose prevalent and active malware families' samples.) Apart from the `MALICIA` dataset, we also developed a malware-downloader to acquire recent malware samples from different infection vectors (such as phishing). This gives us a diverse set of recent malware samples that are not present in `MALICIA` dataset and exist in the wild.

To acquire malware from different feeds we make use of public security forums such as Malware Domain List (`MDL`) and Anti Network Virus Alliance (`ANVA`) [17]. These public forums are used by security analysts and volunteers to report and analyze malicious `URLs`. Our malware-downloader periodically visits these forums and downloads fresh malware samples by visiting the malicious `URLs`. For a downloaded sample, malware-downloader uses the same family label as mentioned in `MDL` or `ANVA` since the links leading to malware

Table 3: Qualitative analysis of HTTP attributes.

Name	Distinct	Missing	Unique	GR
http.user_agent	39	1.2%	0.02%	0.72
http.connection	3	4.0%	0%	0.60
http.accept	6	31.0%	0%	0.40
http.method	2	0%	0%	0.40
http.accept_encoding	4	54.0%	0%	0.30
http.url	3,385	0%	15.0%	0.26
http.host	2,226	0%	6.0%	0.25
http.content_type	5	72.0%	0%	0.18
http.cookie	13	98.0%	0.04%	10^{-4}
http.referer	16	94.0%	0%	10^{-4}
http.accept_language	1	90.0%	0%	0
http.authbasic	2	99.9%	0%	0
http.cache_control	1	74.0%	0%	0
http.content_encoding	1	74.0%	0%	0
http.authorization	2	99.9%	0%	0

samples reported in these forums are manually checked by experts and only verified ground-truth is reported [20].

Table 2 shows our analysis of the dataset of malware families. It presents the number of binaries for each malware family that generate network traffic, number of requests transmitted by malware of each family, as well as the % split of requests according to the network protocols for each family. We found that `HTTP` is the most common protocol used among malware families. However, some malware families do not completely rely on `HTTP`. For instance, we found that malware belonging to `Ramnit` solely uses an unknown binary protocol built on top of `TCP`. Furthermore, `Cutwail` and `Cridex` largely exploits `SMTP` for sending Spam, whereas `Zbot` and `Zeroaccess` heavily rely on `UDP` for malicious communication because of their C&C infrastructure and peer-to-peer nature. In summary, our analysis of recent and active malware samples indicates that the malware families exhibits different network behaviors during execution. Therefore, any scheme based on network behaviors should accommodate different protocols (not only `HTTP` [21]) to accurately classify real-world malware samples into families.

Experimental Setup. For our experiments, we have used the standard implementations of selected classifiers in an open source tool Knowledge Extraction based on Evolutionary Learning (`KEEL`) [2]. The classifiers are trained by using a combination of all extracted network traffic attributes. Afterwards, we used a stratified 10-fold cross validation procedure on requests belonging to different network behaviors extracted from malware dataset.

6. EXPERIMENTS AND RESULTS

In this section we demonstrate the results of our experiments. We first provide qualitative analysis of features obtained from state-space and protocol-aware modelings. This analysis is useful in measuring the effectiveness of algorithms used to classify malware samples into families. We then discuss the classification accuracy of algorithms. Finally, we provide the training and testing time analysis.

Qualitative Analysis of Features. We now analyze some of the discriminative characteristics of the attributes extracted from network behaviors of the malware dataset. We focus on the quality of the extracted attributes using protocol-aware and state-space modelings (see Section 3). Several information and theoretic measures have been used to evaluate the quality of attributes in a given dataset. For our analysis, we employ the widely used gain ratio (`GR`) [9] to analyze the quality of attributes extracted from the protocol-

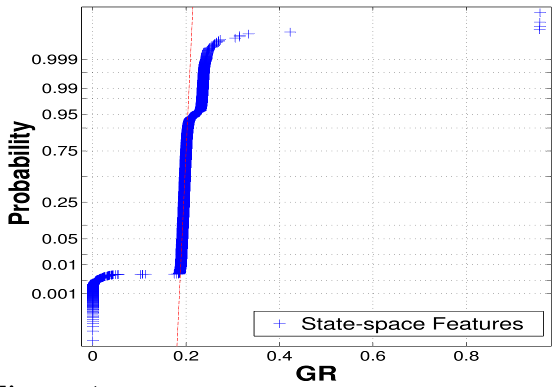

Figure 4: Normal probability plot for gain ratio of [state-space] attributes (unknown protocols).

aware modeling and to select distinctive attributes obtained from the state-space modeling.

Figure 4 shows the normal probability plot for the `GR` of the [*state_space*] attributes. As the value of `GR` ranges from 0 to 1, the values near 1 show a higher discriminative nature of computed transition probabilities and vice versa. Figure 4 shows that there are four attributes showing very high `GR` values. These are attributes present in `UDP` payload (port 16471) of `Zeroaccess` corresponding to the encrypted command `GETL` [31]. Apart from that, majority of [*state_space*] attributes have very low `GR` and can be redundant for classification algorithms. We therefore empirically choose only top 50 attributes with highest `GR` values in [*state_space*] transitions. We also tabulated our analysis on protocol-aware modeling for `HTTP` protocol in Table 3. Table 3 shows attribute names, distinct values of attributes in dataset, % of missing and unique values of attributes in dataset, and finally `GR` of each attribute. The attributes with higher `GR` (like `http.user_agent` etc.) can be regarded crucial for classification of malware families using network behaviors.

Classification Accuracy. The classification accuracy represents the percentage of correctly assigned family labels to malware binaries by using a combination of protocol-aware and state-space attributes. Table 4 shows the per family accuracy achieved by each classifier. It also shows the average accuracy attained by the particular classifier on the complete dataset (Samples).

Training Accuracy. Table 4 shows that all algorithms achieved different training accuracies for each of the 20 selected malware families. Most algorithms achieved high training accuracies for `Winwebsec` (18), `Zbot` (19), and `Zeroaccess` (20). This was an expected result as these families constitutes about 99% of requests generated by all malware samples in our dataset. Therefore, the average accuracy (samples) attained by a majority of the classifiers on the complete malware dataset was quite high. The only exception was SLAVE that achieved very poor training accuracy for `Winwebsec` (18) and `Zeroaccess` (20). The Michigan style UCS achieved training accuracy of 96.49% (per family) in classification of network behaviors exhibited by different malware families. The evolutionary classifiers, GAssist-ADI, SLAVE, and XCS showed poor training accuracies for the majority of malware classes. NB achieved 0% training accuracy in classifying malware belongs to `Foriegn` (6) and `Waledace.C` (15). Similarly, the non-evolutionary classifiers, C4.5 and KNN achieved 0% training accuracy in classify-

Table 4: Classification accuracy in (%) of the selected classifiers on the malware dataset.

	Evolutionary								Non-Evolutionary							
Classifiers	GAssist-ADI		SLAVE		UCS		XCS		C4.5		C-SVM		KNN		NB	
Malware Families	Train	Test	Train	Test	Train	Test	Train	Test	Train	Test	Train	Test	Train	Test	Train	Test
1.Cleaman	9.98	10.20	0	0	100	100	0	0	0	0	100	100	0	0	100	100
2.Coinminer	9.72	12.50	0	0	100	100	0	0	100	100	100	100	100	100	27.78	0
3.Cridex	99.92	99.83	0	0	100	100	51	51.23	100	100	100	94.55	100	100	100	100
4.Cutwail	56.85	33.30	22.22	20	94.75	88.87	1	2.56	64.44	66.67	95.19	69.88	59.75	60	65	65
5.Drstwex.A	100	100	73.33	72.92	100	100	20.70	21.57	100	100	98.04	98.04	100	100	96.08	96.08
6.Foreign	7.41	0	0	0	72.22	83.33	0	0	0	0	100	100	66.67	66.67	0	0
7.Malagent	0	0	0	0	100	73.33	0	0	0	0	100	40	100	100	100	100
8.Onescan	50.20	48.24	0	0	98.17	76.47	0	0	100	100	100	76.47	44.71	44.71	100	98.82
9.Qakbot-AE	41.71	41.54	0	0	100	95.38	0	0	100	98.46	100	95.38	99.66	100	100	100
10.Ramnit	59.26	66.67	100	100	99.07	91.67	0	0	100	100	73.15	70.83	33.33	33.33	58.33	58.33
11.Simda	5	8.33	0	0	96.85	8.33	3.33	5	100	100	100	0	63.70	63.33	100	100
12.Spybot.bfr	54.94	55.56	50	50	100	100	50	50	100	100	100	100	100	100	100	100
13.Spyeye	0	0	0	0	95.96	66.19	1.52	0	98.30	73.86	95.62	79.17	76.33	78.69	24.35	14.39
14.Suspectcrc	68.46	67.71	0	0	97.22	79.17	0.75	0	37.73	35.94	100	64.06	84.84	83.85	99.48	98.96
15.Waledace.C	0	0	0	0	82.54	64.29	0	0	0	0	100	78.57	45.24	50	0	0
16.Waledace.R	2.68	3.45	0	0	93.10	79.31	0	0	0	0	100	62.07	69.73	72.41	100	62.07
17.Webprotection	19.61	23.53	0	0	100	100	0	0	0	0	100	100	0	0	70.59	70.59
18.Winwebsec	99.98	99.88	0	0	99.97	99.35	86.64	86.72	100	99.97	100	99.82	99.91	99.85	100	100
19.Zbot	99.97	99.96	100	100	100	99.99	96.84	96.96	100	100	100	99.57	99.96	99.96	99.83	99.79
20.Zeroaccess	99.77	99.72	49.49	49.49	100	99.88	48.98	98.61	100	100	100	98.93	99.45	99.45	99.33	99.33
Avg.(per Family)	44.27	43.52	17.82	17.71	96.49	85.28	18.04	20.63	65.02	63.75	98.10	81.37	72.16	72.61	77.04	73.17
Avg.(Samples)	99.14	99.19	84.90	84.91	99.96	99.70	94.53	94.53	99.49	99.42	99.99	99.01	99.56	99.55	99.73	99.70

ing `Cleaman` (1) and `Webprotection` (17). C-SVM classifier attained highest training accuracy of 98.10% (per family).

Testing Accuracy. Table 4 shows the testing results for different malware families. The evolutionary classifier UCS achieved the best average testing results with an accuracy of 85.28% (per family). SLAVE had the lowest testing accuracy of 17.71% (per family) and as such was not an adequate algorithm in classifying malware network behaviors. GAssist-ADI and XCS were also below par and achieved a low average testing accuracy of 43.52% and 20.63% (per family) respectively. The non-evolutionary algorithms, C4.5, KNN, and NB achieved average testing accuracy of 63.75%, 72.61%, and 73.17% (per family) respectively. C-SVM was dominant among non-evolutionary classifiers and achieved testing accuracy of 81.37% (per family). However, C-SVM was not able to classify a single variant of malware family `Simda` (11) using network behavior attributes. In conclusion, the Michigan style UCS emerged as the most accurate algorithm for classifying different network behaviors of malware with the prominent accuracy results (99.70% on the complete dataset and 85.28% per family).

For the current problem, it is impossible to achieve complete coverage of malware families by using a single malware network behavior. For instance, malware families like `Drstwex.A` (5) and `Ramnit` (10) solely use `TCP` for malicious communication. Hence, the reasonable design option was to utilize both protocol-aware and state-space attributes for malware family classification. Despite achieving good accuracy on complete dataset, majority of the classifiers achieved low accuracy per family relative to the evolutionary UCS classifier (like NB achieved average accuracy of 99.70% on all malware samples and only 73.17% average accuracy per malware family). This shows that UCS has the ability to transform multi-dimensional knowledge from different malware network behaviors into accurate rules.

Training Time. The training time indicates the time taken by the classifier to build its model on given dataset. The training time is vital for frequently updating the rule-base of the classifiers on additional malware samples. Table 5 shows the time taken by the classifiers during the training phase. SLAVE had consumed more time to evolve its rule as compared to other evolutionary algorithms. The Michigan style LCS (XCS and UCS) spent less time in training than the Pittsburgh style learning classifier GAssist-ADI. The lowest training time was attained by NB, which only took 24.5 seconds to build the model. KNN had no training time as it performed all computations during the testing phase.

Table 5: Average training and testing times.

	Algorithm	Training Time	Testing Time
Evolutionary	GAssist-ADI	34.1min	0.03sec
	SLAVE	>45min	0.02sec
	UCS	8.8min	0.02sec
	XCS	3.7min	0.21sec
Non Evolutionary	C4.5	8min	0.001sec
	C-SVM	27.9min	0.001sec
	Naïve Bayes	24.5sec	0.001sec
	KNN	`n/a`	0.07sec

Testing Time. As for testing time, Table 5 shows that most of the machine learning algorithms took only a fraction of a second to classify a malware family. The C4.5, C-SVM, and NB algorithms had small testing times relative to evolutionary algorithms. The Michigan style XCS had spent more time in testing as compared to all other classifiers. A higher testing time for an algorithm might cause delay in the classification of a malware family.

7. RELATED WORK

While this paper is the first to incorporate evolutionary algorithms in classification of malware families using different network behaviors, there has been an extensive work on malware classification using different techniques. A related survey of these techniques can be found in [4].

Techniques proposed for malware classification using evolutionary classifiers are based on rule learning [28], control flow graph [6], and dependency graph [15] approaches. Recently, the authors of [1] proposed a virus detection clonal algorithm based on clonal selection and genetic algorithm to detect malware. Other technique, IMAD [18] operates by extracting *n*-grams from the system calls made by an executable to detect malware. In contrast to these studies, our work incorporate *different* network behaviors (such as `HTTP, SMTP, UDP, TCP`) from numerous *recent and active* malware families. Also, the goal of our approach is to distinguish between instances of real-world malware families and not to distinguish between benign and malware executables.

There is also a wealth of work that does not involve evolutionary algorithms for malware classification. These studies perform malware classification by incorporating a variety of attributes such as system calls, system changes, network traffic, and screenshots [26, 7, 12]. A number of techniques that involve network-behavior attributes for malware classification using clustering are proposed in [24, 21, 14]. The major drawback of clustering methods is that they do not rely on supervised knowledge to model the information from data. This can be a hurdle in the *automatic classification* of malware and may result in inconsistent performance [26].

8. CONCLUSION

In this paper, we presented an efficient malware family classification system that models the protocol-aware and state-space features from different malware network behaviors as potential input attributes for the various classification algorithms. We performed a comprehensive study of four evolutionary and four non-evolutionary classification algorithms. A side contribution of this paper is also the collection and analysis of network behaviors exhibited by real-world malware samples. To our knowledge, this is the first investigative study of evolutionary and non-evolutionary algorithms to efficiently classify malware families using different network behaviors. The focus of our future work will be to study the classification of benign executables against known malware families using network behaviors.

Acknowledgements.

The authors would like to thank the anonymous reviewers for their insightful comments. This research is partially funded by the Research Fund KU Leuven and by the EU FP7 project NESSoS with the financial support from the Prevention of and Fight against Crime Programme of the European Union (B-CCENTRE). Opinions expressed in this material do not necessarily reflect the views of the sponsors.

9. REFERENCES

[1] S. Afaneh et al. Virus detection using clonal selection algorithm with genetic algorithm (vdc algorithm). *Applied Soft Computing*, 13(1):239 – 246, 2013.

[2] J. Alcalá-Fdez et al. KEEL: A Software Tool to Assess Evolutionary Algorithms for Data Mining Problems. *Soft Comput.*, 13(3):307–318, 2008.

[3] J. Bacardit. Analysis of the Initialization Stage of a Pittsburgh Approach Learning Classifier System. In *Proceedings of the 7th Annual Conference on Genetic and Evolutionary Computation*, pages 1843–1850. ACM, 2005.

[4] Z. Bazrafshan et al. A survey on heuristic malware detection techniques. In *Proceedings of the 5th Conference on Information and Knowledge Technology (IKT)*, pages 113–120. IEEE, 2013.

[5] E. Bernadó-Mansilla and J. M. Garrell-Guiu. Accuracy-based Learning Classifier Systems: Models, Analysis and Applications to Classification Tasks. *Evol. Comput.*, 11(3):209–238, 2003.

[6] D. Bruschi et al. Detecting Self-mutating Malware Using Control-flow Graph Matching. In *Proceedings of the Detection of Intrusions and Malware & Vulnerability Assessment*, pages 129–143. Springer, 2006.

[7] J. Caballero et al. Measuring Pay-per-Install: The Commoditization of Malware Distribution. In *USENIX Security Symposium*, 2011.

[8] T. Cover and P. Hart. Nearest neighbor pattern classification. *IEEE Trans. on Information Theory*, 13(1):21–27, 1967.

[9] T. Cover and J. A. Thomas. *Elements of information theory*, volume 306. Wiley Online, 1991.

[10] Why spam is annoying but malware is frightening. `http://blog.emsisoft.com/2013/11/13/annoying-spam/`.

[11] A. González and R. Pérez. Slave: A genetic learning system based on an iterative approach. *IEEE Trans. on Fuzzy Systems*, 7(2):176–191, 1999.

[12] C. Grier et al. Manufacturing Compromise: The Emergence of Exploit-as-a-service. In *Proceedings of the ACM Conference on Computer and Communications Security*, pages 821–832. ACM, 2012.

[13] T. Hill and P. Lewicki. *Statistics: methods and applications.* StatSoft, 2006.

[14] J. Jang et al. BitShred: Feature Hashing Malware for Scalable Triage and Semantic Analysis. In *Proceedings of the 18th ACM Conference on Computer and Communications Security*, pages 309–320. ACM, 2011.

[15] K. Kim and B.-R. Moon. Malware Detection Based on Dependency Graph Using Hybrid Genetic Algorithm. In *Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation*, pages 1211–1218. ACM, 2010.

[16] Mcafee threats report: First quarter 2013. `http://www.mcafee.com/au/resources/reports/rp-quarterly-threat-q1-2013.pdf`.

[17] Malware Domain List. `http://malwaredomainlist.com/`.

[18] S. B. Mehdi et al. IMAD: In-execution Malware Analysis and Detection. In *Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation*, pages 1553–1560. ACM, 2009.

[19] A. Mohaisen and O. Alrawi. Unveiling Zeus: Automated Classification of Malware Samples. In *Proceedings of the 22nd International Conference on World Wide Web*, pages 829–832, 2013.

[20] A. Nappa et al. Driving in the Cloud: An Analysis of Drive-by Download Operations and Abuse Reporting. In *Detection of Intrusions and Malware, and Vulnerability Assessment*, volume 7967, pages 1–20. Springer, 2013.

[21] R. Perdisci et al. Behavioral Clustering of HTTP-based Malware and Signature Generation Using Malicious Network Traces. In *Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation*, pages 26–26. USENIX Association, 2010.

[22] J. R. Quinlan. Improved use of continuous attributes in C4.5. *Journal of Art. Int. Res.*, 4:77–90, 1996.

[23] M. Z. Rafique and M. Abulaish. xMiner: Nip the Zero Day Exploits in the Bud. In *Proceedings of the 10th International Symposium on Network Computing and Applications*, pages 99–106. IEEE Computer Society, 2011.

[24] M. Z. Rafique and J. Caballero. Firma: Malware clustering and network signature generation with mixed network behaviors. In *Proceedings of the 16th International Symposium on Research in Attacks, Intrusions and Defenses*, pages 144–163. 2013.

[25] M. Z. Rafique et al. Application of Evolutionary Algorithms in Detecting SMS Spam at Access Layer. In *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation*, pages 1787–1794. ACM, 2011.

[26] K. Rieck et al. Learning and Classification of Malware Behavior. In *Proceedings of the 5th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 108–125. Springer-Verlag, 2008.

[27] I. Rish. An empirical study of the naive Bayes classifier. In *Workshop on Empirical Methods in Artificial Intelligence*, pages 41–46, 2001.

[28] M. Z. Shafiq et al. On the appropriateness of evolutionary rule learning algorithms for malware detection. In *Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation Conference*, pages 2609–2616. ACM, 2009.

[29] Sophos report. `http://www.sophos.com/en-us/medialibrary/PDFs/other/sophos-security-threat-report-2014.pdf`.

[30] S. Wilson. Generalization in the XCS classifier system. In *AGPC*, pages 665–674. Morgan Kaufmann, 1998.

[31] J. Wyke. The zeroaccess botnet: Mining and fraud for massive financial gain. `http://www.sophos.com/en-us/why-sophos/our-people/technical-papers/zeroaccess-botnet.aspx`.

[32] Zeus tracker. `https://zeustracker.abuse.ch/index.php`.