Supervised Detection of Infected Machines Using Anti-virus Induced Labels
(Extended Abstract)

Tomer Cohen[1], Danny Hendler[1(✉)], and Dennis Potashnik[2]

[1] Department of Computer Science, Ben-Gurion University of the Negev, Beer Sheva, Israel
hendlerd@cs.bgu.ac.il
[2] IBM Cyber Center of Excellence, Beer Sheva, Israel

Abstract. Traditional antivirus software relies on signatures to uniquely identify malicious files. Malware writers, on the other hand, have responded by developing obfuscation techniques with the goal of evading content-based detection. A consequence of this arms race is that numerous new malware instances are generated every day, thus limiting the effectiveness of static detection approaches. For effective and timely malware detection, signature-based mechanisms must be augmented with detection approaches that are harder to evade.

We introduce a novel detector that uses the information gathered by IBM's QRadar SIEM (Security Information and Event Management) system and leverages anti-virus reports for automatically generating a labelled training set for identifying malware. Using this training set, our detector is able to automatically detect complex and dynamic patterns of suspicious machine behavior and issue high-quality security alerts. We believe that our approach can be used for providing a detection scheme that complements signature-based detection and is harder to circumvent.

1 Introduction

With tutorials for writing sophisticated malicious code, as well as malicious source code and tools for malware generation freely available on the Internet in general and the dark web in particular [10,18], developing new malware is becoming easier and sharing malicious code and exploits between different cyber-criminal projects becomes common.

Moreover, polymorphic and metamorphic malware that utilize dead-code insertion, subroutine reordering, encryption, and additional obfuscation techniques, are able to automatically alter a file's content while retaining its functionality [15,23]. As a consequence, the number of new malicious files is growing quickly. Indeed, the number of new malware released to the wild on January 2017 alone is estimated as approx. 13 million and the number of *known* malware is estimated as approx. 600 million [21]!

S. Dolev and S. Lodha (Eds.): CSCML 2017, LNCS 10332, pp. 34–49, 2017.
DOI: 10.1007/978-3-319-60080-2_3

A direct implication of this high rate of new malware is that anti-virus products, which rely heavily on signatures based on file-contents for identifying malware, must be complemented by detection approaches that are harder to evade. One such approach is to attempt to identify the behavioral patterns exhibited by compromised machines instead of relying only on the signatures of the files they download. This approach is more robust to most malware obfuscation techniques, since new malware variants typically have new signatures but exhibit the same behavioral patterns.

Many organizations employ SIEM (Security Information and Event Management) systems, which are software products and services that consolidate information streams generated by multiple hardware and software sources within the organization. SIEM systems facilitate the real-time analysis of gathered information in terms of its security implications. They are able to enforce enterprise security policies and to generate events and alarms when (customizable) static rules are triggered. The rich data available to contemporary SIEM systems holds the potential of allowing to distinguish between the behavior of benign and compromised machines.

This is the approach we take in this work. Our study is based on data collected by IBM® Security QRadar® system [9]. QRadar collects, normalizes and stores the data, received from various networks and security devices, and also enriches it with additional analytics, such as new events generated by its Custom Rule Engine (CRE). We have developed a detector for compromised machines that automatically mines these log files in a big-data environment, in order to extract and compute numerous features for identifying malicious machine behavior. A key challenge is that of generating a *labelled* set of training examples, each representing the activity of a specific machine during a specific time interval, on which supervised machine learning algorithms can be invoked. Analyzing numerous activity logs manually in order to determine whether or not they are benign is expensive and time-consuming. Moreover, since the behavior of infected machines may vary over time, the process of manual labeling should be repeated sufficiently often, which is unrealistic.

Instead, our detector leverages the alerts issued by the anti-virus running on the client machine in order to automatically identify periods of time in which the machine is likely infected and label them as "black" instances. "White" training instances are generated based on the activity of machines for which no AV alerts were issued for an extended period of time. This allows our detector to automatically and periodically generate labelled training sets. Our work thus makes the following contributions:

- Our experiments prove that malicious behavior of infected machines can be accurately detected based on their externally observed behavior.
- We devise a novel detector for infected machines in big data SIEM environments, that uses anti-virus induced labels for supervised classification.
- We present the results of extensive evaluation of our detector, conducted based on more than 6 terabytes of QRadar logs collected in a real production environment of a large enterprise, over the period of 3.5 months between

1.12.15–16.3.2016. Our evaluation shows that our detector identifies security incidents that trigger AV alerts with high accuracy and indicates that it is also able to alert on suspicious behavior that is unobserved by the AV.

2 Related Work

Some previous work used anti-virus (AV) labels for categorizing malicious executables into malware categories such as bots, Trojan horses, viruses, worms, etc. Perdisci et al. [20] proposed an unsupervised system to classify malware based on its network activities. Nari and Ghorbani [17] proposed a scheme of building network activity graphs. Graph nodes represent communication flows generated by the malware and are marked by the type of the flow (DNS, HTTP, SSL, etc.) and edges represent the dependencies between the flows. Bekerman et al. propose a machine-learning based system for detecting malicious executables. The data set consists of malware network traces, that were tagged based on the detections of an IDS system, and traces of benign flows. Whereas the goal of the aforementioned studies is to detect malicious executables or to categorize them to malware families, the goal of our system is to detect infected machines based on their network behavior.

We now describe several previous works that proposed systems for identifying infected machines. Narang et al. [16] propose a method for detecting a P2P botnet in its dormant phase (in standby for receiving a command from the $C\&C$) by collecting network traces from benign P2P application and P2P botnets and extracting features from the traces. Several studies [3, 11] attempt to detect the life-cycle phase when a newly infected bot searches for its $C\&C$, by leveraging the fact that this search often generates a large number of DNS query failures. Another approach [5, 7] is to leverage the fact that different bots send similar messages to the $C\&C$. Unlike these works, our system does not look for a specific phase in the malware's life-cycle, nor does it limit itself to the detection of specific malware types.

Some previous work examines the behavior of hosts in a time window, extracting features based on the events associated with the host in the time windows and training a machine learning classifier based on these features. Bocchi et al. [4] use tagged data set based on a single day of network traces, captured by a commercial ISP. They use time windows of 30 min when a host is inspected and an instance is generated based on all the events that occurred during the time window. Unlike their detector which operates on offline data, our system is integrated within the QRadar SIEM system. It can therefore use a multitude of *log sources* (described in Sect. 4), reporting about different aspects of the computer's network's behavior. Moreover, out study was conducted based on data collected during a significantly longer period of 3.5 months.

Gu et al. [6] present a system called Botminer, that uses clustering algorithms for detecting infected hosts. A key difference between our system and theirs is that, whereas our detector is designed to detect any type of malware, Botminer is limited to bot detection. Yen et al. [22] propose an unsupervised machine-learning based system to detect anomalous behavior of machines in an enterprise environment. The data is collected using a SIEM system used by EMC.

The authors define a time window of a single day to inspect a host and create an instance based on all the events relating to this host that occurred during this time window.

3 The QRadar Environment

Our detector is based on data collected by an IBM® Security QRadar® SIEM (Security Information and Event Management) system [9], which we shortly describe in this section. SIEM systems are organization-level software products and services that consolidate information streams generated by multiple hardware and software sources and facilitate the real-time analysis of this data in terms of its security implications. The devices that send their reports to the instance of QRadar which we used in this study are the following: Symantec endpoint protection solution, network firewalls, personal firewall, IDS, routers, DNS servers, DHCP servers and authentication servers.

Devices that send streams of information to QRadar are called *log sources*. Some of these devices (e.g. firewalls and AVs) send streams of events. Network devices (e.g. routers and switches), on the other hand, generate streams of *flows* and are therefore called *flow sources*. Events include security-related data, such as firewall denials, ssh unexpected messages, teardown UDP connection, teardown TCP connection, etc. QRadar also enriches event streams by adding new events, generated by its CRE, such as port scanning, excessive firewall denials across multiple internal hosts from a remote host, local windows scanner detected, etc. Flows, on the other hand, report on network traffic. Table 1 lists key flow fields.

Table 1. Flow fields

Field name	Description
Source IP	The IP address of the machine that initiated the flow
Destination IP	The IP address of the destination machine
Source port	The port used by the machine that initiated the flow
Destination port	The port used by the destination machine
First packet time	The time of the first packet
Incoming packets	The number of packets sent by the source
Outgoing packets	The number of packets sent by the destination
Incoming bytes	The total number of bytes sent by the source
Outgoing bytes	The total number of bytes sent by the destination
Direction	Indicates who initiated the flow, a machine that belongs to the enterprise or a machine outside the enterprise
Source IP location	The geographical location of the source IP
Destination IP location	The geographical location of the destination IP
TCP flags	The TCP flags used in the flow session

QRadar's event collector collects the logs from the devices, parses, and normalizes them to unified QRadar events. The normalized stream of data is then passed to an event processor, which enriches the data and generates additional events, according to the custom rules defined in the system. Some events may designate a security threat and trigger an alert, called an *offense*, that is sent to a security analyst. For example, an AV report can trigger such an offense. Another example is a collection of firewall denials generated by a single remote host that may trigger an "excessive firewall denials across multiple hosts from a remote host" offense.

The QRadar instance we worked with can locally save up to one month of events and flows. In order to aggregate more data, normalized data is forwarded to a remote HDFS [1] cluster. The data gathered in the HDFS cluster sums up to more than 2 TB of data each month, from which approximately 1.6 TB are events and the rest are flows. On an average day, more than 2000 unique enterprise user IPs are seen and more than 32 M events and 6.5 M flows relating to these users are collected.

Gathered data is partitioned to hourly files: each hour is represented by two files – one aggregating all flows reported during that hour, and the other aggregating all events reported during that hour. Each event is comprised of fields such as event name, low and high level categories, source and destination IPs, event time, and event payload. The event name designates the event's type. Each event type (identified by the event's event name field) belongs to a single low-level category, which serves to group together several related event types. Similarly, low-level categories are grouped together to more generic high-level categories. Table 2 lists a few events, specifying for each its low- and high-level categories.

We implemented our detector on Spark [2]. Spark is an open source cluster computing framework that is able to interface with Hadoop Distributed File System (HDFS) [1]. Since our feature extraction is done using Spark, our detector is scalable and can cope with big data.

Table 2. QRadar sample events

Event name	Low-level category	High-level category
Virus Detected, Actual action: Left alone	Virus Detected	Malware
Virus Detected, Actual action: Detail pending	Virus Detected	Malware
Teardown UDP connection	Firewall session closed	Access
Firewall allow	Firewall permit	Access
Built TCP connection	Firewall session opened	Access

4 The Detector

Our detector leverages the alerts issued by the anti-virus running on client machines in order to automatically identify time windows in which the machine is likely infected and label them as "black" instances. "White" training instances are generated based on the activity of machines for which no AV alerts were issued for an extended period of time. Each such time window includes events and flows pertaining to the machine under consideration. Before providing a detailed description of our detector and the features it uses, we define more precisely what we mean by the terms flows, black instances, and white instances.

A *flow* is an aggregation of packets that share the same source address, source port, destination address, and destination port. The stopping condition for aggregation differs between udp and tcp. Aggregation of udp packets stops when no new packets arrive for a predefined time. Aggregation of tcp packets stops either when no new packets arrive for a predefined time or when a packet with a set FIN flag has been sent or received.

A *black instance* is a positive example that is provided to the ML algorithm used by our detector. It describes a period of time in which a host is assumed to be infected. The generation of a black instance is triggered upon the occurrence of certain types of anti-virus (AV) events on a host. The black instance encapsulates all the events/flows from/to the host before and after the AV event occurred on it. We have set the length of the time window to 7 h, 6 h before the AV event and 1 h after it.

Events of the following two types trigger the generation of a black instance: "Virus Detected, Actual action: Left alone" and "Virus Detected, Actual action: Details pending". The reason for basing the generation of black instances on these AV events is that their semantics guarantees that a malware is indeed active on the host when the event is generated for it. In both these event types, a malware was detected on the machine but was not deleted by the AV, hence it is likely that the malware was active on the host some time before the AV event and possibly remained active for at least some period of time after the AV event.

The AV's configuration in our system is such that it attempts to delete all detected malware. Nevertheless, in some circumstances, the malware cannot be deleted, either because it is the image of a running process or because the AV lacks permissions for doing so.[1]

A *white instance* is a negative example that is provided to our detector. It describes a period of time in which it is assumed that no malware operates within the host. It encapsulates the events and flows pertaining to a clean host within a time window of 7 h. By clean, we mean that no significant AV event for that machine was reported for 7 days before and after the white instance.

[1] Newer AV versions have the capability of stopping the process and then deleting the file.

Features. As mentioned previously, our data consists of network flows (provided in netflow form) and security-related events. We have defined more that 7000 features to be computed based on a time window of data. The features can be divided to the following groups.

1. Event name #: number of events with the same specific name (that is, events of the same specific type) in the time window.
2. Event name distribution: number of events with the same name divided by the number of all events in the time window.
2. Low level category #: number of events with the same low level category in the time window.
4. Low level category distribution: number of events with the same low level category divided by the number of all events in the time window.
5. High level category #: number of events with the same high level category in the time window.
6. High level category distribution: number of events with the same high level category divided by the number of all events in the time window.
7. Flows on port: the number of flows with the specific port number (split into outgoing ports and incoming ports).
8. Flow port distribution: the number of flows with the specific port number divided by the number of all flows in the time window.
9. Flow statistics: average, max, min, variance and deviation of the aggregation fields of the flows, for example: number of packets in/out, number of bytes in/out, etc.

In order to lower dimensionality, we used well-known feature selection algorithm. We applied 6 algorithms and selected the top 40 features output by each, thus reducing the number of features to 135. The algorithms we applied are: Information Gain [12], Information Gain Ratio [8], Chi Squared, Relief [13], Filter [24] and SVM-based feature selection. Each algorithm contributed approx. 20 unique features and the rest of the features were shared between one or more algorithms.

Constructing the Machine Learning Model. The training set is labelled based on AV reports. To construct black instances, we find all significant AV reports (see the definition of a black instance earlier in this section) available from the data collected by QRadar. Then, *raw black instances* are built around each such event (6 h before and 1 h after the event). Raw instances are the aggregation of all events and flows pertaining to the IP of the machine on which the AV event occurred and fall within the time window. Then, we compute the features, normalize the instance (we provide more details on normalization later) and label the instance as black.

Raw white instances are created based on sampled clean periods of machines. Such instances are normalized similarly and are labeled as white. Based on the set of these training instances, a machine learning model is built. The data set we worked with exhibits imbalance: there were far more white instances than black instances. In order to alleviate this problem, we employed under-sampling. After experimenting with a few under-sampling ratios, we have set the ratio between the black and the white instances in our training set to 1:10 (i.e., there are 10 times more white instances than black instances). This is consistent with the findings of Moskovitch et al. [14], who investigated how the ratio between black and white instances in the training set affects model accuracy in the malware detection domain.

The required number of white instances is obtained by random sampling of a large pool of white instances. This pool is created by generating, per every machine IP appearing in the data, 8 instances per day (centered around hours 24am, 3am, 6am, 9am, 12pm, 15pm, 18pm and 21pm) for each day in the training period.

We invoked the detector on a test set of the instances. The construction of the training set and the test set[2] is done on the HDFS cluster using map-reduce code. The computation of feature selection, model construction and classification are currently done on a stand-alone machine, since these ML algorithms were not available on our HDFS cluster.

The input to our detector are QRadar's events and flows. These are accumulated and aggregated within predetermined time windows, per every machine under consideration. Then, features are extracted and their values are normalized, resulting in instances to be classified. Next, an ML algorithm is invoked on these instances, using the most recent model available, resulting in classification of the instances, based on which detection warnings are issued when necessary.

Data Filtering. The QRadar data we received consists of events and flows for all the machines in the enterprise network. These include servers (e.g. DNS servers and mail servers), endpoint computers, and smartphones. The AV reports which we use for labelling, however, are only generated on endpoint computers. Moreover, the network behavior of servers and smartphones is substantially different from that of endpoint computers in terms of flow and event type distributions. In our evaluation we therefore focused on modelling and detecting infected endpoint computers (both desktops and laptops) and filtered out data from servers and smartphones.

Some of the training instances created by the algorithm had little or no activity, which is not helpful for differentiating between malicious and benign activity. Therefore, when constructing the training set, we filtered out all instances whose level of activity (in terms of numbers of flows and events) was below a certain threshold.

Some events always accompany significant AV event of the types "Virus Detected, Actual action: Left alone" or "Virus Detected, Actual action: Details pending". We had to eliminate features based on these events in order to avoid over-fitting that would detect AV events rather than suspicious network behavior.

[2] More details on the test set are provided in Sect. 5.

Normalization of Data Values. If the resolution of feature values is too fine, then over-fitting may result due to the fact that the model may classify two instances differently based on very small differences in field values. To mitigate this potential problem, we normalized feature values in the following manner.

– Features that count the number of events/flows from any specific type were normalized by applying the log function and then rounding the result to the closest integer value.
– Distribution-features (whose values represent fractions) were rounded up to the nearest second decimal place.

Setting Time Window Length. A key issue affecting detection accuracy is that of setting the length of the time windows that define training and detection instances. As described in Sect. 4, black instances are built around significant AV events indicating that a malware is running on the machine for which the event is issued. In general, however, we do not know for how long the malware is active before and after the AV event and the extent to which its activity is reflected in the network behavior of the infected host. The following questions have to be addressed in this regard.

1. How long should the time window extend before the AV event?
2. How long should the time window extend after the AV event?

Setting the length of the time window before and after the AV event involves striking a good balance in the following inherent tradeoff: an overly small window may imply losing significant information characterizing malicious activity, whereas an overly large window may imply incorporating data that is irrelevant. Let us call the period within the time window that precedes the AV event the *earlier period* and the period following the AV event the *later period*. We conducted several experiments in order to optimize these periods. Based on the results of these experiments, we chose to define time windows of 7 h–6 h before the AV event and 1 h after it.

5 Evaluation

In this section we report on the results of our evaluation. We created a training set based on QRadar data from 1.12.2015–29.2.2016. We use undersampling with a ratio of 1:10 between black and white instances to overcome the imbalance exhibited by our dataset. We then created a classification model based on this training set and evaluated its accuracy on the rest of the data, spanning the period 1.3.2016–16.3.2016 (henceforth referred to as the *evaluation period*). We conducted a few experiments, described in the following.

Instance-Based Experiment. We created a test set that consists of all the time windows built around significant AV events that were reported on the first half of March 2016 (there were 59 such instances) and all *white instances* during this period. White instances are created in a manner similar to that described in Sect. 4. More precisely, 8 instances are constructed per IP per day starting 24am, every 3 h. Those instances whose time interval does not contain any significant AV event are designated as white. The total number of machines that appear in the test set is 4987 and the total number of instances in the test set is 285,494.

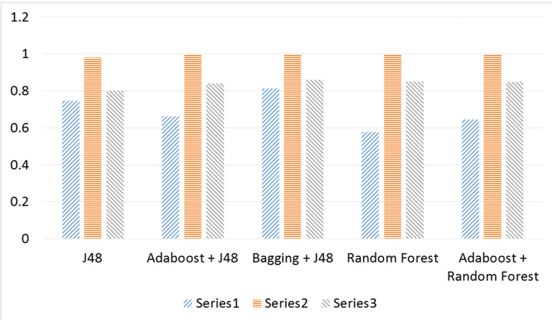

Fig. 1. Detection results for instance-based experiment.

We evaluated the following machine learning algorithm implemented in WEKA [19]: Random Forest, Decision tree (J48), Adaboost+J48, Bagging+J48, Random Forest, and Adaboost+Random Forest. The results are presented in Fig. 1. Best results were obtained by the Bagging+J48 algorithm, which yields a TPR of 0.81 on black instances and a TPR of 0.996 on white instances, with ROC area 0.86.

Compromised Machine Detection Experiment. As mentioned in Sect. 4, events and flows are bound to an endpoint IP. In the enterprise network, however, the IP of a machine changes on a regular basis. Consequently, we cannot track the activity of a machine over an extended period of time solely based on the IPs associated with events/flows.

In order to address this problem, our code constructs a mapping between IP/time pairs and machine MAC addresses. This mapping is constructed using DHCP events and personal firewall events reported to Q-Radar. Unfortunately, DHCP and firewall events were not available for all machines in the course of the first half of March, but we were able to construct the mapping for 10 machines on which a total of 32 black instances occurred (which we henceforth refer to as *compromised machines*) and for additional 2110 machines on which no such events occurred (which we henceforth refer to as *clean machines*).

In the compromised machine detection experiment, we evaluate the extent to which the classifier alerts on compromised machines during the

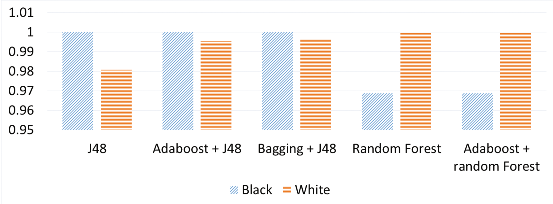

Fig. 2. Detection results for the compromised machine detection experiment.

evaluation period. In order to do so, we use a test set that contains all the instances (8 per day) constructed for all mapped machines. We evaluate classification results as follows.

- If the detector alerts (classifies as positive) during the first half of March on one or more instances of a compromised machine, all black instances that occurred on that machine during the evaluation period are considered true positive, otherwise they are all considered false negative.
- For all instances that are from clean machines, if the instance is alerted on it is considered false positive, otherwise it is considered true negative.

The results of this experiment are shown by Fig. 2. In terms of true positive rate, 3 algorithms (J48, Bagging+J48, Adaboost+J48) alert on all compromised machines (hence all 32 black instances are true positive), whereas the other two algorithms (random forest and Adaboost+random forest) alert on 9 out 10 compromised machines and only miss a single machine, on which a single black instance occurred during the evaluation period (hence 31 black instances are true positives).

All algorithms exhibit excellent detection rate on white instances but the random forest algorithms (with and without Adaboost) are the clear winners, both exceeding true positive rate of 0.9997 (!). In absolute numbers, Adaboost+random forest has only 29 false positives and random forest only has 31 (out of a total of approx. 135,000 white instances), all of which occurred on a set of 14 clean machines.

5.1 Early Detection Experiment

In this section, we evaluate the extent to which our detector is able to alert on suspicious activity before the AV does. In order to be able to track machine activity across day boundaries, we conduct also this evaluation for the set of machines for which a mapping exists throughout the evaluation period. In this experiment, we consider a black instance as a true positive, if our detector alerted on the machine on which it occurred within ± 3 days of the event.

Similarly to the compromised machine detection experiment, we use a test set that contains all the instances (8 per day) constructed for all mapped machines. First, we compute true positive and false positive rates as follows.

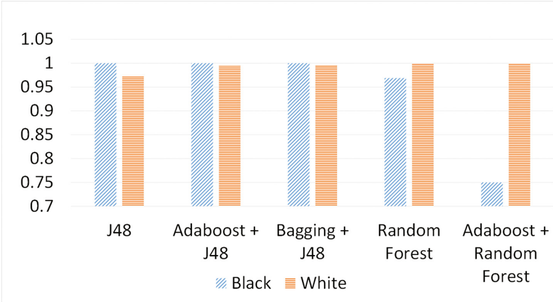

Fig. 3. True positive/negative results for a ±3 days alert period.

- For every black instance, if the detector alerts (classifies an instance as positive) on that machine within ±3 days of the black instance, it is considered true positive, otherwise it is considered false negative.
- For all instances that are at least 3 days afar from all black instances (if any) on their machine, if the instance is alerted on it is considered false positive, otherwise it is considered true negative.

The results are shown in Fig. 3. Comparing with Fig. 2, true positive results are identical for all algorithms except for Adaboost+random forest, for which the number of true positives is now down to 24 (as compared with 31 in the compromised machine detection experiment). True negative results are also similar to those presented in Fig. 2. The random forest algorithms are the winners and both exceed true negative rate of 0.9997. In absolute numbers, Adaboost+random forest has 31 false positives and random forest has 38 (out of a total of approx. 135,000 white instances), all of which occurred on a set of 14 clean machines.

As we described above, a black instance is considered a true positive if the detector alerts on that machine within ±3 days. We say that the detector provides *early detection* for a black instance, if it alerts on the same machine on some instance that precedes the black instance by at most 3 days. If the algorithm provides early detection for an algorithm, then the *early detection period* is the length of time between the black instance and the earliest alert on that machine that precedes it by at most 3 days.

Clearly, an instance for which the detector provides early detection is a true positive. Based on the results presented by Figs. 2 and 3, we conclude that the Random Forest algorithm strikes the best balance between true positive and true negative rates. In the early detection experiment, we evaluated its capability of providing early detection.

Overall, out of the 31 true positives, the Random Forest algorithm provides early detection for 25. The average early detection period is approx. 22 h. Figure 4 presents a histogram of the early detection period achieved by the algorithm.

5.2 Evaluation Using X-Force Exchange

IBM's X-Force Exchange is a cloud-based threat intelligence platform. It offers an API that allows the user to make queries regarding suspicious IPs and URLs (among other services). When querying about an IP/URL, X-Force Exchange returns a few scores in the range $[0 - 10]$ that quantify the IP's reputation w.r.t. several risk categories. The higher the score, the more suspicious is the IP/URL. The risk categories relevant for the evaluation we describe next are: *Bots*, *Botnet Command and Control Server*, and *Malware*.

We used X-Force Exchange in order to obtain an alternative method of evaluating the quality of our detector that is orthogonal to the features it uses. We did this by querying X-Force Exchange on IPs and URLs that are communicated with during instances in our test set. Our hypothesis was that black instances, as well as instances alerted on by our detector, would possess statistically-significant X-Force Exchange lower reputation levels (and therefore higher risk scores).

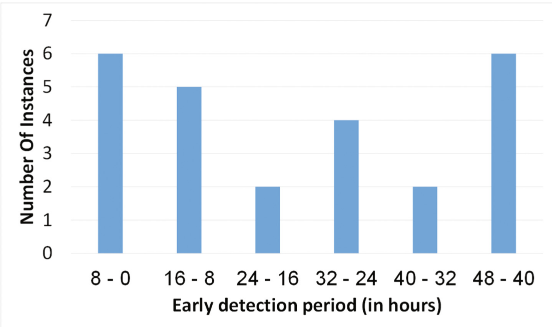

Fig. 4. Early detection histogram for the Random Forest algorithm.

Table 3 presents the average scores obtained for suspected (alerted), black, and white instances in the 3 relevant categories. As expected, the average score of black instances is significantly higher than that of white instances for all categories and the gap is especially large for the Botnet C&C category. Suspected instances on which our detector alerts obtain scores that are also significantly higher than those of white instances and, for the Bots and Malware categories, even higher than those of black instances.

Table 3. X-Force average scores

Category	Suspected	Black	White
Bots	4.211538	3.246154	1.969231
Malware	5.812	5.7	4.24
Botnet command and control	3.015385	4.6	1.323077

Table 4. X-Force p-values

Category	Black vs. Suspectet	Black vs. White	Suspectet vs. White
Bots	0.223193	0.046219	0.014266
Malware	0.161492	8.28E−08	1.42E−08
Botnet command and control	0.031646	1.04E−06	0.024139

We have also checked the statistical significance of the differences in score by computing their p-values and show the results in Table 4. As anticipated, the differences between the scores of black and white instances, as well as those of suspected and white instances, are statistically significant. On the other hand, the differences between black and suspected instances in the Bots and malware categories are insignificant, but they are significant for the botnet C&C category, where the grade of black instances is higher. Collectively, these results provide a clear indication that at least part of the presumed false positives of our detector do indicate suspicious host behavior that is not alerted by the AV.

6 Discussion

In this work, we presented a novel detector for infected machines in big data SIEM environments, that uses anti-virus induced labels for supervised classification. Our detection uses features that were selected out of more that 7000 features, computed based on time windows of QRadar data, containing reports on machines' flows and events.

We also present the results of extensive evaluation of our detector, conducted based on more than 6 terabytes of QRadar logs collected in a real production environment of a large enterprise, over the period of 3.5 months. Our evaluation shows that our detector identifies security incidents that trigger AV alerts with high accuracy. Moreover, it is able to provide early detection for a majority of these events. Our evaluation also indicates that our detector is able to alert on suspicious behavior that is unobserved by the AV.

One direction for future work is to find additional features that can further improve the accuracy of our detection approach. One possible way of doing this is to use the reputation of IPs/URLs that are communicated with during a time-window. These were used by us for evaluation, but not for constructing a detection model.

Another interesting direction is to conduct an experiment for observing the rate of concept drift exhibited by the models learnt using our approach and leverage its results for optimizing the duration and frequency in which models are learnt. Finally, it would also be interesting to check the extent by which models learnt on one QRadar system are applicable to other QRadar systems.

Acknowledgments. This research was supported by IBM's Cyber Center of Excellence in Beer Sheva and by the Cyber Security Research Center and the Lynne and William Frankel Center for Computing Science at Ben-Gurion University. We thank Yaron Wolfshtal from IBM for allowing Tomer to use IBM's facilities, for providing us the data on which this research is based, and for many helpful discussions.

References

1. Hadoop distributed file system. http://hadoop.apache.org/
2. Spark cluster computing. http://spark.apache.org/
3. Antonakakis, M., Perdisci, R., Nadji, Y., Vasiloglou II, N., Abu-Nimeh, S., Lee, W., Dagon, D.: From throw-away traffic to bots: Detecting the rise of DGA-based malware. In: USENIX Security Symposium, vol.12 (2012)
4. Bocchi, E., Grimaudo, L., Mellia, M., Baralis, E., Saha, S., Miskovic, S., Modelo-Howard, G., Lee, S.-J.: Magma network behavior classifier for malware traffic. Comput. Netw. **109**, 142–156 (2016)
5. Dietrich, C.J., Rossow, C., Pohlmann, N.: CoCoSpot: clustering and recognizing botnet command and control channels using traffic analysis. Comput. Netw. **57**(2), 475–486 (2013)
6. Gu, G., Perdisci, R., Zhang, J., Lee, W., et al.: BotMiner: clustering analysis of network traffic for protocol-and structure-independent botnet detection. In: USENIX Security Symposium, vol. 5, pp. 139–154 (2008)
7. Gu, G., Zhang, J., Lee, W.: BotSniffer: detecting botnet command and control channels in network traffic (2008)
8. Hall, M.A., Smith, L.A.: Practical feature subset selection for machine learning (1998)
9. IBM: IBM Security QRadar SIEM. http://www-03.ibm.com/software/products/en/qradar-siem/
10. iicybersecurity: International institute of cyber security. https://iicybersecurity.wordpress.com
11. Jiang, N., Cao, J., Jin, Y., Li, L.E., Zhang, Z.-L.: Identifying suspicious activities through DNS failure graph analysis. In: 2010 18th IEEE International Conference on Network Protocols (ICNP), pp. 144–153. IEEE (2010)
12. Kent, J.T.: Information gain and a general measure of correlation. Biometrika **70**(1), 163–173 (1983)
13. Kira, K., Rendell, L.A.: The feature selection problem: traditional methods and a new algorithm. In: AAAI, vol. 2, pp. 129–134 (1992)
14. Moskovitch, R., Stopel, D., Feher, C., Nissim, N., Elovici, Y.: Unknown malcode detection via text categorization and the imbalance problem. In: IEEE International Conference on Intelligence and Security Informatics, ISI 2008, pp. 156–161. IEEE (2008)
15. Musale, M., Austin, T.H., Stamp, M.: Hunting for metamorphic JavaScript malware. J. Comput. Virol. Hacking Tech. **11**(2), 89–102 (2015)
16. Narang, P., Ray, S., Hota, C., Venkatakrishnan, V.: PeerShark: detecting peer-to-peer botnets by tracking conversations. In: 2014 IEEE Security and Privacy Workshops (SPW), pp. 108–115. IEEE (2014)
17. Nari, S., Ghorbani, A.A.: Automated malware classification based on network behavior. In: 2013 International Conference on Computing, Networking and Communications (ICNC), pp. 642–647. IEEE (2013)

18. Deep Web News. https://darkwebnews.com
19. Weka 3: Data mining software in Java. University of Waikato. http://www.cs.waikato.ac.nz/ml/weka/
20. Perdisci, R., Lee, W., Feamster, N.: Behavioral clustering of http-based malware and signature generation using malicious network traces. In: NSDI, vol. 10, p. 14 (2010)
21. AV TEST: The independent it-security institute. https://www.av-test.org/en/statistics/malware/
22. Yen, T.-F., Oprea, A., Onarlioglu, K., Leetham, T., Robertson, W., Juels, A., Kirda, E.: Beehive: Large-scale log analysis for detecting suspicious activity in enterprise networks. In: Proceedings of the 29th Annual Computer Security Applications Conference, pp. 199–208. ACM (2013)
23. You, I., Yim, K.: Malware obfuscation techniques: a brief survey. In: 2010 International Conference on Broadband, Wireless Computing, Communication and Applications (BWCCA), pp. 297–300. IEEE (2010)
24. Yu, L., Liu, H.: Feature selection for high-dimensional data: a fast correlation-based filter solution. In: ICML, vol. 3, pp. 856–863 (2003)