On the security of machine learning in malware C&C detection: a survey

JOSEPH GARDINER and SHISHIR NAGARAJA, Security Lancaster Research Centre, Lancaster University

One of the main challenges in security today is defending against malware attacks. As trends and anecdotal evidence show, preventing these attacks, regardless of their indiscriminate or targeted nature, has proven difficult: intrusions happen and devices get compromised, even at security-conscious organizations. As a consequence, an alternative line of work has focused on detecting and disrupting the individual steps which follow an initial compromise and that are essential for the successful progression of the attack. In particular, a number of approaches and techniques have been proposed to identify the command and control (C&C) channel which a compromised system establishes to communicate with its controller.

A major oversight with many of these detection techniques is the design's resilience to evasion attempts by the well-motivated attacker. C2 detection techniques make widespread use of a machine learning (ML) component. Therefore, to analyse the evasion resilience of these detection techniques we first systematize works in the field of C&C detection, and then, using existing models from the literature, go on to systematize attacks against the machine learning components used in these approaches.

Categories and Subject Descriptors: C [**Computer Systems Organization**]; C.2 [**COMPUTER-COMMUNICATION NETWORKS**]; C.2.0 [**General**]: Security and protection ; I.2.6 [**Learning**]; D.4.6 [**Security and Protection**]: Invasive software

General Terms: Security, Algorithms, Performance

Additional Key Words and Phrases: Command and Control channels, botnets, network intrusion, data mining, machine learning

ACM Reference Format:
ACM Comput. Surv. V, N, Article A (January YYYY), 38 pages.
DOI:http://dx.doi.org/10.1145/0000000.0000000

1. INTRODUCTION

In this survey, we focus on the evasion resilience of the machine learning component(s) of command and control detection systems. Malware detection and mitigation is an established problem. In the last several years, the number of attacks, their sophistication, and potential impact have grown substantially. On one hand, indiscriminate attacks have continued to flourish: these attacks are financially motivated, are responsible for the compromise of large numbers of machines, and result in the theft of financial data, such as credit card numbers and online banking account credentials [Franklin et al. 2007], or used in carrying out attacks including Distributed Denial of Service (DDoS). Opportunist attacks usually result in the formation of a botnet, a coordinated collection of infected hosts under the control of an attacker.

Author's addresses: Joseph Gardiner and Shishir Nagaraja, School of Computing and Communications, InfoLab21, Lancaster University, Lancaster, UK, LA1 4WA; email:{j.gardiner1, s.nagaraja}@lancaster.ac.uk
Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies show this notice on the first page or initial screen of a display along with the full citation. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, to redistribute to lists, or to use any component of this work in other works requires prior specific permission and/or a fee. Permissions may be requested from Publications Dept., ACM, Inc., 2 Penn Plaza, Suite 701, New York, NY 10121-0701 USA, fax +1 (212) 869-0481, or permissions@acm.org.
© YYYY ACM 0360-0300/YYYY/01-ARTA $15.00
DOI:http://dx.doi.org/10.1145/0000000.0000000

At the same time, targeted attacks have emerged as a new threat. These attacks target specific organizations or individuals with the intent of obtaining confidential data, such as contracts, business plans, and manufacturing designs [Gardiner et al. 2014].

Statistics and anecdotal evidence indicate that *preventing* attacks, either indiscriminate or targeted, is difficult. For example, news reports have indicated that even security conscious, well funded organizations have fallen victims to attacks [Gardiner et al. 2014].

Considering the difficulties in effectively preventing attacks, defenders have looked at ways of *detecting* and *disrupting* the individual steps that follow an initial compromise and which are essential for the successful progression of an attack. This is the so-called kill chain approach to defence [Hutchins et al. 2010]. In particular, considerable effort has been spent in identifying the establishment of Command & Control (C&C) channels, i.e. the communication channel through which attackers control compromised devices and receive feedback from them (for example any collected sensitive information).

A typical detection system consists of three main components: data collection, feature extraction and a separation module. We discuss this in more detail in Section 3, but we provide a brief overview here. Data collection represents the measurement step - the collection of input to the system, for example netflow data from an network monitor. The feature extraction step processes this data into a form that is usable by the separation module. The output of this will usually be a set of data points representing entities that need to be separated, such as hosts on the network or observed domain names.

The separation module is responsible for making decisions based upon the data. This can take two approaches - one being to label data points as either benign or malicious (most systems will provide lower level detail for malicious points), the other is to group data points that share similar properties. This module is usually an implementation of a machine learning (ML) algorithm (in most cases, an existing, well known algorithm taken from the literature). The versions of these algorithms in use are not designed for an adversarial environment. Detection techniques using these algorithms do not consider the case where an adversary is actively attempting to avoid detection.

While some detection papers are providing discussion on evasion resistance, there is rarely any practical evaluation of the system's evasion resilient properties. Meanwhile, there is a growing collection of literature discussing attacks against various machine learning algorithms. In Section 6, we bring out attack papers against a number of machine learning algorithms that are commonly used in C&C detection papers.

To achieve these goals, we start by providing a generalised model for C&C detection in section 3. Next, we review approaches for the detection of C&C activity, categorizing the different techniques in use (section 4). We then switch to an analysis of different techniques that attackers may use to evade existing defences by first looking at how to categorize attacks (section 5) and then by looking at existing literature in the space of attacks against machine learning (section 6), and finally identify open challenges and areas of research (Section 7).

Contributions:

— We provide a comprehensive review of C&C detection techniques (Section 4).
— We identify weaknesses of the C&C detection systems by systematizing attacks against the machine learning components (Sections 5 and 6), making use of models defined in the field of adversarial machine learning.
— We reason why secure ML algorithms are not in use in C&C detection systems (Section 7).

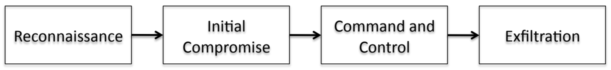

Fig. 1. Malware attack life cycle.

— We identify open research challenges in the use of ML for C&C detection (Section 7.6).

2. THE COMMAND AND CONTROL DETECTION PROBLEM

2.1. Malware phases

Today, malware attacks are both indiscriminate, in the form of botnets, and targeted, evasive, and aimed at obtaining and exfiltrating sensitive data from specific individuals or organisations. How are these attacks carried out in practice? While the specific attack steps and their naming may vary across publications, the literature agrees on the general structure of malware attacks, which is commonly represented as a sequence of steps similar to those of Figure 1.

The command & control (C&C) phase is where the adversaries leverage the compromise of a system. More precisely, compromised systems are forced to establish a communication channel back to the adversary through which they can be directly controlled. The C&C channel enables an attacker to establish a "hands-on-keyboard" presence on the infected system (via so-called remote access tools), to install additional specialized malware modules, and to perform additional malicious actions (e.g. spread to other machines or start a denial of service attack). This channel can either be centralised with a single (or small set of) control servers access by the malware, or decentralised in a peer-to-peer botnet where there is no central server present.

2.2. Focus of this paper

In this survey, we focus on the evasion resilience of the machine learning component(s) of command and control detection systems. Note that while we focus on the C&C phase, the data exfiltration stage uses many of the same channels as the C&C stage and so detection systems often cover both phases.

Command and control detection systems are distinct from host-based detection systems. A C&C detection system will usually operate at a network gateway, where network traffic is collected, rather than on a per-host basis. The detection problem is particularly challenging due to the vast amount of legitimate network traffic generated by normal network use (in a medium to large organisation this could easily reach terabytes a day). In the case of targeted malware attacks, the C&C traffic may only represent a minute percentage of the total traffic volume.

A further issue, in particular in the case of targeted attacks, is the desire of the attacker to remain undetected. While indiscriminate attacks (for example botnets) are relatively difficult to hide in the general case, it can be assumed that the well-motivated targeted attacker will put effort into evading the particular detection methods in use.

A common feature of almost all C&C detection systems is the machine learning component used to make decisions about packets, or hosts, through the use of classifying or clustering techniques. While detection systems do not purely comprise of a machine learning algorithm, it is one of the core components. It may not be possible for an attacker to evade the other stages of detection (as discussed in Section 3), so the logical next step is to attack the core machine learning component. However, in the research describing the design of such detection systems, there is little, if any, consid-

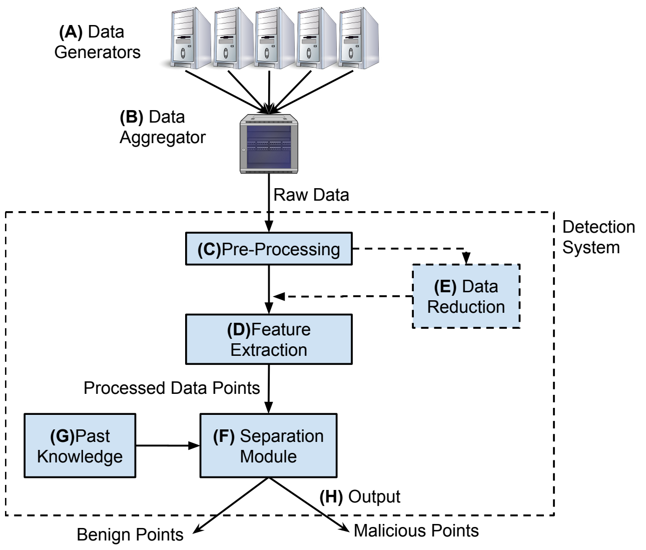

Fig. 2. Generalised architecture of C&C detection systems

eration given to the security of the machine learning component. Meanwhile, there is an increasing volume of literature describing attacks against the ML algorithms used in these systems. While the attacks are often described in the non-specific scenario, many of the attacks could be transferable to the C&C case.

3. GENERALISED ARCHITECTURE FOR C&C DETECTION

We now present a generalised architecture for a typical C&C detection system, shown in Figure 2, and give a brief description of each component. This is a representation of a deployed system, where any training phases have already occurred.

3.1. Architecture

*Data generators **(A)**.* The data generators represent the entities that produce the data evaluated by the detection system. These can take many forms. In the most common case, these will be individual hosts within a network that generate traffic, which can be assumed to be producing both malicious and benign data points. The data generators could also be honeynets or sandboxes, running malware in a controlled condition to extract it's behaviour. In a large network, there could be tens or hundreds of thousands of traffic generators.

*Data aggregator **(B)**.* The data aggregator collects data from the individual data generators into a single stream. The aggregator could be a network monitor on the edge of a large network, or on the ISP. Depending on the size of the monitored network, the aggregator may be hierarchical and be made up of different levels of aggregators at different vantage points. The aggregator could be the monitor around the honeynet/sandbox. It could also be covering external services, such as DNS.

The data aggregator will usually apply the first round of filtering on the data. The major round of processing will be the removal of unwanted data. The unwanted data will be anything not needed for detection, and will typically be components such as the payload of packets. For example, if the aggregator is a traffic monitor, the monitor will often output the data as netflow records, and may apply sampling

(sflow) in order to control the amount of data collected due to storage/processing limitations.
It is important to note that the aggregator is outside of the detection system. The aggregator may be serving multiple detection systems, as well as network monitoring tools, simultaneously, and as such will be configured accordingly.
*Pre-processing **(C)**.* The pre-processing step represents any processing of the data before the data goes through the feature extraction step. This could take many forms, for example removing data that is not useful for detection from packet data.
*Feature extraction **(D)**.* This step extracts the features required for the machine learning component from the data to convert the processed data into a format that can be used by the selected algorithm. The features that are extracted are a design choice of the system, chosen to maximise the detection accuracy.
*Optional: Data reduction **(E)**.* Some systems will often make use of extra steps in order to reduce the size of the data, and remove possible false positives. There are many techniques for doing this, but one of the most common is the use of whitelists in order to remove known benign data points, blacklists to remove known benign points (such as malicious domains) or use of a simple signature-based filter to remove known, easy-to-detect malware.
*Separation Module **(F)**.* The separation module is the key component of the detection system, and is responsible for separating the input data points into a minimum of the broad classes of "benign" and "malicious". In many systems this goes further, by splitting the two into finer categories, for example matching a malicious data point to a specific malware variant.
The separation module can take many forms. In some cases it may simply be an anomaly detector that identifies unusual behaviour compared to past behaviour. In the majority of cases it will be some form of machine learning algorithm, either supervised or unsupervised. For example, a supervised approach will use labelled data or signatures in order to assign categories to the new data points, while in the unsupervised approach clustering would be applied in order to separate the data. The separation module may apply multiple ML algorithms in multiple stages to achieve its goal.
If a classifier is in use, the classifier would have been trained using a labelled set of data before deployment. If an on-line clustering algorithm is in use, this module will include already clustered data to which the new point is added.
*Past knowledge **(G)**.* The past knowledge component is used for supervised approaches and represents the existing information which is fed into the system in order to classify the new data. In the case of an anomaly detection algorithm, the past knowledge would be previous behaviour that can be used for comparison purposes. If a classifier is used for the separation module, this could be a labelled dataset used to train the classifier, or a set of signatures. If a clustering algorithm is used, this past knowledge can be used to identify malicious and benign clusters.
*Output **(H)**.* The system will usually, at the highest level, output two sets of points: malicious and benign. However, in many cases the separation will be more fine grained, with data points assigned more specific classes (for example malware variant) in the supervised example, or groups of similar data points in the unsupervised example. In some cases, in particular in the case of an IDS, alerts will be outputted that identify hosts/entities that need to be investigated. These alerts may include extra information to assist network administrators in their investigations.

4. REVIEW OF CURRENT C&C DETECTION APPROACHES

Given the range of C&C design techniques, there is much interest in the design of techniques to localise C&C communication traffic by exploiting its special nature.

C&C detection falls broadly into three categories: signature-based, classifier-based and clustering–based. Signature-based detection systems attempt to recognise known patterns of behaviour in new behaviour, classifiers attempt to label samples using a model trained on past data, while clustering-based detection methods attempt to divide the malicious behaviour from the legitimate through statistical means.

We do not go into explicit discussion about C&C techniques in this survey, however we refer the reader to [Gardiner et al. 2014] for a survey on C&C techniques, including all those mentioned in this section.

4.1. Evaluation criteria

There are two primary measures of the success of a C&C detection system, the true positive (TP) rate and the false positive (FP) rate. The true positive rate measures the percentage of malicious samples that are labelled correctly as malware, while the false positive rate measures the number of legitimate samples that are incorrectly labelled as malware. There are of course other metrics used to evaluate machine learning, however the TP and FP rates are the most commonly provided amongst C&C detection literature and so we use these for comparison purposes. We also link the detection systems with the attacks against machine learning algorithms presented in Section 6.

4.2. Measurement and data collection

When detecting malware C&C, the selection of which data to collect and analyse is extremely important. For example, various detection methods require different levels of detail in the data. As networks scale, it will get progressively challenging to store all traffic — a requirement of most enterprise C&C detection techniques. Thus, if C&C traffic traces go unrecorded, then detection systems cannot work. While some techniques have been proposed to overcome the scalability limitations by developing sampling techniques, they do so without considering evasion resilience requirements. Also, little attention has been paid to tuning measurement in response to C&C evasion.

Effective monitoring and data collection are crucial for detection techniques. Traffic monitoring is performed by routers, commonly using Netflow [Cisco Systems Inc. 2016] feature or the sFlow feature. Alternatively, standalone measurement devices observing traffic via network mirroring devices or splitters (optical or electrical) are more flexible than in-router methods [Cranor et al. 2003]. In both cases, traffic traces are exported to collectors which store the traces.

There are monitoring systems that claim to be able to store all of the data. This will be affected by scalability issues. Even if all of the data can be stored (which is not guaranteed on larger networks), the processing of the data (such as applying an C&C detection technique) is far less scalable and will almost certainly encounter issues on a large system.

4.3. Signature-based methods

In signature-based detection methods, malware C&C is detected by looking for known patterns of behaviour. Signatures are generated for known malware samples, and then new traffic is compared to these signatures [Jacob et al. 2011].

Signatures are generated by analysing confirmed C&C traffic collected from various sources. The primary sources are honeynets and sandboxes. Malware is run in controlled conditions, and its activity recorded. Most aspects of the malware's network behaviour can be included in a signature, from statistical properties such as flow sizes, to detailed information such as packet contents.

4.3.1. Communication pattern detection. Malware variants often have very particular protocols when it comes to communication. These are often noticeably different compared to legitimate traffic, both in packet contents and in the behaviour of the communication. This makes signature based detection methods very good for detecting known variants of malware. Many different pieces of malware may also be based upon a common component, meaning that a single signature can be used to detect multiple pieces of similar malware. A popular method for detection is to produce signatures based upon the contents of packets. It is often the case that packets of data involved in the C&C of malware will be almost identical across multiple hosts. Even though some malware families use encryption in their communications, that encryption is usually a simple, lightweight algorithm (as the encryption is often for obscurity rather than security), so there are similarities among different ciphertexts. For example, in the work of Rieck et al. [2010], in which n-gram based signatures are generated for the payloads of malware that is run under controlled conditions in a sandbox. Signatures are also generated for legitimate traffic, and with this method the system can achieve detection rates of close to 95%, with a false positive rate of close to zero when running on a network gateway. Rossow and Dietrich [2013] extend this idea by providing a system for handling encrypted C&C channels. They leverage the fact that many malware variants use hard-coded, simple keys, and so they extract these keys through reverse engineering and attempt to decrypt all packets before pattern matching is applied.

Zand et al. [2014] propose a system for automatically generating text-based signatures for botnet detection. The system works by extracting common strings from observed traffic, and then ranking the collected strings by first clustering network traces, and then calculating the information gain for each string based upon the entropy of it's appearance in clusters. The highly-ranked strings can then be used for generating signatures. The system is tested on traffic collected from 1.4 million malware samples. After manually analysing the top 100 strings, the system found 29 good signatures, 41 signatures for benign traffic and 30 unknown, which is an improvement on previous work [Kim and Karp 2004].

Further to this, Rafique and Caballero [2013] proposed a system for large-scale automatic signature generation. The system uses network traces collected from sandboxes and produces signatures for groups of similar malware, covering numerous protocols. This system is able to identify numerous malware examples with a high TP rate, and experiences a low false positive rate due to the specificness of the signatures generated. The signatures are designed to be exported to intrusion detection systems such as Snort for use in on-line detection.

BotHunter [Gu et al. 2007] is a system for identifying compromised hosts based upon the actions they perform, more specifically the pattern of infection and initial connection to a C&C server. There are 5 steps to this pattern: inbound scan, inbound exploit, binary download, outbound C&C communication and outbound infection scanning (for propagation). These steps are identified as being a good generalisation of the typical infection model for a botnet. Detection is performed by looking for combinations of these actions within a certain time period. Tested on a live campus network and the system is able to achieve a 95% TP rate and low false positive rate.

4.3.2. DNS traffic analysis. There has been a large amount of work which attempts to provide a detection mechanism that can identify domains associated with malware at the DNS level. As we have seen, DNS is used by a large amount of malware that makes use of a centralised command and control structure.

Nelms et al. [2013] propose ExecScent, a system for identifying malicious domains within network traffic. The system uses traces of known malware samples to create signatures. The signatures are not just based upon domains, but also the full HTTP requests associated with them. This system is unique, however, in that to reduce false positives the signatures are tailored to the network that they will be used on, based upon the background network traffic. This accounts for the variance in the behaviour on different networks.

4.3.3. Malicious server detection. A slightly different approach is to attempt to detect the servers used for command and control directly. One approach for this is to use probing. Nappa at al. [2014] propose Cyberprobe, a system for automatically generating signatures for malware families by collecting traffic within a honeynets (and collecting data from public sources), clustering the traffic and creating a set of signatures for each cluster. The signatures are used to create probes. The system then probes IPs, and matches the responses to signatures to identify malicious servers. The system is extremely scalable, and can perform a scan of the whole IPv4 address space. Xu et al. [2014] provide a follow-up piece of work which creates the signatures by applying dynamic binary analysis to malware samples. This has the benefit that signatures can be extracted without a working server (a limitation of network trace-based approaches). Both of these approaches were successful in identifying multiple new servers for certain malware families, with 0% FP rates.

4.4. Classifier based methods

In a similar fashion to signature-based methods, classifier-based methods use past observations to assign labels to new samples. A classifier is trained using a dataset of labelled data with each point represented by a set of features. The labels for each data point in the dataset are known, making this an example of supervised learning.

The training data can be collected from many sources. Again, malicious data can be collected fro honeypots and sandboxes, or cam be collected from actual traffic that has been identified as malicious. Legitimate data is often collected from live network traffic that has any known maliciousness removed, or taken from known public sources. For example, sets of legitimate domains can be found by looking at the Alexa rankings.

4.4.1. Communication pattern detection. Rahbarinia et al. [2013] provide a two-step system for identifying hosts participating in malicious P2P behaviour. First, a **boosted decision tree** classifier identifies hosts that exhibit any P2P behaviour. Then, a 2-step process identifies the P2P network that the host belongs to. A classifier is trained for each different P2P application (the actual classifier is interchangeable, although they test with the **KNN**, **gaussian** and **parzen** classifiers), and then the host traffic is passed through all classifiers. If one classifier outputs a score above a threshold, the host is viewed as running that application. A **random forest** classifier is used to solve cases where two applications are matched. The first stage is able to identify P2P hosts with a TP rate of up to 90%, which can be boosted to 98.6% if the random forest classifier is used with a short time window. The effectiveness of the application classifiers varies effectiveness. For example, the legitimate applications and Zeus botnet can be identified with a 90% TP rate and at most 3% FP rate. The Storm and Waledac botnets achieve lower TP rates at 45% and 40% respectively, but with lower FP rates. The total misclassification error is 0.68%.

4.4.2. DNS traffic analysis. Exposure [Bilge et al. 2011] is a system for applying large-scale, passive DNS traffic analysis in order to identify malicious domains (not limited to those related to C&C behaviour, rather those involved in any malicious behaviour). The system extracts 15 features (time-based, DNS-based, TTL-based and domain name based). A training set is built using sources of known benign and malicious domains (Alexa ranking, blacklists etc). This is then used to train the **J48** classifier (an implementation of C4.5 decision trees). The system is tested on 100 billion DNS queries, using 10-fold cross validation. The system achieves detection rates of up to 98.5%, with a false positive rate of around 1%. The authors state that an adversary could possible evade the system through the use of rate-limiting DNS queries from infected hosts or by using uniform TTL values, although it is argued that these are unlikely to occur as they will lead to a drop in the performance of the malware.

Kopis [Antonakakis et al. 2011] makes use of the global view of the upper DNS hierarchy. In Kopis, a classifier is built that, instead of looking at the domains' IP and name, looks at the hosts that make the DNS requests. This leverages the fact that malware-related domains are likely to have an inconsistent, varied pool of requesting hosts, compared to a legitimate domain which will be much more consistent. The geographic location is also taken into account: requesters inside large networks are given higher weighting as a large network is more likely to contain infected machines. A feature set representing this information is used to train a **random forest** classifier. When tested on 5 months of data taken from two authoritative name servers, the system achieved up to a 98% TP rate with a 0.5% FP rate, and was even able to identify a new botnet based in China, which was later removed from the internet.

While not explicitly analysing DNS traffic, Ma et al. [2009] test the effectiveness of three popular classifiers **(bayes, SVM and logistic regression (LR)**) in identifying malicious domains. Datasets were created by combining blacklist data (for malicious domains) and online directory data (for benign domains) and then producing a dataset using a number of automatically extracted features. When tested using a 50-50 training/testing split, the SVM and LR algorithms achieved error rates of 1-3%. The interesting point is that the Bayes classifier, often chosen for its speed and scalability, achieved a higher error rate of 2.5-5%. The authors also tested the case where test data from a different dataset (different directory and blacklist)to the dataset that was used for training is used. The error rate was increased to 44%, which shows that in some cases training and testing on data from the same source may not give accurate real-world results.

4.4.3. Malicious server detection. Probing is not the only technique used for detecting servers. Bilge et al. [2012] propose DISCLOSURE, a system for identifying malicious servers from netflow data, extracting features related to flow sizes, client access patterns and temporal behaviour. In particular, the system is designed to work on very large (ISP level) datasets, for which sampling may have been applied. Labelled netflow records are used to train a **random forest** classifier, which is able to achieve detection rates of 60-70%, with false positive rates of 0.5-1%. To reduce false positives, information about ASs (autonomous systems) is taken from three public malicious server lists (FIRE, EXPOSURE and Google Safe Browsing), and a reputation score is calculated for each AS using the information from all three sources. Servers found on networks with an AS score below a threshold are assumed to be false positives and so are ignored, as the associated network is assumed to not participate in malicious activities. The authors also test the evasion resilience of the system, by creating simulated netflow traffic for two botnets that introduce random delays between network connections, and random padding to vary flow lengths, making the botnets appear more like benign servers. By feeding this netflow data into their classifier, they were able to a) successfully detect the botnets, b) improve general detection rates and c) detect botnets that were not previously detected by their system (the netflow data that the detection was applied to was different to the data inputted into the classifier).

4.5. Clustering based methods

The main disadvantage of using a signature or classifier based detection method is that these systems are usually not as effective at detecting new, or updated, malware due to an inherent assumption of stationary data. As malware changes freuquently, every time a new variant of malware is discovered, or an existing piece updates itself, the signatures have to be recreated or the classifier retrained. If the new variant is not discovered, then it is unlikely to be detected by these systems. Clustering based systems can account for unknown behaviour. In these systems, the algorithms attempt to separate different patterns of behaviour, without necessarily knowing itself what is malicious or benign. The output of clustering is often used to produce signatures.

One particular issue in detecting malware in large organisations is the problem of ”dirty” logs. This refers to the large variety in logs that are kept by different systems which are often incompatible and inconsistent, plus may also contain a large amount of duplication between different sets of logs. Yen at al. [2013] propose a solution to this in the form of Beehive. In Beehive, log data is first normalised; for example hosts are all mapped to IP addresses and timestamps are all converted to UTC. A set of features are then extracted for each host, which can be tuned to the setting, but as an example contain traffic volume information and enterprise policy breaches (for example accessing blocked pages). A iterative variant of **k-means** clustering is then applied which results in a set of outliers which are then labelled as suspicious. These hosts are then subject to manual investigation. When tested on a large enterprise network, the system produced alerts which were 25% malicious, 45% policy violations and 35% labelled as other.

Zhang et al. [2014] propose a system for detecting P2P botnets that differs from previous attempts in that it is unsupervised, meaning no knowledge of existing malicious behaviour is required. The system first applies clustering of flows in order to identify hosts that have taken part in P2P activity, and then applies two layers of filtering and then **hierarchical clustering** in order to group hosts that are taking part in a malicious P2P network. They show that it can separate malicious P2P traffic from benign with a 100% TP rate, and a low 0.02% FP rate.

In Botgrab, Yahyazadeh and Abadi [2014] also cluster flows using a custom online flow clustering algorithm, but then apply a reputation engine to identify hosts that have a negative reputation. Reputation is based upon participation in coordinated activities, an indication of botnet membership. The reputation engine can be combined with knowledge of hosts participating in malicious activities (for example DDoS) to achieve a true positive rate of at least 97% and FP rate of at most 2.3% (if malicious activities knowledge is not used this is reduced to 92% and 2.02%).

Yen and Reiter [2008] propose TĀMD, a system for identifying candidate groups of infected computers within a network by aggregating similar flows. Three aggregators extract aggregates of hosts based upon communication destinations, packet payloads and the host platform (OS). In particular, the destination aggregator makes use of a **k-means** clustering variant. This system is able to identify all of the malicious hosts in most cases (when VM outputs are inserted into background traffic), except for an IRC botnet which achieved a 87% detection rate.

4.5.1. DNS traffic analysis. We have already discussed signature and classifier based detection systems that make use of DNS traffic analysis. We will now explore systems that use clustering based methods. One proposed detection method is to make use of the reputation of domain names to decide if they are related to malicious activities [Antonakakis et al. 2010]. In this system (Notos), domains are clustered first by the IP addresses associated with them, and secondly according to similarities in the syntactic structure of the domain names themselves. The **k-means** clustering algorithm is used. These clusters are then classified as malicious or not based upon a collection of whitelists and blacklists: domains in a cluster that contains blacklist domains are likely to be malicious themselves. This system is run on local DNS servers and can achieve a true positive rate of 96% and a low false positive rate.

In an attempt to identify domains accessed as the result of a DGA, Schiavoni et al. [2014] apply clustering to domains extracted from DNS queries. The system first filters and removes domains that are human pronounceable, The remaining domains are then clustered with those collected from blacklists using the **DBSCAN** clustering algorithm, using features related to the IPs that the domains resolve to. These clusters are used to extract fingerprints, which are used to match new domains to known DGAs. When tested on a 3-month dataset taken from the DNSDB, with Conficker, Torpig and Bamital domains inserted, the system could filter out 50%, 35% and 62% of the DGA domains respectively, resulting in recall between 81 and 94%. The system also contains an intelligence module, that is able to track the evolution of IPs that groups of domains point to in order to monitor changes in botnet behaviour.

4.5.2. Fast flux detection. In a fast flux network (FFSN) the command and control server is hidden behind a proxy of numerous compromised hosts. Performing DNS queries on the domain of the server will return a large, and constantly changing, set of IP addresses. As you may expect, this type of behaviour is relatively easy to detect.

There are some differences between fast-flux service networks (FFSNs) and content delivery networks (CDNs) [Holz et al. 2008]. To detect a FFSN is a simple process, due to the two characteristics of an FFSN: short TTL values in DNS responses and non-overlapping DNS responses.

It is also possible to automatically detect which domains belong to the same FFSN. In Fluxbuster, Perdisci et al. [2012] apply **hierarchical** clustering to domains so they are grouped according to overlap in the returned IP addresses. By then comparing the clusters to previously labelled data, they can be classified (using the **random forest** classifier as flux or non-flux, revealing domains that make use of the same network. Tested on five months of live traffic data, the system is able to identify domains with a less than 1% FP rate,

4.6. Hybrid detection systems

BotMiner [Gu et al. 2008] is a system for detecting infected hosts without previous knowledge of specific botnets. In this system, bots are identified by clustering hosts that exhibit similar communication and (suspected) malicious activities. Activities are monitored using custom signatures for the Snort IDS. The clustering (**x-means**) groups hosts according to the botnet that they belong to, using the fact that hosts within the same botnet are likely to exhibit similar communication patterns, and will usually perform activities synchronously (such as DDoS attacks). When tested on 10 days of university traffic, with the traffic of 8 botnets inserted, the system achieved a 100% TP rate for 6 out of the 8 botnets, with the other two achieving TP rates of 99% and 75%. The system achieves a low false positive rate.

Antonakakis et al. [2012] describe a system for identifying previously unseen domain generation algorithms (DGAs) by taking advantage of the fact that DGAs result in large amounts of Non-Existent Domains (NXDomain) responses, and bots within the same botnet will generate similar NXDomain traffic. The system has a 3 step process. First, domains are clustered in two different ways using the **x-means** algorithm, first by string-based features of the domain name, and secondly by domains which share source hosts. These two sets of clusters are then combined through intersection, and any clusters small enough discarded. These clusters are then passed to a filtering step, that makes use of a multi-class variant of the **Alternating Decision Trees** (ADT) classifier (trained on previously discovered DGAs) to remove known DGAs. The system then produces a Hidden Markov Model (HMM) for each DGA that can be used to evaluate single domain names. When tested on the traffic data from a US ISP over a 15 month period 360,700 NXDomains were discovered, queried by 187,600 distinct hosts. The system identified 12 DGAs, of which 6 were new. For assigning domains to known DGA algorithms using the HMMs, TP rates of 99.7% and FP rates of 0.1% can be achieved.

4.7. Graph-based detection

A slightly different approach for detection is to make use of graph based detection approaches. These are becoming important due to the ability of graphs to be able to naturally represent communication patterns. While these approaches do not directly relate to the discussion on ML attacks, we include to provide examples of upcoming non machine learning based detection methods.

Several works [Collins and Reiter 2007; Iliofotou et al. 2008; Iliofotou et al. 2009; Zhao et al. 2009; Jelasity and Bilicki 2009] have applied graph analysis to detect botnets. The technique of Collins and Reiter [Collins and Reiter 2007] detects anomalies induced in a graph of protocol specific flows by a botnet control traffic. They suggest that a botnet can be detected based on the observation that an attacker will increase the number of connected graph components due to a sudden growth of edges between unlikely neighbouring nodes. While it depends on being able to accurately model valid network growth, this is a powerful approach because it avoids depending on protocol semantics or packet statistics. However this work only makes minimal use of spatial relationship information. Additionally, the need for historical record keeping makes it challenging in scenarios where the victim network is already infected when it seeks help and hasn't stored past traffic data. Illiofotou et al. [2008; 2009] also exploit dynamicity of traffic graphs to classify network flows in order to detect P2P networks. It uses static (spatial) and dynamic (temporal) metrics centred on node and edge level metrics in addition to the largest-connected-component-size as a graph level metric. Botgrep [Nagaraja et al. 2010] proposes a data mining technique to discover P2P graphs based on searching for expander graphs using random walks. Botyacc [Nagaraja 2014] performs a similar operation, by producing a dual graph of the communication graph, applying the Laplace-Beltrami operator to reduce the dimensionality of the data and the applying random walks to extract P2P networks. This can achieve detection rates of up to 99%, with an false positive rate below 0.1%.

Invernizzi et al. [2014] make use of a graph-based approach in order to detect the binary download stage of the malware infection process in large scale networks. Information collected from HTTP traffic is used to build undirected neighbourhood graphs, where nodes represent IP addresses, domain names, FQDNs, URLs, URL paths, file names, and downloaded files (represented as hashes). A graph is generated for each host that exhibits malware-related behaviour (such as domain-fluxing). Graphs are then assigned a metric, based upon the graph properties, on the likelihood that the candidate is malicious. When tested on a week-long ISP dataset, the system can achieve 60% precision and 90% recall on the malicious class, and 99.69% precision and 98.14% recall on the benign class, and is successful in identifying a large number of malicious downloads.

Manadhata et al. [2014] incorporate belief propagation into a graph model for detecting malicious domains. An undirected graph is produced, where nodes are hosts and domains, and edges go from hosts to domains they access. Using a ground truth instantiated from public whitelists and blacklists, belief propagation is applied until stabilisation occurs, with a domain labelled

Table I. Performance, and ML technique, of a variety of detection systems. Final two colums represent section numbers in this survey (NGE = not explicitly given in paper)

System	Target	TP Rate	FP Rate	ML Algorithms	Confirmed ML Attacks	Possible ML Attacks
Notos [Antonakakis et al. 2010]	Domains	96.8%	0.38%	k-means clustering		6.2.1, 6.2.2, 6.4.1, 6.4.1
Kopis [Antonakakis et al. 2011]	Domains	73.6-98.4%	0.3-0.5%	Random Forest classifier	6.1.1, 6.1.2, 6.1.3	6.3.3
Exposure [Bilge et al. 2011]	Domains	98.5%	1%	J48 decision trees (C4.5 variant)		6.1.1, 6.1.2, 6.3.3
Pheonix [Schiavoni et al. 2014]	DGA	NGE (80-94% recall)	NGE	DBSCAN clustering		6.2.1, 6.2.2, 6.4.2
Antonakakis et al. DGA [2012]	DGA	99.7%	0.1%	x-means clustering, Alternating decision trees		6.2.1, 6.1.1, 6.2.2, 6.4.1
FluxBuster [Perdisci et al. 2012]	FFSN	NGE	<1%	Hierarchical clustering, C4.5 decision trees	6.2.1, 6.4.1, 6.4.2	6.2.2
Zhang et al. [2011]	P2P	100%	0.02%	Flow clustering (distance-based)		6.2.2, 6.2.1, 6.4.1, 6.4.2
Zhang et al. [2014]	Stealthy P2P	100%	0.2%	BIRCH clustering, Hierarchical clustering	6.2.1, 6.4.1, 6.4.2	6.2.2
PeerRush [Rahbarinia et al. 2013]	P2P	90%	<3%	Decision trees, KNN, Gaussian and Parzen classifiers, Random Forest classifier	6.1.1, 6.1.2, 6.1.3, 6.2.2	
Firma [Rafique and Caballero 2013]	Multi-protocol C&C	NGE	0.00001% (live traffic)	Custom clustering		6.2.1, 6.2.2, 6.4.1, 6.4.2
Botgrab [Yahyazadeh and Abadi 2014]	Hosts	97%	2.3%	Custom online flow clustering		6.2.1, 6.2.2, 6.4.1, 6.4.2
Beehive [Yen et al. 2013]	Hosts	NGE	NGE	k-means variant		6.2.1, 6.2.2, 6.4.1, 6.4.2
TĀMD [Yen and Reiter 2008]	Hosts	87-100%	NGE	k-means clustering		6.2.1, 6.2.2, 6.4.1, 6.4.1
BotMiner [Gu et al. 2008]	Hosts	75-100%	NGE (low, 0.03%)	x-means clustering, signatures (SNORT)		6.2.1, 6.2.2, 6.4.1, 6.3.3, 6.4.1
Disclosure [Bilge et al. 2012]	Servers	60-70%	0.5-1%	Random Forest classifier	6.1.1, 6.1.2 6.1.3	

as malicious if the final belief value is above a threshold. This can achieve a TP-rate of 95.2% and an FP rate of 0.68%. They also show that the system can be run in near real time (processing 3 hours of data takes 16 minutes) meaning that malicious domains could be detected within 16 minutes of them first appearing.

5. CATEGORIZING ATTACKS AGAINST MACHINE LEARNING

As is evident in Section 4, many (if not most) detection systems make use of a machine learning algorithm (or set of algorithms) as the main component of the system for either identifying or isolating C&C traffic. Before we discuss specific attacks against the machine learning algorithms in section 6, we first discuss how to categorise attacks using various models from the literature. We also point out some of the weaknesses of the ML algorithms, and define the general threat model.

5.1. Weakness of ML algorithms

Early command and control detection systems were designed for detection of indiscriminate malware attacks, however the world has since changed. There has been an increase in more sophisticated, targeted malware attacks that specifically try to evade specific detection/ML systems. According to Verizon [Verizon RISK Team 2013], these now make up 25% of all attacks. The ML algorithms used in the early detection systems as discussed in Section 4 is in their design intentions. The ML algorithms were designed for use in situations where an adversary is not trying to have an impact on the outcome of the algorithm. The same machine learning algorithms are still in use, even in work published within the last year, whereas the attack field has evolved from indiscriminate to targeted attackers. Even though the security community [Barreno et al. 2010; Barreno et al. 2008] agree that a threat model should be constructed before the learning component is designed, this step is missing from the ML component design step in almost all detection systems. Performance of the learning component in ideal conditions is usually the primary consideration when choosing an algorithm. An interesting paper from Mersinal et al. [2015] discovers through surveying that IT security professionals (non-academics) are risk adverse, and will generally favour security over operability. This contradicts what we see in C&C detection papers, where performance in terms of both processing time and TP and FP rates is the driving force behind the choice of algorithm, with a lack of proper security considerations.

5.2. Attacker

The attacks described in this section are, in most cases, too advanced for the indiscriminate attacker, for example an independent botmaster using a purchased rootkit. For the indiscriminate attacker, the target for attack is in the most case individuals and small businesses who are unlikely to have any sort of detection system past simple, consumer anti virus products that can be easily evaded.

For the well-resourced and well motivated (e.g. state sponsored) targeted attacker, these attacks are well within their capability. These attackers are knowledgeable professionals who can put in the effort required to carry out the attacks, due to the large "profit" (in terms of money, information and disruption depending on the target) that may be gained.

5.3. Categorising attacks

Barreno et al. [2006; 2011] devise a set of properties for categorising attacks against machine learning algorithms (in particular, supervised-learning algorithms) (Table II). Their categorisation takes into account the goals of the attack in terms of the specificity (**targeted** or **indiscriminate**), the security violation being instigated (**integrity** or **availability**) and the influence of the attack (**causative** or **exploratory**).

5.4. Attack goals

At the broadest level, the attacker will have one of two goals:

— The attacker will want to achieve anonymity in the sense that they want to hide their presence.
— The attacker wants to compromise the integrity of the detection technique so that it becomes unusable.

While both attacks will achieve the same end goal of the attacker avoiding detection, in the second case the defenders will be aware that they are under attack, alerting them to the attacker's presence. In the first case, however, the attacker's presence is not revealed.

Table II. Properties of attacks against supervised learning classifiers according to Barreno et al. [2006]

		Description
Influence	Causative	Alter the training process through influence over the training data
	Exploratory	Do not change training data, but use techniques such as probing or offline analysis to discover information
Specificity	Targeted	Focus a particular point, or set of points
	Indiscriminate	Attacker has flexible goal, such as to increase false positives
Security Violation	Integrity	Results in intrusion points being labelled as normal (false negatives)
	Availability	Results in many classification errors that increases both false negatives and false positives, resulting in system becoming unusable.

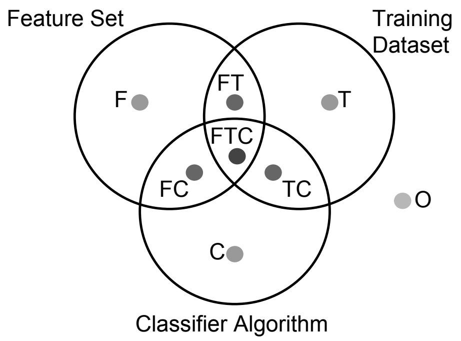

Fig. 3. Levels of attacker knowledge according to Srndic and Laskov [Šrndic and Laskov 2014]. O indicates attacker has little knowledge of any component.

These goals can be expressed more precisely using the Barreno model, and subsequent expansion by Biggio et al. [2014b]. Biggio et al. expand the Barreno model to represent attacks against classifiers, but similar goals apply to attacks against clustering algorithms. The Biggio model makes use of the specificity and security violation classifications from the Barreno model, with the addition of the “privacy” security violation, indicating a case where an attack will try to gain information from the classifier.

5.5. Attacker knowledge

One of the main considerations when discussing an attacker is the level of knowledge that an attacker has about the target system. In their attack against the PDFRate system, Šrndic and Laskov [2014] split this into knowledge about the classifier (and it’s parameters), the feature set in use and the training set. The attack knowledge is then a combinations of these, with knowledge of all three being the “perfect” case. This is illustrated in Figure 3. There is also the case that the attacker can gain no knowledge at all. This model can be transferred to the clustering example, by replacing “training dataset” with “test dataset” and “classifier algorithm” with “clustering algorithm”. It is worth noting, however, that knowledge of the test dataset in a clustering algorithm is different to that of the training data used in a classifier. The training data used within a classifier is pre-existing and may be publicly available so could easily be gained by the attacker. However, the input to a clustering algorithm (test dataset) includes all the new data (in the network example the new traffic data) so is more likely to only be estimated by the attacker.

Biggio et al. [2014b]. classify attacker knowledge specifically for the classification setting with more detail. As well as knowledge of the (1) training data and (2) feature set, they also incorporate the knowledge of the (3) learning algorithm and its decision function, (4) the decision function and its parameters and (5) the feedback available from the classifier (for example the class labels applied to sample queries from the attacker).

The attacker knowledge can vary between systems. There is a difference in systems that stem from published research, to commercially designed systems. Research papers that describe the design of systems are likely to, at a minimum, contain the ML algorithm in use, plus also the training/test data used (which may be publicly available) and at least a subset of the features used. Any production system that is released as open source software reveals the ML algorithm and feature set, although the training data may be user specific. For commercial, closed source software, probing attacks or reverse engineering can be used in order to discover at least partial information about the system [Corona et al. 2013].

The targeted attacker can improve their knowledge through research and social engineering techniques. For example, the motivated attacker could learn of contracts between an organisation and detection system provider, leading them to learn the detection system in use.

5.6. Attacker Capability

It is also important to consider the capability of the attacker. We assume that the attacker has no access to the control (ML) algorithm itself, rather they are limited to modifying the input (training or testing) data.

The ability of the attacker to change the data is in part limited by the specific application under consideration. For example, in the case of PDF files certain features can be changed freely, while changing some features has a direct impact on other features [Šrndic and Laskov 2014].

Biggio et al. [2014b] define the capability of an attacker (against classification algorithms) using four measures:

(1) The attacker influence in terms of causative or exploratory.
(2) Whether (and to what extent) the attack affects the class priors.
(3) The amount of and which samples (training and testing) can be controlled in each class.
(4) Which features, and to what extent, can be modified by the adversary.

This is also applicable to the clustering scenario (minus (2)).

5.7. Key terms

Some key terms that we will use throughout the rest of this section are as follows:

Learner. This refers to the machine learning algorithm
Surrogate learner. A surrogate learner is a copy of the learner produced by the adversary for testing attacks before they are carried out against the target. The surrogate has differing levels of completeness based upon the attacker knowledge. For example, the dataset in use could be an estimation of the one used by the target, or the learning algorithm, if unknown, could be substituted.
Production learner. The production learner is the instance of the learner in use by the target.

6. A SURVEY OF ATTACKS AGAINST MACHINE LEARNING ALGORITHMS

In this section we discuss attacks, taken from from the literature, against the machine learning algorithms used in the malware C&C detection systems found in Table I. We group attacks by their goal (evasion or poisoning), and by the group of algorithms affected (classifiers or clustering). We then present a number of techniques for each, give examples of demonstrated attacks from the literature and discuss any limitations of the attacks. Attacks are categorized according to the models defined in section 5. In section 7 we discuss the impact of these attacks in C&C detection problems.

We also look at some of the existing host-based techniques used by current malware to evade detection.

6.1. Evasion attacks - classifiers

6.1.1. Mimicry attack. Wagner and Soto [2002] introduced the idea of a mimicry attack, an attack in which the goal is to craft an attack point that appears as a benign point, hence is an example of an **exploratory, integrity** attack, that can be either **targeted** or **indiscriminate**. The attack either aims to move the attack point into the benign area (indiscriminate), or attempts to mimic a particular benign point (targeted). The attack attempts to modify the features of the attack point such that an anomaly detector or classifier labels the point as benign. The attack is limited by the ability of the attacker to modify the features under consideration by the learner. This attack typically does not require knowledge about the classification algorithm in use, as the attack is focused on exploiting knowledge of the distribution of malicious and benign points.

Affected algorithms. The mimicry attack works by effectively reducing the distance between the attack point and benign points. The attack has been demonstrated against the random forest, Bayes and SVM classifiers. As random forest is a multiple decision tree based classifier, the attack should also be effective against single decision tree-based algorithms such as C4.5. Biggio et al. [2013] demonstrate a gradient descent attack which theoretically works against any classifier with a differentiable discriminant function, including SVMs and neural networks.

Demonstrated attacks. Wagner and Soto [2002] demonstrate the mimicry attack against a host-based IDS, with a goal of identifying any traces of system calls that are accepted by the IDS, but still carry out some malicious activity. The attack can be performed by either adding no-ops (system calls that have no side effect on the system) or by removing system calls that cause alarms by replacing them with a different sequence of calls, that achieve the same end result (in exploiting the system). An example of this is to replace a sequence that opens up a root shell, with one that adds a new root user to the shadow file. The attack is mimicry due to the fact that the attackers find a sequence of calls that are accepted (labelled as benign) by the IDS, and so the attack “mimics” a legitimate sequence of system calls. The attack is tested against the pH IDS [Somayaji and Forrest 2000], by modifying the autowx attack script exploit. The attack is shown to be able to successfully evade the IDS.

Šrndic and Laskov [2014] demonstrate a mimicry attack against PDFRate [Smutz and Stavrou 2012], a system for the detection of malicious PDF files. PDFRate uses the **random forest** classifier to assign a score to PDF files, indicating their maliciousness. The system extracts 202 features by reading the file at the byte level, and applying regular expressions to extract the feature values. By inserting data into to the space between the cross-reference table and the trailer in the PDF file, attacks can be performed that will influence the output of PDFRate, but will not affect the rendering of the file in a normal pdf reader, which will parse the file and skip that data. To perform a mimicry attack, *the modifiable features are set to match those of a benign file*. The attack, using a surrogate classifier, tries different benign files until one is found that suitably reduces the score outputted by the surrogate. The paper focuses on attacks where the feature set is known (around 70% of the features are described in the original paper). Attacks are tested first on a local version of PDFRate that is built using the available knowledge based upon the adversary, and then the best strategy is applied and the result uploaded to PDFRate, with a goal of reducing the score assigned to the file. The attacker can modify the values of 35 features and increment a further 33. This is partly limited by the inter-dependability of the features. The attacks when applied onto files, which are then uploaded to PDFRate, can reduce the outputted score by 28-42% for attacks. Interestingly, when using a surrogate classifier using **SVM** rather than random forest, and then uploading the attack points to PDFRate, the scores outputted by PDFRate are still significantly reduced.

Biggio et al. [2013] demonstrate a mimicry attack with a gradient descent component against the **SVM** classifier and a **neural network**. The attack is considered in both the perfect knowledge (PK) situation, where the attacker knows all components of the classifier, and the limited knowledge (LK), where they only know which classifier is in use and the feature set, but not the training data or trained classifier (although they can produce an estimated surrogate training set). The attack modifies feature values until the attack point changes label. This is performed using a gradient descent approach with a component that favours points that imitate benign points. Including the mimicry attack makes the attack sub-optimal (when compared to the non-mimicry version described in section 6.1.2). The attack is tested on two datasets, a dataset of handwritten digits (represented as grey scale images) for the perfect knowledge case using a linear SVM classifier, and for the case of malicious PDF detection (with features extracted according to previous work by Maiorca et al. [2012]) in both the PK and SK cases, using both SVM and neural networks. The handwritten digits were modified by changing the grey scale pixel values, which the PDF files were modified by adding objects (and therefore keywords). In the handwritten digits example using SVM with the linear kernel, the attack when containing the mimicry component is slower (requiring more iterations), but can cause the digits to appear as another, and in the limited knowledge case could lead to a higher probability of evading the target classifier, requiring up to 50 modifications to achieve a false negative (FN) rate of 1. In the malicious PDF example, using linear SVM, the mimicry attack is effective in both the PK and LK cases. 50 modifications (iterations of the gradient descent) increases the FN rate to 1 or 0.75 for the PK and LK cases respectively. For the RBF kernel, 15 modifications still increases the FN rate up to around 0.8 in the PK case, or 0.6 in the LK case.. The neural network is more susceptible to the mimicry attack, requiring only 20 modifications in the perfect knowledge case to result in an FN rate of 1, while in the limited knowledge case an FN rate of 0.5 can be achieved with 50 modifications. When not using the mimicry component, 50 modifications in the PK case only results in a FN rate of 0.2. It is explained that as the pure gradient descent only finds a local minimal, this may not be enough to evade, however when incorporating the mimicry component the attack point is drawn towards an area densely populated by benign points.

Traffic morphing, which shares similarities with the mimicry attack, has been shown to be effective against the **Bayes** classifier. Wright et al. [2009] show that it can be enough to simply emulate the traffic feature(s) that is the focus of the detection system. They show that this works in the case of identifying web pages by traffic volume using the bayes classifier, for which the accuracy can be reduced from 98% to 4%, or 63% if the classifier is trained with attack samples.

Attack limitations. The attack is limited in two ways. First, there could be features that are not modifiable to the extent required to perform the attack. There is also a limit on how far the malicious sample can be changed — the sample still needs to serve it's original, malicious purpose. For example, in network traffic destination IP addresses cannot be chosen arbitrarily. The attack also requires knowledge of what is seen as normal in the production learner, which may not be available (although this can usually be estimated).

6.1.2. Gradient descent attacks. The gradient descent attack is a relatively common attack within the literature. The gradient descent attack is applied to both classifiers and clustering algorithms, and applies a gradient descent function in order to find a state for the attack point that achieves the desired result (point misclassified). Gradient descent is an optimisation algorithm that aims to minimise functions by iteratively moving a point (by changing parameter values) in the negative direction of the function gradient. In the evasion case, these are examples of **exploratory integrity** attacks. The typical approach is to generate an attack point and test its effectiveness on a surrogate learner. If the point is not sufficient, the gradient descent is applied, a new point generated and that tested until the desired result is achieved. In some cases, the gradient descent can be combined with a mimicry component to launch a targeted attack.

Affected algorithms. Because the gradient descent attack is not targeting a particular design component of classifiers, the attack is applicable to most classifiers. The attack finds any attack point that achieves the desired result for the attacker. For example, the technique used by Biggio et al. [Biggio et al. 2013], while only tested using SVM and neural networks, is theoretically applicable to any classifier with a differentiable discriminant function. We found example works (discussed below) against the SVM and random forest classifiers, as well as neural networks.

Demonstrated attacks. Šrndic and Laskov [2014] apply a gradient descent-kernel density estimation (GD-KDE) attack against the PDFRate system for detecting malicious PDF files (as described in section 6.1.1). The GD-KDE attack is used as a comparison to the mimicry attack. The GD-KDE attack is used to generate attack points using a surrogate classifier. They assume that the attacker does not know the classifier, and so replace the random forest classifier in their surrogate with an SVM classifier. The GD-KDE attack is able to reduce the score outputted by PDFRate by 29-35%, demonstrating the effectiveness against both the **SVM** and **random forest** classifiers.

Biggio et al. [2013] propose an attack based upon gradient descent that works at test time that theoretically works against any classifier with a differentiable discriminant function, including **SVM** and **neural networks**. The gradient descent can be combined with a mimicry component, as we discussed in section 6.1.1. When tested on the handwritten digits dataset in the perfect knowledge (PK) case using a linear SVM classifier, the attack is able to successfully change the outputted label with a limited number of iterations. In the malicious PDF detection example the attack is able to increase the FN rate to close to 1, with under 15 modifications to the attack point for the linear SVM with perfect knowledge (full knowledge of classifier, feature set and training data). With limited knowledge (knowledge of classifier and feature set only), 40 modifications are required, but 15 modifications still increases the FN rate up to 0.8. The results are only marginally worst for the RBF kernel. The neural network is more robust, 50 modification only increase the FN rate to around 0.3 for the perfect knowledge case, and 0.1 for the limited knowledge case. The authors mention that the gradient descent often does not successfully find an attack point for the neural network as for the majority of samples the local minimum found by the gradient descent is too far from the decision boundary to cause misclassification, and so the gradient descent terminates early.

Attack limitations. As with the mimicry attack, this attack is limited by the amount the attacker can modify the feature values. The attack also requires enough knowledge in order to be able to build a surrogate classifier in order to test candidate attack points. The more accurate the surrogate, the higher the chance of success of the attack.

6.1.3. *Other attacks.*

Genetic programming. Xu et al. [2016] apply a genetic programming approach to evading PDF malware classifiers, more specifically PDFRate [Smutz and Stavrou 2012] and Hidost [Šrndic and Laskov 2013] systems. PDFRate makes use of the **Random Forest** classifier (as described in section 6.1.1), while Hidost makes use of **SVM**. Hidost, according to its authors, is robust against adversaries, only suffering from 2 additional false positives (out of 5000 samples) under the "strongest conceivable mimicry attack". In contrast to previous work, the attack assumes no knowledge on behalf of the attacker, with the attacker only having access to a black box implementation of the target (the deployed system). Also in contrast to previous work, where it is assumed attackers can only increase feature values (in order to maintain malicious behaviour), the attacker in this scenario is able to modify feature values in many ways. The attack works by generating attack traces that represent series of operations of the original file to cause misclassification. Traces are computed using an iterative approach, with random operations applied to the file (such as inserting a page from a benign file). The modifications are evaluated using a fitness function, which incorporates the score from the target detector, and whether or not the file still carries out its intended malicious behaviour (evaluated using an oracle in the form of a sandbox). The attack is instantiated by taking a set of malicious files as an initial population, and then applying the modifications. Previously found successful traces are applied to some files to create new starting points to aid in finding attacks. The attack is tested on 1348 malicious pdf files, and is shown to be able to evade both PDFRate and Hidost with a 100% success rate. Attack traces generated for Hidost, but submitted to PDFRate, were able to evade detection in 77.6% of cases, whereas in the reverse case only 2 seeds were able to evade detection. The authors speculate this is due to the different feature sets in use by each system.

6.2. Evasion attacks - clustering

6.2.1. Mimicry attacks. The mimicry attack works by effectively reducing the distance between the attack point and benign points. While the attack has primarily been demonstrated against classifiers, it should also be effective against any clustering that uses a distance function. For example, the attack would be effective against the hierarchical clustering algorithm. The required effort on behalf of the attacker will in part depend on the cut parameter in use (this is the level in the hierarchy from which clusters are read, the lower the cut the more fine-grained the clustering), which in part dictates the distance between, and number of, clusters. The mimicry attack could also be used against the k-means clustering algorithm. The main limitation against k-means is that if a sufficiently high value of k is used, then the output clusters will be tight, meaning that the attack point will have to mimic the target point almost completely.

Demonstrated attacks. Biggio et al. [2013] define an obfuscation attack against single-linkage **hierarchical clustering** that is functionally equivalent to a mimicry attack. The attack works by selecting a target cluster, that the attack point should be clustered with. A line is then drawn between the original attack point, and the nearest point within the target cluster, and the new attack point is chosen from a point on this line no more than a maximum distance from the original attack point. This is tested in the perfect knowledge (full knowledge of the algorithm, features and data) case, using the MNIST handwritten digit dataset of greyscale images (as discussed in section 6.1.1). The attack is able to successfully merge attack samples into the target cluster with limited modifications.

6.2.2. Gradient descent attacks. The gradient-descent approach can also be used in an evasion attack for clustering algorithms. As with the classification example, the attacker will require a surrogate classifier. The attacker will then need to move the data point they wish to hide until it is merged into a benign cluster. As before, this is a **exploratory integrity** attack. The attack requires a large amount of knowledge to be successful, in particular relating to the shape of the benign data. The attack should be effective against any distance based clustering algorithm as long as the local optimum attack point that is found is close enough to a benign cluster.

Attack limitations. For best results, the attacker will require knowledge of the benign data that the attack point will be clustered with. Without any knowledge of the other test data, the local optimum attack point may not be close enough to benign clusters to be successfully merged. The attacker could estimate this data using previous data which they have access to, however this will potentially reduce the effectiveness of the attack (depending on how much the test data compares to the surrogate dataset).

Knowledge of the parameters of the clustering algorithm may also have an impact. For example, the threshold used within hierarchical clustering dictates how close the attack point will need to be to the benign points to successfully merge. Similarly, the k value used in k-means will dictate how fine grained the clustering is - a larger k value will require the attack point being closer to the benign points. If the attacker were to compute the attack point with different parameter values to the test clustering algorithms, the attack may not be successful.

6.3. Poisoning attacks - classifiers

6.3.1. Label Flipping attacks. Label flipping attacks are a form of poisoning attack with a goal of introducing label noise into the training data by flipping labels. The attacker is able to cause an amount of the legitimate samples to be labelled as malicious in the training data, or an amount of the malicious samples to be labelled as legitimate. The amount is down to the capability of the attacker. Label flipping is an example of a **causative integrity/availability** attack that attempts to maximise classification error. There have been many attempts at designing ML algorithms to provide robustness against random label noise arising from noisy data sets, but these designs do not consider the adversarial case where the attacker attempts to maximise the classification error. The extent to which labels can be flipped is restricted by a budget (the number of labels that can be flipped).

In the real world, the attacker would need to have an impact on the data collection phase. There are a number of ways they could do this. One of the most common methods for collecting malware data is through the use of honeypots and sandboxes. Malware has been shown to be able to identify when it is being run in a virtual environment (see section 6.6 “Evading dynamic analysis systems”), so the malware could be programmed to engage only in legitimate behaviour when in a sandbox, which would then create legitimate samples that are labelled as malicious.

Affected algorithms. As discussed below, the attack has been demonstrated on the SVM classifier, although it should affect any classification algorithms affected by label noise.

Demonstrated attacks. Xiao et al. [2012] discuss a causative attack against **SVMs** which makes use of label flipping in order to poison the training set. In this attack, the adversary flips labels of training points, constrained by a budget, with attack points chosen to inflict the maximum loss. On an artificial, two-dimensional dataset the attack is able to increase the error rate up to 32% (on a binary class dataset with a 50/50 split), which is not far from reducing the classifier to a random guess. When tested on a set of 10 real world datasets (downloaded from the LIBSVM website), the attack is able to reduce the classifier to a random guess (a 50% error rate) by flipping 10% of the labels of the training set. An interesting observation is that the attack is tested on an SVM classier trained with both a linear kernel, and an RBF kernel, and is found to be much more susceptible to attack when using the RBF kernel. The authors speculate that this due to the fact that when using the RBF kernel instances are mapped to the infinite dimensional feature space, and so instances are more sparsely distributed. Therefore flipping a label has a greater effect on the hyperplane. This indicates that even variations on an ML algorithm have different tolerances to attack. The attack is also effective to a lesser degree against the label noise robust SVM (LN-SVM) algorithm [Biggio et al. 2011].

Attack limitations. The primary limitation of the attack is the budget limiting the number of points in the training set that can have their labels flipped. The more restrictive the budget, the more restricted the attack. The attack also requires some knowledge on behalf of the attacker of the classifier and training data in use. If this information is limited, then the attack becomes far more difficult for the attacker.

6.3.2. Gradient descent attacks. Gradient descent-based poisoning attacks impact the **availability** of the learner. These are **causative** attacks, which can be either **targeted** or **indiscriminate** in nature.

As opposed to the evasion case (see sections 6.1.2 and 6.2.2), where attack points are moved in order to be misclassified, in the posioning case the gradient descent attack inserts attack points into the training data chosen to maximise the impact on the classifier performance, with a goal of reducing performance to the level that the classifier is unusable. The common approach is to start with a benign point, flip it's label and move it (according to the gradient descent function) to maximise the objective function.

Affected algorithms. The gradient descent-based poisoning attack has been demonstrated against the **SVM**.

Poisoning attacks work by effectively introducing adversarial noise into the training data. In that sense, a gradient descent-based attack is theoretically effective against any classification algorithm that is not tolerant to noise. Even if an algorithm is tolerant to random noise, it may be possible to apply the attack in a controlled manner and still successfully carry out the attack, although the complexity will be increased.

Demonstrated attacks. Biggio and Laskov. [2012] use a poisoning attack to attack **SVM**. The attack assumes knowledge of the training set used by the learner. The attacker inserts attack points by flipping the label of a point in the target class to create an initial attack point, and then applies a gradient descent to find an optimal attack point which is then added to the training set. When evaluated on an artificial, two-dimensional dataset the attack can achieve a classification error of up to 0.06 for a linear kernel, and 0.035 for the RBF kernel. The attack is also tested on the handwritten digit dataset mentioned previously, using an SVM with the linear kernel, which achieves a classification error of 0.1 to 0.3 after 200 iterations of the gradient descent algorithm.

Attack limitations. The attack is limited by the amount of noise that the attacker can inject. One example is the attackers ability to cause data points to be labelled as they require. For example, when attacking a spam filter, the spam label usually arises from the user, and so the attacker is reliant on the oracle labelling the points in the way the attacker desires.

6.3.3. Dictionary attacks. A dictionary attack is a specific form of poisoning attack that can be carried out against classifiers trained using token-based features. In this attack, malicious data is inserted into the training set that contains tokens (for example words) that feature in benign data. The intention is that the malicious data will be discovered, and included in future training sets. The goal is then to cause benign data to be misclassified as malicious, indicating a **causative availability** attack. This can both be **indiscriminate** (any benign point can be misclassified) or **targeted** (the attacks want to cause a particular benign sample to be misclassified). Nelson et al. [2008] provide attacks against SpamBayes[1], a system for identifying spam email messages. Two causative availability attacks are proposed against the system. The first, an indiscriminate dictionary attack, sends emails to a destination covered by SpamBayes which contains words likely to appear in legitimate emails. These spam emails will be included in the training data when the system is retrained (which happens periodically), increasing the likelihood that legitimate emails will be labelled as spam. In the optimal case, the emails will contain the full English dictionary. As a refinement, the email can just contain a dictionary closer to the victims word distribution.

The second attack, a targeted attack, assumes knowledge of a specific email that the attacker does not want the intended recipient to read, by causing it to appear in the victims spam folder. This attack involves including words specific to that email in the attack emails, hence the attack effectiveness is limited by the attackers knowledge of the structure of the target email. In the optimal case, this will be the full dictionary attack, which will contain all of the target words.

When tested on a large dataset, the attacks are proved to be effective. The dictionary attack, when attack emails make up 1% of the dataset (achievable by a large scale attacker), can cause 90% of the legitimate emails to be labelled as unsure or spam. The targeted attack can, with 30% knowledge of the target email, change classification in 60% of cases.

The SpamBayes algorithm is shared by BogoFilter [2] and SpamAssassin [3], and so the proposed attacks should have a large impact on these systems as well.

6.4. Poisoning attacks - clustering

6.4.1. Bridging attacks. Bridging attacks are a form of poisoning attack that has been demonstrated against clustering algorithms. As with the other poisoning algorithms, they are examples of **causative integrity/availability** attacks.

The attack introduces points into the space between clusters, with a goal of causing clusters to split and merge. In hierarchical clustering, introducing points between clusters causes the inter-cluster distance to be affected, which can then cause clusters to merge.

[1]http://spambayes.sourceforge.net/
[2]http://bogofilter.sourceforge.net/
[3]http://http://spamassassin.apache.org/

Affected algorithms. The attack has been demonstrated against variant of the **hierarchical clustering** algorithm. The attack should be effective against any clustering algorithm that use the inter-cluster distance. The attack may have an effect against centroid-based algorithms such as **k-means** clustering, although to a lesser extent. The introduction of points between clusters will have an effect on the final placement of centroids, which could cause points to be clustered with points which they would not have previously been clustered with. The clusters are unlikely to merge (depending on the structure of the data and initial location of the centroids), but if the attack points are chosen carefully the attack points could cause clusters to merge.

Demonstrated attacks. The Malheur system is the target of work by Biggio et al. [2014]. Malheur [Rieck et al. 2011] applies **single (or complete) linkage hierarchical clustering** to MIST malware behaviour reports, which contain sequences of host-based system events, where individual execution flows of threads and processes are grouped in a single, sequential report.

The paper assumes the scenario in which the attacker has perfect knowledge, and reduces the attack to an optimisation problem, in which the objective function to be maximised is the distance of the clusters formed when under attack from the clusters formed in absence of the attack. The attack goal is to cause clusters to merge until the system becomes unusable. The attack is run on a local version of the system, where all possible attack points are considered, and only the one that has the largest impact is used. The attack is applied iteratively, continuously adding points until the objective function is maximised. The attacks also propose two variations, which are less computationally expensive as they instead estimate the effect of the attack points, rather than re-run the clustering algorithm. To generate attack samples, an existing malware sample is taken and features are manipulated by increasing their value (in order to maintain the malicious capability of the samples).

When tested on two datasets, the same dataset used in the original Malheur paper and a new malware dataset collected by the authors, the bridge attack is able to reduce the number of clusters from 40 to 5 with only 2% of injected samples. The estimation-based approaches require more injected samples, but are far less computationally expensive. To verify the results, two random based attacks (one that generates an attack point at random, one that chooses the best of a number of random attack points), are shown to be ineffective. Finally, an attack that works as the bridge attack, but minimises the F-measure, achieves comparable results to the bridge attack.

Very similar attacks are applied to three different datasets in [Biggio et al. 2013]: the banana-shaped dataset from PRTOOLS, a malware C&C dataset and a dataset of handwritten digits. As before, perfect knowledge on behalf on the attacker is assumed. The attacks aims to cause clusters to split or merge to impact on the availability of the algorithm. The attacks are compared against a random approach. On the banana shaped dataset, the bridge attacks are found to outperform the random attacks. The best approach results in a lower number of final clusters than in the unpoisoned dataset. On the malware dataset, the results are similar. On the digits dataset, the random approaches are shown to be ineffective. The three bridge based approaches cause significant fragmentation of the final clusters.

Attack limitations. To achieve maximum results, the attack requires perfect knowledge of the target classifier, which can be an unrealistic assumption in the real world. The attack could still be successful if a surrogate data set is used to approximate the target classifier, although this has not been explored in the literature. The attacker only requires knowledge of one other cluster in order to perform the bridge attack. This cluster could be estimated with a high accuracy (for example the traffic to "facebook.com" will be similar wherever it appears so may be reliably estimated).

6.4.2. Gradient descent attacks. As with classifiers, gradient descent poisoning attacks on clustering algorithms are **causative availability** attacks. In the clustering case, attack points are inserted to cause clusters to split and possibly merge with others, reducing of the accuracy the clustering. Attack points are created and the most effective points are found by maximising the objective function using a gradient descent approach.

Affected algorithms. This class of attack has been demonstrated against **hierarchical clustering**.

As this type of attack exploits the distance function used within the clustering algorithms, it is theoretically applicable to other types of clustering, such as k-means. However, whereas a single attack point can be sufficient to cause a cluster to split in algorithms that use intra-cluster distance (such as hierarchical clustering), to cause a split in centroid-based algorithms would likely require a larger number of attack points to significantly shift the centroids.

Demonstrated attacks. The **complete-linkage variant of hierarchical clustering** is also attacked by Biggio et al. [2014]. The attack is again a poisoning attack with perfect knowledge on behalf of the attacker. In this attack, attack points are added to the edge of clusters with a goal of causing the cluster to split, and possibly merge with another. As before, the attack is run iteratively with the goal of maximising the objective function (which represents the distance between the clustering output under attack compared to not under attack). There are also two variants that use estimation to reduce the number of iterations required. This attack is tested against three datasets (taken from [Biggio et al. 2013]). In the first, taken from PRTools the extend attack causes the clustering to result in a smaller number of more distinct clusters. The second dataset, which contains malware C&C behaviour represented by 6 features, gives similar results. Finally, a dataset of handwritten digits, represented by 28x28 grey-scale images converted into 784 dimensional data, is used. In this case, the estimation attacks are more effective than the fully-iterative attack.

Attack limitations. As in the classification case, the attack is limited by the amount of noise that the attacker can inject. Depending on the density of the data, the attacker may be required to introduce a large number of attack points to cause significant degradation in performance.

6.5. Other attacks

6.5.1. Attacks on automatic signature generation (ASG). Perdisci et al. [2006a] propose an attack against Polygraph [Newsome et al. 2005], a system for automatically generating signatures for the detection of polymorphic worms. The attack assumes that an attacker who controls host A, and wants to infect host B, will send a malicious flow containing the attack code. The hope is that host B will be used for collecting flows for input into Polygraph.

The attack works by sending extra flows along with all malicious flows, that are constructed to not actually perform any attack, but contain specifically crafted data that will cause the generated signatures to only include data related to the protocol framework, rather than invariants related to the worm, effectively rendering the signature useless. The false flows are generated by taking a copy of the malicious flow, and applying a number of transformations over it. These include randomly permuting bytes and injecting substrings from legitimate flows.

The attack is tested on a recreation of Polygraph using the Apache Knacker exploit for the polymorphic worm, with flows collected from the university network for use as innocuous traffic. The original Polygraph paper claims that 80% noise is required to affect the system, but the authors find that 1 false flow per malicious flow can result in useless signatures up to 44% of the time. If two flows are used, this increases to up to 85%.

Newsome et al. [2006] propose a causative integrity attack against conjunction learners (of which Polygraph is an example), dubbed red herring attacks. The attack involves including spurious features into attack samples, that will be included in signatures. To evade detection, the attacker then stops including the extra features. This attack only works on conjunction learners, and is not effective against other machine learning algorithms such as Bayes. For Bayes learners, they propose a causative availability attack, dubbed the correlated outlier attack. In this attack, the attack sample includes spurious features that appear in legitimate samples. This increases the Bayes score of legitimate samples, leading to a choice of either high false positives or false negatives.

Chung and Mok [2006; 2007] propose "allergy attacks" against the Autograph [Kim and Karp 2004] signature generator. Allergy attacks have a goal of performing a DOS attack, by leading the generator to produce signatures that match to normal traffic. The attack is performed by first sending attack traffic that causes Autograph to mark the node as suspicious, and then traffic that resembles legitimate traffic. The attack can be supplemented with a corpus so that it will also affect Polygraph and similar ASG systems. The authors show that the attack is successful with only a small number of packets required to cause a target site to be blacklisted.

6.5.2. Attacks on intrusion detection systems. Fogla et al. [2006] propose polymorphic blending attacks against the PAYL, a byte-frequency based intrusion detection system [Wang and Stolfo 2004; Wang et al. 2006]. The attack extends polymorphic attacks by adding a blending component that, like mimicry attacks, matches the statistics of mutated attack instances to normal traffic profiles to evade byte frequency-based networks anomaly IDS systems. This is done by "encrypting" the packet payloads by using a substitution table that replaces bytes in the original packet with bytes in the target distribution. Padding is also added, and packets can be split to match the normal distribution. The substitution table is sent along with the packet, as well as a "decrypter" that can recover the original packet. The attack also ensures that the attack distribution matches the normal distribution despite the table and decrypter being included. The attack is tested on both the 1-gram and 2-gram variants of PAYL, using the MS03-022 vulnerability as the base attack. The normal traffic distribution is taken from 1 day collected from a university network, and the training set for PAYL is taken from a further 14 days. The attacks are tested with target packet lengths of 418,730 and 1460. The attacks are shown to be effective in reducing the anomaly score outputted by PAYL to below thresholds that would realistically be used. They also find that increasing the number of packets used by the attack, meaning smaller individual packets, further reduced the anomaly score as the smaller packets could be made to more closely match the normal distribution. The authors provide a formal analysis of the attack in [Fogla and Lee 2006].

6.6. Emerging host-based evasion techniques

We will now briefly review some techniques on the host side used by malware in the wild to evade detection. Recently there has been an increasing amount of malware that provide methods of evading current detection and analytical systems.

Evading signatures. Traditional defence systems (such as traditional anti-virus and intrusion detection systems) often rely on signatures to detect attacks or malicious code. A signature characterizes a known attack by defining its characteristics. For example, in the context of malware, a signature could be a regular expression that matches the bytes found in a specific malicious file. Unfortunately, a number of obfuscation techniques have been proposed (and are used extensively) to counter signature-based detection. For example, polymorphism [Hosmer 2008] is a technique that enables an attacker to mutate an existing malicious binary and create a completely new version from it, while retaining its original functionality but remaining undetected by current signatures. The anti-virus vendor Kaspersky recently reported detecting more than 2 unique malicious samples per second, likely the result of extensive application of polymorphic techniques [Kaspersky 2013].

Evading dynamic analysis systems. To overcome the limitations of signature-based analysis of malicious code, researchers use dynamic analysis tools, also called sandboxes [Egele et al. 2012]. These tools execute a binary in an instrumented environment and classify it as either benign or malicious depending on the observed behaviour.

To thwart automated dynamic analysis, malware authors have developed a number of checks (so-called “red pills”). To detect sandboxes, the checks leverage differences in execution characteristics between a real host and virtualised environments [Ferrie 2007] or emulated systems [Paleari et al. 2009]. Further, malware may execute its malicious payload or specific parts of its code only when some “trigger” fires, i.e., only when some specific precondition is satisfied [Moser et al. 2007]. As another evasion technique, malware use stalling code [Kolbitsch et al. 2011] which delays execution of malicious activity just long enough that the automated analysis system stops the analysis having observed benign activity only, and moves on. This technique simply leverages the fact that, to analyse a large volume of programs, an analysis system must bound the time it spends executing a single sample to a limited time (in the order of few minutes).

Evading reputation systems. Another defensive approach that has gained traction in the last few years is the use of reputation information for network entities (servers or domain names). Malware authors have a crude but effective attack against such reputation blacklists: they use a certain server or domain for malicious purposes only for a very limited amount of time. Recent data from researchers at Google shows that this strategy is well in use: they studied domains hosting exploit kits used in drive-by-downloads and found that their median lifetime is only 2.5 hours [Grier et al. 2012]. Clearly, an effective blacklist should be able to detect the malicious domain and distribute this knowledge to all the enforcement devices before the domain has been abandoned.

7. DISCUSSION

7.1. Why is secure machine learning not in use for C&C detection?

it is unclear why secure machine learning algorithms are not in more widespread use within the field of C&C detection. Below, we discuss some of the possible reasons why secure ML is not in use.

Lack of awareness:. A simple and obvious reason is simply a lack of awareness of secure ML algorithms. Unless one specifically looks towards the secure machine learning field, they are unlikely to be aware of the more secure algorithms, or even that the vulnerabilities in the simpler algorithms exist.

Ease of access:. More specifically, we mean access to an implementation of the code, either through an open source piece of available software, API or an algorithm that is easy to implement by an academic. Even if a piece of software or API exists, it may not be available for the desired scenario without large amounts of pre processing on the data to ensure it is in the correct format. If we compare this with the popular algorithms such as k-means or SVM, these have many implementations (in both software and libraries) that can easily be applied to almost any dataset.

Reduced performance:. It may be the case that secure ml algorithms, when compared with the non-secure counterparts, achieve lower performance in terms of both processing time, and in the key metrics (TPR/FPR and similar). When evaluating detection systems, these metrics are key selling points that elevate new work above previous. **While this assumption is not true for all works, there are some examples in the literature.For example, Brückner and Scheffer [2011] demonstrate an increase in execution time in the multi=classifier system from $\sim 10^1$ to more than 10^3 seconds using the largest training set. In terms of accuracy, some works [Zhang et al. 2015; Biggio et al. 2011] show a reduction in performance when not under attack, while others show a slight increase [Biggio et al. 2010].**

Lack of clear security metrics:. When choosing a machine learning algorithm, it is common to look to the standard performance metrics (true/false positive rates, precision, recall, computation efficiency and so on). In a non-secure setting, these are usually sufficient in choosing one algorithm over another. However, as far as we are aware, there are currently no clear metrics for measuring the evasion resilience of machine learning algorithms. We feel that a lack of these metrics is one of the reasons why more secure machine leaning algorithms are not in use — there is currently no way to see a measurable benefit to their use over the simpler algorithms. This is not a trivial problem due to the large range in threat models which need to be considered, as the threat model in use impacts on the achieved level of evasion resilience.

Lack of evidence of attacks:. **The reason for not using secure algorithms could be purely down to the fact that there is little to no evidence of attacks targeting machine learning in practical applications. Attackers are likely to evade detection by not very complex detection systems using simpler attacks. Of course in the attacker-defender arms race we should expect attackers to become more sophisticated as detection systems become more complex so it is only a matter of time before attacks against machine learning become more commonplace.**

7.2. Difficulty of applying attacks in C&C

Almost all of the attacks against machine learning algorithms we discussed in section 6 were applied to simplified use cases using easily modifiable data points. In many of the cases, the features could easily be modified to carry out the attack. In the malware C&C case the features may not be as modifiable as in some other examples. For example, if we look to a simple C&C channel that sends data to a centralised server over tcp. Some features, such as any relating to the packet contents (length, n-grams etc.), or the number and size of packets, are easy to modify by the attacker. However, other feature values, for example representing IP addresses, port numbers or protocol flags, are not as easily modifiable, as they could either have an impact on the functionality of the communication channel if changed, or not be under the control of the attacker.

Carrying out poisoning attacks that require influencing the training data may also prove to be difficult. The attacker would have to ensure that their attack points are collected by the system engineers to be included. As an example, this may rely on the attackers malware being caught in a honeypot, and it may be difficult for an attacker to ensure this occurs.

7.2.1. Availability of attacker knowledge. In all of the attacks discussed in section 6 the attacker requires some level of knowledge in order to perform the attack. The amount of knowledge required varies between the different attacks. For example, the mimicry attack can be successful using surrogate datasets and algorithms that do not necessarily match those in use at test time. However, for attacks such as those based upon gradient descent, more specific knowledge is required for the attack to be effective.

One issue is therefore in the real world how much of this knowledge is available to the attacker? In part this depends on the target (organisation) of the attacker. The attacker may be able to work out through social engineering techniques the specific detection systems in use, and therefore gain information about the algorithms, and possible feature sets in use, either through information published by vendors or through reverse engineering. If the target system is an off-the-shelf product, the attacker could simply purchase a copy to use as a surrogate. It is well known that malware authors test new malware against popular anti-virus systems prior to deployment.

Knowledge of the specific training data in use also depends on the specific target. The training data may not be made public by the system designers, or the training data could be tailored to suit a particular customers network (to maximise performance). In the real world the attacker is far more likely to be able to create a surrogate training by making use of public datasets and incorporating benign date that is expected to appear.

7.3. Effectiveness against full C&C detection systems

While machine learning algorithms are a core component of C&C detection systems, the ML algorithm is not the only step. Many of the systems will apply various levels of **pre- and post-processing** to the data, including but not limited to whitelisting/blacklisting, noise reduction and data sampling. **Clustering algorithms are often used as a way to separate data in order to create signatures afterwards, which may be tolerant to some degree of error**. The attack examples discussed in section 6 usually focus on a scenario where the test samples only undergo feature extraction, and are then fed directly into the machine learning algorithm. Therefore it is unclear how much extra steps in the C&C detection system will influence the effectiveness of the discussed attacks. Of course, some of these extra components may also aid the attacker. For example, Botminer [Gu et al. 2008] (a C&C detection system that uses x-means clustering as the main ml algorithm) generates some of its input data through the alerts thrown out by an anomaly detector. In the process of attempting to attack the clustering algorithm, for example through the use of a mimicry attack, the attacker may also inadvertently evade the anomaly detector, meaning that they are far less likely to be detected (in the case of Botminer there are other data sources although the anomaly detector inputs play a major role).

7.4. Limitations of current detection approaches

In surveying the field of C&C detection approaches, we found a number of limitations.

Limited datasets. Acquiring useful datasets is a well known problem for malware researchers. Legal requirements often mean that datasets cannot be released once created, meaning that each new system will usually need to have a dataset created for it. This data may not be a true representation of a real-world situation, indicating that the observed performance may not be repeatable in the real-world.

Limited reproducibility of datasets. Typically C&C detection systems are evaluated using network traces collected from real-world networks, from smaller university networks up to the ISP level. These datasets are almost never released for use by others, due to contractual agreements and privacy concerns. This makes it difficult to reproduce the evaluation of these detection systems.

Testing on known malware. A problem which is especially evident in systems that rely on classifiers is the problem of testing on a small collection of known malware samples. These samples are used to generate signatures, and then the traffic from the same samples used to generate a test dataset. This can lead to higher detection rates than would be seen in the more realistic situation when the classifier is trained only on related malware families.

Lack of scalability testing. Due to finite resource both in terms of data availability and processing resources, the testing of detection systems is often small-scale in nature. or large scale but run in batch mode taking a long time. Many of these systems are designed with the intention of being installed within a corporate network or even an ISP where the volume of data to be processed may be far larger, and closer to real time detection is required. Some of the designed systems, while working well in a small-scale setting, may break down in a larger scale environment.

7.5. Attack defences

We will now cover a few approaches to defending against the attacks discussed in Section 6. Note that a complete survey on this is out of scope of this work, and sop we only focus on defences relevant to C2 detection systems. For a more detailed discussion on defences against attacks we refer the reader to [Biggio et al. 2014a].

Defending against ML attacks is a difficult task. Defenders can take both reactive and proactive approaches [Biggio et al. 2014a]. In a reactive approach, the defender observes an attack, and then incorporates countermeasures into the existing system. In a proactive approach, the defender anticipates the adversary's strategy and develops countermeasures before deployment. The majority of defences are proactive in nature, attempting to limit the capability of the adversary before deployment. In particular, the two common approaches are the use of multi classifiers systems (MCSs), and the use of ideas from game theory to predict and model the attack at training without having access to attacker data.

One of the most common approaches to defend against evasion attacks on classifiers is to make use of multi classifier systems (MCSs) [Kolcz and Teo 2009; Zhang et al. 2015; Biggio et al. 2010], in which the combined output of a number of classifiers are used to make a decision. The different classifiers vary by either using different subsets of the feature set or different subsets of the training data. One variation of MCSs is to make use on one-class classifiers, which can be more effective in identifying attacks. One-class classifiers produce a tight decision surface around the target objects (for example the benign objects), and output how much a datapoint matches that target, rather than outputting a label from a set. These work particularly well when combined with two-class classifiers, as described by Biggio at al. [2015]. They find that, while two-class classifiers achieve better accuracy when not under attack, one-class classifiers achieve better accuracy while under attack. The classifiers are all based on SVMs. By combining two one-class and one two-class classifier, a more robust MCS is produced. Performance of the MCS is less effected when under attack, n particular good word insertion (GWI)/bad word obfuscation (BWO) attacks against spam filters and adding features in PDF files, compared to a single SVM classifier. Perdisci et al.[2006b] make use of a set of one-class SVM classifiers to provide resistance against mimicry attacks in payload-based intrusion detection systems (such as PAYL). When tested against polymorphic blending attacks [Fogla et al. 2006] the SVM-based multi classifier IDS is able to detect the occurrence of all the attacks with a 0.5% FP rate (correctly identifying 99.2% of all attack packets). Biggio et al. [2010] evaluate the robustness of two approaches to MCSs: bagging (in which each classifier is trained with a subset of the training data) and the random subspace method (RSM, where each classifier is trained with all the data, but a subset of the feature set). The two approaches are tested using linear SVM and logistic regression (LR) in the spam detection scenario, under the GWI-BWO evasion attack (with both perfect and limited knowledge). When not under attack, the bagging and RSM MCSs almost always outperform the single classifiers. Under the perfect knowledge attack, the bagging method reduces the impact of the attack for both the SVM and LR classifiers, while the RSM method only improved the robustness of LR. In the limited knowledge case, LR is improved slightly by both the bagging and RSM methods, while SVM is improved by bagging only under the limited knowledge attack.

Zhang et al. [2015] provide a method for performing feature selection while providing robustness to evasion attacks. Feature selection, where only a subset of the possible features are used for training, is used to reduce the time and computational complexity of the algorithm, and provide better learning on smaller training sets. This can however lead to further evasion if the adversary is aware of the features that are selected, due to the fact that there are less features to attack. The proposed method, wrapper-based adversarial feature selection (WAFS) is based upon the popular forward feature selection and backward feature elimination algorithms. The algorithm is tested in the spam detection scenario using a linear SVM classifier and in the PDF malware detection scenario using SVM with the RBF kernel. the algorithm is shown to produce more robust classifiers when subjected to gradient-descent based evasion attacks, when compared to the forward feature selection, backward feature selection and adversarial backward feature selection algorithms.

Kolcz and Teo [2009] use an approach to MCS that involves a feature re-weighting step, as an alternative to feature selection. In the re-weighting step, the classifier is trained in two passes. In the first pass, the classifier (SVM or logistic regression (LR)) is trained, outputting feature weights. In the second pass the classifier is retrained, using feature weights inverse to the feature importance, derived from the weights outputted in the previous pass. When tested on spam email detection under GWI/BWO evasion attacks, a MCS built using 10 LR classifiers, differing by the included features (each has 50% of the total feature set), is shown to be able to withstand the attack to a higher degree than standard LR, a single re-weighted LR or fscale-LR (a variant of a worst-case feature noise injection algorithm).

Game theory has been used in order to incorporate knowledge of attacker capability into learning algorithms. Broadly, the goal is to find an equilibrium representing the optimal strategy against the opponent (where no player benefits from deviating from the set of actions represented by the equilibrium). One such approach is to incorporate Stackleberg games: sequential games in which the follower (adversary) can observe and react to the leaders (learner) action. Brückner and Scheffer [2011] apply a Stackleberg prediction game (SPG) to the field of spam email detection. The Stackleberg equilibrium is found using an optimisation function. The approach when tested on four email datasets, is shown to outperform SVM, LR and Nash LR in accurately identifying spam emails. In a static prediction game, the learner and adversary both act simultaneously, without prior information on the opponents move. Brückner and Scheffer [2009] provide a method for finding the unique Nash equilibrium in a static game, focused on the email spam detection example. The equilibrium is found using a convex loss function. Brückner et al. [2012] revise [Brückner and Scheffer 2009] to repair the theorem dictating under which conditions an unique Nash equilibrium for a game exists, and develop Nash logistic regression (NLR) and Nash SVM (NSVM). When tested on email spam detection, the Nash LR and SVM variants outperform the non-Nash variants.

A different approach is to incorporate attack data into the training set to increase classifier robustness. Biggio et al. [2011] propose a system that addresses a lack of attack data for use in training by making use of generative classifier and incorporating a model of the expected attack into the training phase. The technique is evaluated in two settings. First, it is evaluated in the biometric identity verification scenario, where the LLR fusion rule is extended with the model. The extended LLR is shown to be slightly less effective when compared to standard LLR when not under attack, but achieve significantly better performance when under attack. The technique is also tested in the spam filtering case using a modified Naive Bayes text classifier (as used in SpamBayes) against the GWI/BWO attack. The modified Naive Bayes is shown to outperform the standard Naive Bayes. In both cases the defence reduces the effect of the attack rather than provide complete protection.

Machine unlearning [Cao and Yang 2015] is a reactive approach that enables the defender to remove compromised (poisoned) data from a training set without having to retrain the classifier. This approach is effective, as long as the defender knows what to unlearn. Therefore the defender needs some form of attack detection to know what to unlearn.

Another common approach is to take an existing machine learning algorithm, and modify it to provide resilience to a particular adversary [Dalvi et al. 2004], for example through retraining frequently. This approach is limited by the fact that it is an arms race between the defenders keeping the system up-to-date and attackers producing new attack variants.

Noise robustness techniques in ML have been around for a number of years. In the case of clustering, these techniques are generally focused on the handling of outliers or Gaussian noise [Chintalapudi and Kam 1998; Yang and Wu 2004; Li et al. 2007; Böhm et al. 2006], whereas in the case of classification the goal is to handle label noise [Biggio et al. 2011; Bhattacharyya 2004; Denchev et al. 2012]. However, these techniques are focused on "naturally occurring" noise rather than adversarial noise, and so the techniques are likely to be circumventable by the well-motivated attacker.

There are of course limitations to the mentioned defences. In almost all cases, the defences do not provide immunity to the discussed attacks. While total immunity is impossible without already knowing which attack samples are which, the level of protection is often not enough. The usual result is that the effect of the attack is reduced, but the attacks can still be successfully performed, just with more effort on behalf of the attacker.

In almost all cases, the defences are evaluated against a single attack with one or two limited datasets. Spam email detection is the motivating example in the majority of the defence papers mentioned above, with good word insertion/bad word obfuscation attacks being common. The emails are usually converted into a bag-of-words feature vector made up of binary values. This evaluation methodology leaves two points to consider. What is the effectiveness of the defences against further attacks? And how well will the defences work in situations with far more complex, non-binary features such as in the case of malware C&C detection? There is a lack of evaluation in other security applications, such as C&C detection, where ML use is common. Part of this issue is down to the authors of detection systems themselves, as where they may evaluate with a few different ML algorithms, the set of algorithms will consist of non-secure variants. While spam detection and PDF malware detection (one of the other common use cases in the defence papers) both share similarities to some C&C detection systems, such as those performing analysis on packet payloads, others, such as those focused on complex features such as packet timings and fields and malware behaviour represent far more complex problems.

Some of the defences, namely those based on game theory, rely on the attacker performing as expected by the attacker model. However, in reality the attacker may not behave as expected, for example using the feedback of the learning algorithm to formulate future attacks [Biggio et al. 2014a]. Probabilistic models such as those described in [Biggio et al. 2011] rely on an accurate prediction of the behaviour of the attacker. However, as in any security problem, the attacker may orchestrate an attack that has not been considered.

7.6. Open challenges

We combine the above into two key open challenges:

— The clean slate design of machine learning techniques that incorporate considerations to an adversary are required, as opposed to adding layers on top of existing non-adversarial ML techniques.
— Although in this survey we have focused on the detection step, a well-resourced attacker can also evade the measurement step when current sampling techniques are used [Gardiner and Nagaraja 2014]. There is an need for the development of evasion-resilient sampling techniques, for example techniques for tuning measurement to respond to C&C detection to maximise the effectiveness of detection techniques.

8. CONCLUSION

Both academia and industry have been fighting malware C&C communication channels for close to a decade now. From time to time, experts have proclaimed that the problem has been solved, only to find their confidence has been misplaced due to subsequent attacks. With the wide deployment and support for signature-based techniques, first applied to intrusion detection, research has been focussing on applying machine learning techniques to C&C detection. Much of the detection effort has focussed on traffic behaviour which is not intrinsic to the functioning of the botnet. Consequently, C&C designers will find it trivial to bypass protections based on this assumption. We first observed attackers moving to decentralised architectures for the benefits of scalability and resilience. Now the attackers are adopting traffic analysis resistance techniques from anonymous communications literature.

A second challenge is the scalable collection of traffic traces. A number of the detection systems discussed in this survey require full, rather than samples, data. With increasing traffic rates it will soon become hard to store all traffic thus forcing defenders to rely on estimation via sampling techniques. For certain data types, such as DNS logs, storing all of the data is easily achievable, whereas storing packet payloads is not scalable. Sampling in itself provides a possible method of evasion for the attacker — if the attacker can evade sampling, which may be a simpler task, they evade detection no matter which ML algorithm is in use.

In the light of these challenges, the problem of characterising C&C traffic behaviour from sampled traffic, requires a shift of perspective. Researchers need to take a step back to focus on the big picture. First, the challenges of building secure measurement techniques have not received the necessary attention in the security community – the 'needle-in-the-haystack' problem is challenging and some approaches have been outlined from sampling theory but these do not work in an adversarial setting. Apart from sampling techniques, the measurement architecture has to be open and extensible, allowing network wide co-ordination to focus measurement resources on attack traffic rather than trying to work out broad trends as has historically been done.

Next, there has been extensive research into the application of machine-learning techniques whilst making some critical assumptions: (a) Existing techniques are too specialised, so it's hard to make a good case for deployment of these. Instead, we need a *flexible extensible framework* where an ensemble of detection algorithms can be activated as needed instead of operating many detection systems in parallel; (b) Most existing techniques use static datasets whereas detection techniques must operate on streaming data where fresh updates arrive every few minutes. Efficient techniques to update previous results with the new data are an important consideration for performance and hence deployment. (c) The use of robust machine-learning techniques which can withstand variance and high-dimensionality is very important. While the research direction of applying learning theory is promising, existing solutions are still impractical. Our analysis and summarisation of current techniques show that performance, and especially evasion resilience, are the main barriers to wide adoption. We stress the significance of these properties in the real world to the security research community.

REFERENCES

Manos Antonakakis, Roberto Perdisci, David Dagon, Wenke Lee, and Nick Feamster. 2010. Building a Dynamic Reputation System for DNS. In *Proc. of the USENIX Security Symposium*.

Manos Antonakakis, Roberto Perdisci, Wenke Lee, Nikolaos Vasiloglou, and David Dagon. 2011. Detecting Malware Domains at the Upper DNS Hierarchy. In *Proc. of the USENIX Security Symposium*.

Manos Antonakakis, Roberto Perdisci, Yacin Nadji, Nikolaos Vasiloglou, Saeed Abu-Nimeh, Wenke Lee, and David Dagon. 2012. From Throw-Away Traffic to Bots: Detecting the Rise of DGA-Based Malware. In *Proc. of the USENIX Security Symposium*.

Marco Barreno, Peter L. Bartlett, Fuching Jack Chi, Anthony D. Joseph, Blaine Nelson, Benjamin I.P. Rubinstein, Udam Saini, and J. D. Tygar. 2008. Open Problems in the Security of Learning. (2008).

Marco Barreno, Blaine Nelson, Anthony D. Joseph, and J. D. Tygar. 2010. The Security of Machine Learning. *Mach. Learn.* (2010).

Marco Barreno, Blaine Nelson, Russell Sears, Anthony D. Joseph, and J. D. Tygar. 2006. Can Machine Learning Be Secure? (2006).

C. Bhattacharyya. 2004. Robust classification of noisy data using second order cone programming approach. In *Intelligent Sensing and Information Processing, 2004. Proc. of International Conference on*.

Battista Biggio, SamuelRota Bul, Ignazio Pillai, Michele Mura, EyasuZemene Mequanint, Marcello Pelillo, and Fabio Roli. 2014. Poisoning Complete-Linkage Hierarchical Clustering. In *Structural, Syntactic, and Statistical Pattern Recognition*. Springer Berlin Heidelberg.

Battista Biggio, Igino Corona, Zhi-Min He, Patrick P. K. Chan, Giorgio Giacinto, Daniel S. Yeung, and Fabio Roli. 2015. *Multiple Classifier Systems: 12th International Workshop, MCS 2015*. Chapter One-and-a-Half-Class Multiple Classifier Systems for Secure Learning Against Evasion Attacks at Test Time.

Battista Biggio, Igino Corona, Davide Maiorca, Blaine Nelson, Nedim rndi, Pavel Laskov, Giorgio Giacinto, and Fabio Roli. 2013. Evasion Attacks against Machine Learning at Test Time. In *Machine Learning and Knowledge Discovery in Databases*.

Battista Biggio, Giorgio Fumera, and Fabio Roli. 2010. Multiple classifier systems for robust classifier design in adversarial environments. *International Journal of Machine Learning and Cybernetics* (2010).

B. Biggio, G. Fumera, and F. Roli. 2011. Design of robust classifiers for adversarial environments. In *Systems, Man, and Cybernetics (SMC), 2011 IEEE International Conference on*.

Battista Biggio, Giorgio Fumera, and Fabio Roli. 2014a. Pattern recognition systems under attack: design issues and research challenges. *International Journal of Pattern Recognition and Artificial Intelligence* (2014).

B. Biggio, G. Fumera, and F. Roli. 2014b. Security Evaluation of Pattern Classifiers under Attack. *IEEE Transactions on Knowledge and Data Engineering* (2014).

Battista Biggio and Pavel Laskov. 2012. Poisoning attacks against Support Vector Machines. In *In International Conference on Machine Learning (ICML)*.

Battista Biggio, Blaine Nelson, and Pavel Laskov. 2011. Support Vector Machines Under Adversarial Label Noise. (2011).

Battista Biggio, Ignazio Pillai, Samuel Rota Bulò, Davide Ariu, Marcello Pelillo, and Fabio Roli. 2013. Is Data Clustering in Adversarial Settings Secure? (2013).

Battista Biggio, Konrad Rieck, Davide Ariu, Christian Wressnegger, Igino Corona, Giorgio Giacinto, and Fabio Roli. 2014. Poisoning Behavioral Malware Clustering. (2014).

Leyla Bilge, Davide Balzarotti, William Robertson, Engin Kirda, and Christopher Kruegel. 2012. Disclosure: Detecting Botnet Command and Control Servers through Large-Scale NetFlow Analysis. In *Proc. of the Annual Computer Security Applications Conference (ACSAC)*.

Leyla Bilge, Engin Kirda, Christopher Kruegel, and Marco Balduzzi. 2011. EXPOSURE: Finding Malicious Domains Using Passive DNS Analysis. In *Proc. of the Symposium on Network and Distributed System Security (NDSS)*.

Christian Böhm, Christos Faloutsos, Jia-Yu Pan, and Claudia Plant. 2006. Robust Information-theoretic Clustering. (2006).

Michael Brückner, Christian Kanzow, and Tobias Scheffer. 2012. Static Prediction Games for Adversarial Learning Problems. *J. Mach. Learn. Res.* (2012).

Michael Brückner and Tobias Scheffer. 2009. Nash Equilibria of Static Prediction Games. In *Advances in Neural Information Processing Systems 22*.

Michael Brückner and Tobias Scheffer. 2011. Stackelberg Games for Adversarial Prediction Problems. (2011).

Yinzhi Cao and Junfeng Yang. 2015. Towards Making Systems Forget with Machine Unlearning. (2015).

K.K. Chintalapudi and Moshe Kam. 1998. A noise-resistant fuzzy c means algorithm for clustering. In *Fuzzy Systems Proc., 1998. IEEE World Congress on Computational Intelligence., The 1998 IEEE International Conference on*.

Simon P. Chung and Aloysius K. Mok. 2006. Allergy Attack Against Automatic Signature Generation. In *Recent Advances in Intrusion Detection*.

Simon P. Chung and Aloysius K. Mok. 2007. Advanced Allergy Attacks: Does a Corpus Really Help. (2007).

Cisco Systems Inc. 2016. Cisco IOS Netflow. (2016). http://www.cisco.com/web/go/netflow.

M. Patrick Collins and Michael K. Reiter. 2007. Hit-List Worm Detection and Bot Identification in Large Networks Using Protocol Graphs. (2007).

Igino Corona, Giorgio Giacinto, and Fabio Roli. 2013. Adversarial attacks against intrusion detection systems: Taxonomy, solutions and open issues. *Information Sciences* (2013).

Chuck Cranor, Theodore Johnson, Oliver Spataschek, and Vladislav Shkapenyuk. 2003. Gigascope: a stream database for network applications. (2003).

Nilesh Dalvi, Pedro Domingos, Mausam, Sumit Sanghai, and Deepak Verma. 2004. Adversarial Classification. (2004).

Vasil Denchev, Nan Ding, Hartmut Neven, and S.v.n. Vishwanathan. 2012. Robust Classification with Adiabatic Quantum Optimization. (2012).

Manul Egele, Theodoor Scholte, Engin Kirda, and Christopher Kruegel. 2012. A Survey on Automated Dynamic Malware Analysis Techniques and Tools. *Comput. Surveys* 44, 2 (2012).

P. Ferrie. 2007. *Attacks on More Virtual Machine Emulators*. Technical Report. Symantec.

Prahlad Fogla and Wenke Lee. 2006. Evading Network Anomaly Detection Systems: Formal Reasoning and Practical Techniques. (2006).

Prahlad Fogla, Monirul Sharif, Roberto Perdisci, Oleg Kolesnikov, and Wenke Lee. 2006. Polymorphic Blending Attacks. , Article 17 (2006). http://dl.acm.org/citation.cfm?id=1267336.1267353

Jason Franklin, Vern Paxson, Adrian Perrig, and Stefan Savage. 2007. An Inquiry into the Nature and Causes of the Wealth of Internet Miscreants. In *Proc. of the ACM Conference on Computer and Communications Security (CCS)*.

Joseph Gardiner, Marco Cova, and Shishir Nagaraja. 2014. Command and Control: Understanding, Denying and Detecting. (2014). work commisioned by CPNI, available at http://c2report.org.

Joseph Gardiner and Shishir Nagaraja. 2014. On the Reliability of Network Measurement Techniques Used for Malware Traffic Analysis. In *Security Protocols XXII*.

Chris Grier, Lucas Ballard, Juan Caballero, Neha Chachra, Christian J. Dietrich, Kirill Levchenko, Panayiotis Mavrommatis, Damon McCoy, Antonio Nappa, Andreas Pitsillidis, Niels Provos, M. Zubair Raque, Moheeb Abu Rajab, Christian Rossow, Kurt Thomas, Vern Paxson, Stefan Savage, and Geoffrey M. Voelker. 2012. Manufacturing Compromise: The Emergence of Exploit-as-a-Service . In *Proc. of the ACM Conference on Computer and Communications Security (CCS)*.

Guofei Gu, Roberto Perdisci, Junjie Zhang, and Wenke Lee. 2008. BotMiner: Clustering Analysis of Network Traffic for Protocol- and Structure-Independent Botnet Detection. In *Proc. of the USENIX Security Symposium*.

G. Gu, P. Porras, V. Yegneswaran, M. Fong, and W. Lee. 2007. BotHunter: Detecting Malware Infection Through IDS-Driven Dialog Correlation. In *Proc. of the USENIX Security Symposium*.

T. Holz, C. Gorecki, K. Rieck, and F. Freiling. 2008. Measuring and Detecting Fast-Flux Service Networks. In *Proc. of the Symposium on Network and Distributed System Security (NDSS)*.

Chet Hosmer. 2008. Polymorphic & Metamorphic Malware. In *Proceedings of the BlackHat Conference*.

Ling Huang, Anthony D. Joseph, Blaine Nelson, Benjamin I.P. Rubinstein, and J. D. Tygar. 2011. Adversarial Machine Learning. (2011).

Eric M. Hutchins, Michael J. Clopperty, and Rohan M. Amin. 2010. *Intelligence-Driven Computer Network Defense Informed by Analysis of Adversary Campaigns and Intrusion Kill Chains*. Technical Report. Lockheed Martin Corporation.

M. Iliofotou, M. Faloutsos, and M. Mitzenmacher. 2009. Exploiting Dynamicity in Graph-based Traffic Analysis: Techniques and Applications. (2009).

M. Iliofotou, P. Pappu, M. Faloutsos, M. Mitzenmacher, G. Varghese, and H. Kim. 2008. Graption: Automated Detection of P2P Applications using Traffic Dispersion Graphs (TDGs). (2008).

Luca Invernizzi, Stanislav Miskovic, Ruben Torres, Sabyaschi Saha, Sung-Ju Lee, Christopher Kruegel, and Giovanni Vigna. 2014. Nazca: Detecting Malware Distribution in Large-Scale Networks. (Feb 2014). http://seclab.cs.ucsb.edu/media/uploads/papers/invernizzi_nazca_ndss14.pdf

Gregoire Jacob, Ralf Hund, Christopher Kruegel, and Thorsten Holz. 2011. JACKSTRAWS: Picking Command and Control Connections from Bot Traffic. In *Proceedings of the 20th USENIX Conference on Security*.

M. Jelasity and V. Bilicki. 2009. Towards Automated Detection of Peer-to-Peer Botnets: On the Limits of Local Approaches. (2009).

Kaspersky. 2013. Ask An Expert: The Brainstorming. http://blog.kaspersky.com/ask-an-expert-the-brainstorming/. (2013).

Hyang-Ah Kim and Brad Karp. 2004. Autograph: Toward Automated, Distributed Worm Signature Detection. (2004).

Clemens Kolbitsch, Engin Kirda, and Christopher Kruegel. 2011. The Power of Procrastination: Detection and Mitigation of Execution-Stalling Malicious Code. In *Proc. of the ACM Conference on Computer and Communications Security (CCS)*.

Alexsander Kolcz and Choon Hui Teo. 2009. Feature Weighting for Improved Classifier Robustness. (2009).

Zhenguo Li, Jianzhuang Liu, Shifeng Chen, and Xiaoou Tang. 2007. Noise Robust Spectral Clustering. In *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on*.

Justin Ma, Lawrence K. Saul, Stefan Savage, and Geoffrey M. Voelker. 2009. Beyond Blacklists: Learning to Detect Malicious Web Sites from Suspicious URLs. (2009).

Davide Maiorca, Giorgio Giacinto, and Igino Corona. 2012. A Pattern Recognition System for Malicious PDF Files Detection. (2012).

PratyusaK. Manadhata, Sandeep Yadav, Prasad Rao, and William Horne. 2014. Detecting Malicious Domains via Graph Inference. In *Computer Security - ESORICS 2014*, Mirosaw Kutyowski and Jaideep Vaidya (Eds.). Lecture Notes in Computer Science, Vol. 8712. Springer International Publishing, 1–18. DOI:http://dx.doi.org/10.1007/978-3-319-11203-9_1

Konstantinos Mersinas, Bjoern Hartig, Keith Martin, and Andrew Seltzer. 2015. Experimental Elicitation of Risk Behaviour amongst Information Security Professionals. (2015).

Andreas Moser, Christopher Kruegel, and Engin Kirda. 2007. Exploring Multiple Execution Paths for Malware Analysis. In *Proc. of the IEEE Symposium on Security and Privacy*.

Shishir Nagaraja. 2014. Botyacc: Unified P2P Botnet Detection Using Behavioural Analysis and Graph Analysis. In *Computer Security - ESORICS 2014*, Mirosaw Kutyowski and Jaideep Vaidya (Eds.). Lecture Notes in Computer Science, Vol. 8713. Springer International Publishing, 439–456. DOI:http://dx.doi.org/10.1007/978-3-319-11212-1_25

Shishir Nagaraja, Prateek Mittal, Chi-Yao Hong, Matthew Caesar, and Nikita Borisov. 2010. BotGrep: Finding P2P Bots with Structured Graph Analysis. (2010).

Antonio Nappa, Zhaoyan Xu, Juan Caballero, and Guofei Gu. 2014. CyberProbe: Towards Internet-Scale Active Detection of Malicious Servers. (February 2014).

Terry Nelms, Roberto Perdisci, and Mustaque Ahamad. 2013. ExecScent: Mining for New C&C Domains in Live Networks with Adaptive Control Protocol Templates. In *Proc. of the USENIX Security Symposium*.

Blaine Nelson, Marco Barreno, Fuching Jack Chi, Anthony D. Joseph, Benjamin I. P. Rubinstein, Udam Saini, Charles Sutton, J. D. Tygar, and Kai Xia. 2008. Exploiting Machine Learning to Subvert Your Spam Filter. (2008).

J. Newsome, B. Karp, and D. Song. 2005. Polygraph: automatically generating signatures for polymorphic worms. (May 2005). DOI:http://dx.doi.org/10.1109/SP.2005.15

James Newsome, Brad Karp, and Dawn Song. 2006. Paragraph: Thwarting Signature Learning by Training Maliciously. (2006).

R. Paleari, L. Martignoni, G. Fresi Roglia, and D. Bruschi. 2009. A Fistful of Red-Pills: How to Automatically Generate Procedures to Detect CPU Emulators. In *Proc. of the USENIX Workshop on Offensive Technologies (WOOT)*.

Roberto Perdisci, Igino Corona, and Giorgio Giacinto. 2012. Early Detection of Malicious Flux Networks via Large-Scale Passive DNS Traffic Analysis. *IEEE Transactions on Dependable and Secure Computing* 9, 5 (2012).

Roberto Perdisci, David Dagon, Wenke Lee, Prahlad Fogla, and Monirul Sharif. 2006a. Misleading Worm Signature Generators Using Deliberate Noise Injection. (2006).

Roberto Perdisci, Guofei Gu, and Wenke Lee. 2006b. Using an Ensemble of One-Class SVM Classifiers to Harden Payload-based Anomaly Detection Systems. (2006).

M. Zubair Rafique and Juan Caballero. 2013. FIRMA: Malware Clustering and Network Signature Generation with Mixed Network Behaviors. In *Proc. of the Symposium on Recent Advances in Intrusion Detection (RAID)*.

Babak Rahbarinia, Roberto Perdisci, Andrea Lanzi, and Kang Li. 2013. PeerRush: Mining for Unwanted P2P Traffic. In *Detection of Intrusions and Malware, and Vulnerability Assessment*.

Konrad Rieck, Guido Schwenk, Tobias Limmer, Thorsten Holz, and Pavel Laskov. 2010. Botzilla: Detecting the "Phoning Home" of Malicious Software. In *Proc. of the ACM Symposium on Applied Computing (SAC)*.

Konrad Rieck, Philipp Trinius, Carsten Willems, and Thorsten Holz. 2011. Automatic Analysis of Malware Behavior Using Machine Learning. *J. Comput. Secur.* (2011).

Christian Rossow and Christian J. Dietrich. 2013. ProVex: Detecting Botnets with Encrypted Command and Control Channels. In *Proc. of the Conference on Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA)*.

Stefano Schiavoni, Federico Maggi, Lorenzo Cavallaro, and Stefano Zanero. 2014. Phoenix: DGA-Based Botnet Tracking and Intelligence. In *Detection of Intrusions and Malware, and Vulnerability Assessment*.

Charles Smutz and Angelos Stavrou. 2012. Malicious PDF Detection Using Metadata and Structural Features. (2012).

Anil Somayaji and Stephanie Forrest. 2000. Automated Response Using System-call Delays. (2000).

Verizon RISK Team. 2013. *2013 Data Breach Investigations Report*. Technical Report. Verizon.

Nedim Šrndic and Pavel Laskov. 2013. Detection of Malicious PDF Files Based on Hierarchical Document Structure. (2013).

Nedim Šrndic and Pavel Laskov. 2014. Practical Evasion of a Learning-Based Classifier: A Case Study. (2014).

David Wagner and Paolo Soto. 2002. Mimicry Attacks on Host-based Intrusion Detection Systems. (2002).

Ke Wang, Gabriela Cretu, and Salvatore J. Stolfo. 2006. Anomalous Payload-based Worm Detection and Signature Generation. (2006). http://dx.doi.org/10.1007/11663812_12

Ke Wang and Salvatore J. Stolfo. 2004. Anomalous Payload-Based Network Intrusion Detection. In *Recent Advances in Intrusion Detection*.

Charles V. Wright, Scott E. Coull, and Fabian Monrose. 2009. Traffic Morphing: An Efficient Defense Against Statistical Traffic Analysis. In *In Proc. of the 16th Network and Distributed Security Symposium*. IEEE, 237–250.

Han Xiao, Huang Xiao, and Claudia Eckert. 2012. Adversarial Label Flips Attack on Support Vector Machines. (aug 2012).

Weilin Xu, Yanjun Qi, and David Evans. 2016. Automatically Evading Classifiers - A Case Study on PDF Malware Classifiers. (2016).

Zhaoyan Xu, Antonio Nappa, Robert Baykov, Guangliang Yang, Juan Caballero, and Guofei Gu. 2014. AutoProbe: Towards Automatic Active Malicious Server Probing Using Dynamic Binary Analysis. (2014).

Moosa Yahyazadeh and Mahdi Abadi. 2014. BotGrab: A negative reputation system for botnet detection. *Computers and Electrical Engineering* 0 (2014).

Miin-Shen Yang and Kuo-Lung Wu. 2004. A Similarity-Based Robust Clustering Method. *IEEE Trans. Pattern Anal. Mach. Intell.* (2004).

Ting-Fang Yen, Alina Oprea, Kaan Onarlioglu, Todd Leetham, William Robertson, Ari Juels, and Engin Kirda. 2013. Beehive: Large-scale Log Analysis for Detecting Suspicious Activity in Enterprise Networks. (2013).

Ting-Fang Yen and Michael K. Reiter. 2008. Traffic Aggregation for Malware Detection. (2008).

Ali Zand, Giovanni Vigna, Xifeng Yan, and Christopher Kruegel. 2014. Extracting Probable Command and Control Signatures for Detecting Botnets. (2014).

F. Zhang, P.P.K. Chan, B. Biggio, D.S. Yeung, and F. Roli. 2015. Adversarial Feature Selection Against Evasion Attacks. *Cybernetics, IEEE Transactions on* (2015).

Junjie Zhang, R. Perdisci, Wenke Lee, Xiapu Luo, and U. Sarfraz. 2014. Building a Scalable System for Stealthy P2P-Botnet Detection. *Information Forensics and Security, IEEE Transactions on* (2014).

Junjie Zhang, Roberto Perdisci, Wenke Lee, Unum Sarfraz, and Xiapu Luo. 2011. Detecting Stealthy P2P Botnets Using Statistical Traffic Fingerprints. In *Proceedings of the IEEE/IFIP Conference on Dependable Systems and Networks (DSN)*.

Y. Zhao, Y. Xie, F. Yu, Q. Ke, Y. Yu, Y. Chen, and E. Gillum. 2009. BotGraph: Large Scale Spamming Botnet Detection. (2009).