The Journal of Systems and Software 85 (2012) 1650–1672

Contents lists available at SciVerse ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

Malware characteristics and threats on the internet ecosystem

Zhongqiang Chen[a], Mema Roussopoulos[b], Zhanyan Liang[c], Yuan Zhang[d], Zhongrong Chen[e], Alex Delis[b,*]

[a] Yahoo! Inc., United States
[b] University of Athens, Greece
[c] Guangxi Univ. of Finance & Economics, China
[d] Florida State University, United States
[e] Shire US Inc., United States

ARTICLE INFO

Article history:
Received 6 September 2011
Received in revised form 7 February 2012
Accepted 7 February 2012
Available online 28 February 2012

Indexing Terms:
Malware propagation mechanisms and payloads
Malware characteristics and categorization
Support vector machines
Self-organizing maps

ABSTRACT

Malware encyclopedias now play a vital role in disseminating information about security threats. Coupled with categorization and generalization capabilities, such encyclopedias might help better defend against both isolated and clustered specimens.In this paper, we present Malware Evaluator, a classification framework that treats malware categorization as a supervised learning task, builds learning models with both support vector machines and decision trees and finally, visualizes classifications with self-organizing maps. Malware Evaluator refrains from using readily available taxonomic features to produce species classifications. Instead, we generate attributes of malware strains via a tokenization process and select the attributes used according to their projected information gain. We also deploy word stemming and stopword removal techniques to reduce dimensions of the feature space. In contrast to existing approaches, Malware Evaluator defines its taxonomic features based on the behavior of species throughout their lifecycle, allowing it to discover properties that previously might have gone unobserved. The learning and generalization capabilities of the framework also help detect and categorize zero-day attacks. Our prototype helps establish that malicious strains improve their penetration rate through multiple propagation channels as well as compact code footprints; moreover, they attempt to evade detection by resorting to code polymorphism and information encryption. Malware Evaluator also reveals that breeds in the categories of *Trojan, Infector, Backdoor*, and *Worm* significantly contribute to the malware population and impose critical risks on the Internet ecosystem.

1. Introduction

An avalanche of malware has appeared on the Internet over the last decade whose goal is to compromise the confidentiality, integrity and availability of infected computing systems (Fredrikson et al., 2010). The astonishing growth in malware population is articulated by pertinent encyclopedias: for example, by March 2009, 340,246 specimens of malware are listed in Trend Micro (2011); of which, only 37,950 species were collected before 2000 and 123,802 breeds were compiled in 2006 alone. The survivability of malware on the Internet ecosystem depends on a number of factors, one of which is the use of multiple penetration channels. The Nimda worm, for instance, not only has had much success in corrupting numerous machines by camouflaging itself in *HTML*-based Email messages (CERT, 2001), but has also used backdoors installed by the Code Red II and Sadmind/IIS codes to break into vulnerable machines (CERT, 2001). Besides email and backdoors, security vulnerabilities found in popular web and database services provide malware with a large set of potential penetration channels and greatly increase the speed of penetration as these services eliminate any human intervention in malware replication (Fredrikson et al., 2010). For example, the worm Slammer was reported to have exploited a buffer overflow in *MS-SQL Server Desktop Engine* to infect more than 90% of vulnerable hosts within ten minutes (Moore et al., 2003).

Malware often locates victim systems using an array of mechanisms including host scanning, hit-list gleaning, and network snooping (Paxson et al., 2002). In the process, affected systems are bombarded by voluminous network traffic and a large amount of their computational resources are consumed, inevitably jeopardizing their productivity (Weaver et al., 2003). When successful, the malware frequently implants in infected machines extra pieces of code termed payloads that typically lead to information leakage, distributed denial-of-service attacks, and security degradation

* Corresponding author. Tel.: +30 210 727 5212; fax: +30 210 727 5214
E-mail addresses: zqchen@yahoo-inc.com (Z. Chen), mema@di.uoa.gr (M. Roussopoulos), liangzhanyan@163.com (Z. Liang), yzhang@math.fsu.edu (Y. Zhang), chenzhongrong@hotmail.com (Z. Chen), ad@di.uoa.gr (A. Delis).

0164-1212/$ – see front matter © 2012 Elsevier Inc. All rights reserved.
doi:10.1016/j.jss.2012.02.015

(Skoudis and Zeltser, 2003). To extend its life span, malware not only obfuscates its code with cryptographic techniques, but also resorts to polymorphism and metamorphism (Kawakoya et al., 2010; Leder et al., 2009). In this regard, each of the produced clones diversifies in terms of code sequence and functionality, defeating anti-malware products that exclusively rely on pattern matching for malware detection (Walenstein et al., 2010). By encroaching deeply into the operating systems and applications, malware may stubbornly resist containment and could dramatically increase the difficulty of its removal even after it is detected (Costa et al., 2005).

An obvious defense against malware is to eliminate security vulnerabilities in Internet applications. Unfortunately, stringent product requirements for time-to-market routinely yield loophole-stricken applications and dampen incentives for developing bug-free systems. In addition, the abundance of "kiddy scripts" and cyberspace attack tools, readily available and often in source code, exacerbate the situation, expedite the discovery of new loopholes, and help in the formation of new attack variants (Fredrikson et al., 2010). Consequently, the population of security loopholes has exploded and ample attack opportunities continue to exist for malware species. Other defenses against malware that have been proposed include traffic throttling, domain blacklisting, and end-system collaboration (Thomas and Nicol, 2010; Williamson, 2002). These however require coordination amongst different autonomous domains making them difficult to achieve in practice. As a result, the most common practice is to build malware "defense-frontiers" by using security products such as firewalls, anti-malware applications and intrusion detection/prevention systems. By and large, these products mainly rely on detecting telltale patterns termed signatures derived from already known malicious species and thus, they are ineffective against strains that exploit zero-day vulnerabilities and whose characteristics are not yet available (Walenstein et al., 2010). Finally, to be effective, frequent applications of software patches and signature updates for security products are necessary. Unfortunately, the potential service interruption and side effects of such patches often leads to indefinite postponement, thus further increasing the vulnerability of systems (Sidiroglou and Keromytis, 2005).

Serving as critical resources against cyber-attacks, security advisories such as the Trend Micro and Symantec encyclopedias play a key role as they usually provide detailed descriptions on behavior of known species, which help malware risk analysis and unveil potential threats to the Internet. Unfortunately, today's encyclopedias do not currently offer any automated categorization and generalization functionalities; meanwhile, classifying newly discovered species as well as updating the encyclopedias themselves calls for substantial manual intervention. With limited or no automated operations, it is extremely difficult to rapidly determine the main malware propagation channel, which is critical in achieving timely response. In the same vein, it is equally cumbersome to derive from current security advisories the malware distributions with respect to specific taxonomic features such as *Carried Payload* or *Containment Difficulty* without significant manual work.

In this paper, we propose and implement Malware Evaluator an *automated categorization framework*, that (a) defines taxonomic features to quickly and efficiently characterize malware life-cycle aspects, (b) automatically classifies malware species, tracks their evolution, and evaluates their impact on Internet, (c) helps users develop policy and measures to counter malware attacks, and (d) identifies and categorizes zero-day attacks with the help of other automated malware analysis tools such as *Norman Sandbox* and *GFI Sandbox*. Using machine learning and data mining techniques, Malware Evaluator empowers encyclopedias such as Trend Micro and Symantec with automated categorization and summarization capabilities. Our framework automatically extracts taxonomic features and collects training data from Trend Micro and Symantec encyclopedias to build learning models with support vector machines (SVMs) and gradient boosting decision trees (GBDTs) (Friedman, 1999; Vapnik, 1999); it also visualizes classifications through self-organizing maps (SOMs) (Kohonen, 2000).

We select attributes to help form the feature space of malware species based on their potential information gain. To contain the dimensions in the feature space, we use word-stemming and stop-word removal techniques. Our framework defines uni-labeled as well as multi-labeled taxonomic features: in the former, only a single label can be attached to each malware while in the latter, a piece of malware may simultaneously belong to multiple categories. For each taxonomic feature, Malware Evaluator attempts to construct models with a variety of machine learning methods, and chooses the model with the best performance. More specifically, Malware Evaluator first decomposes the modeling problem into a set of tasks and builds a SVM binary classifier for each task to help differentiate a feature category from the rest; it then constructs a multi-class SVM categorizer and a GBDT classifier on the same training data; finally, it evaluates the resulting models in terms of classification accuracy, precision, and recall, and picks the best performer for the taxonomic feature in question. The constructed learning models help Malware Evaluator systematically organize malware species in a hierarchical manner based on taxonomic features. Moreover, the unique learning and summarization features of Malware Evaluator make it possible to quickly and automatically classify yet-to-be-discovered breeds and assist in evaluating the malware evolution and its risks.

While experimenting with our Malware Evaluator prototype implementation, we have quickly and efficiently confirmed that (1) malware species typically employ multiple attack avenues to improve penetration rate and carry a variety of payloads to maximize the return-on-investment, (2) malware avoids detection by making use of evasion techniques including encryption, polymorphism, and metamorphism, and finally, (3) *Trojan, Infector, Backdoor*, and *Worm* constitute by far the most significant threats to the cyberspace. With Malware Evaluator we eliminate the laborious and time-consuming manual inspection and analysis of existing malware encyclopedias that would otherwise be needed to make such kinds of observations. Moreover, we are able to make observations on the *evolution* of malware over time, enabling us to easily identify trends in malware characteristics, something not currently possible via manual inspection. Finally, by integrating with automated malware analysis tools, the framework is able to recognize zero-day threats that exploit vulnerabilities unknown to the public.

The rest of our presentation is organized as follows: Section 2 sketches related work and Section 3 presents our framework on malware categorization. The classifications based on adopted taxonomic features for each stage of the malware life cycle – creation, penetration and activation, discovery and eradication – are discussed in Sections 4, 5, and 6 respectively. We outline in Section 7 the results regarding malware evolution, its risk to the Internet ecosystem over time, and the applications of Malware Evaluator in real world situations. Finally, we conclude in Section 8.

2. Related work

Malicious software is an all-time high threat to the Internet ecosystem as it can irreparably damage computing systems. Should private information (i.e., passwords, sensitive data, secret business plans, etc.) be exposed, confidentiality is jeopardized. Integrity is questioned if lineage and origin of data cannot be properly authenticated; meanwhile, systems availability deteriorates should services fail to deliver in a timely manner, rendering it imperative to detect and prevent malware. The essential challenge in identifying malware species is to capture the characteristics that

distinguish them from benign software. Any telltale sequence of bytes in malicious binaries or events manifested by malware can obviously act as patterns for malware detection (Shieh and Gligor, 1997). Malware's traces left on infected machines including modifications to file system, manipulations on registry databases, or network activities, which are typically obtained by static analysis of malicious binaries, are also exploited to improve detection accuracy (Moser et al., 2007). Unfortunately, static analysis is hardly effective in fighting against malware species that employ polymorphism and metamorphism as identifying obfuscating and self-modifying code essentially presents an NP-hard problem (Moser et al., 2007). As a result, dynamic analysis is used to reveal the behavior of a malware species by running it under a controlled environment termed sandbox (Willems et al., 2007). The information collected in a sandbox not only enhances the reliability of malware detection, but also makes it possible to identify malware variants with polymorphism and metamorphism. The analysis results of malware species are usually assembled into an encyclopedia – a comprehensive account of malware. For instance, the Trend Micro encyclopedia listed more than 340,246 by the end of last decade (Trend Micro, 2011). The continual increase in the malware population, however, makes it next to impossible for an individual to evaluate every species encountered. This calls for exchange and correlation of security information among encyclopedias.

To address differences in naming conventions and terminologies among security advisories, an effort to introduce "common" malware terminology was reported in Lindqvist (1999). Unfortunately, the sophistication in both malware characteristics and complex behavior renders the creation of a unified nomenclature a challenge. For instance, the simple term *malware* has numerous definitions: in Helenius (2002), malware is defined as a program designed to be intentionally harmful while in Brunnstein (1999), malware is any piece of code whose functionality deviates from its specification. In addition, specific naming conventions for species used by vendors and institutes greatly differ and often lead to misunderstandings (Scheidl, 1999). An early tiered virus naming system (Scheidl, 1999) labels each specimen along three dimensions namely, family, group, and variants. However, it is counterproductive for researchers to wait for available and standardized names for newly discovered malware before conducting pertinent analysis, documentation, and likely quarantine. The way malware is defined certainly designates its universe: the latter in Bontchev (1998) consists of *Bomb, Trojan, Virus*, and *Worm* while in Boney (1999), the universe includes *Trapdoor* and *Backdoor* as well.

Ideally, a malware taxonomy should help identify breeds and clarify differences among species (Landwehr et al., 1994). To be of value, such a taxonomy should arrange species based on features that possess properties of objectivity, operability, and exhaustiveness (Krsul, 1998). An objective taxonomic feature is intrinsic to the specimen under study and should be able to be automatically extracted. Operability requires a feature to be observable and measurable ensuring repeatability. Exhaustiveness warrants the complete coverage of all species in the universe (Lindqvist, 1999). Perhaps more important is that the process of cataloguing species in a taxonomy should be deterministic and specific (Lindqvist and Jonsson, 1997); determinism ensures the automation of the categorization procedure, while specificity guarantees both uniqueness and unambiguity in the produced classification. The complicated nature of contemporary malware makes it challenging to define unambiguous features applicable to the entire population. The key factor *Intent of Creator* of Helenius (2002) in determining the legitimacy of a program is subjective and difficult to measure; the same applies to the feature *User Consent*, the differentiator between malware and benign software (i.e., cleanware) in Rutkowska (2006). The above features render the categorization process observer-dependent and therefore non-deterministic. The lack of determinism and specificity constitute major obstacles for deploying automated classification schemes to achieve pragmatic malware risk analysis.

The current malware growth-rate renders categorization efforts that require manual intervention ineffective; for instance, manually categorizing all entries of the Trend Micro encyclopedia with respect to a variety of taxonomic features would be a daunting task. In Lee and Mody (2006), a malware classification is built on case-based reasoning and designates a piece of code malicious should its minimum distance from samples in malware space be within a specific threshold. Using the unsupervised learning technique of self-organizing map (SOM) (Kohonen, 2000), DeLooze (DeLooze., 2004) groups vulnerabilities into two-dimensional grid according to their textual similarities creating vulnerability clusterings. In Venter et al. (2008), a prototype is described that uses a SOM to initially group vulnerabilities, calibrate the map with labeled samples, and finally, exploit the outcome as a classifier. Reports generated by dynamic malware analysis can be also used to cluster malware specimens (Bailey et al., 2007; Lee and Mody, 2006). In particular, text description generated by dynamic analyses on the behavior of each malware species is transformed into a sequence of words, and then species are grouped together according to their sequential distances; the latter are typically measured by normalized "compression" or "edit" distances. Obviously, the resulting set of malware clusters can serve as a classifier should species within the same cluster come from the same family and therefore, they share similar characteristics. By nature, such clustering methods are unsupervised learning techniques and their grouping process receives no "external" guidance. For instance, by applying clustering based on normalized compression distance to a set of 3698 malware species, 403 clusters are obtained, of which more than 51% has only a single member (Bailey et al., 2007). Evidently, the weak generalization and skewed clustering result is hardly useful in malware classification. Other machine learning and data mining methods including support vector machines and graph kernels are also occasionally applied to malware analysis and categorization (Rieck et al., 2008; Wagner et al., 2009).

Our main objective in proposing Malware Evaluator is to automate the classification process and offer malware generalization. In contrast to prior work, we treat classification as a *supervised* learning problem and build models for taxonomic features using Support Vector Machines (SVMs) and Gradient Boosting Decision Trees (GBDTs); the former minimize classification errors on selected samples (Schoelkopf and Smola, 2002; Vapnik, 1999), while the latter harvest the aggregate effect of an ensemble of weak learners (Friedman, 1999). Compared to alternative learning methods including Naive Bayes and neural networks, SVMs have proved to be a superb choice for tasks involving voluminous and high-dimensional data as well as sparse feature vectors frequently encountered in real applications (Joachims, 1999). The vast malware population and its diverse behavior undoubtedly result in a feature space with very-high dimensions making feature selection a must in order to contain computational intensity. Techniques such as information gain, *Chi*-square, and odds ratio have been successfully used in text classification (Forman, 2003). Moreover, the efficiency on solving the quadratic programming optimization while training SVM binary classifiers has been much improved. For instance, SVM-Light (Joachims, 1999) and SVM-Torch (Collobert and Bengio, 2001) accelerate the training process with working sets and data caching. The SVM modeling complexity can be further reduced with heuristics and domain-specific optimizations (Ferris and Munson, 2003; Keerthi and DeCoste, 2005). Similarly, parallel and distributed computing techniques can be employed to speed-up the constructions of GBDT models (Tyree et al., 2011; Ye et al., 2009).

When a taxonomic feature contains more than two categories, the learning task should differentiate multiple classes and necessitates a multiclass learning model (Allwein et al., 2000). Through a multiclass-to-binary reduction method, this multiclass problem can be decomposed into a set of binary classification tasks each of which distinguishes a category from the rest (Hastie and Tibshirani, 1998). Similarly, the multiclass learning problem can be solved by partitioning categories into opposing subsets using error-correcting (e.g., Hamming) codes so that binary classifiers are trained on subsets instead of individual classes (Dietterich and Bakiri, 1995; Platt et al., 2000). By treating the multiclass learning problem as a monolithic constrained optimization task having a complex quadratic objective function (Bredensteiner and Bennet, 1999), a single multiclass categorizer can be built with the *multiclass-optimization* method. Here, the constrained objective function is decomposed into a series of "small tasks" with each task involving only a subset of training data or constraints and hence, can be solved analytically (Boser et al., 1992; Crammer and Singer, 2001). Malware Evaluator builds SVM models using both the multiclass-to-binary reduction and multiclass-optimization methods while it provides SOM-based visualization for its classifications.

3. The proposed Malware Evaluator framework

In this section, we present the salient features of Malware Evaluator. We initially discuss the stages of malware life-cycle, then provide the rationale for our design choices and outline feature vectors, SVM, GBDT, and SOM-based classifications, learning models and training sets used.

3.1. The malware life-cycle

The ongoing explosion of malware population is accompanied by the emergence of a very large number of variants. Encyclopedias treat malware and variants differently: Symantec advisory lists threats on a family-basis while the Trend Micro maintains separate entry for each piece of either malware or its variant. Hence, while Symantec encyclopedia lists 10,217 families, Trend Micro maintains 340,246 entries (Trend Micro, 2011; Symantec, 2009). The fast growth rate necessitates that encyclopedias cross-reference each other to share and correlate facts since it has become vastly difficult for any single encyclopedia to thoroughly analyze every specimen encountered. For example, Table 1 presents the *WORM_SDBOT.APO* entry from Trend Micro and it evidently references advisories including Kaspersky, Symantec, and McAfee.

Table 1 also shows that Trend Micro provides sufficiently detailed information and organizes its entries in a standardized manner. The description on each worm features three distinct parts: *General, Description*, and *Detail*. *General* and *Detail* maintain fields that characterize the specimen in question. For example, the feature *Distribution Potential* in *General* indicates the geographic spread of the strain; *Payload* in *Detail* describes reported activities conducted by the breed on compromised systems. Compared to other advisories, the Trend Micro encyclopedia defines a rich set of *taxonomic features* to characterize malware species. In addition, its structured organization can be leveraged to enable systematic and automated processing on collected facts. Therefore, we designate Trend Micro as our main information resource for our prototype and transform it into an automated malware classifier.

All malware specimens share a similar life-cycle that consists roughly of the following phases:

1. *Creation*: A fully functioning malware requires complex computer knowledge and programming skill when built from scratch. The availability of source code for certain species, however, makes it straightforward to develop new strains. The modularized design of many specimens also facilitates the generation of variants.
2. *Penetration & Activation*: Once a piece of malware has surfaced in the Internet ecosystem, it has to permeate into victims. Often, its built-in penetration engine makes malware self-propagating and self-replicating. Malware activities are materialized by a variety of payloads implanted on infected machines, which could be automatically unleashed or activated with specific conditions such as time, events, or user operations.
3. *Discovery & Eradication*: Malware may eventually expose itself by its own pernicious activities and visible traces. To avoid detection, malware may resort to evasive techniques such as information encryption and code polymorphism. By infiltrating deeply into OS and applications of infected systems, malware could resist containment and removal.

Strains may exhibit a variety of diverse behaviors in their lifetime ranging from sheerly annoying to extremely malicious. Based on functionality and behavior, Trend Micro encyclopedia categorizes species into groups listed in Table 2.

Comprising 55.02% of the entire repertoire, *Trojans* are by far the largest class. By circumventing normal authentication processes,

Table 1
The *WORM_SDBOT.APO* and *JS_FEEBS.GJ* entries in Trend Micro encyclopedia.

WORM_SDBOT.APO	*JS_FEEBS.GJ*
General	
Malware type: Worm; Aliases: Virus.Win32.Virut.n (Kaspersky), W32/Virut.remnants (McAfee), W32.Virut.B (Symantec), W32/Virut.Gen (Avira), Mal/Heuri-D (Sophos); In the wild: Yes; Destructive: Yes; Language: English; Platform: Windows 2000, XP; Encrypted: Yes Damage potential: High; Distribution potential: High;	Malware type: JavaScript; Aliases: Worm.Win32.Feebs.gen (Kaspersky), W32.Feebs (Symantec), HTML/Feebs.Gen (Avira), Troj/FeebDl-K (Sophos); In the wild: Yes; Destructive: Yes; Language: English; Platform: Windows 98, ME, NT, 2000, XP, Server 2003; Encrypted: Yes; Damage potential: High; Distribution potential: Medium;
Description	
Installation and Autostart Techniques: Upon execution, this memory-resident worm drops a copy of itself as the file BOXUKITEP.EXE in the Windows system folder. It then executes its dropped copy before it terminates and deletes itself. To enable its automatic execution at every system startup, it creates the following registry entries: . . . *Network Propagation and Exploits*: This worm spreads via network shares. It uses NetBEUI functions to get available lists of user names and passwords. It then lists . . . Aside from the obtained credentials, it also uses the following list of user names and passwords: . . .	*Arrival and Installation*: This malicious JavaScript is embedded in a malicious Web site and runs on a system when a user visits the said Web site. It may also arrive as an attachment to spammed email messages. It creates the following registry entries as part of its installation . . . *Download Routine*: When executed, it displays a fake loading page that mimics any of the following Web-based email providers, saying that there is no available . . . *Antivirus Retaliation*: This malicious JavaScript deletes antivirus and security-related registry subkeys from the following key: . . .
Detail	
Memory resident: Yes; Size of malware: 37,888 Bytes; Compression type: Morphine; Payload 1: Terminates processes; Trigger condition 1: Upon execution; Payload 2: Compromises system security;	File type: Script; Size of malware: Varies; Payload 1: Deletes antivirus and security-related registry keys; Payload 2: Downloads files;

Table 2
TrendMicro: Categories of Malware Species.

ID	Name	Description	Num	Pct
1	Trojan	self-contained programs with benign appearance but hidden malicious features	181711	55.02
2	Backdoor	consist of client/server parts, the latter on victims waits for commands from the former	47230	14.30
3	File Infector	virus attaching itself to executables such as EXE and COM to propagate	36536	11.06
4	Worm	self-contain, self-replicate, propagate via networks without human intervention	36331	11.00
5	JavaScript	written in JavaScript and infect other programs in the same language	9196	2.78
6	VBScript	written in VBScript and infect other programs in the same language	5647	1.71
7	HTML Script	written in scripts supported by Web services and propagate via Web	4614	1.40
8	Macro	written in macro programming languages and propagate by attaching to other files	4377	1.33
9	Boot Virus	embed in master boot records (MBR) and is activated when infected systems start	1812	0.55
10	Batch Virus	a virus replicating through multiple channels such as infecting files and boot sectors	1621	0.49
11	Others	this category includes attack tools, malicious mobile code, and jokes	1173	0.36

the *Backdoor* category contributes 14.30% of the collection. Categories *Infector* and *Worm* provide approximately 11.00% a piece to the population. Finally, *Boot Virus* and *Batch Virus* categories are only sparsely populated. The already very large malware population coupled with its high-growth rate make it nearly impossible for a single advisory to effectively capture the entire universe of specimens. This inadvertently leads to bias in collections. For instance, *Trojans* contribute 55.02% to the Trend Micro repertoire but they make up only 29.23% of the Symantec collection.

3.2. Design rationale for Malware Evaluator

The basic template followed by Trend Micro entries consists of three parts as mentioned earlier (Table 1). The *General* part characterizes the specimen using distinct *taxonomic features* including *Malware Type, Language*, and *Damage Potential*. The *Description* part offers an overview of the strain and outlines reported installation mechanisms, activities conducted, and symptoms of infected systems. The *Detail* section analyzes each breed with respect to file compression method, storage footprints and specific-payload data. Information about *taxonomic features* of all encyclopedia entries must be provided by domain experts, making the task labor-intensive and time-consuming. This results in a small portion of entries having values for each defined feature; for instance, only 43,548 out of 340,246 strains list a value for the *Encryption* feature and less than 12.56% of entries are labeled with respect to the *Destructiveness* feature. It is simply impossible to manually categorize the large population in Trend Micro based on a variety of taxonomic features within a reasonable period of time. Malware Evaluator helps in this direction as it automatically derives taxonomic features for species and offers an automated malware categorization and generalization process.

In addition to the features extracted from Trend Micro, our framework defines new characteristics to cover the entire malware lifecycle. For example, we designate features *Attack Avenue, Carried Payload* and *Containment Difficulty* to illustrate malware behavior in stages *Penetration & Activation* as well as *Discovery & Eradication* of its lifespan. Some taxonomic features take a number of values (or classes): the *Damage Potential* feature ranges in the *Low, Medium* or *High* classes, while the newly adapted feature *Carried Payload* assumes thirteen (13) classes including *Execute Arbitrary Command, Terminate Process*, and *Compromise Security*. A taxonomic feature is multi-label if a specimen can be simultaneously assigned to multiple categories of the feature, and uni-label otherwise. In our framework, feature *Carried Payload* is multi-labeled since a malware breed could carry multiple payloads simultaneously to conduct a variety of malicious activities. For instance, worm *WORM_SDBOT.APO* not only kills other normal processes, but also deactivates many security applications. In contrast, feature *Distribution* is uni-labeled as its categories are mutually exclusive, consequently, worm *WORM_SDBOT.APO* is only in category *High* while script *JS_FEEBS.GJ* is considered to have *Medium* distribution capability.

The Malware Evaluator performs categorization in three phases:

- **Malware representation:** the textual description for a specimen in the Trend Micro encyclopedia is converted into a format suitable for machine learning.
- **Construction of learning model:** the framework creates a machine learned model for each taxonomic feature based on the training data automatically collected from both Trend Micro and Symantec encyclopedias.
- **Unlabeled-species classification:** the constructed learning models automate the categorization on unlabeled malware from Trend Micro repertoire as well as strains that may be discovered in the future.

The categorization with respect to a variety of features not only assists the assessment of malware risks to cyberspace, but also helps analyze malware evolution and develop counter-attack measures. Moreover, we visualize malware categorizations with self-organizing maps (SOMs) to further uncover specimen-intrinsic properties.

3.3. Creating malware feature vectors

To transform an encyclopedia entry into a representation suitable for processing by machine learning algorithms, Malware Evaluator first applies a tokenization process to the text of each entry. The text is treated as a bag of words without taking token positions into account. A "feature vector" for the malware in question is then formed where each distinct token is considered as an attribute with its occurrence frequency as the value. The feature space is the assembly of feature vectors for all species and its dimensionality is the key determinant for the computational complexity of the learning task. To reduce the dimensions of feature space, we use a stopword-elimination process to filter out tokens that have only grammatical function but do not add new meaning to the text (Fox, 1992). The stopwords are typically articles, conjunctions, pronouns, and common prepositions. With the stopword-elimination process in place, tokens such as "upon", "this" and "itself" are excluded for the feature vector of the entry *WORM_SDBOT.APO* (Table 1).

Using the *Porter Stemming* algorithm(Porter, 1980), Malware Evaluator further decreases the number of feature space dimensions; the algorithm first conflates tokens into their common stem roots by stripping plurals, past participles, and other suffixes and subsequently, converts them into their lower-case counterparts. For instance, tokens in *WORM_SDBOT.APO* such as "install", "technique" and "automatic" are transformed into stems "instal", "techniqu", and "automat", respectively. The number of feature space dimensions can be dramatically reduced

by selecting attributes that have significant information gain, which measures the contribution to entropy reduction of a feature (Forman, 2003). In particular, the information gain of a term t with respect to a feature having a set of categories $\{l_i, i=1, 2, \ldots, k\}$ can be expressed as $G(t) = -\sum_{i=1}^{k} P(l_i)\log P(l_i) + P(t)\sum_{i=1}^{k} P(l_i|t)\log P(l_i|t) + P(\bar{t})\sum_{i=1}^{k} P(l_i|\bar{t})\log P(l_i|\bar{t})$, where $P(l_i)$ is the probability of category l_i, $P(t)$ is the probability of term t appeared in documents while $P(\bar{t}) = 1 - P(t)$ and finally, $P(l_i|t)$ and $P(l_i|\bar{t})$ are the conditional probabilities of category l_i given term t and $\bar{t}$, respectively. An alternative to the above tokenization approach is the use of *n-gram*ming to create attributes in feature vectors; in this, the text of a malware entry is initially tokenized and then every token is swept with a *n-byte* sliding window to generate its *n-grams*. The resulting *n-grams* are subjected to the aforementioned stemming.

To eliminate biases due to difference in sizes of malware entries, Malware Evaluator normalizes every feature vector to unit length by dividing the weight of each attribute in the feature vector with the vector's Euclidean length. Our experiments show that the classification performance can be improved by scaling each attribute with its inverse document frequency (IDF). More specifically, the weight w_i of the ith attribute of feature vector v with d dimensions is computed as: $w_i = c_i \log(N/n_i)/\sqrt{\sum_{j=1}^{d}(c_j \log(N/n_j)^2}$; here, c_i is the raw term frequency of the ith attribute of vector v, N is the total malware population, and n_i is the number of malware entries that contain the token represented by the ith attribute (Salton and Buckley, 1988). With the above stopword elimination and word stemming, we reduce the feature space dimensionality from 32,975 to 19,556; this can be further reduced to 17,601, should 90% of the attributes with highest information gain be selected.

3.4. Automating malware classification

In order to automate the malware classification, the proposed framework resorts to state-of-the-art machine learning techniques including supervised learning methods such as SVM and GBDT as well as unsupervised learning methods such as SOM. Meanwhile, to attain the best classification performance, the framework employs diversified modeling methods simultaneously based on the same set of training data. The set of training data can be represented as $T = \{\bar{x}_i, y_i\}, (i = 1, \ldots, m)$, here, each data point $\bar{x}_i \in R^d$ with d features has a true label (or class) $y_i \in Y = \{l_1, \ldots, l_k\}$. With the training data at hand, a supervised learning task is to construct a model that balances the classification performance on T and its generalization capability on unseen examples, while a unsupervised learning task tries to find hidden structure in T.

Learning models built by the *Support Vector Machines (SVMs)* attempt to minimize the classification errors on a set of randomly selected samples (Vapnik, 1999). When the label set is $Y = \{l_1 = -1, l_2 = +1\}$, the constructed learning model is a binary classifier separating data points in positive (+1) category from its negative (-1) counterparts with a hyperplane that maximizes the summation of its shortest distances to the closest positive and negative examples. By expressing the separating hyperplane as $\overline{w}\cdot\bar{x} + b = 0$ with the weight vector $\overline{w} \in R^d$, inner-product operator $(\cdot)$, and bias b, the objective function of an SVM-binary classifier is to minimize $\|w\|^2/2 + C_b\sum_i^m \xi_i$, where parameter C_b is a penalty to classification errors, and $\xi_i(\xi_i \geq 0, i = 1, \ldots, m)$ is a non-negative slack variable for the ith sample in T so that $\overline{w}\cdot\bar{x}_i + b \geq +1 - \xi_i$ for positive samples and $\overline{w}\cdot\bar{x}_i + b \leq -1 + \xi_i$ for negative samples. In practice, the objective function is transformed into its Wolfe dual form to maximize $\sum_i^m \alpha_i - 1/2\sum_{i,j}^m \alpha_i\alpha_j y_i y_j \bar{x}_i\cdot\bar{x}_j$, subjected to $0 \leq \alpha_i \leq C_b$ $(i = 1, \ldots, m)$ and $\sum_i^m \alpha_i y_i = 0$. Parameter α_i $(i = 1, \ldots, m)$ is a non-negative multiplier for each constraint. The objective function in the Wolfe form is convex and its constraints also form a convex set, rendering it a convex quadratic programming (QP) problem, thus, its solution is given by $\overline{w} = \sum_i^m \alpha_i y_i \bar{x}_i$. Data points with their corresponding $\alpha_i > 0$ form the set of support vectors $S = \{s_1, s_2, \ldots\}$. An unseen example $\bar{x}$ is assigned label +1, if formula $\sum_{i=1}^{|S|} \alpha_i y_i \bar{s}_i \cdot \bar{x} + b$ is positive, and label -1 otherwise.

If the label set $Y = \{l_1 = 1, \ldots, l_k = k\}$ has $k > 2$, multiclass-SVM learning model is built using two approaches: *multiclass-to-binary reduction* and *multiclass-optimization* methods. In the multiclass-to-binary reduction method, the learning problem is reduced to a set of binary classification tasks and a binary classifier is independently built for each label l_k with the one-against-rest training technique (Hastie and Tibshirani, 1998). In constructing training data for the classifier designated to label l_k, data points with label l_k are considered as positive while the remaining samples are treated as negative; in this manner, we create an SVM-binary learner. Hence, the resulting learning model by the multiclass-to-binary reduction method consists of k binary classifiers (termed SVM-binaries). In contrast, the multiclass-optimization method defines a monolithic objective function with complex constraints covering all classes so that a single multiclass categorizer, termed SVM-multiclass, is created. Similar to a SVM-binary classifier, the objective function for a multiclass categorizer can be defined to minimize $\|W\|^2/2 + C_m\sum_{i=1}^m \xi_i$ subject to $\overline{W}_{y_i}\cdot\bar{x}_i + \delta_{y_i,r} - \overline{W}_r\cdot\bar{x}_i \geq 1 - \xi_i$ $(\forall i, r)$. Here, W is a matrix of weights with size $k \times n$, $\overline{W}_r$ is the rth row of W, $\delta_{i,j}$ is a loss function that generates an output of 1 if $i = j$ and 0 otherwise, and parameter C_m controls the balance between training errors and classification accuracy. Similarly, the objective function of the multiclass categorizer is also converted to its Wolfe dual form for the derivation of its solution (Crammer and Singer, 2001).

From the Wolfe dual form of the objective function for an SVM classifier and its corresponding solution, we can observe that data points appear only with the form of inner-product (i.e., $x_i \cdot x_j$). By specifying a mapping function $\Phi : R^d \mapsto H$, we can transform feature vectors of training data from R^d into a space H with higher dimensions so that the model constructed in H only depends on data points through functions of the form $\Phi(x_i)\Phi(x_j)$. With a kernel function $K(x_i, x_j) = \Phi(x_i)\Phi(x_j)$, we can replace $x_i \cdot x_j$ by $K(x_i, x_j)$ in objective functions and their constraints, and so build the learning model in space H by using K without explicit computation of Φ. Along the same lines, the label of an unseen example can be obtained by computing inner-products of its feature vector and parameter w (or W) via function K instead of Φ.

It is well established that the performance of a machine learning method is governed by a set of tunable parameters such as kernels used in SVM, which could be polynomial, radial basis functions (RBFs), or sigmoid functions (Vapnik, 1999); therefore, the performance may vary along with changes in parameter settings. Also, it is unlikely for a modeling method to deliver satisfactory performance for all learning tasks. Thus, in Malware Evaluator, a malware classification model is also built with *Gradient Boosting Decision Tree (GBDT)* technique, which is an ensemble of weak prediction learners – decision trees (Friedman, 1999). For the given training set T, the GBDT method seeks an approximation $\hat{F}(\bar{x})$ to a function $F^*(x)$ that minimizes the expected value of some specified loss function $L(y, F(\bar{x}))$, which could be squared-error, absolute error, or binomial log-likelihood function. The approximation $\hat{F}(\bar{x})$ can be expressed as a weighted sum of functions $b_i(x)$ chosen from a base set B of decision trees (i.e., the weak learners), and GBDT attempts to minimize the average value of the loss function on T. Starting with a model consisting of a constant function $F_0(\bar{x})$, GBDT incrementally expands the model in a greedy and iterative fashion. More specifically, at the kth iteration, GBDT would select a decision tree $b_k(\bar{x})$ to partition the input space into r disjoint regions $R_{jk}(j = 1,$

2, ..., r, and r is the number of leaf nodes in the tree) and predicts a constant value b_{jk} for each region, so that the decision tree $b_k(\bar{x})$ can be written as $b_k(\bar{x}) = \sum_{j=1}^{r} b_{jk} I(x \in R_{jk})$, where $I(.)$ is the identity function. Then γ_m is chosen to minimize the loss function $\gamma_m = arg\min_{\gamma} \sum_{i=1}^{n} L(y_i, F_{m-1}(\bar{x}_i) + \gamma b_m(\bar{x}_i))$, and the model is updated as $F_m(\bar{x}) = F_{m-1}(\bar{x}) + \gamma_m b_m(\bar{x})$.

Fitting the training set too closely can lead to degradation of the model's generalization ability. One regularization parameter to control data overfitting is the number of gradient boosting iterations (i.e., the number of trees in the model). Similarly, the number of terminal (or leaf) nodes in decision trees can control the maximum level of interactions between variables in the model and, therefore, is able to affect data overfitting. Another regularization method is to use parameter ν, the learning rate, to manipulate model's generalization ability by modifying the update rule as $F_m(\bar{x}) = F_{m-1}(\bar{x}) + \nu \cdot \gamma_m b_m(\bar{x})$, here $0 < \nu \leq 1$. By fitting a base learner on a subset rather than the whole set of training data at each iteration, randomness is introduced into the GBDT algorithm and may help prevent overfitting, thus, such a subsampling technique can regularize the learning process as well.

In contrast to SVM and GBDT, *Self-Organizing Maps (SOMs)* are unsupervised learning methods that transform data point from a high-dimensional space into a low-dimensional one (Kohonen, 2000); during this transformation, similar features of the former space retain their spatially-clustered nature in the latter. Consisting of components called "neurons" that are typically arranged into a hexagonal or rectangular grid, an SOM associates each neuron with a weight vector that has the same dimensionality d as the feature vectors of input data. In training an SOM with competitive learning methods, neurons in different parts of the map are activated by distinct input patterns, while adjacent nodes respond similarly to the same stimulus as follows: Euclidean distances between the feature vector of each input and weight vectors of all neurons in the map are computed and the cell with the smallest distance is designated as the best matching unit (BMU) for the input in question. The weights of the BMU and its neighboring neurons in the SOM lattice are subsequently adjusted so that they resemble the input vector and could become winners with high probability when similar input instances are encountered in the future.

The weight vector w_i for node i decreases with time as well as with its distance from the BMU according to the formula: $w_i(t+1) = w_i(t) + h_{ic}(t)\alpha(t)\{x(t) - w_i(t)\}$, where t is the training epoch, $x(t)$ is the input vector, $h_{ic}(t)$ is the neighborhood function with c representing the BMU, and $\alpha(t)$ is the learning rate that monotonically diminishes with t. The simplest form of neighborhood function $h_{ic}(t)$ is known as *Bubble*; in this, a neighborhood set $N_c(t)$ is defined for each node c and $h_{ic} = 1$ if node $i \in N_c(t)$ or $h_{ic} = 0$ otherwise. Another neighborhood kernel used by Malware Evaluator is the *Gaussian* function expressed as $h_{ic}(t) = \exp(-(\| r_c - r_i \|^2)/(2\sigma^2(t)))$ with r_c and r_i the radius vectors of nodes c and i, respectively, and $\sigma(t)$ a monotonically decreasing function of time t. The monotonicity of α and σ guarantees that the neighborhood for a neuron shrinks over time and vanishes completely at the end of the training process. The fact that a SOM not only adjusts the weight of the winner but also those of its neighboring cells leads to the capability of spatial clustering and topology preservation. By calibrating the SOM lattice with labeled input data so that the BMU for an input sample inherits the label of this sample, the map can serve as a classifier by marking unknown data points with labels of their associated BMUs.

3.5. Developing training data and learning models

Malware Evaluator creates a SVM learning model for each taxonomic feature based on the training data it automatically collects from both Trend Micro and Symantec encyclopedias. The produced training data may be inadequate due to the fact that these two collections that categorize information on taxonomic features are manually generated by domain experts. Malware Evaluator attempts to overcome this issue by expanding the training data with entries that match telltale patterns unique to taxonomic features. The sparse malware feature vectors resulting from the brief malware entry descriptions justify our choice for SVMs in Malware Evaluator. SVMs are a superb choice when it comes to learning tasks that demonstrate dense concepts and sparse feature vectors.

Training data for a taxonomic feature can be expressed as $T = \{\bar{x}_i, y_i\}$, ($i = 1, \ldots, m$), where $\bar{x}_i \in R^d$ is the feature vector for the ith data point and $y_i \in Y = \{l_1, \ldots, l_k\}$ is its true label. The materialization of features and their corresponding T is labor-intensive and requires significant amount of time when performed manually. Malware Evaluator automates the collection of training data by taking advantage of taxonomic features defined in the Trend Micro malware encyclopedia as shown in the *General* and *Detail* parts of worm *WORM_SDBOT.APO* and script *JS_FEEBS.GJ* (Table 1). Trend Micro-extracted training data is insufficient for the creation of a robust learning model as only a small fraction of entries provides information pertinent to the taxonomic feature in question. To address this, our framework enlarges the size of its training data by mining resources in the reference pool of the Trend Micro encyclopedia such as Symantec. In this regard, the Symantec advisory entries *W 32.Virut.B* and *W 32.Feebs* are referenced by *WORM_SDBOT.APO* and *JS_FEEBS.GJ* of Trend Micro respectively; and the corresponding Symantec entries provide information on features including *Distribution* and *Threat Containment*. The Malware Evaluator can be also configured to extend training data with Trend Micro entries that match keywords unique to the taxonomic feature in question.

With the training data for a taxonomic feature at hand, Malware Evaluator can create an SVM-based learning model. If set Y contains only two labels, a binary learning model is materialized. For a taxonomic feature with $|Y| > 2$, the learning problem is decomposed into $|Y|$ binary classification tasks with the multiclass-to-binary reduction method, and subsequently $|Y|$ binary classifiers are obtained with the one-against-rest training method. For a uni-label taxonomic feature, Malware Evaluator creates a multiclass categorizer with the multiclass-optimization training method. Malware Evaluator saves the generated learning models to classify unlabeled malware breeds or newly identified species. For a multi-label taxonomic feature, an unlabeled malware could be assigned to multiple categories as long as the corresponding binary classifiers produce positive values. In contrast for a uni-label taxonomic feature, Malware Evaluator places a malware into the category with the largest output if the model consists of multiple binary classifiers or the class with the highest confidence when a multi-class model is used.

Our framework employs an n-fold cross-validation method to evaluate the performance of a learning model in terms of classification accuracy, precision, recall, and F_β measure. The training data set T is first partitioned into n equally-sized groups $\{T_i, i = 1, \ldots, n\}$, and each group assumes the same malware distribution as T so that each partition contains samples from all possible classes. Subsequently, the cross-validation process carries out the following steps in the ith of its n iterations:

- **Training Phase**: partition T_i is held out to act as the validation set while the remaining $n-1$ partitions are combined together to form a new training set $A_i = \bigcup_{j \neq i} T_j$, used to build a learning model L_i.
- **Labeling Phase**: data points in validation set T_i are labeled with the learning model L_i; a sample in T_i is classified correctly if its assigned label by L_i coincides with its true label.

– **Measuring Phase**: performance metrics including classification accuracy, precision, recall, and F_β-measure for the model L_i are computed.

Classification accuracy is defined as the ratio of the number of correctly classified examples over the size of the validation set. The F_β-measure computes the weighted harmonic mean of precision P and recall R with the formula $F_\beta = (1+\beta^2)PR/(\beta^2 P+R)$. The precision P for a class is the ratio between the number of correctly labeled samples over the total number of samples that are assigned to the class. Recall R is the ratio of correctly labeled samples over the total number of samples that actually belong to the class. By setting $\beta = 1$ so that precision P and recall R are considered to be equally important, we obtain the F_1-measure as $F_1 = 2PR/(P+R)$. Metrics such as classification accuracy and precision for a learning model are the average of measures obtained in the n iterations of the cross-validation process.

3.6. Key taxonomic features

In Malware Evaluator, a set of taxonomic features are defined to characterize the behavior of malware in various stages of its life cycle and each taxonomic feature captures a certain aspect of malware. Table 3 presents the main taxonomic features Malware Evaluator uses; column *Feature* designates the feature name, column *Description* provides a brief summary of the characteristics of the feature and column *Stage* describes the phase in malware life cycle with which this taxonomic feature is associated. For instance, taxonomic feature *Attack Avenue*, which belongs in the *Penetration*-stage of the malware lifecycle, tracks the propagation channels, such as Emails and instant messengers, employed by malware to invade victim systems.

For each taxonomic feature, multiple machine learning techniques including SVM-binary, SVM-multiclass, GBDT, and SOM, are used to generalize the characteristics of malware. Based on performance measures including classification accuracy, precision, and recall, the best model is chosen for every specific taxonomic feature. Column *Model and Accuracy* of Table 3 lists the Malware Evaluator-selected machine learning model for the taxonomic feature in question. For example, the model constructed by the SVM-binary method has the best categorization performance for taxonomic feature Targeted Language, and its classification accuracy is 99.84%. The performance of a machine learning method is typically regulated by a bundle of tunable parameters (e.g., the size of forest in GBDT, the learning rate in SOM, and the kernel in SVM), and therefore, it may vary along with changes in parameter values. In addition, it is impractical to traverse the entire space formed by the combination of tunable parameters, which could be infinite; thus, there may not be a model that is always the winner under all possible settings. Hence, multiple models do appear in Column *Model and Accuracy*. Detailed discussions on taxonomic features of Table 3 are presented in forthcoming sections as column *Section* indicates.

4. The inception of the malware life-cycle

In this section, we elaborate on the taxonomic features that characterize the behavior of malware during its *Creation* stage.

4.1. Affected computing platforms

Just like any other application, malware materializes its services with APIs provided by the OS. Should a specimen be able to penetrate these APIs, it can then successfully encroach on many variants of the OS in question. For instance, script *JS_FEEBS.GJ* is capable of compromising six members of the *Windows* OS family

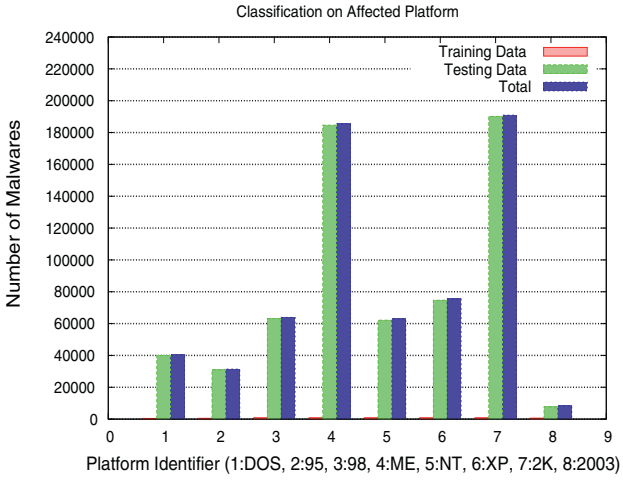

Fig. 1. Classification on Affected Platform with stemming and stopword elimination.

including *Win98* and *Server 2003*. To improve penetration, malware attempts to adjust its attack strategy and pack a different set of code depending on the targeted OS, leading to divergent behavior across different platforms. In this respect, *XML_BADBUN.A* installs *JavaScript* and *IRC* scripts when the victims belong to the *Windows* family while it pushes *Python* and *Perl* scripts to *Linux* distributions. Although a plethora of OSs operate in the Internet ecosystem, opportunistic malware authors often aim at platforms with large user bases or rich applications. Species targeting *Linux, Unix*, and *Macintosh* are only occasionally observed. In contrast, *Microsoft* OSs including *DOS, Win95, Win98, ME, NT, XP, 2000*, and *2003* (including newer versions such as *Vista* and *Win 7*) make up the most favorite targets. Hence, Malware Evaluator defines *Affected Platform* as a core taxonomic feature in terms of OS and focuses on the above eight *Windows* members; the latter are designated as the categories with identifiers 1–8. We treat the *Affected Platform* feature as multi-labeled; *JS_FEEBS.GJ* (of Table 1) clearly shows that malware simultaneously penetrates multiple OSs.

We automatically construct the training data required for the *Affected Platform* feature by extracting Trend Micro entries that have information on the field *System Affected*. For instance, the worm *WORM_SDBOT.APO* (Table 1) may act as candidate for training data of the categories *Win2000* and *XP*. We then generate feature vectors for entries in the training data through the tokenization process and build the *Token-based Learner* model by using the multiclass-to-binary reduction method of Section 3. In the process, we also enable stemming and stopword removal. By categorizing species in the Trend Micro repertoire with the above *Token-based Learner* model which essentially consists of eight binary classifiers, we obtain the malware distribution of Fig. 1.

Clearly, *Win2000* and *WinME* have been the most popular targets while *XP, NT*, and *Win98* also attract many assaults. In contrast, only a few species manage to intrude *Win2003* mainly due to its relatively small user base. The sum population for categories *Win2000* and *WinME* alone is obviously larger than the total number of entities in the Trend Micro encyclopedia (i.e., 340,246), indicating that some malware strains indeed bombard multiple computing platforms at the same time.

Fig. 2 depicts the categorization performance by the eight *Token-based Learner* binary classifiers. The categorization accuracy of the binary learning models ranges between 97.51% achieved by the *WinME* learner to 100.00% attained by the *DOS* classifier. Clearly, all binary classifiers deliver similar categorization precision. Their recall and F_1-measures, however, differ significantly; the *Win2000* categorizer performs the worst with recall 85.25% and F_1-measure 91.36%. The average classification accuracy for the eight binary

Table 3
Taxonomic features, models, and their categorization accuracy.

Stage	Feature	Description	Model and accuracy	Section
Creation	Affected Platform	Vulnerable Operating systems (OSs)	SVM-binaries (98.34)	4.1
	Targeted Language	Attacked Languages	SVM-binaries (99.84)	4.2
	File Type	Types of files in malware	SVM-binaries (98.83)	4.3
Penetration	Attack Avenue	Installation mechanisms to infect systems	SVM (93.12), GBDT (98.63)	5.1
	In the Wild	Executables or source code publicly accessible	GBDT (98.56), SVM (100.00)	5.2
Activation	Damage Potential	Expose confidentials, destroy integrity	SVM-multiclass (99.94)	5.3
	Destructiveness	Damage file systems, stability, and productivity	SVM-multiclass (99.66)	5.4
	Carried Payload	Malicious activities after invading victims	SVM-binaries (96.65)	7.3
Discovery	Message Compression	Compressed files or network traffic	SVM (98.84)	6.1
	Information Encryption	Cryptographic methods employed by malware	SVM-multiclass (99.81)	7.4
	Memory Resident	Stay in main memory after execution	SVM-multiclass (99.31)	7.4
Eradication	Removal Difficulty	Skill, tools, and expertise to remove malware	SVM (96.63), GBDT (98.24)	6.2
	Containment Difficulty	Skill, tools, and expertise to quarantine malware	SOM (95.78), GBDT (97.05)	7.4

classifiers is 98.34%, meanwhile, the average precision, recall, and F_1-measure are 98.95%, 89.38%, and 93.90%, respectively.

To investigate the impact on the classification performance of feature spaces created with tokenization and n-gram processes, we also build an *Ngram-based Learner* model that is trained on feature vectors formed with n-*grams* ($n=3$). We observe that all *Token-based Learner* binary classifiers consistently outperform their counterparts in the *Ngram-based Learner* model. For example, the *NT* classifier in *Token-based Learner* achieves categorization accuracy 97.73%; this is better than the 94.31% attained by its *Ngram-based Learner* counterpart. The average classification accuracy of 98.34% attained by the *Token-based Learner* model is higher than the 95.98% of the *Ngram-based Learner*. Hence, Malware Evaluator uses the tokenization process along with the *Token-based Learner* model as its *default option*.

4.2. Multilingual systems targeted by malware

OSs and applications that make use of national languages often become malware targets. Computing systems and components with localization features turned on frequently introduce language-specific security loopholes and so provide the required launch pads for attacks. For instance, entry *CVE-2008-0730* in the dictionary of common vulnerabilities and exposures (CVEs) records loopholes in input methods for Korean, Thai, and Simplified/Traditional Chinese for *Sun Solaris10* that may grant unauthorized accesses. Similarly, the vulnerability in the Japanese *PHP* Gallery Hosting described in *CVE-2007-5733* may allow for the execution of arbitrary code on affected systems. As a result, opportunistic malware authors often create species that take advantage of language specific features and therefore, they infect only a certain user population. Malware Evaluator defines Targeted Language as a feature that not only designates the language-specific malware behavior but also helps determine the origin, geographical distribution, and infectious species population; the exploitation of language specific loopholes typically requires cultural background and knowledge, implying a strong connection between malware creators and regions where the exploited languages prevail.

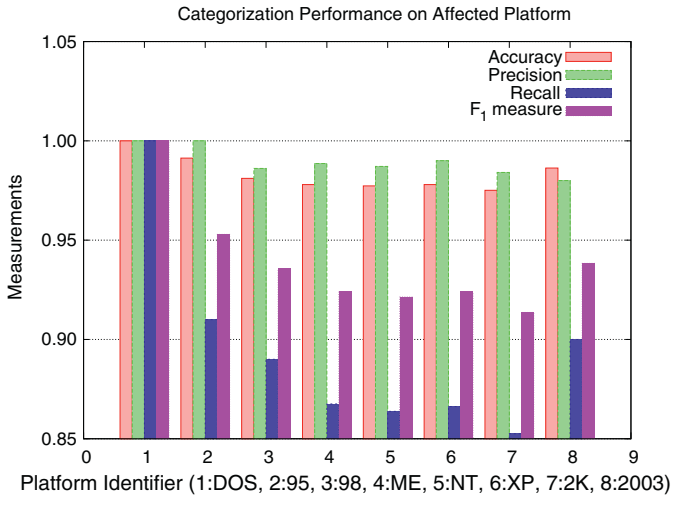

Fig. 2. Classification accuracy, precision, recall, and F_1-measure for Affected Platform.

The taxonomic feature Targeted Language is multi-labeled since a specimen could simultaneously exploit multiple language-specific features. To maximize return-on-investment, opportunistic creators tend to attack languages with large populations. However, it is not uncommon for languages spoken by smaller groups to fall victim occasionally. The latter clearly indicates how widespread malware species are. For instance, hosts using Czech localization are targeted by file infector *HELLWEEN*, while worm *WORM_LASTWORD.A* and Trojan *TROJ_JIGA.A* attack machines localized in Bosnian and Vietnamese languages, respectively. To collect ample training data for a robust learning model, we focus on languages with larger populations as well as rich samples in the Trend Micro encyclopedia, and select Chinese, English, German, Japanese, Portuguese, and Spanish as the categories for the Targeted Language feature; we assign numeric identifiers 1 to 6 to the above languages in respective order.

We retrieve entries with information on field *Language* from the Trend Micro encyclopedia to generate training data for feature Targeted Language. By using the multiclass-to-binary reduction method and setting parameter $C_b=300$, which delivers the best trade-off between F_1-measure and training time in our experiments, we construct a learning model that consists of six binary classifiers, each of which recognizes one category from the Targeted Language feature. Fig. 3 depicts the malware distribution among the different categories obtained by the constructed model. Our main observation is that 79.34% of the species attack English-based computing platforms, while the next frequently targeted language, Chinese, attracts only 15.80% of assaults. Invasions into OSs tailored to Japanese, Spanish, Germany, and Portuguese only occur occasionally. The average classification accuracy achieved by the six binary classifiers of the learning model is 99.84%, and the average precision, recall, and F_1-measure are at 99.68%, 99.11%, and 99.39%, respectively, indicating that the categorization by the model is by and large bias-free.

To evaluate the impact of parameter C_b on the categorization performance, we construct a series of learning models by varying C_b in the range of [50, . . ., 400] and derive their classification accuracy,

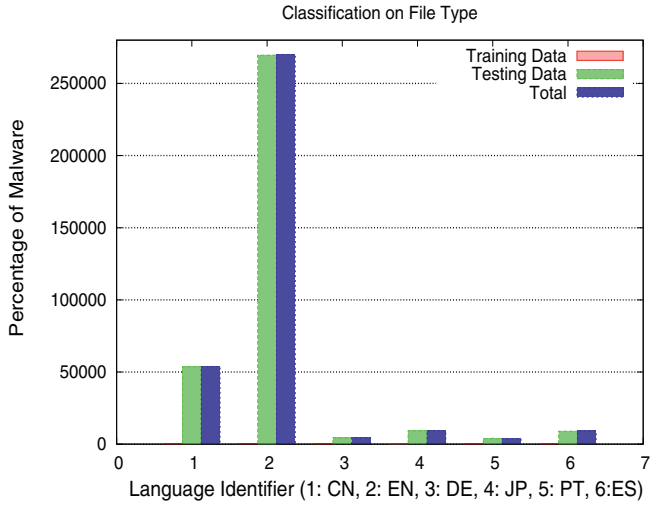

Fig. 3. Malware distribution on feature Targeted Language.

precision, recall, and F_1-measure shown in Fig. 4. The insensitivity of the categorization accuracy to parameter C_b is evident as it performs almost perfectly in the entire chosen C_b spectrum. On the contrary, recall and F_1-measure can be improved significantly by increasing parameter C_b from 50 to 100, however, the enhancement is marginal beyond $C_b > 100$. As expected, a larger C_b imposes heavier penalty on any training error committed by the model and forces the latter to make fewer categorization mistakes to minimize the objective function. Consequently, large C_b values deliver better classification performance at the cost of a smaller margin around the separating hyperplane.

4.3. File types used to encapsulate malware

To be effective once within compromised systems, malware encapsulates its code in one or more file formats that are compatible with those of the targeted environment. A number of attacks use scripting languages such as text-based *JavaScript* and *Perl* to enable rapid development; scripts are essentially independent of the underlying OS. Nevertheless, OS-specific file formats prevail as they help improve code efficiency and provide backward compatibility. In this respect, the executable and link format (`ELF`) is unique to *Linux*, while dynamic linked library (DLL) objects are prevalent in the *Windows* OS family. Obviously, a piece of malware becomes ineffective should it land on working environments incompatible with its own file format(s). Malware Evaluator introduces the File Type taxonomic feature to indicate frequent formats used by malware; classes of the File Type include: *BAT, DLL, DOC, DOS, EXE, HTML, MACRO, PE*, and *SCRIPT*; we assign them identifiers in the range 1, . . ., 9 respectively. As it is common practice for malware to pack a bunch of different type files together, this taxonomic feature is multi-labeled. For instance, in the package of backdoor *BKDR_BIFROSE.YK*, file types *EXE* and *PE* co-exist.

The Trend Micro designated *File Type* field helps Malware Evaluator in the creation of training data for the feature in discussion. For instance, specimen *JS_FEEBS.GJ* (Table 1) is collected into the set of training data as its *Detail* section clearly indicates its file type to be script. By activating stemming and stopword removal, we build a learning model termed *Stemming* with the help of the multiclass-to-binary reduction method. Using the Trend Micro repertoire, we establish that the malware distribution has the following characteristics: 79.95% of the species ship as *PE* files, while 68.29% encapsulate themselves in the *EXE* format. In contrast, only a very small portion of strains pack as *DOC* and *BAT* mainly due to their direct human readability. By evaluating the categorization accuracy of the binary classifiers for the *Stemming* model; we can observe that the highest accuracy is attained by the *DOS* classifier while the lowest by the *PE* learner.

To evaluate the impact word stemming and stopword-elimination have on the classification performance, we build two additional learning models: *No-Stemming* and *No-Stemming-Stopword*. The respective names indicate how these two models function. Our analysis reveals that most binary classifiers in the *No-Stemming* outperform their *Stemming* counterparts. The average classification accuracy of models for *Stemming, No-Stemming* and *No-Stemming-Stopword* models stand at 98.83%, 98.98%, and 98.87% respectively. The fact that the *No-Stemming* and *No-Stemming-Stopword* models offer slightly higher classification accuracies than *Stemming* demonstrates the overhead of the stemming process. In the same manner, the slight deterioration in the categorization accuracy by *No-Stemming-Stopword* if compared to *No-Stemming* manifests the harmful impact of adding stopwords into the feature space. The *No-Stemming* model generates a 36,621-dimensional feature space which is notably larger than the 32,975 dimensions of *Stemming*; *No-Stemming-Stopword* further enlarges the feature space to contain 36,824 attributes as it treats stopwords as features. By default, Malware Evaluator carries out word stemming and stopword removal so that the size of the produced feature space does not adversely affect the model training time.

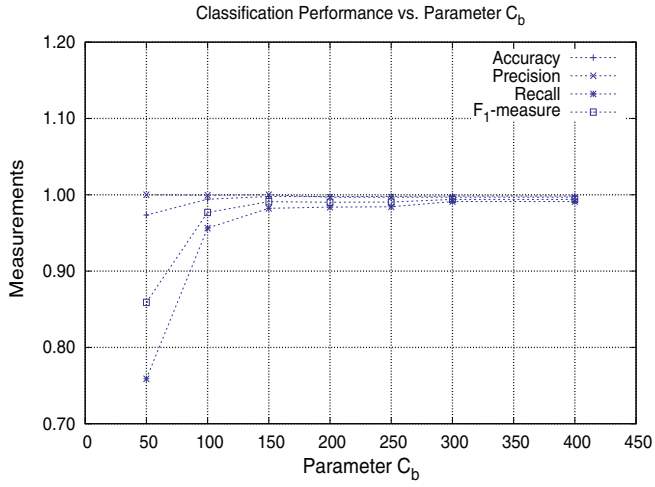

Fig. 4. Classification performance on feature Targeted Language.

5. Penetration and activation

By sowing its code across wide geographical regions and using compromised systems to attack others, malware infects hosts in the Internet with a domino effect. In this section, we identify taxonomic features that play a major role during the *Penetration & Activation* stage of malware life-cycle.

5.1. Propagation mechanisms

At first, a strain often infects a small set of hosts or *seeds* which are then used as launching pads for creating epidemics. The channel used to ship malicious code between hosts is known as the penetration mechanism. This mechanism is materialized through a set of malware components – target selector, penetrator, and code transporter – that coordinate actions among themselves. The selector probes Internet hosts to discover new targets; as soon as a candidate is identified, the penetrator tries to gain an entry by means of vulnerability exploitation or backdoors installed by other species.

Table 4
Major penetration channels utilized by malware species.

ID	Channel	Description	Sample keywords
1	Exploit	Exploits loopholes such as buffer overflow	exploit, vulnerability
2	Email/Attachment	Embed in email bodies or attachments as links or files	email attachment, spam
3	Network Share	camouflage as shared files in trusted networks	shared folder, network share
4	Peer-to-Peer	Ship as shared files in P2P networks such as KaZaA	peer-to-peer, Gnutella
5	Instant Messenger	Spread via instant messengers such as AOL and MSN	internet relay chat, Skype
6	Drive-by Download	Hide in web pages to infect visitors' systems	drive-by, malicious web site
7	Dropped-by Malware	bundle and spread with other software package	dropped by malware

Once the target is corrupted, the code transporter moves the entire malware body into the victim. Malware may become ineffective in bringing about epidemics should all potential propagation channels be blocked. Thus, it is vital to be able to classify penetration mechanisms and so we introduce the Attack Avenue taxonomic feature. Our comprehensive analysis on species listed in Trend Micro has led to the adoption of the main categories listed in Table 4 as far as this feature is concerned.

The Trend Micro encyclopedia does not define the Attack Avenue feature and so, Malware Evaluator resorts to pattern matching to create the required training data. We search the Trend Micro malware encyclopedia for patterns specified in column *Sample Keywords* of Table 4; we consider any entry that yields a match as candidate for training data. In this respect, entry *WORM_SDBOT.APO* (Table 1) makes it into the training data as it matches the *network share* pattern. Feature Attack Avenue is multi-labeled as a malware specimen can simultaneously deploy multiple installation mechanisms. Once training data are in place, we construct a feature space with all possible attributes and build the learning model *Full-Feature* with the help of the multiclass-to-binary reduction method. Fig. 5 shows the malware categorization that *Full-Feature* delivers. Email/attachments make up the prime penetration channel for malware spread as more than 52.15% Trend Micro species are embedded in Email messages as links, macros, or attached entities. By encapsulating themselves into web pages and exploiting misconfigured Web-server hosts that use *ActiveX* and *Java Applets*, approximately 13.00% of malware breeds can infect unguarded systems; this is mostly achieved via drive-by downloads that automatically fetch contaminated web-content (Cova et al., 2010). Peer-to-peer and Instant Messaging (*P2P/IM*) applications also significantly contribute to the proliferation of malware. With the exception of Email/attachments, all other penetration avenues appear to be equally attractive to species; this is an indicator that malware seizes every possible opportunity to infect the Internet ecosystem.

To analyze the effect of the feature space sizes on the classification performance, we build three additional learners based on attribute sets that contain top-50%, top-60%, and top-70%, respectively, of the features with the highest information gains. Fig. 6 shows the performance of the newly trained models along with that of the *Full Feature* model. Overall, differences in classification accuracy, precision, recall, and F_1-measure for the four models seem to be insignificant although the model based on top-70% of features offers slightly better classification precision. It is also interesting to observe that the learner with the full feature set actually slightly deteriorates in its performance compared to the model built on top-70% attributes, indicating that attributes with low information gains are in fact noise and interfere with the learning capability of the generated model. In comparison, with categorization accuracy 98.63%, the GBDT model built on the same training data delivers much better classification performance (i.e., Table 3).

5.2. In the wild

When a malicious specimen is either active or its code is encountered in real environments, Trend Micro marks it as found *In the Wild*; otherwise, the specimen is considered to be "at the zoo" and unavailable to the public. For instance, as both *WORM_SDBOT.APO* and *JS_FEEBS.GJ* (Table 1) are observed in the real world and hence are found in the wild. To cause Internet epidemics, a specimen needs to be activated and unleashed in the wild. In this respect, Malware Evaluator uses taxonomic feature In the Wild, which comprises two exclusive categories *Yes* and *No*, to characterize the malware availability. By collecting training data for the *In the Wild*

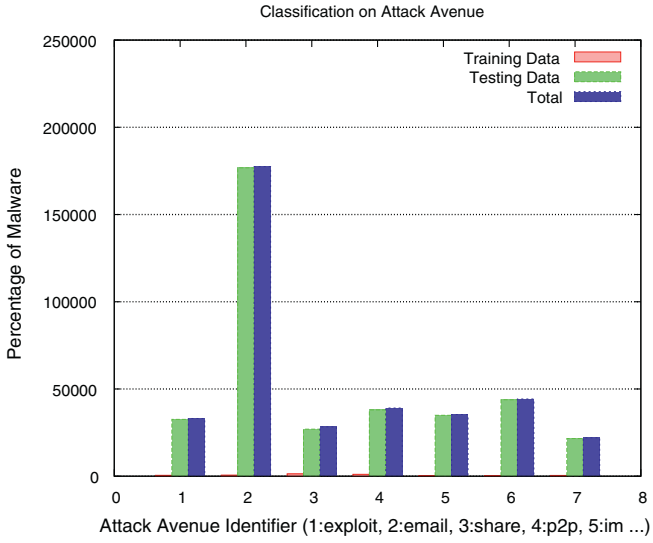

Fig. 5. Malware distribution based on the Attack Avenue feature.

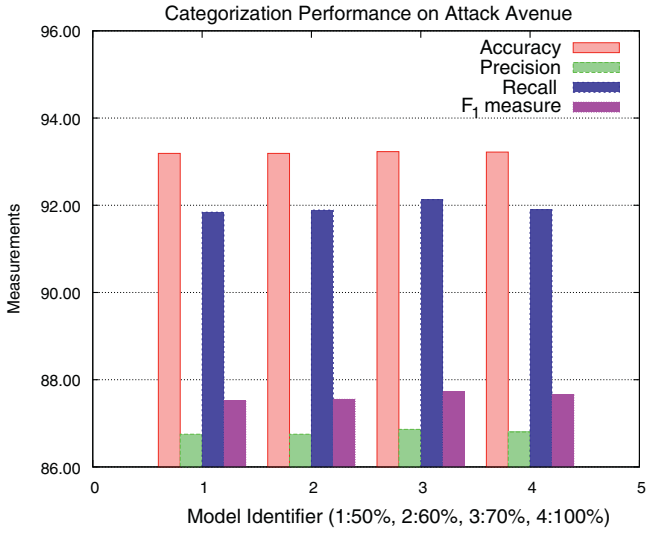

Fig. 6. Classification performance on Attack Avenue feature.

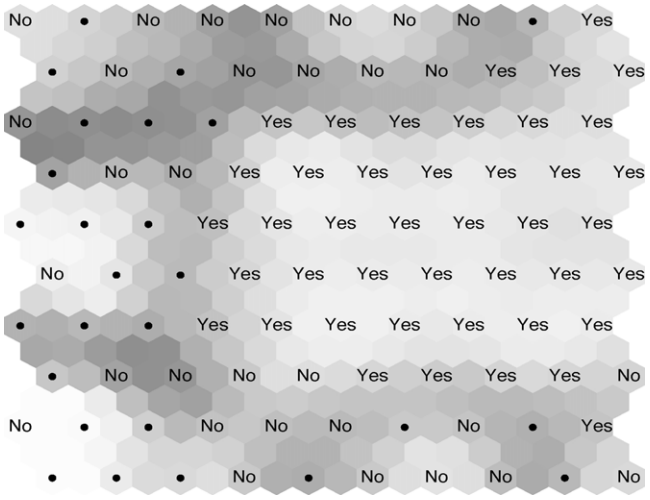

Fig. 7. SOM U-matrix for feature In the Wild.

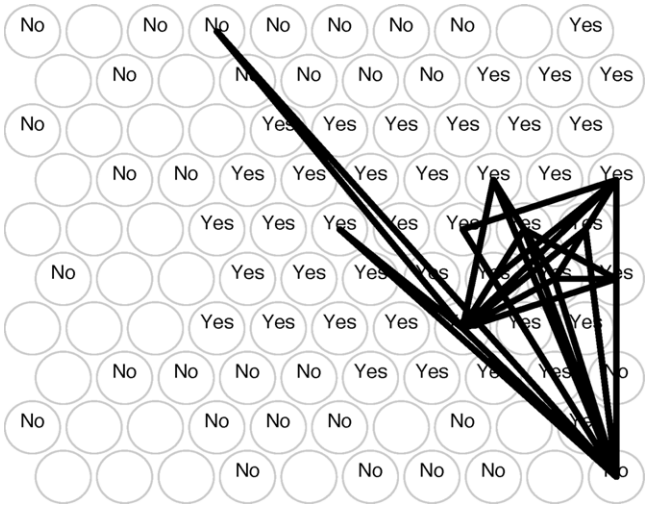

Fig. 8. Genealogical trajectory of In the Wild.

feature from Trend Micro, we generate a SOM map with a 10×10 grid of hexagonal-shaped neurons using the following procedure:

1. **Initialization**: Weight vectors of neurons, which have the same dimensions as feature vectors of samples in training data, are initialized with random values.
2. **Training**: By using neighborhood function *Bubble* and learning rate $\alpha = 0.05$, weight vectors of neurons are updated according to training data with competitive learning techniques described in Section 3.4.
3. **Refinement**: The map is further refined by repeating the *Training* step with a fine-grained learning rate $\alpha = 0.02$.
4. **Calibration**: The best matched unit (BMU) for each sample in the training data is identified. The BMU inherits the sample's label as well.

By representing each node with its average distance to its closest neighbors and further mapping it to gray scale in [0, 100] with the darkest color 0 representing the longest distance, we obtain the U-matrix for the SOM depicted in Fig. 7. Neurons labeled *Yes* occupy 43 out of 100 nodes and are mainly located at the middle right of the map, while the 31 nodes for category *No* scatter around the boarders along with unlabeled cells. The *No*-labeled nodes have darker colors compared to those of the *Yes* class implying that nodes in category *Yes* look more similar and homogeneous due to their smaller average distances among themselves.

In the Trend Micro encyclopedia, the specimen family *JS_FEEBS* has 271 members, each of which has its own unique name consisting of three parts: malware type, family name, and suffix. The malware type is typically an acronym for the genre described in column *name* of Table 2. The family name identifies a group of specimens that share the same code base and therefore have similar characteristics and behavior. Members in the same family are differentiated by suffixes that are assigned in alphabetical or numerical order according to their discovery date. Therefore, the *JavaScript*-encapsulated strains *JS_FEEBS.GI* and *JS_FEEBS.GK* are the immediate ascendant and descendant of the *JS_FEEBS.GJ* instance in its family. Obviously, the chronological relationship of species in the same malware family can be obtained by sorting their names in ascending alphabetical order. With the sorted list of species in a family at hand, we can identify the best matching unit (BMU) in a SOM map for each family member, and connect BMUs in the chronological orders of their corresponding specimens to obtain a curve termed "genealogical trajectory", which outlines the evolution of the malware family in question with respect to a specified feature.

Fig. 8 shows the genealogical trajectory of the family *JS_FEEBS* in reference to the In the Wild feature. The BMUs of the 271 family members only fill 12 out of 100 SOM neurons, indicating that many species in fact select the same cells as their BMUs. By designating the upper left corner of the SOM as the origin and upper edge the x axis while left edge the y axis, we can derive that 171 specimens fall into the cell at (9, 9), while 42 are trapped into node (7, 6). In addition, only two occupied nodes at (3, 0) and (9, 9) are *No*-labeled while the remaining belong to category *Yes*. BMUs for the oldest members in the *JS_FEEBS* family –*JS_FEEBS-1*, *JS_FEEBS.A*, and *JS_FEEBS.D* – are located at (7, 6), (5, 4), and (9, 9), respectively. Meanwhile, cells (7, 6) and (5, 4) are labeled as *Yes* whereas node (9, 9) is in class *No*, indicating that some variants are not successfully released in real-world environments. Furthermore, specimen *JS_FEEBS.GJ* outlined in Table 1 chooses neuron (8, 5) as its BMU, while both of its immediate ascendant and descendant *JS_FEEBS.GI* and *JS_FEEBS.GK* are represented by BMU at (9, 9), resulting in line segments from (9, 9) to (8, 5) and (8, 5) to (9, 9). Therefore, species in the *JS_FEEBS* family appear to keep switching between in-the-wild and at-the-zoo, manifesting its meandering evolution route.

With the same training data, we also build a number of classifiers based on the multiclass-optimization method by varying C_m in the range of $[10^1, 10^8]$; C_m essentially controls the trade-off between training errors and classification accuracy. Fig. 9 depicts the relationship between $\log(C_m)$ and categorization performance. The classification accuracy starts off at 96.92% when $C_m = 100$ and steadily improves while increasing $\log(C_m)$; it reaches 99.33% when $\log(C_m) \geq 6$. In a way reminiscent of the C_b parameter of the binary SVMs, C_m trades training errors for margins of separating hyperplanes: a large C_m imposes heavy punishment on training errors committed by SVM models and compels the latter to reduce training errors, leading to a higher classification accuracy.

Based on the same set of training data, we also construct a series of classifiers by using Gradient Boosting Decision Tree (GBDT) algorithm. Every GBDT classifier comprises a set of n decision trees, and each tree has 4 leaf nodes. By varying n in the range of [2, 10], we obtain a sequence of GBDT models, and evaluate their categorization performance, which is presented in Fig. 10. With only as few trees and leaf nodes as 2 and 4, a GBDT model is able to achieve quite remarkable classification performance. Compared to models created by SVM method, all GBDT classifiers behave much better in terms of classification accuracy, precision, recall, and F_1 measure if parameter C_m below 10^5 is used in SVM model generation. Unfortunately, the performance of GBDT models only improves

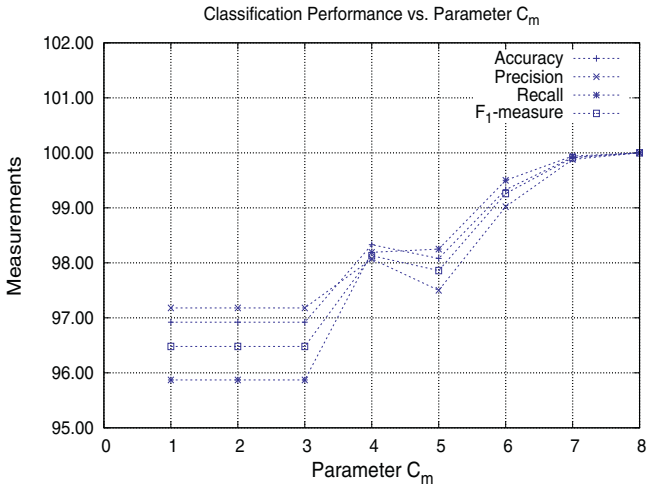

Fig. 9. Classification performance by SVM on In the Wild.

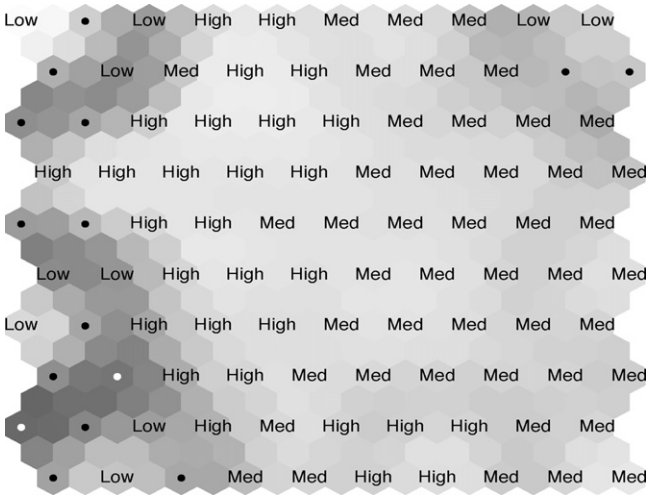

Fig. 11. The U-matrix for feature Damage Potential.

marginally by increasing the number of trees or leaf nodes due to data overfitting issue.

5.3. Damage potential

Certain species are specifically created to delete or corrupt files, causing direct damage to the integrity of victim hosts. Others such as in the *Backdoor* and *Trojan* families work mostly in "background" and gradually consume both CPU-cycles and bandwidth eventually affecting the victim's productivity. To hide its malicious activities, malware typically alters configuration files, lowers security settings, or changes the behavior of utilities on victim systems. Some species even aggressively terminate security applications including anti-virus and firewall services so that attacks go completely undetected. We therefore propose the Damage Potential feature to characterize the inherent malware capability of compromising the confidentiality and integrity of a system. The uni-labeled feature Damage Potential is made up of following three categories:

1. *High*: species corrupt system data or files beyond recovery. Furthermore, members of this category could gather sensitive information or generate heavy network activities.
2. *Medium*: data and files contaminated by strains in this category can only be recovered with special utilities and tools. Breeds may automatically execute arbitrary commands, create massive spams and terminate security applications.
3. *Low*: malware only modifies non-mission-critical data/files and modifications can be readily undone without resorting to specialized tools. Other side-effects can be reversed by simply restarting the system.

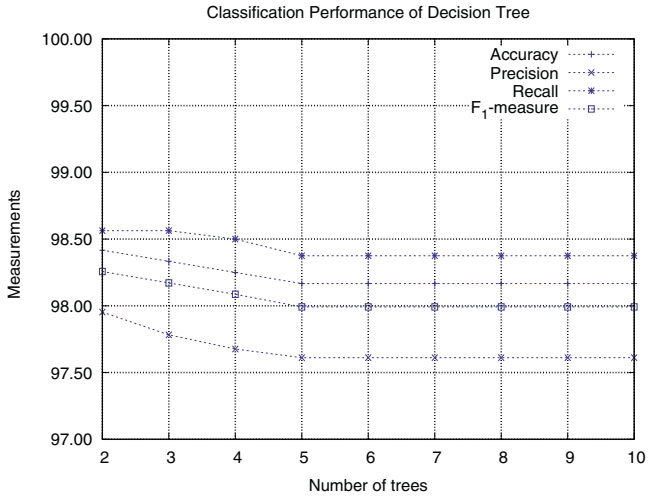

Fig. 10. Classification performance by GBDT on In the Wild.

We automatically gather training data for the Damage Potential feature from Trend Micro entries that maintain information in the corresponding field. The SVM learning model built with the multiclass-optimization method helps us understand the malware distribution on feature in discussion: 70.48% malware strains have low damage potential, rendering them security noise in the ecosystem. On the contrary, about 14.89% and 14.63% of the species have medium and high damage potential to the cyberspace respectively; they both call for effective mechanisms to prevent from breaking out. Fig. 11 shows the U-matrix for the SOM that consists of a 10×10 grid of neurons and is created based on the same training data. Apparently, neurons with the same labels tend to cluster together. Here, nodes for classes *Low* and *Medium* mainly reside at the left and right of the SOM, while the narrow gap between them belongs to category *High*. The relatively dark color for *Low*-occupied region reveals that the average distance between its neurons are larger than those for classes *Medium* and *High*. The continuous estates for category *Medium* point out that its members share similar features and are so more homogeneous compared to the other two classes.

In the construction process of the SOM map, weight vectors of neurons are adjusted according to input samples so that characteristics of the latter could be preserved in certain cells of the map. It is therefore reasonable to expect that a dominant attribute in the feature space should still remain its significance in the SOM lattice with high probability. In the proposed framework, each attribute of a weight vector actually represents a token that appears in the Trend Micro repertoire. We define the plane map for the SOM's *i*th attribute as the lattice formed by the *i*th component in the weight vector of every neuron. To determine dominant attributes for Damage Potential, we thoroughly analyze its plane maps and evaluate the contribution of each token. Fig. 12 presents the plane map for token *exploit*, which is a major contributor to the feature in question. The figure is generated by mapping attribute weights into gray levels in [0, 1] with 0 the darkest and 1 the lightest. A cell assumes a gray value of 0.5 if its converted gray level is less than 0.5 to improve the readability. The observation that token *exploit* assumes extremely large values for neurons of class *High*

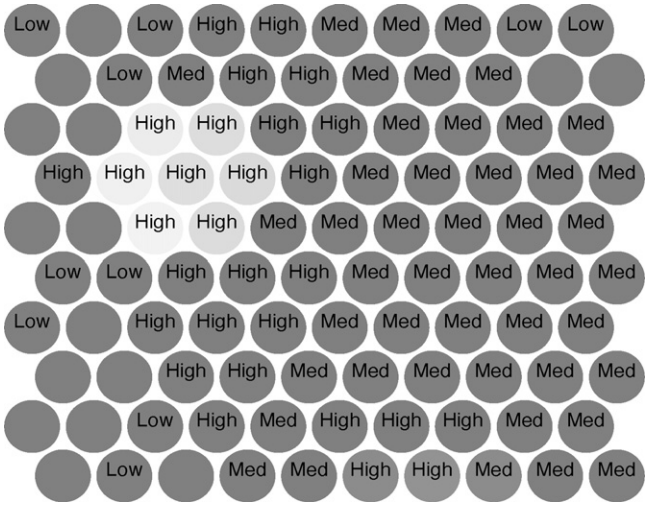

Fig. 12. Plane map of token *exploit* on Damage Potential.

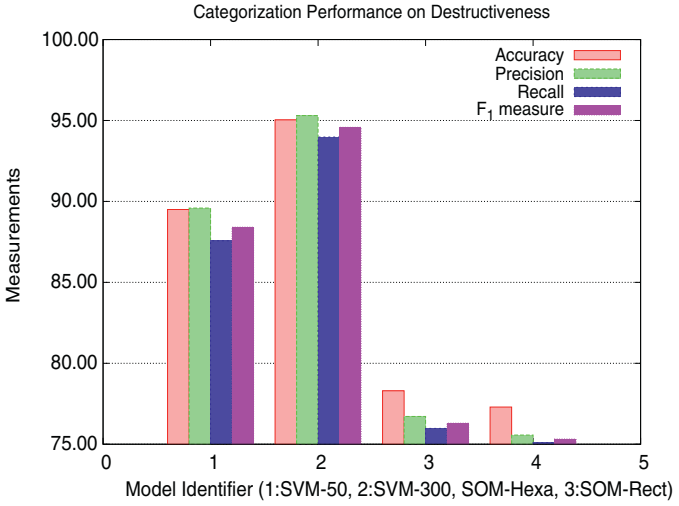

Fig. 13. Classification performance on Destructiveness by SVM and SOM.

renders the token a significant distinguisher for the feature. In addition, the strong clique formed by the 7 nodes that have highest values for token *exploit* is centered at cell (2, 3) and has average gray value 0.83, much higher than the average of the entire plane – 0.10. Therefore, token *exploit* is considered the dominant attributes for the Damage Potential feature and can act as a strong category differentiator.

5.4. Destructiveness

A piece of malware is deemed destructive if it either corrupts a victim's file system or even worse formats the entire hard drive. Malware is also destructive should it excessively consume resources and affect the availability of the victim system. Through compromised systems, malware can also create distributed denial of service (DDoS) attacks typically using intensive network traffic equally affecting availability. Malware Evaluator defines the Destructiveness feature to capture the effect malware imposes on the system availability. The uni-labeled Destructiveness feature consists of two categories: *Yes* and *No*. For instance, worm *WORM_SDBOT.APO* and script *JS_FEEBS.GJ* (Table 1) are destructive; on the contrary, their ascendants *WORM_SDBOT.APL* and *JS_FEEBS.GH* are examples for the *No* category.

We create training data by retrieving Trend Micro entries that have information on the *Destructiveness* field. We then construct *SOM-Hexa*, a SOM map whose neurons are in hexagonal shape. The resulting SOM map can act as a classifier to label unseen samples by letting the input sample inherit the label of its best matched unit (BMU). By changing the neuron lattice from hexagonal to rectangular shape, we obtain another SOM map termed *SOM-Rect*. With the same training data, we also build two SVM learning models named *SVM-50* and *SVM-300* with parameter C_b set to 50 and 300 respectively. Fig. 13 presents the categorization performance of *SVM-50* and *SVM-300* along with those attained by *SOM-Hexa* and *SOM-Rect*. Clearly, model *SOM-Hexa* outperforms *SOM-Rect*, but models *SVM-50* and *SVM-300* dramatically exceed their SOM counterparts.

With the same training data at hand, we also construct a group of GBDT models that have the same settings including learning rate and number of leaf nodes but differ in the number of trees which vary in the range of [10, 200]. Resorting to 10-fold cross-validation method, we measure the classification performance of the resulting GBDT models and compare it with that of SVM models. Our experiments indicate that the categorization measurements of GBDT models are quite similar after the number of trees increases to 40. More specifically, the classification accuracy is around 84.65%,

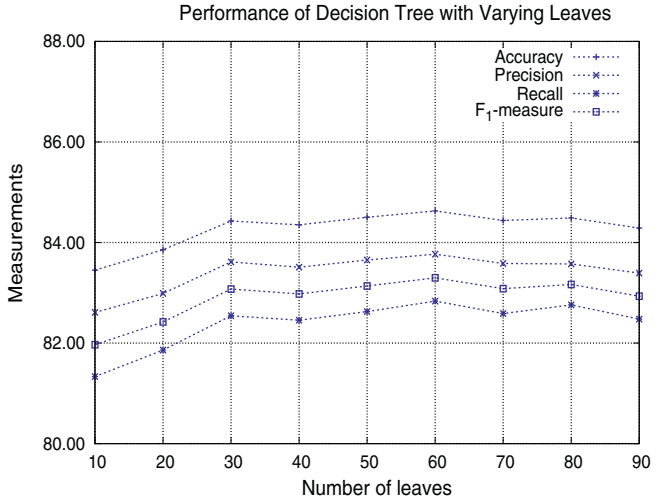

Fig. 14. Impact by number of leaves on classification performance for Destructiveness.

while precision and recall are about 83.80% and 83.02%, respectively. By comparing the performance of GBDT models with that of SVM and SOM classifiers shown in Fig. 13, we can observe that GBDT outperforms SOM, but is inferior to SVM. To further investigate the impact on the categorization performance by the number of leaf nodes in the GBDT models, we train a sequence of GBDT models by fixing the number of trees to be 200 but vary their numbers of leaf nodes in [10, 90]. The classification performance of the resulting GBDT models is presented in Fig. 14. The best performance is achieved when the GBDT model consists of 60 leaf nodes in each of its 200 trees. In contrast, the worst performer is the GBDT model that has only 10 leaf nodes in its trees. It is also obvious from the figure that the performance of GBDT models is relatively insensitive to the number of leaf nodes when the latter is beyond 30. In comparison, the model built by SVM-multiclass method based on the same training data delivers the best classification performance with accuracy of 99.66% (shown in Table 3).

6. Discovery and eradication

The specimen life expectancy is mostly determined by the time spent without being detected. To avoid discovery, malware attempts to maintain control over victims for as long as possible by

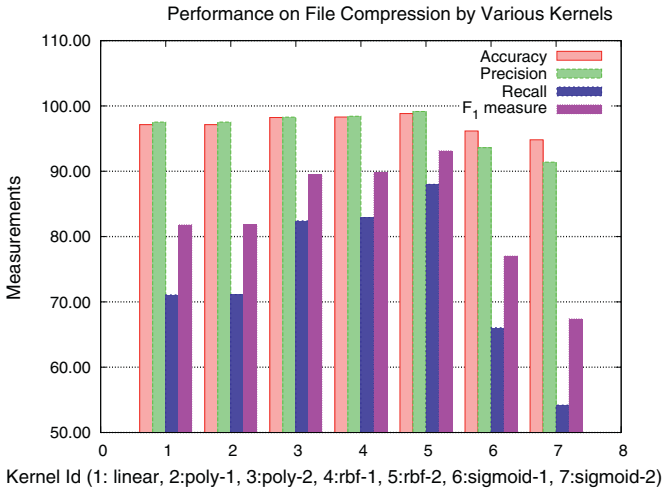

Fig. 15. Impact on performance for feature Compression Type by different kernels.

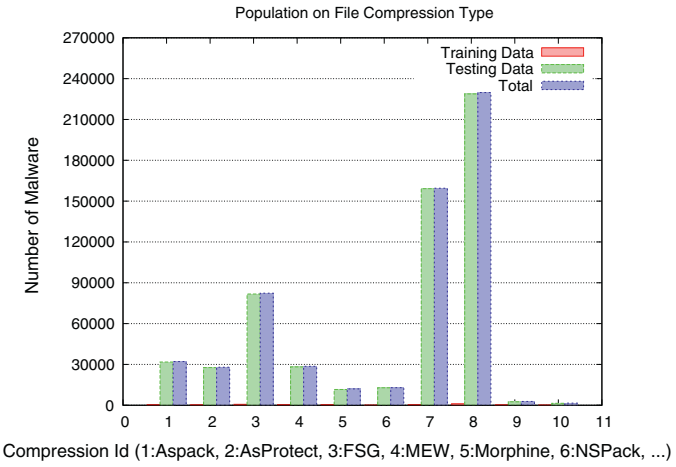

Fig. 16. Malware distribution in the categories of Compression Type with RBF kernel.

typically employing cryptographic techniques to scramble its data and communication channels. Malware also tries to defeat firewalls and related security applications by compressing its executables using a variety of file packing methods. In this section, we elaborate on the features that characterize malware behavior during its *Discovery & Eradication* stage.

6.1. File compression

Malware routinely applies compression on its executables to reduce both its storage footprint and network transmission times. To this way, malware obfuscates its content and original digital structure diminishing its exposure to scrutiny. In general, file compression may help deter reverse engineering and elude detection by pattern-based anti-malware devices. To achieve the same effect as uncompressed binaries, malware usually transforms its executable into a self-extracting archive that consists of the compressed file and a decompression engine to unpack the file and then transfer control to it on the fly. In Malware Evaluator, we designate the File Compression feature to help characterize the malware behavior on how packing executables occurs. As it is impractical to enumerate all potential compression algorithms especially when proprietary packing mechanisms are involved, we focus on the *most popular* compression methods listed in Table 5.

The training data for the multi-labeled File Compression feature are constructed from the Trend Micro encyclopedia. For instance, *WORM_SDBOT.APO* worm (Table 1) compacts its files in the *Morphine* format before shipping. By using the multiclass-to-binary reduction method with parameter $C_b = 500$ and a linear kernel, we build a SVM model termed *Linear*, and present its categorization performance in Fig. 15. Categorization accuracy is at 97.13%, and its precision, recall, and F_1-measure are at 97.53%, 71.08%, and 81.81%, respectively.

To evaluate the impact on categorization performance under different SVM kernel functions, we train learning models with various kernels whose performance is shown in Fig. 15. The used kernels are:

1. *Polynomial kernels* in the form of $(\overline{x}_i \cdot \overline{x}_j + 1)^d$ with variable exponent d and for brevity, we only describe results with $d = 1$ and 2 (termed *poly-1* and *poly-2* in Fig. 15).
2. *Radial basis functions (RBFs)* expressed as $\exp(-\gamma\|\overline{x}_i - \overline{x}_j\|^2)$ with various γ; Fig. 15 only shows the results for $\gamma = 1 and 2$.
3. *Sigmoid functions* in the format of $\tanh(\overline{x}_i \cdot \overline{x}_j + c)$ with tunable parameter c and the results in Fig. 15 are for $c = 0.5$ and 1 (denoted as *sigmoid-1* and *sigmoid-2*).

Fig. 15 depicts that sigmoid functions deliver the worst categorization performance, while RBFs outperform other kernel types especially in terms of recall and F_1-measures. Compared to linear kernels, the construction of learning models with RBFs demands many more CPU-cycles, therefore, Malware Evaluator employs linear kernel functions as its default option. The malware distribution for feature File Compression generated by the learner *RBF-1* with $C_b = 500$ is portrayed in Fig. 16. Evidently, 64.95% species resort to *UPX* compression method, while 49.09% and 24.36% employ *PECompact* and *FSG*, respectively, rendering them the most popular packing methods. In comparison, compression approaches *Upack* and *Upolyx* are much less frequently used.

6.2. Removal difficulty

Although it is important to identify implanted malware, it is even more critical to quarantine and eventually eradicate the detected artifacts completely. To extend its life span even after being detected, a specimen may resort to a variety of methods to resist containment and removal. By contaminating file systems and registry databases with multiple files and keys, malware could permeate deeply into infected machines essentially increasing the difficulty of being completely removed. Once about to be removed, malware often resists by destroying host applications. To completely remove malware from infected machines, specimen-installed files should be expunged while corrupt data have to be sanitized or recovered. The removal process also involves deletion of registry entries created by malware and termination of processes associated with the specimen in question. The purging of various specimen types calls for diverse skills, utilities as well as coordination among geographically distributed sites as far as network-aware species are concerned.

The Malware Evaluator defines the Removal Difficulty feature to characterize both skills and technologies required for malware elimination. We designate three levels *Difficult, Moderate*, and *Easy*; the *Difficult* class is reserved for species that demand diverse experience and specialized techniques for malware elimination, while the removal of strains in class *Moderate* involves some kind of expertise, and little or no expertise is necessary for breeds in the last *Easy* category. We use the Symantec security advisory to collect training data for the Removal Difficulty feature as such information is unavailable in Trend Micro. For instance, *W32.Virut.B* worm is easy

Table 5
File compression methods by malware of Trend Micro encyclopedia.

ID	Compression	Description
1	Aspack	A Win32 executable compressor capable of reducing file size and resisting reverse engineering
2	AsProtect	A tool designed to compress and encrypt applications and counteract reverse engineering
3	FSG	A file compressor especially suitable for small *EXE* or *ASM* files
4	MEW	A EXE-packer based on LZMA and ApPack methods and having a good compression ratio
5	Morphine	A simple packer/cryptor that is usually used to hide malicious code
6	NSPack	an executable compressor for exe, dll, ocx, and scr files, also pack 64-bit executables
7	PECompact	A compressor for code, data, and import/export table with proprietary compression algorithm
8	UPX	Ultimate Packer for eXecutables is an open-source packer performing in-place decompression
9	Upack	Upack is a file packer based on LZMA compression
10	Upolyx	A scrambler that transfers UPX packed files through its polymorphic decryption engine

to be removed from infected systems; in contrast, strain *W32.Feebs* is rather difficult to eradicate completely. With the training data at hand, we build a SVM learning model for the Removal Difficulty feature using the multiclass-optimization reduction method, and observe that among 340,246 species in the Trend Micro encyclopedia, 248,599 (about 73.06%) can be easily removed from infected systems. In contrast, 26.90% malware strains require advanced utilities and complex skill for their complete elimination, some further demand experienced security experts with customized and specially-crafted tools for their removal.

6.3. Categorization performance of combined models

Thus far, the machine learned models are built individually for each taxonomic feature; similarly, the constructed models for a taxonomic feature are verified independently of other features. It is more efficient, however, if models of all taxonomic features are built based on the same set of training data and their performance is evaluated using the same set of validation data. One benefit of using the same training data set and validation data set for all taxonomic features is that the categorization performance for models of taxonomic features as a whole (denoted as combined model) can be easily determined. Suppose that, for instance, separate models are constructed for taxonomic features in the set { Damage Potential, Attack Avenue }, and the resulting two models are used together to classify a sample whose set of true tags is {High, {Exploit, Email} }. If the set of labels given by the combined model is {High, {Exploit} }, then the combined model mislabels the sample. Similarly, the combined model still misbehaves if the assigned set of labels is {High, {Email} }, {High, {Exploit, Email, P2P} }, or {Low, {Exploit, Email} }. Thus, the categorization performance of the combined model can only be as good as the weakest classifier in the combined model.

Although the idea of building and verifying models of taxonomic features on a single set of training data and a single set of validation data is tempting, it is difficult to implement in practice. First, to make such a model-building method feasible, each data point in the training data set and validation data set must have values on all taxonomic features. Unfortunately, such a constraint drastically reduces the number of candidates for training data and validation data. For example, in Trend Micro, less than 13% of its entries have values for taxonomic features Encryption or Destructiveness, and less than 11% of the entries have values for both features. The percentage of entries with information on more than three taxonomic features could be much lower, making it unlikely to gather enough training data for the combined model. Second, even if a set of training data could be successfully constructed, it is highly unlikely that the distribution of values for every taxonomic feature in the training data set is similar to that in the entire repertoire. Finally, the data of the training set may overfit certain taxonomic features, but may be under-populated for others; this thus, affects the categorization performance of all models as a whole. Therefore, it is unrealistic to train models for all taxonomic features with a universal set of training data.

Model validation faces the same dilemma as training data collection as it is extremely difficult to obtain a set of adequate data to fit all taxonomic features. As a compromise, we obtain a picture of the classification performance of the combined model carried out the following methodology:

- we build a model for each individual taxonomic feature independently,
- we collect a set of validation data such that each data point in the set has tags for all taxonomic features,
- we use the collected set of validation data to measure the categorization performance of the combined model.

We apply the above procedure to evaluate the combined model formed by taxonomic features in stages *Penetration* and *Activation* listed in Table 3. The average classification accuracy of the five models in the set stands at 98.98% while the classification accuracy of the combined model derived by the afore procedure is at 95.45%. Clearly, the combined model performs slightly worse than the weakest classifier in the set of models, which has categorization accuracy 96.65%, but much worse than the average accuracy delivered by the models. In the same manner, the average classification accuracy of models for all taxonomic features depicted in Table 3 is 98.86%, while the classification accuracy of the combined model consisting of all models is 91.54%, indicating that the performance of the combined model deteriorates by about 7.32% compared to the average performance of the models.

7. Malware risks and trends

Malware Evaluator not only categorizes species with respect to a variety of taxonomic features but also helps evaluate threats and shed light on malware evolution. In this section, we first identify and categorize an array of evasive techniques used by species to resist eradication. We then carry out summarization on malware characteristics along the File Size and Discovery Date dimensions to reveal trends about malware footprints and propagation. We also conduct malware classification according to Attack Avenue, Carried Payload, and Discovery Date features to better understand how species tend to attack targets. Finally, we analyze the risks imposed by malware.

7.1. Enhanced malware evasive capabilities

Anti-malware products typically identify species by searching files or network traffic for specific telltale patterns termed signatures. To derive such signatures, experts often analyze malware code and trace its behavior through reverse engineering, debugging, and disassembling. Patterns, however, can be readily changed by malware creators with the help of encryption, polymorphism, and metamorphism (Leder et al., 2009). Malware could also integrate anti-debugging and anti-reverse-engineering

techniques rendering its code very challenging to dissect and analyze. Malware Evaluator defines the Concealment feature that helps characterize malware behavior on how it tries to hide its existence and conceal its spurious activities. The feature contains the following categories:

- *Stealth*: Stealth malware deliberately attempts to conceal changes it manages to accomplish on victim systems. Various stealthy techniques exist to hide alterations in files, processes, registry settings, and activity traces. For instance, a strain may intercept and modify system calls to remove its appearance from lists of files or processes, and "restore" original sizes on infected files.
- *Polymorphism/Metamorphism*: A polymorphic strain automatically morphs its code into different mutant versions without changing underlying functions when clones are generated. Metamorphism not only dynamically changes code sequences, but also constantly modifies functionalities.
- *Armor*: Strains of this class integrate anti-debugging and anti-reversed-engineering techniques into their code to slow down or even prevent altogether the analysis of their functions and behavior conducted by experts through disassembly, traces, and sand-boxing.
- *Obfuscation*: Members of this category conceal their presence in infected systems by a variety of obfuscation methods including misleading names for their files, creation of hidden folders, and randomized file names or communication ports used.

Through pattern searching in the text body of the Trend Micro entities, we construct training data for feature Concealment, build its respective classifier, and then build a malware hierarchy based on taxonomic features Concealment and Discovery Date. The analysis on the hierarchy points out that, after its first peak in 2000, the *Stealth* category appears to steadily decline to a valley around 2006 and then re-gains momentum. Similarly, obfuscation methods are frequently employed and the population of class *Obfuscation* reach its peak in 2006. In comparison, the use of anti-debugging and anti-reverse-engineering techniques only becomes significant in 2008–2009, indicating increased effectiveness in their fight against the anti-malware community. It is worth observing that *polymorphism* and *metamorphism* fail to become major techniques for malware concealment mainly due to their design and implementation complexity.

Encryption techniques are not only critical for code poly-/metamorphism, but also essential for data and network traffic scrambling. Multiple layers of encryption are also used for malware to further complicate its identification. Although encryption helps species evade detection by anti-malware products, it does so at the cost of code complexity and degraded performance in terms of penetration rate and execution speed. The malware hierarchy with respect to taxonomic features Information Encryption and Discovery Date indicates that the majority of species actually propagate in plaintext. This is indicative of the fact that performance is still the predominant design criterion in malware development. The percentage of strains created each year that employ cryptographic methods is at the level of approximately 10% over the past decade. This implies that encryption still remains an effective option in eluding detection.

7.2. Use of compact malware footprints

The size of specimen footprint characterized by feature File Size in Malware Evaluator clearly affects both the malware functionality and penetration speed. For a specimen entering victim systems through bundling with other software, its transportation to infected hosts inevitably slows down if a large code footprint is required. The footmark is also constrained when it penetrates targets by taking advantage of security vulnerabilities such as buffer overflows as the exploitation can only be successful if the malicious code is within a certain range of sizes. A sizable malware also requires large storage space in the compromised systems leaving a visible trace that facilitates its detection. Overall, specimen authors try to optimize their artifacts so that a trade-off between storage consumption and malware functionality can be achieved.

The mutant behavior and customized functionalities of malware complicate the definition of the File Size feature. First, malware could tailor its footprint according to dynamics in affected systems including OSs present and network bandwidth availability. Second, malware is capable of transferring files in an amortized fashion so that a piece of stub code is initially installed which later downloads the rest of the code on demand. Periodic self-updates also fluctuate the malware size during its life-cycle. Finally, it is common for strains to compress their files to enable efficient transportation and storage. For instance, the *WORM_SDBOT.AAB* has a memory footprint of 218,624 Bytes when compressed but unfolds to 2,060,288 Bytes once unpacked. In Malware Evaluator, we represent a specimen with its smallest uncompressed footprint, should it manifest diversified behavior in its sizes over time.

For the File Size feature, we specify the following seven categories: 5, 25, 50, 100, 300, 800 and >800 in *KBytes*. We construct training data by extracting Trend Micro entries that have information in the *Size of Malware* field and converse the respective sizes into the above seven buckets. For example, the footprint of 37,888 Bytes for *WORM_SDBOT.APO* makes it a member of the 50 bucket. We then build an SVM model for File Size based on the derived training data. The feature File Size and its malware categorization help us investigate the relationship between species genre and storage footprint. To this end, we classify malware according to feature Malware Type and then categorize each group with respect to feature File Size, and present portion of classification results in Fig. 17. The cumulative distributions for *Backdoor* and *Trojan* demonstrate strong similarity, that is, 47.40% and 50.63% of their members have memory footprints of up to 100 KBytes. The *Infector* and *Worm* categories also demonstrate similar accumulated distributions. By considering a breed to have a compact footprint when its memory consumption is less than 100 KBytes, we can observe that about half of strains in *Backdoor* and *Trojan* are footprint-compact. For the *Infector* and *Worm* classes the number of footprint-compact breeds is even higher at 78.72% and 82.87% respectively. Obviously, the majority of species manifest

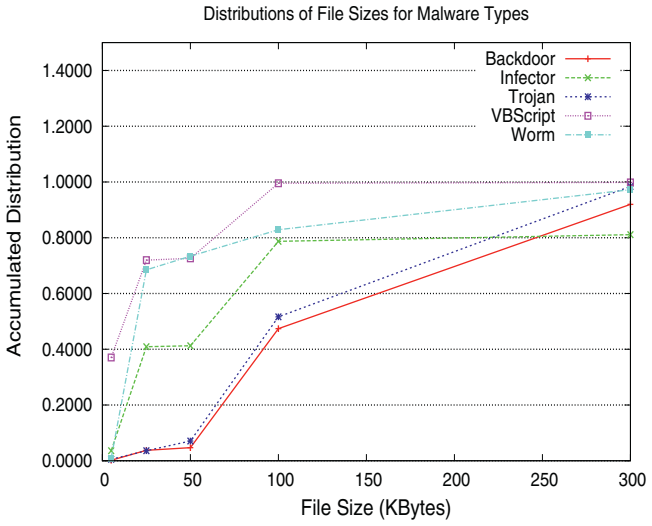

Fig. 17. Cumulative malware distribution on Malware Type and File Size.

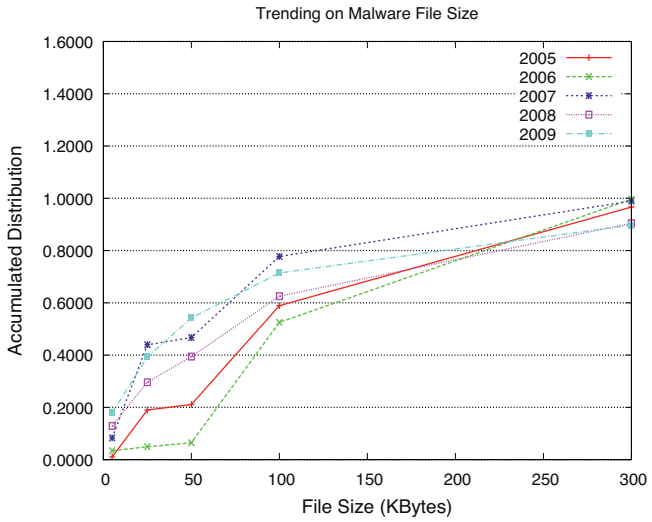

Fig. 18. Accumulated malware distribution on Discovery Date and File Size.

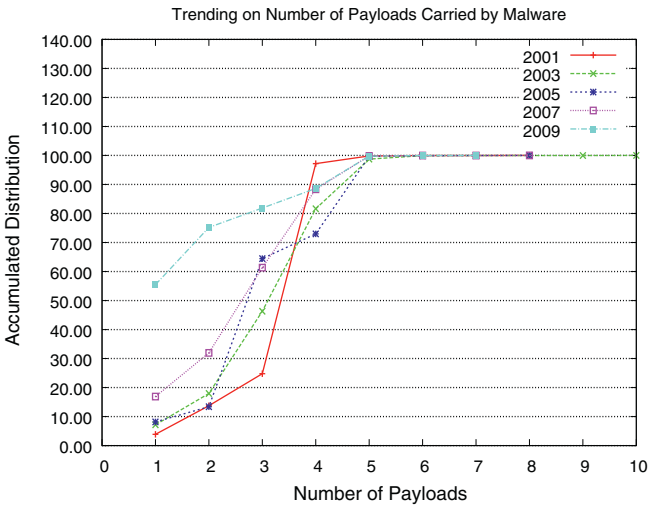

Fig. 19. Accumulated distribution of malware with respect to number of payloads.

tiny footprints to attain widespread effect through light payloads. On the other hand, some strains in *Backdoor* and *Trojan* categories typically attempt to reside permanently on victim systems calling for a much larger set of payloads and necessitate heavy footprints.

To obtain a better insight into the temporal trend with regards to malware footprint, we classify malware using the File Size and Discovery Date features. Fig. 18 shows that species tend to have reduced storage footprints chronologically: only 3.39% of species require storage less than 5 KBytes in 2006. This number becomes 8.33% in 2007 and jumps up to 12.91% and 17.89%, respectively for 2008 and 2009. Similarly, about 52.51% of the malware population born in 2006 are footprint-compact, while 77.73% and 71.43% of the species discovered in 2007 and 2009 are footprint-tight, further confirming the trend for reduced footprints. Malware storage consumption distributions for 2008 and 2009 are heavy-tailed as 9.55% and 10.27% of their members fall outside the range of [0, 300] *KBytes*. In contrast, only 3.35%, 0.35%, and 1.00% of species discovered in 2005, 2006, and 2007 consume storage larger than 300 KBytes.

7.3. Diversified attack avenues and multiple payloads

Malware typically carries a variety of payloads including those listed in Column *payload* of Table 6. Here, the categorization regarding the Carried Payload and Discovery Date features demonstrates that payloads do evolve over time. The majority of the 2000-born species is designed to affect the stability of victims. In 2006, information theft becomes a popular payload and file-download was the most favorite activity in 2009. By sorting categories according to Carried Payload on a yearly basis, we obtain the top three most frequently employed payloads. In 2000, the top three types of payload were *System Instability, File Manipulation*, and *File Download*; in 2005, the top three were *File Manipulation, Integrity Impact*, and *Process Termination*. Evidently, it is difficult to find out a common payload that appears in the top-three payloads throughout the years.

Table 6 also shows that payloads *Message Display, Proxy Service*, and *Command Execution* become "out of fashion" as they typically generate visible effects revealing the very existence of the specimens. In contrast, payloads *File Manipulation, Information Leakage*, and *File Download* are heavily utilized. For instance, about 20.29% of the 2007-born species carry payload of *File Download*; this becomes 32.21% and 29.38%, respectively, for 2008 and 2009. It is worth pointing out that the *Security Degradation* payload fails to become a lethal weapon in malware's arsenal. Obviously, improved detection capabilities of anti-malware products weaken the effectiveness of attacks. Moreover, visible specimen effects caused by disabling firewalls or blocking accesses to anti-malware sites also lead to rapid reaction by administrators.

The summation of species over all rows and columns of Table 6 yields a population much larger than the total number of entries in the Trend Micro repertoire (i.e., 340,246), implying that a number of strains carry multiple payloads simultaneously. Our malware classification based on the number of payloads carried reveals that 12.34% of strains uses only one type of payload. In contrast, 51.28% of species actually pack four different payload types in their distributions, while about 10% carry more than four different payloads. For instance, *TROJ_AOL.BUDDY.P* can unleash nine types of payload and *WORM_AGOBOT.AU* features 10 types of payload in its arsenal including *Configuration Change, Process Termination*, and *Email Spam*. The evolution on malware payload can be further analyzed by classifying strains with respect to Carried Payload and Discovery Date features depicted in Fig. 19. From those species born in 2001, only 3.90% have a single payload (i.e., singleton) and 72.43% carry four payloads. As we move to 2005, 2006 and 2009, the singletons steadily increase at 8.17%, 16.89%, and 55.47% respectively. On the other hand, the malware population that use multiple types of payload significantly increases: although in 2001 only 2.90% species carry more than four payloads, in 2003 and 2005 the corresponding percentiles are at 18.35% and 27.03%.

The evolving malware attack avenues can be analyzed with the help of the categorization on features Discovery Date and Attack Avenue shown in Fig. 20. The "*Attack Avenues = 1*" bars depict species that penetrate target systems using a single attack channel; Fig. 20 depicts that most one-attack-channel malware were created in 2006. The peak for "*Attack Avenues = 2*" occurs in 2005, while the spikes for "*Attack Avenues = 3*" and "*Attack Avenues = 4*" coincide in 2009. The existence of multiple pinnacles in the bar chart signifies that opportunistic authors use any penetration mechanisms they have at their disposal. Furthermore, a significant portion of strains discovered in 2008 and 2009 penetrate targets with more than two attack mechanisms, signifying the chronological trend for multiple spread channels. The combination of malware characteristics including reduced storage footprint, multi-payloads, and diversified attack avenues clearly underlines the modularity as well as the maturity in malware creation.

Table 6
Malware classification based on type of payload and discovery date.

ID	Payload	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009
1	*System Instability*	20156	7229	761	2519	5246	976	1193	12139	477	143
2	*Security Degradation*	125	2802	267	92	228	536	7213	328	166	102
3	*Email Spam*	6519	15689	811	3753	1898	824	645	7803	183	91
4	*File Manipulation*	17036	15313	5445	13518	13940	42723	109993	10650	2252	1119
5	*Information Leakage*	8211	25297	3924	8412	14219	19530	109108	11702	1933	598
6	*Proxy Service*	4210	2171	152	251	402	780	403	1455	827	895
7	*DDoS Attacks*	1621	351	1074	3269	1196	13753	1447	3202	257	22
8	*Process Termination*	5449	11017	2392	8731	3760	39802	2469	4538	545	200
9	*Command Execution*	1056	816	455	607	1992	1344	3483	486	153	436
10	*Download*	9167	25113	3180	9047	16928	6212	111092	17466	4525	1807
11	*Integrity Impact*	5096	3401	3175	9454	6533	42011	10550	5619	766	238
12	*Configuration Change*	9040	14143	3049	5287	11366	5290	108531	6708	1705	385
13	*Message Display*	3260	2462	1011	2198	1757	600	7402	3991	260	115

7.4. Evaluating risks imposed by malware

Efficient incident responses and effective defense strategies necessitate malware risk assessment. However, the diverse behavior and sophisticated characteristics of malware make it a challenge to objectively ascertain threats to the Internet ecosystem. Consequently, domain experts undertake the time-consuming task of evaluating malware risks. In this context, our dissection shows that less than 15% of the species in the Symantec encyclopedia are analyzed with respect to their threats on the Internet, and nearly all Trend Micro entries assume the default category *Low* for the feature Risk. To automate the evaluation on malware risks, Malware Evaluator defines the Risk Level feature. This feature attempts to quantify the threat imposed by malware based on the characteristics, which cover the entire specimen life-cycle, listed in Table 7.

To quantify the contribution of the above characteristics to the Risk Level feature, we design a scoring system to estimate the risk of each malware. This scoring mechanism is configurable and its default settings appear in Column *Default Scoring Method* of Table 7. For instance, the *WORM_SDBOT.APO* worm of Table 1 attacks two different OSs causing 2 points to be added to its risk score; in addition, each of its high damage and distribution potential contributes 3 points to its risk score as well. By summing up the contributions from all features described in Table 7 for *WORM_SDBOT.APO*, we obtain risk-score 26. Based on our extensive evaluation with the proposed scoring approach, we designate five categories for the Risk Level feature:

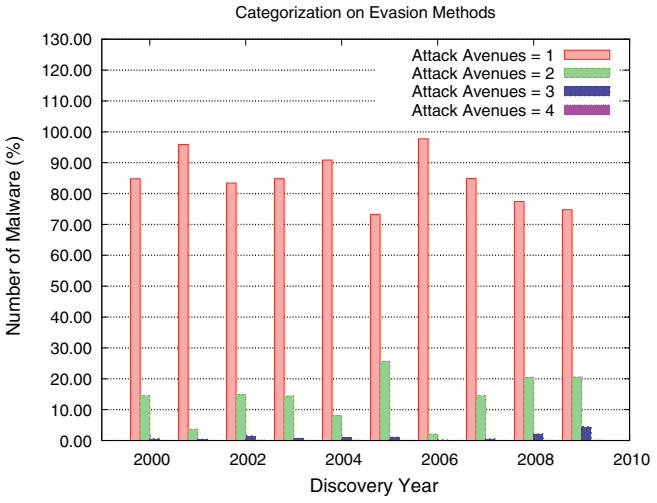

Fig. 20. Number of installation venues employed by malware.

1. *Extremely Critical*: this category accommodates the most dangerous malware species that possess significant damaging power and high distribution potential. Specimens of this category typically have risk scores >32 (configurable).
2. *Highly Critical*: this group contains malware strains that are highly dangerous, difficult to contain, and their risk scores are above the threshold 28.
3. *Moderately Critical*: this class may carry multiple payloads or employ multiple attack channels, and have medium system impact or distribution potential. The default threshold for risk score is larger than 22.
4. *Mildly Critical*: This category holds malware specimens that have limited damage or distribution potential, and their risk scores are greater than a specified threshold (greater than 15 by default).
5. *Slightly Critical*: Members of this group pose little or negligible threat.

The thresholds on risk scores for the above-described categories are derived as follows. First, 2000 species are randomly selected from entries in Symantec which have information on field *Risk Level*. Their risk scores are then computed according to the rules defined in Table 7. Finally, they are sorted based on their risk scores and thresholds are picked to minimize the number of mis-classified species. Our categorization with respect to the Risk Level feature based on the above-described criteria allows us to determine that both *WORM_SDBOT.APO* and *JS_FEEBS.GJ* of Table 1 are part of the *Moderately Critical* category. In comparison, *WORM_SDBOT.CZX, TROJ_VB.BLA*, and *BKDR_IRCBOT.QB* are placed in the *Extremely Critical* class, while species *W97M_HILITE.A, IRC_DOLLY.B*, and *WORM_SDBOT.HD* are *Highly Critical*.

The malware classification with respect to Malware Type and Risk Level helps us better understand risks imposed by various types of malware. The majority of malware labeled *Extremely Critical* originate from malware of *Trojan, Infector, Backdoor* and *Worm* types. Member of the *Highly Critical* category emanate mostly from the *Infector* malware type and the *Moderately Critical* class mainly gets its population from the *Backdoor, Worm* and *Trojan* families. By ordering malware types according to their percentages of species in categories *Extremely Critical* and *Highly Critical*, we can clearly observe that *Infector* imposes the most serious threat to the Internet thus far, while the next three dangerous types are *Trojan, Backdoor* and *Worm*. To analyze the evolution of risks imposed by malware, we categorize malware according to features Risk Level and Discovery Date. The analysis on the resulting classification reveals that the majority of species discovered in 2001, 2004, 2006, 2008, and 2009 are only mildly critical, while most of the species created in other years are considered as moderately critical. We have encountered slightly critical strains mainly discovered in 2000, and the lion's share of species with label of highly and extremely critical are from 2001 and 2009 respectively. By computing the annual ratio

Table 7
Features used in Malware Evaluator to evaluate malware risk.

Stage	Feature	Description	Default scoring method
Creation	Affected Platform	Operating systems (OSs) vulnerable to malware	Number of OSs affected
	Targeted language	Languages attacked by malware	Number of affected languages
Penetration	In the Wild	Malware in executables or source code publicly accessible	Yes = 2, No = 1
	Distribution	Areas affected by malware (Low, Medium, High)	Low = 1, High = 3
	Attack Avenue	Installation mechanisms to infect systems	Number of attack channels
Activation	Damage potential	Expose confidential information or destroy system integrity	Low: 1, Medium: 2, High: 3
	Destructiveness	Damage file systems, stability, and productivity	No: 1, Yes: 2
	Carried payload	Malicious activities after invading victims	Number of payloads
Discovery	Information encryption	Cryptographic methods employed by malware	No: 1, Yes: 2
	Message Compression	Compress files or network flows to avoid detection	number of compression types
	Memory resident	Stay at main memory after execution	No: 1, Yes: 2
Eradication	Containment difficulty	Skill, tools, and expertise required to quarantine malware	Easy = 1, Difficult = 3
	Removal difficulty	Skill, tools, and expertise required to remove malware	Easy = 1, Difficult = 3

between extremely/highly and moderately/mildly/slightly critical malware, we find that the ratio reaches its peak in 2001, then falls into a valley around 2005; it then rises again in 2008 and forms another peak in 2009. It is therefore reasonable to expect that malware risk fluctuates over time, demanding constant monitoring of its evolution as well as continuous effort toward containment and eradication.

7.5. Malware evaluator in action

In this section, we discuss how Malware Evaluator can be used in practice using two Trend Micro-provided case studies. In the first of these two studies, company A, has installed a computer security system on its machines to detect and prevent malware attacks. Although no serious intrusion has ever occurred, the large volume of attempted attacks reported by the computer security system is a heavy burden to its site security office. We use Malware Evaluator to help find measures to reduce the volume of attempted attacks. Obviously, an attack cannot be launched if all of its potential propagation channels are effectively cut off. Thus, we extract all attempted attacks reported by the security system in the past six months, most of which are members of the *WORM_SDBOT* family (refer to Table 1 for one of its species *WORM_SDBOT.APO*), evaluate their attack venues using Malware Evaluator, and derive the distribution of attack venues shown in Fig. 21. Clearly, more than 35% of the attacks use network share as launching pads, while about 40% of the attempted intrusions come from IM and P2P applications. Based on this information, we suggested to the company to limit the use of network share, IM, and P2P in its work environment, and as a result, the attempted attacks reported in company A have significantly decreased.

In our second case study, similar to company A, company B also deploys a network security device to shield its intranet from external attacks. Unfortunately, this network security device was accidentally misconfigured to work on detection mode instead of prevention mode, that is, attacks are detected and reported but still allowed to pass through the device. As a result, the intranet was invaded and infected by malware. Depending on the seriousness of the invasion, the company wanted to determine which measure to adopt: complete re-installation of fresh systems on all machines or removal of malware from infected machines. To evaluate the impact of the aftermath, we analyzed, with the help of Malware Evaluator payloads carried by malware species detected by the network security device, and generated the distribution of payloads, which is delineated in Fig. 22. We observed that the most visible impact of malware payloads on the infected systems is the compromise of their integrity, implying that critical system components and configurations had been corrupted. In addition, more than 15% of the payloads carried by malware may have exposed sensitive information such as passwords and business secrets to attackers. Moreover, the security services including firewalls and anti-viruses on the infected machines had been deactivated or

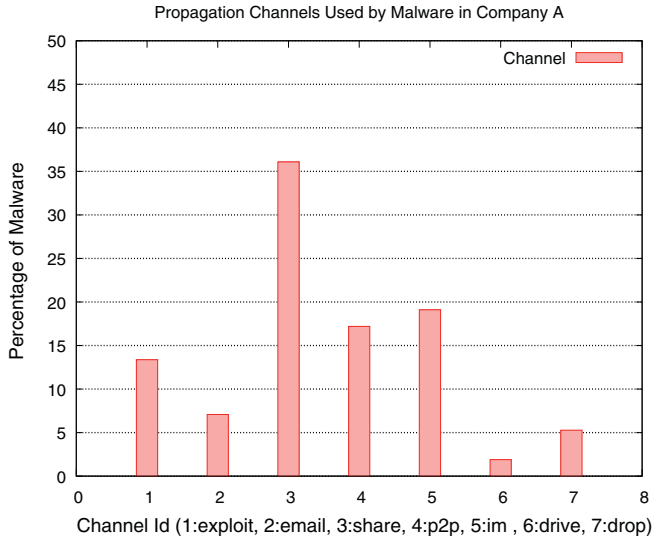

Fig. 21. Penetration channels of malware in company A.

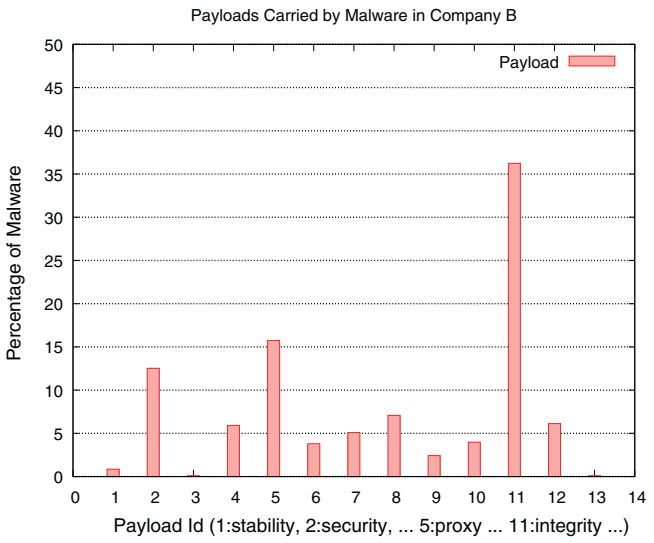

Fig. 22. Payloads carried by malware in company B.

Table 8
Interaction between Malware Evaluator and other malware analysis tools.

#	Sandbox	Sandbox output (excerpt)	Damage P.	Destructiveness	Attack Avenue
1	Norman	[Changes to filesystem] * Creates file C:/WINDOWS/SYSTEM32/ . . . [Changes to registry] * Creates value 11Removal" = . . . in key "HKLM/Software/Microsoft/. . ." [Network services] * Looks for an Internet connection. * Connects to "some.domain.com" on port 9889 (TCP). [Security issues] * Possible backdoor functionality . . .	High	No	Share, DroppedBy
2	Norman	[General information] * File might be compressed. [Changes to filesystem] * Creates file C:/WINDOWS/Jammer2nd.exe. [Changes to registry] * Creates value "Jammer2nd" = "C:/WINDOWS/ . . ." in key "HKLM/Software/Microsoft/Windows/CurrentVersion/Run". [Process/window information] * Will automatically restart after boot . . .	High	Yes	Exploit,
3	GFI	Filesystem: New Files (2): Creates File: C:/DOCUMENT/ . . . Find Files (12): %SystemRoot%/system32/*, %SystemRoot%/ . . . Opened Files (16): /./PIPE/lsarpc, %SystemRoot%/system32 Creates Mutex (8): Name: RasPbFile, Opens Mutex (1) Process: Creates Process – Filename () CommandLine: (%SystemRoot Registry: Reads (15): HKEY_LOCAL_MACHINE/SOFTWARE/ . . .	High	Yes	Exploit

contaminated (see payload type 2 in Fig. 22). Based on the above analysis, we recommended to company B that all systems be reinstalled.

Malware Evaluator can also play a significant role in identifying and categorizing zero-day attacks, which are newly created and thus are not recognized by any computer security products. By integrating Malware Evaluator with automated malware analysis tools such as *Norman Sandbox* and *GFI Sandbox*, we identify and categorize zero-day attacks with the following procedure: (a) the sample of the potential zero-day attack is fed into *Norman Sandbox* or *GFI Sandbox* so that its behavior is automatically analyzed; (b) the analysis report created by *Norman Sandbox* or *GFI Sandbox* acts as input to Malware Evaluator, where the report is rewritten and then treated as a bag of words to form attribute vector and labels are generated for its taxonomic features listed in Table 3; (c) the labels of taxonomic features for the sample can be used by security experts, researchers, or administrators as a guide to further study the characteristics of potential attack. In Table 8, we present several cases to demonstrate the interaction between Malware Evaluator and automated malware analysis tools *Norman Sandbox* and *GFI Sandbox*.

When we receive the first sample of Table 8, we scan it with popular anti-malware products but none identifies it as malware. We then go through the above-described procedure for the sample. Column *Sandbox Output (Excerpt)* shows the analysis report generated by *Norman Sandbox*, and labels assigned by Malware Evaluator are presented in Columns *Damage Potential*, *Destructiveness*, and *Attack Avenue*. As the sample could bring heavy damage to a system when infected, it is highly likely to be a malware specimen. Further investigation by security experts confirms that the sample under study is a new variant of the worm *SPYBOT* family. The report for the second sample delivered by *Norman Sandbox* indicates that the sample might insert itself into the victim system in compressed format. After landing on the infected host, the sample pollutes victim's file system with multiple files, creates a mutex, and changes system's registry. Furthermore, the sample also modifies the infected system's configurations so that it can survive system reboot. Such behavior of the malicious sample clearly destroys the integrity of victim hosts as its label is "High" for taxonomic feature Damage Potential assigned by Malware Evaluator. By scanning the sample with anti-malware products, the sample turns out to be a new variant of *WORM_NETSKY*, a mass-mailing worm. Extensive manual analysis on this sample by experts revealed that the sample can infect hosts not only by exploiting vulnerabilities, but also through emails.

In the same manner, Malware Evaluator can also interact with *GFI Sandbox* to recognize and classify zero-day malware species as demonstrated in sample 3 of Table 8. Although the output format of *GFI Sandbox* is different from *Norman Sandbox*, both create report on various behavioral aspects of the given sample. The high damage and destructive impact caused by the sample reveal with high confidence that the sample is a malware. In-depth analysis by experts indicated that the sample probably was a newly created backdoor that could infect systems through IRC in addition to exploiting vulnerabilities in victim machines. To further demonstrate the capability on categorization of newly discovered malware samples, we build a model, with the help of Malware Evaluator to discriminate *SPYBOT* family from other malware species. Being fed with sample 1 of Table 8 to the model, the model accepts the sample to be its member; on the contrary, it firmly rejects both samples 2 and 3. Similarly, the machined learned model constructed to recognize the worm *WORM_NETSKY* family can successfully identify sample 2 in Table 8 as part of it but decline the other samples.

8. Conclusions and future work

In this paper, we present Malware Evaluator, a framework that transforms malware encyclopedias such as Trend Micro into an automated classifier which not only clusters species according to taxonomic features, but also helps evaluate malware evolution and detect zero-day attacks. Our framework treats malware classification as a supervised learning task, builds models for taxonomic features with support vector machines (SVMs) and gradient boosting decision trees (GBDTs), and visualizes malware categorizations with self-organizing maps. The textual description of each entry in the Trend Micro repertoire is collapsed into a bag of words so that the entry in question can be represented with a feature vector; in this vector, each word is an attribute having value the word occurrence-frequency in the entry. We reduce dimensions of feature space with stopword removal and word stemming, while selecting attributes according to their information gains further condenses the space footprint. Training data are automatically extracted from both Trend Micro and Symantec encyclopedias.

The key feature of Malware Evaluator is that it is an *automated* classifier, thus enabling observations about malware characteristics and evolution to be made quickly, something not previously possible with existing encyclopedias which require manual categorization and analysis. Classifications on taxonomic features that characterize malware in its entire lifecycle demonstrate that Malware Evaluator delivers high accuracy, precision, recall, and F_1-measurements. The trend analysis conducted with Malware Evaluator points out that malware spares no effort in shrinking its storage footprint to improve propagation speed and

reduce visible traces left on infected systems. To elude detection, species often resort to obfuscation and stealth techniques, while the integration of anti-debugging and reverse engineering deterrence mechanisms into malware code further defeats anti-malware products that rely on telltale pattern matching for malware identification. Malware Evaluator also exposes the malware trend to penetrate hosts with diversified attack avenues and subsequently unleash multiple payloads to enhance its damaging effects on infected systems. Furthermore, the risk assessment we conducted using Malware Evaluator has established that *Trojan, Infector, Backdoor*, and *Worm* impose critical threats to the Internet ecosystem and in all likelihood will continue to dominate the malware landscape in the future. Finally, the use of Malware Evaluator in real-world cases demonstrates its ability to help defend against threats and recognize zero-day attacks.

We plan to extend our work in a number of directions: we intend to design a much larger set of taxonomic features with properties of determinism and specificity to thoroughly characterize malware species while keeping the classification process automated. We will also attempt to strengthen our presentation with visualization techniques such as Generative Topographic Maps and Smoothed Data Histograms. In this way, intricate structures revealed by malware categorizations could be effectively projected. We plan to investigate additional feature selection methods such as mutual information and *Chi*-square to possibly offer a better trade-off between dimensionality of feature space and categorization performance. To enhance the integration of Malware Evaluator with other automated malware analysis tools, we aim to thoroughly evaluate the performance of Malware Evaluator on texts generated by software such as *Norman Sandbox* compared to human-created descriptions. Finally, we will make publicly available our toolkit to help monitor malware dynamics in the Internet ecosystem and forecast the evolution of species as well as their characteristics.

Acknowledgements

We are grateful to reviewers for their comments and Peter Wei of *Trend Micro Inc.* for fruitful discussions on the proposed framework. This work has been partially supported by the European Commission *D4Science II FP7* Project and the ERC Starting Grant Project (no 279237).

References

Allwein, E.L., Schapire, R.E., Singer, Y., Kaelbling, P., 2000. Reducing multiclass to binary: a unifying approach for margin classifiers. Journal of Machine Learning Research 1, 113–141.

Bailey, M., Oberbeide, J., Andersen, J., Mao, Z.M., Jahanian, F., Nazario, J., 2007. Automated classification and analysis of internet malware. In: The 10th Symposium on Recent Advances in Intrusion Detection (RAID'07), Gold Coast, Australia, September. Springer, pp. 178–197.

Boney, D.G., 1999, June. The Plague: An Army of Software Agents for Information Warfare. Technical Report. Department of Computer Science, School of Engineering and Applied Science, Washington DC.

Bontchev, V.V., 1998. Methodology of Computer Anti-Virus Research. Technical Report. University of Hamburg, Germany.

Boser, B.E., Guyon, I.M., Vapnik, V.N., 1992. A training algorithm for optimal margin classifiers. In: Proceedings of the Fifth Annual ACM Workshop on Computational Learning Theory, Pittsburgh, PA, USA, July. ACM Press, pp. 144–152.

Bredensteiner, E.J., Bennet, K.P., 1999. Multicategory classification by support vector machines. Computational Optimizations and Applications 12 (December), 53–79.

Brunnstein, K., 1999. From antivirus to antimalware software and beyond: another approach to the protection of customers from dysfunctional system behaviour. In: Proceedings of the 22nd National Information System Security Conference, Arlington, VA, USA, October. NIST, pp. 1–14.

CERT, 2001. CERT Advisory CA-2001-26 Nimda Worm, http://www.cert.org/advisories/CA-2001-26.html.

Collobert, R., Bengio, S., 2001. SVMtorch: support vector machines for large-scale regression problems. Journal of Machine Learning Research (JMLR) 1, 143–160.

Costa, M., Crowcroft, J., Castro, M., Rowstron, A., Zhou, L., Zhang, L., Barham, P., 2005. Vigilante: end-to-end containment of internet worms. In: Proceedings of the 20th ACM Symposium on Operating Systems Principles (SOSP2005), Brighton, United Kingdom, October. ACM, New York, NY, USA, pp. 133–147.

Cova, M., Kruegel, C., Vigna, G., 2010. Detection and analysis of drive-by-download attacks and malicious Javascript code. In: Proceedings of the World Wide Web Conference, Raleigh, NC, USA, April. ACM, pp. 1–10.

Crammer, K., Singer, Y., 2001. On the algorithmic implementation of multiclass kernel-based vector machines. Journal of Machine Learning Research 2 (December), 265–292.

DeLooze, L.L., 2004. Classification of computer attacks using a self-organizing map. In: Proceedings of the Fifth Annual IEEE SMC Workshop on Information Assurance, June. IEEE, pp. 365–369.

Dietterich, T.G., Bakiri, G., 1995. Solving multiclass learning problems via error-correcting output codes. Journal of Artificial Intelligence Research 2 (January), 263–286.

Ferris, M., Munson, T., 2003. Interior-point methods for massive support vector machines. SIAM Journal of Optimization 13 (3), 783–804.

Forman, G., 2003. An extensive empirical study of feature selection metrics for text classification. Journal of Machine Learning Research 3, 1289–1305.

Fox, C., 1992. Lexical Analysis and Stoplist – Data Structures and Algorithms. Prentice Hall.

Fredrikson, M., Jha, S., Christodorescu, M., Sailer, R., Yan, X., 2010. Synthesizing near-optimal malware specifications from suspicious behaviors. In: 2010 IEEE Symposium on Security and Privacy, Oaklan, CA, USA, May. IEEE, pp. 45–60.

Friedman, J.H., 1999. Greedy function approximation: a gradient boosting machine. Annals of Statistics 29, 1189–1232.

Hastie, T., Tibshirani, R., 1998. Classification by pairwise coupling. The Annals of Statistics 26 (2), 451–471.

Helenius, M., 2002. A system to support the analysis of antivirus products' virus detection capabilities. Tampereen Yliopisto, ISBN 951-44-5370-0.

Rutkowska, J., 2006. Introducing Stealth Malware Taxonomy, http://invisiblethings.org/papers/malware-taxonomy.pdf.

Joachims, T., 1999. Making large-scale support vector machine learning practical. In: Schoelkopf, B., Burges, C., Smola, A. (Eds.), Advances in Kernel Methods – Support Vector Learning. MIT Press, Cambridge, MA, pp. 169–184 (Chapter 11).

Kawakoya, Y., Iwamura, M., Itoh, M., 2010. Memory behavior-based automatic malware unpacking in stealth debugging environment. In: 2010 5th International Conference on Malicious and Unwanted Software (Malware), Nacy, Lorraine, October, pp. 39–46.

Keerthi, S., DeCoste, D., 2005. A modified finite newton method for fast solution of large scale linear SVMs. Journal of Machine Learning Research (JMLR) 6, 341–361.

Kohonen, T., 2000. Self-Organizing Maps, third edition. Springer, Berlin.

Krsul, I.V., 1998, May. Software Vulnerability Analysis. Technical Report. Purdue University.

Landwehr, C.E., Bull, A.R., McDermott, J.P., Choi, W.S., 1994. A taxonomy of computer program security flaws, with examples. ACM Computing Surveys 26 (September (3)).

Leder, F., Steinbock, B., Martini, P., 2009. Classification and detection of metamorphic malware using value set analysis. In: 2009 4th International Conference on Malicious and Unwanted Software (Malware), Montreal, QC, October, pp. 39–46.

Lee, T., Mody, J., 2006. Behavioral classification. In: Proceedings of the EICAR Conference, Hamburg, Germany, April/May. European Expert Group for IT Security, pp. 1–17.

Lindqvist, U, 1999. On the Fundamentals of Analysis and Detection of Computer Misuse. Technical Report. Department of Computer Engineering, Chalmers University of Technology, Goteborg, Sweden.

Lindqvist, U., Jonsson, E., 1997. How to systematically classify computer security intrusions. In: Proceedings of the 1997 IEEE Symposium on Security & Privacy, Oakland, CA, November. IEEE Computer Society Press, pp. 154–163.

Moore, D., Paxson, V., Savage, S., Shannon, C., Staniford, S., Weaver, N., 2003. Inside the slammer worm. IEEE Magazine of Security and Privacy 1 (July/August (4)), 33–39.

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

Paxson, V., Staniford, S., Weaver, N., 2002. How to own the internet in your spare time. In: Proceedings of the 11th USENIX Security Symposium (Security'02), San Francisco, CA, USA, August, pp. 1–19.

Platt, J.C., Cristianini, N., Shawe-Taylor, J., 2000. Large margin DAGs for multiclass classification. Advances in Neural Information Processing Systems 12, 547–553.

Porter, M.F., 1980. An algorithm for suffix stripping. Program 14 (3), 130–137.

Rieck, K., Holz, T., Willems, C., Dussel, P., Laskov, P., 2008. Learning and classification of malware behavior. In: The 5th International Conference on Detection of Intrusion Detections and Malware, and Vulnerability Assessment (DIMVA'08), Paris, France, July. Springer, pp. 108–125.

Salton, G., Buckley, C., 1988. Term-weighting approaches in automatic text retrieval. Information Processing and Management 24 (5), 513–523.

Scheidl, G., 1999, July. Virus Naming Convention 1999 (VNC99). Technical Report. http://members.chello.at/erikajo/vnc99b2.txt.

Schoelkopf, B., Smola, A.J., 2002. Learning with Kernels. The MIT Press, Cambridge, MA.

Shieh, S.-P., Gligor, V.D., 1997. On a pattern-oriented model for intrusion detection. IEEE Transactions on Knowledge and Data Engineering 9 (4), 661–667.

Sidiroglou, S., Keromytis, A.D., 2005. Countering network worms through automatic patch generation. Security & Privacy 3 (November–December (6)), 41–49.

Skoudis, E., Zeltser, L., 2003, November. Malware: Fighting Malicious Code. Prentce Hall PTR, ISBN 0131014056.

Symantec, 2009. Virus Encyclopedia of Symantec, http://www.symantec.com/business/security_response.
Thomas, K., Nicol, D.M., 2010. The Koobface Botnet and the rise of social malware. In: 2010 5th International Conference on Malicious and Unwanted Software (Malware), Nacy, Lorraine, October, pp. 63–70.
Trend Micro, 2011. Virus Encyclopedia of Trend Micro, http://threatinfo.trendmicro.com/vinfo/virusencyclo.
Tyree, S., Weinberger, K.Q., Agrawal, K., 2011. Parallel boosted regression trees for web search ranking. In: International World Wide Web Conference 2011, Hyderabad, India, March/April. ACM, pp. 387–396.
Vapnik, V.N., 1999. The Nature of Statistical Learning Theory: Information Science and Statistics. Springer, New York.
Venter, H.S., Eloff, J.H.P., Li, Y.L., 2008. Standardizing vulnerability categories. Computers and Security 27 (May–June (3)), 71–83.
Wagner, C., Wagener, G., State, R., Engel, T., 2009. Malware analysis with graph kernels and support vector machines. In: 2009 4th International Conference on Malicious and Unwanted Software (Malware), Montreal, QC, October, pp. 63–68.
Walenstein, A., Hefner, D.J., Wichers, J., 2010. Header information in malware families and impact on automated classifiers. In: 2010 5th International Conference on Malicious and Unwanted Software (Malware), Nacy, Lorraine, October, pp. 15–22.
Weaver, N., Paxson, V., Staniford, S., Cunningham, R., 2003. A taxonomy of computer worms. In: Proceedings of The First ACM Workshop on Rapid Malcode (WORM'03), Washington, DC, USA, October. ACM, pp. 11–18.
Willems, C., Holz, T., Freiling, F., 2007. CWSandbox: towards automated dynamic binary analysis. Security and Privacy 5 (March–April (2)), 32–39.
Williamson, M.M., 2002. Throttling viruses: restricting propagation to defeat mobile malicious code. In: Proceedings of the 18th Annual Computer Security Applications Conference (ACSAC), Las Vegas, NV, December, pp. 1–8.
Ye, J., Chow, J.H., Chen, J., Zheng, Z.H.,2009. Stochastic gradient boosted distributed decision trees. In: Proceeding of the 18th ACM Conference on Information and Knowledge Managment. ACM, pp. 2061–2064.

Zhongqiang Chen received his PhD in Computer Science from the Polytechnic Institute of NYU, in Broklyn, NY in 2003. Since then, he has worked as member of the technical staff at Fortinet, Inc. and currently is a senior software engineer at Yahoo! Inc. in Sunnyvale, CA. His research interests are in information retrieval, system security, and data analysis.

Mema Roussopoulos is an Assistant Professor of Computer Science at the Department of Informatics and Telecommunications at the University of Athens in Athens, Greece. She completed her PhD in Computer Science and was a Postdoctoral Fellow in the Computer Science Department at Stanford University. She was an Assistant Professor of Computer Science on the Gordon McKay Endowment at Harvard University. She then was a faculty member at the Department of Computer Science at the University of Crete and an Associated Researcher at the Institute of Computer Science at FORTH. Her interests are the areas of distributed systems, networking, mobile computing, and digital preservation. She is a recipient of the CAREER award from the National Science Foundation, a Starting Grant Award from the European Research Council, and the Best Paper Award at ACM SOSP 2003.

Zhanyan Liang received her BS in Actuarial Science from Guangxi University of Finance and Economics in China in 2011 and she currently works in the fields of finance and insurance. Her research interests are in applying mathematical and statistical methods to data mining and system emulation.

Yuan Zhang is currently a PhD candidate in Financial Mathematics at Florida State University (FSU) in Tallahassee, FL. His research interests are in data mining, system modeling and machine learning. He holds an MS in Mathematics from FSU and a BS in Finance with minor in Mathematics from Shanghai Jiaotong University, in China. He is the member of American Mathematical Society.

Zhongrong Chen is currently a Senior Analyst of Shire US Inc. in Pennsylvania, USA. He has been working in the fields of data mining and system modeling in the past decade. He holds a Master degree in Industrial Engineering from the University of Missouri, Columbia, MO and a Master degree in Finance from the Webster University in St. Louis, MO. He is a member of the Global Association of Risk Professionals.

Alex Delis is currently a Professor of Computer Science at the University of Athens in Athens, Greece. His research interests are in distributed systems, data management and software systems. He has been an Associate and Assistant Professor at Polytechnic Institute of NYU as well as a Senior Lecturer and a Lecturer at Queensland University of Technology. He holds a PhD in Computer Science from the University of Maryland in College Park and has been the recipient of an NSF-CAREER Award, an outstanding paper in the IEEE Int. Conf. on Distributed Computing Systems and the Maurice V. Wilkes Medal of the British Computer Society. He has also been a Fulbright Fellow.