Towards Intelligible Robust Anomaly Detection by Learning Interpretable Behavioural Models

Gudmund Grov[1,2], Marc Sabate[3], Wei Chen[2], and David Aspinall[2,4]

[1] Norwegian Defence Research Establishment (FFI), Norway
Gudmund.Grov@ffi.no
[2] School of Informatics, the University of Edinburgh, UK
{David.Aspinall,wchen}ed.ac.uk
[3] EPCC, the University of Edinburgh, UK
m.sabate@epcc.ed.ac.uk
[4] The Alan Turing Institute, London, UK

Abstract

Network anomaly detection for enterprise cyber security is challenging for a number of reasons. Network traffic is voluminous, noisy, and the notion of what traffic should be considered malicious changes over time as new malware appears. To be most useful, an anomaly detection algorithm should be *robust* in its performance as new types of malware appear: maintaining a low false positive rate but raising alarms at traffic patterns which correspond to malicious behaviour; and provide *intelligible* alarms that present their reasoning to support both the analysis of the alarms and necessary incident response.

In this paper we investigate new methods for building anomaly detectors using *interpretative behavioural models* which, we argue, can capture "normal" behaviours at a suitable level of abstraction to provide robustness, in addition to being inherently intelligible as they are interpretable for the security analyst. We consider two such models: a simple Markov Chain model with minimal behavioural structure and a Finite State Automata (FSA) with more structure, and show how these can be learned from normal network traffic alone. Our results show that the FSA performs better than common classifier methods with comparable results to standard Botnet detection methods. The results also indicate that the additional structure in the FSA is important. The FSA shows promise for robustness, although further work (with more data) is needed to fully explore this.

1 Introduction

Machine learning has been applied to computer security for some time, particularly to detect suspicious activity. For example, suspicious network traffic can be used to trigger alarms for intrusion detection systems. Two desirable features for such intrusion detection systems are *robustness* and *intelligibility*.

Robustness Network traffic is voluminous, noisy, and the notion of which traffic should be considered malicious changes over time as new malware appears. If benign network traffic has very regular patterns and malware has very different traffic also in a constant form, the detection problem is not too challenging. With labelled training data representing ground truth, we can train a classifier to separate malicious from benign behaviour. If malicious samples are not available (or assumed to change) a model of "normal" behaviour can be learned and malicious behaviour corresponds to *anomalies* that do not fit the model. This is known as *anomaly detection*. Good results can be obtained when training and testing data represent traffic from similar points in time. In reality, both types of traffic are moving targets: "normal" traffic evolves as software and devices are updated, and malicious traffic evolves as new types of

malware appear. This makes it difficult to produce anomaly detectors which are *robust*, in the sense that they remain accurate as traffic patterns evolve.

Although particular network flows may change, we hypothesise that there are underlying behaviour patterns which are more constant. Intuitively, an updated word processor package acts similarly to its predecessor and has similar network communication patterns; a new ransomware behaves similarly to other ransomware, when viewed abstractly enough.

Intelligibility Alarms raised by anomaly detectors may be genuine in the sense of correctly detected malicious traffic, or false due to incorrect classification of "normal" traffic as malicious. Within a security operations center (SOC), raised alarms need to be investigated – often manually by a SOC analyst [22]. Anomaly detection systems therefore face a key challenge in transforming the alarms/results into "actionable reports" for the analyst – a problem termed the *semantic gap* by Sommer & Paxson [26]. The main recommendation given by Sommer and Paxson to overcome this semantic gap is to "*understand what the system is doing*". Such insight into the reasoning behind triggering an alarm will both support the investigation of the given alarm, and also be used to rule out future false alarms, or automate incident response by updating an incident response system with new known malicious patterns.

Generally, there are two ways to achieve such intelligible anomaly detectors [29]: by generating explanations from the underlying machine learning (ML) models, or by using machine learning models that are interpretable by the analyst.

The advantage of generating explanations is that powerful black-box machine learning algorithms can be utilised. This will, however, involve communicating complex computational processes to humans – and the generated explanation, often a simplification, may be inherently different from the way the machine learning system operates [29]. The explanation may not therefore well reflect the reasoning process that caused the alarm, and consequently make it difficult to update the system for future similar alarms.

1.1 Overall approach & experimental setting

The objectives of this paper is two-fold. Firstly,

> *focusing on normal network traffic alone, we propose learning* behavioural models *that capture high-level invariants of network patterns.*

Secondly, instead of employing powerful state-of-the-art machine learning approaches, we base our work on learning simpler models that are interpretable for a human. More specifically,

> *we propose learning* interpretable *behavioural models which maintains a high level of detection performance.*

We learn and compare two such models and use them for anomaly detection: a simple Markov Chain (MC) with minimalistic behavioural structure; and a Finite State Automata (FSA) which exhibits more structure.

To evaluate our approach we use a dataset, called CTU-13 [11], of network traffic represented as NetFlows, which capture details of how IP traffic flows through a network. We develop a baseline experiment which resembles common approaches to detection (without the use of behavioural models) and compare and contrast our models to them. By comparing the result from the MC and the FSA we also gain some insight to the relative merits of behavioural structure. As we are using interpretable models we only focus on performance, and not intelligibility, in the evaluation.

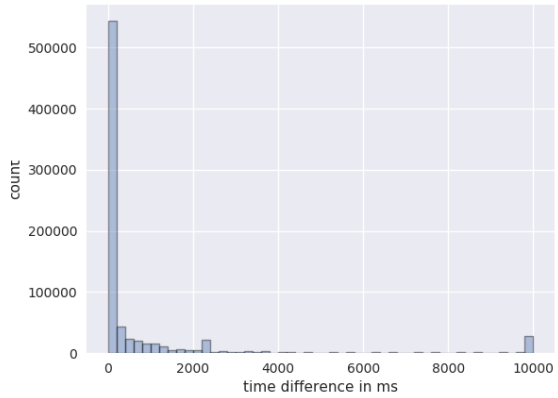
In our approach, the NetFlows are first grouped into *sessions*, where a session should contain NetFlows that are deemed related. In the training phase, the behavioural models are extracted from the sessions. We can then compute a similarity score for a new session with respect to these models and use this score to detect anomalies.

The CTU-13 Dataset Stratosphere Lab (Prague) [11] has made a laboratory generated dataset available to study botnet detection. It consists of more than ten million labelled NetFlow records captured on lab machines for 13 different botnet attack scenarios. These include spam, clickfraud, portscand and DDoS using P2P, IRC and HTTP.

Each NetFlow is labelled based on the source IP address. In the experiments, certain hosts are infected with a botnet and any traffic arising from such a host is labelled as *Botnet* traffic. Traffic from uninfected hosts is labelled as *Normal*. All other traffic is *Background*, as one cannot classify it. More detail about of the dataset and the different scenarios can be found in appendix A.

NetFlow sessions. CTU-13 NetFlows capture all traffic passing through a sensor in a network. Such unstructured traffic is unlikely to exhibit behavioural patterns. Therefore, we group the NetFlows by host according to their source IP addresses. Of course, a single host may generate network flows that are unrelated to each other, because of different functionalities within a single application or because of different applications. Therefore, we group related traffic together in an abstraction we call a *session*.

A perfect session grouping would require (unavailable) information from the top layers of the network stack. As a first approximation, we use the start time of NetFlows, judging flows that are "close in time" to be in the same session. The relative time difference in the CTU-data is shown by the following histogram:



Around 90% of NetFlows start less than 5 seconds after the previous NetFlow. We therefore define a session as:

> *If a NetFlow starts less than five seconds after the previous NetFlow for that host then they are in the same session; otherwise a new session is started.*

Our sessions bear similarities to the 'simple expiry model' in [24] with added robustness over the NetFlows' length in time as our deltas are between the end time of one NetFlow and start time

of the next one, where at the deltas in the 'simple expiry model' are between start times only. Appendix A contains additional information of the sessions segmented by scenarios, infected and normal hosts.

Baseline experiment In our baseline experiments we use three different supervised machine learning algorithms: Multilayer Perceptron (MLP), Random Forest (RF), and Support Vector Machines (SVM) with radial basis function kernels [3]. As further explained in section 3.1.1, the dataset is split into a training set and test set and the classifiers are trained on both normal and malicious traffic in the training set and evaluated on the test set.[1]

For each session, features are extracted and fed into the training process. The baseline experiment is designed to compare with our new approach, and should there resemble common approaches. Using a survey by Miller & Busby-Earle [20] as a guide, we used the following aggregated measures:

- the total number of NetFlows and IP packets;
- the total number of NetFlows to the most frequently connected destination address within the session;
- the average number of IP packets per NetFlow;
- the average number of bytes per IP packet;
- the total number of ICMP, RTP, TCP and UDP NetFlows.

We would like to determine *if a given session on a given host is benign or malicious*. However, the labelling of CTU-13 is too coarse as it only tells us which hosts are infected; traffic originating from infected hosts may be non-malicous. For example, some NetFlow captures were started before the botnets began operating, meaning there will be a time window where all traffic on an infected host is normal. There are log files associated with the scenarios that might be used to solve this discrepancy, but this would need elaborate pre-processing.

Instead, for prediction and testing purposes, we add a post-processing step for each host and for each session. This will take into account previous sessions to decide whether the host is infected. In the CTU-13 dataset, since the duration of the scenarios is only a few hours long, this is done by updating a running average of the prediction of the classification models for each session, taking into account previous sessions within the scenario. Thus, we turn the overall classification problem into deciding

if a given host is benign or malicious,

for which we have ground truth.

1.2 Related work

The CTU-13 dataset has been used to assess multiple botnet detection systems [11], however their treatment of the dataset deviates significantly with how we are treating it, rendering a direct comparison with this work infeasible.[2]

There has been a range of work on building models of network traffic to characterise botnets. Anomaly detectors using these models have demonstrated good performance, e.g., state transition models used in BotHunter [14] and in [30]. BotHunter is built on abnormal behavioural models and complex anomaly detection rules. This deviates from our work as their model has

[1] This gives these baseline approaches an advantage over our behavioural models which are only trained on normal traffic from the training set.

[2] Our work deviates from [11] as we are using only normal traffic, which we derive sessions from that are used for anomaly detection.

to be provided and maintained, whilst we can automatically infer it. Moreover, the models used by BotHunter capture abnormal behaviour while our approach capture normal behaviour. More recently, CTU-13 has been used to extract probabilistic real-time automata for anomaly detection [23] and to build Markov Chains to model botnet behaviour [12].

Chen et al [5, 6] have captured high-level FSA from static analysis of malware source code, and shown robustness for the derived classifiers. Although not directly relevant for our work, automata learning has been applied to learn botnet C&C protocols for botnets [7], and to learn behaviour models from logs [9].

In contrast to the above studies, which focused on modelling malicious behaviours, we train our behavioural models using only *normal traffic* to model *benign behaviour*. We are not familiar with any other work to that addresses learning of behavioural models from normal network traffic to detect malicious activities.

Common approaches to botnet detection attempt to characterise networks with suitable aggregate values such as traffic volume, entropy of destination ports [17], and statistical variables on flows [13]. Models of normal or abnormal behaviour can be trained by exploring patterns between these features [14, 30]. Detection is usually by a threshold function on the deviation from, or the similarity to, these trained models. Our work deviates from this by using behavioural, and not aggregated features.[3] For comparison we do however use aggregate features in our baseline experiment.

Learning of rules and signatures has been studied in the context of malware detection [4], however this general line does not fit with our notion of extracting behavioural models.

A considerable amount of work has been conducted to generate explanations from machine learning systems – see e.g. [15] for a survey. This work often comes under the heading 'eXplainable AI' (XAI)[4]. One of the most relevant work for us is [28] where a FSA is generated to explain a neural network. We return briefly to this subject of explanation in section 4.

1.3 Main contributions and structure of paper

This paper shows that it is possible to learn interpretable behavioural models from network traffic and that these can improve roubustness of anomaly detectors. We also show novel application of learning MC and FSA, and a new detection method based on them. For several evaluation metrics, the FSA receives better scores than standard classifiers, even when they are trained using ground truths while the FSA is trained on normal traffic only. Finally, we show the importance of structure in the models by showing that a more structured FSA outperforms the simpler MC.

In the next section we describe how the learning approach for both the FSA and MC, and how to use them for anomaly detection. Section 3 evaluates our approach by comparing the performance of the MC, FSA and baseline experiment using the CTU-13 dataset. We conclude and discuss further work in section 4.

2 Learning interpretable behavioural models

In this section we will describe the interpretable behavioural models and how they are learned. Section 3 gives details on how they are used. We introduce two different models: Markov

[3] Albeit, as explained in section 2.3, we do also attempt an experiment combining aggregated and behavioural features.

[4] See DARPA's XAI programme https://www.darpa.mil/program/explainable-artificial-intelligence. Several approaches are outlined in a presentation of the programme [16].

Chains (MC) and Finite State Automata (FSA). The advantage of MC is its simplicity, both in learning and usage, while FSA exhibits structure, meaning it can capture the termporal ordering of Netflows.

Common to both of them is the need for a descriptive "action" that has to be derived from each NetFlow in a session, which is discussed first.

Extracting actions A behavioural model provides insight into the composition of actions, where an action represents the atomic step/transition of a model. The choice of a suitable action from a NetFlow is therefore crucial: if the action is too specific then the model is likely to be too concrete and overfitted to the training data; if the action is too generic, then we may not be able to sufficiently distinguish between benign and malicious behaviour (i.e., over-generalisation). As a first approximation we simply used the network *protocol* to represent an action.[5] To illustrate, consider the following selection of NetFlows from CTU-13:

```
Proto,SrcAddr,Sport,Dir,DstAddr,Dport,TotPkts,TotBytes,SrcBytes,Label
-----------------------------------------------------------------------
tcp,147.32.84.118,3038,->,2.215.205.93,6881,124,124,Background
tcp,147.32.84.118,3168,->,2.215.205.93,6881,186,186,Background
tcp,147.32.84.118,3312,->,2.215.205.93,6881,186,186,Background
udp,46.217.68.251,20962,<->,147.32.86.116,19083,136,75,Background
udp,147.32.84.229,13363,<->,208.88.186.4,34033,2241,1643,Background
```

For simplicity, assume these five NetFlows form a session. In a given NetFlow the first element of the listing is the protocol. For this example the session becomes the following sequence of actions, which are used to train a model:

```
tcp; tcp; tcp; udp; udp
```

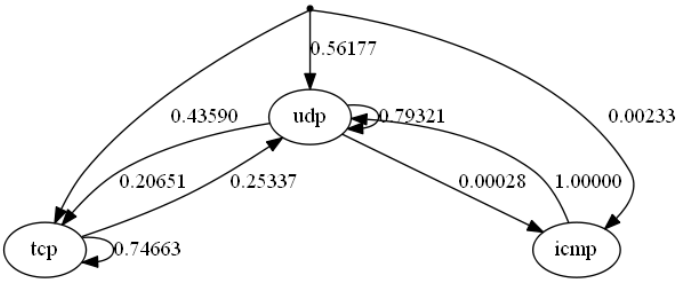
2.1 Learning Simple Markov Chains (MC)

Our first model is a simple Markov Chain (MC) which ignores the overall structure of a session and only considers the probability of two consecutive actions. The MC has the following properties:

- Each type of action becomes a (unique) state.
- There is a state representing the initial (empty) action.
- An edge between two states is labelled by a probability.

The MC is trained from the training set by counting the number of times two actions follow each other, and the probability on a given edge is computed by dividing the counted values with the total number of outgoing transitions for the given state. The following MC captures one particular host in the training set:

[5] Initial experiments with additional use of port number resulted in very large and specialised models, and were therefore abandoned. Finding the most suitable notion of action for particular model types remains future work.



Here, a node represents an action of a sequence, which in this case is either, the TCP, UDP or ICMP protocol. An edge gives the probability that the next NetFlow is the stated type of action. For example, the probability that a TCP NetFlow is followed by another TCP is almost 75%. The unlabelled node at the top is the initial state, meaning for example that the probability that the first action is UDP is about 56%. Given a new session, we compute the following measures of similarity to the trained MC:

- the *total probability*, i.e., the product of each pair of actions; and
- the *average probability*, i.e., the sum of each probability divided by number of actions.

2.2 Learning Finite State Automata (FSA)

The MC ignores the context of actions with respect to what has happened before, and what will happen afterwards. To capture more structure of the behaviour we use a richer Finite State Automata (FSA) to represent the learned model.[6]

Learning automata generally either follows an *active approach*, where the learning system interacts with a teacher following the L* algorithm [1], or a *passive approach* based around state-merging [2, 21]. Active learning has been shown to be applicable within network security to infer protocols, e.g. C&C protocols for botnets [7]. These approaches are characterised by a system acting as a teacher which can be queried during training. Passive learning tends to be applied in scenarios like ours, where only data is available and there is no system to query. This has been particularly successful within software engineering [10], where software models are derived from logs of their execution. As in our case, there is limited scope for negative examples to constrain the generalisation/inference process. We therefore follow the passive learning approach.

We use the *Evidence-Driven State Merging* (EDSM) approach as described in [18, 27, 10]. EDSM follows the following steps:

1. Compute a Prefix Acceptor Tree (PTA), essentially a Trie where the common prefix of each action sequences are combined. This is the initial automata.
2. Where there are two states that are "mergeable" then merge them. Continue until there are no two states that can be merged.

Note that merging two states always generalises the FSA — any valid behaviour will remain valid while new behaviour not captured before can become valid. (If used in a setting where malicious behaviour was used for training, the merging phase would include a test to ensure that the FSA did not capture any bad behaviour; if this was the case, the state merge would be rolled back.)

Two important questions are (*i*) how to find two potential states to merge, and (*ii*) how to decide how similar they are.

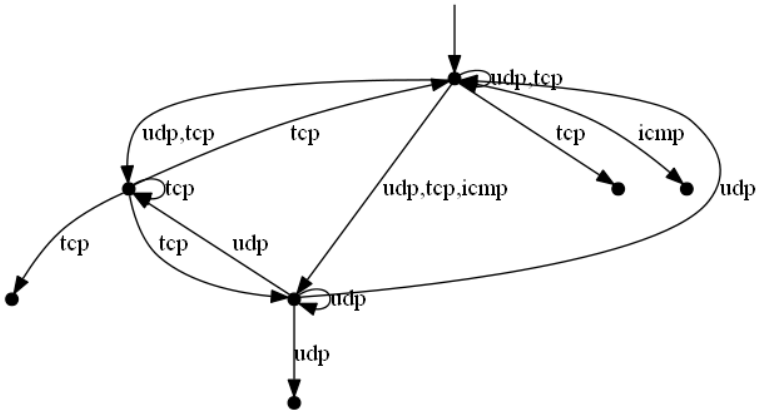
[6] Learning minimal automata (with both positive and negative examples) is known to be NP-hard. We don't need minimal models so more efficient methods are possible.

A naïve approach to (*i*) is to try all possible pairs of states. This will give $\binom{n}{2}$ possibilities, where n is the number of states, which will obviously not scale. Instead we use the *Blue Fringe* algorithm to reduce the number of possible pairs to merge [18, 10].

We follow a similar approach to [27] to compute a similarity measure to support (*ii*). This is achieved by comparing the similarity of outgoing transitions, increasing the measure for each edge labeled by the same action. This is applied recursively for the target states up to a given threshold. In addition, we subtract any disjoint transition[7] thus reducing the similarity (this adjustment was not discussed in [27]). We can then set a threshold for this measure to decide how aggressive merging should be.

There are several ways to configure the algorithm: choosing the number of steps of look-ahead, the treatment of disjoint transitions, and the threshold for the overall score. In our experiments in section 3 we look three recursive steps ahead, reducing the similarity measure in the presence of disjoint transitions. We set a threshold of zero, meaning there must be more similar than disjoint transitions.

The following FSA, where edges are labelled by actions, has been trained from the same host as the MC shown previously:



Once we have merged the states into a more abstract FSA, we can derive the following measures of similarity for a new session:

- Whether or not the action sequence is a *valid* word of the automaton.
- The *edit distance* (a form of Levenshtein distance) of the sequence. This is a measure of the minimum number of changes (insertions, deletions or substitutions) to the actions that are required for it to be valid for the automata. A distance of zero means that no changes are needed; the maximum distance is the length of the sequence. We compute a normalised version by dividing the distance by the length of the sequence, returning a measure between zero and one.

2.3 Integrating behavioural models with aggregate features (1CSVM$_B$)

Aside from using similarity measures from the MC and FSA models directly as classifiers, we can also use them as input features in standard machine learning algorithms. We use the similarity measures as input together with the aggregated features used in the baseline experiment.[8] This enables anomaly detection using *one-class SVM* [25], with only normal traffic used for training.

[7] A disjoint transition is an outgoing edge from one state that is not present in the other.

[8] As opposed to the baseline experiment, we still only learn from normal traffic.

One-class SVM treats all observations in the training set as belonging to one class, and it finds a discriminative boundary around the normal observations in the training set. It uses the radial basis function as a kernel. When a new observation does not fall within the learned boundary it is classified as anomalous.

Note that this approach cannot be considered to be interpretable, as it is essentially a one-class SVM with behavioural features.

3 Evaluation

We evaluate our detectors based on their performance. We compare our detectors using intepretative behavioural models with the baseline approaches described in section 1.1 and contrast the results of the FSA and the MC with each other. The process and phases for the evaluation are first described in section 3.1, before we give the results in section 3.2. Finally, we make a short digression in section 3.3 with a brief discussion on evaluating intelligibility.

3.1 Evaluation phases

Here, we briefly describe the four phases of the evaluation: data preprocessing; training; prediction; and evaluation.

3.1.1 Data preprocessing

For each host, the sessions are extracted as described in section 1.1. The dataset is then split into a training set and a test set. The training set only includes three scenarios[9], where just one malware (called *Rbot*) is deployed. The test set includes the 10 remaining scenarios. This setting aims to validate that our behavioural models trained on benign sessions can generalise better to detect any type of malicious behaviour (by predicting it as abnormal) than standard classifiers trained on a specific case of malicious behaviour. It is crucial to the experiment that the test set includes new and unseen botnets, as we are predominantly investigating robustness of detection techniques.

Due to the limited number of non-infected hosts in the dataset, we exclude one of them from the training set to avoid misleading good predictions where the classifier would be predicting the actual host of the session.

For each session the action sequences are extracted for both the training and test sets. This is used to train and evaluate the interpretative behavioural models. For the SVM, Random Forest and Multilayer Perceptron models, the aggregated features per session is extracted for both sets.

3.1.2 Training

We only use normal hosts to train the interpretative behavioural models, and the one-class SVM augmented with features from the FSA (edit distance) and MC (probability and average probability). We train the SVM, RF, and MLP classifiers using both normal and infected hosts.

3.1.3 Prediction

The trained models provide a session-level score for each session of the test set. This score is different for each model:

[9] These are scenarios 3, 4, and 10 as described in appendix A.

- The supervised machine learning models (SVM, RF and MLP) provide a probability (log-probability for SVM) of the session being malicious or not.
- The FSA model provides the edit distance of a session with respect to all the learned automata (one for each host); these are averaged into a single score.
- The MC model provides a combination of the probability and average probability of a session of the learned MC.
- The one-class SVM provides the distance of the session to the disriminative boundary. This distance is positive for the predicted normal sessions, and negative for the anomalous sessions.

From these scores, we get the host-level classification taking into account an accumulated average of the scores. The resulting score is turned into a binary prediction by finding the threshold value that maximises the F_β score of the model, a weighted harmonic mean of precision and recall.

3.1.4 Evaluation

Our goal is to correctly classify malicious hosts, thus:

- a true positive (TP) is a 'correctly classified infected host';
- a true negative (TN) is a 'correctly classified normal host';
- a false positive (FP) is a 'normal host incorrectly classified as infected'; and
- a false negative (FN) is an 'infected host incorrectly classified as normal'.

The false positive rate (FPR) and the true positive rate (TPR) are defined standardly as:

$$FPR = \frac{FP}{TN + FP} \qquad TPR = \frac{TP}{TP + FN}$$

Evaluation is undertaken on the whole test set – also for scenarios where different malware are executed. The following classification metrics are used to evaluate the models using the test set:

- **Area under the Curve (AUC)**: the area under the ROC curve. It illustrates the relationship between FPR and TPR at different threshold settings of the classifier's output.
- **Precision**: percentage of predicted positives that are true positives, i.e., $\frac{TP}{TP+FP}$.
- **Recall**: percentage of positives predicted as positives, i.e., $\frac{TP}{TP+FN}$.
- F_β **score**: $(1+\beta^2) \cdot \frac{\text{recall} \cdot \text{precision}}{\beta^2 \cdot \text{precision} + \text{recall}}$.

If $\beta = 1$, the F_β measure corresponds to the harmonic mean of precision and recall. Setting $\beta < 1$ lends more weight to precision, while $\beta > 1$ favours recall. While our goal is to prove robustness, which means a weight on recall, we do not want to have a high level of robustness at the expense of precision. We therefore choose $\beta = 0.6$ to avoid a high number of false positives.

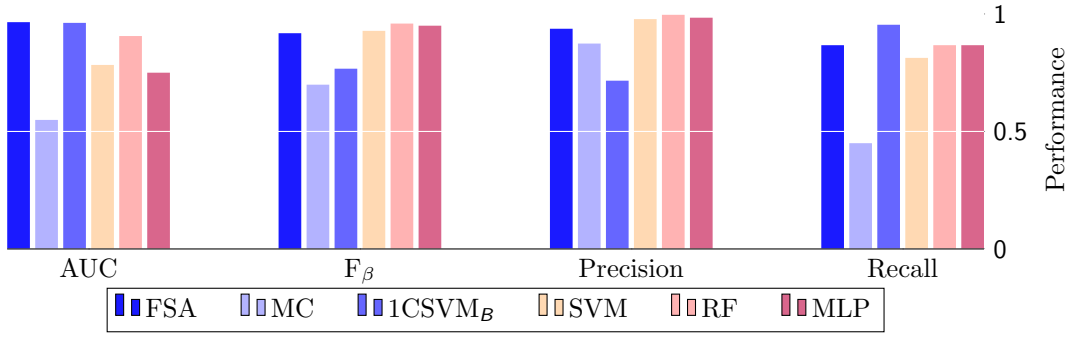
3.2 Results of the experiments

We have performed classic anomaly detection by only training our interpretative behavioural models on benign behaviour, whilst the baseline classifiers were trained on labelled data. The following table shows the overall results:

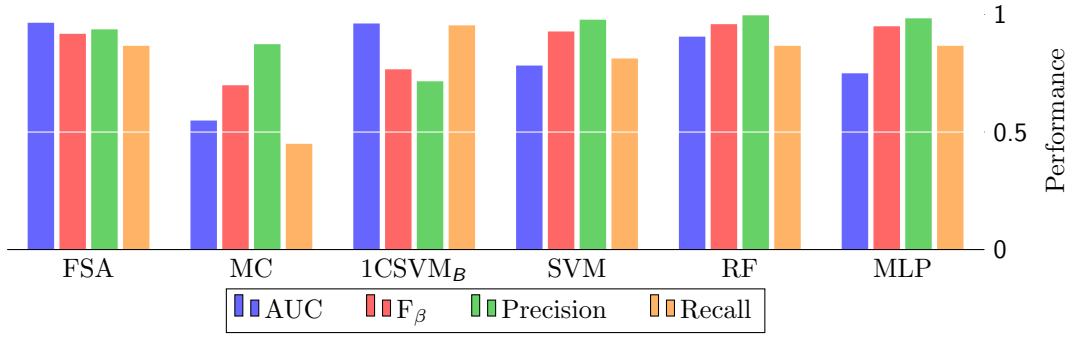
	AUC	**F_β**	**Prec.**	**Recall**
FSA	0.965	0.918	0.937	0.867
MC	0.549	0.699	0.874	0.450
$1CSVM_B$	0.962	0.767	0.716	0.954
SVM	0.783	0.928	0.978	0.813
RF	0.906	0.959	0.997	0.867
MLP	0.750	0.950	0.984	0.867

The one-class SVM that combines aggregate features with features from the FSA and MC is labelled '$1CSVM_B$'

The table shows the respective percentages using the four classification metrics used, with the behavioural models above the line and the baseline classifiers below the line. The results are depicted graphically, structured by each metric, in the following histogram:



For each metric, the three leftmost bars, coloured by different shades of blue, are the three interpretative behavioural models; while the three red-shaded bars on the right are the baseline experiment. As an alternative visualisation, we also depict the result by grouping the results structured by the underlying model:



As the results show, both the FSA and the one-class SVM are reliable methods to detect malicious sessions. The result for the MC, on the other hand, is poorer. This indicates that one either needs more structure in the composition of NetFlows (as in the FSA) or aggregate features (as used in the binary classifiers). When MC is combined with features from the FSA in the one-class SVM, the highest recall is achieved. We have not investigated the degree to which the improvement is due to the MC.

With respect to robustness, recall is the most important measure as it identifies the degree to which new malware is detected. For this metric, one-class SVM with augmented FSA and MC features actually had a higher recall, but performs considerably worse than the FSA in terms of precision, with a high number of false positives. The FSA performs better than all classifiers

for the AUC score, while the F_β score is similar even as precision is favoured. SVM, Random Forest and MLP provide a good classification in terms of precision, where the score is higher than the interpretative behavioural models. One reason may be that the ground truth used in training has more impact for precision. We suspect that we can reduce the over-generalisation of the learned FSA, and thus increase recall, by configuring the correct level of similarity when merging states. This remains future work.

Our false positive rates are comparable to those of BotHunter, while our experiments indicate that our approach is more robust and is likely to perform better on new unseen malicious behaviour. In [23], CTU-13 was used to extract probabilistic real-time automata of botnet behaviour. As for our FSA, a high recall on the CTU-13 data in [23], albeit using a larger training set compared with ours, which indicates that behavioural models of both benign and malicious patterns show promise in detecting new malware.

3.2.1 Summary & discussion

We have not focused on the runtime of our model constructions, but make the following notes.[10] Training the MC is a simple iteration over the data, which is of linear cost. Training the FSA depends on the size of the initial PTA, and varies from less than a minute to around 30 minutes. Computing the edit distance (FSA) and probabilities (MC) is instantaneous.

Besides their performance, another advantage of the FSA with respect to the other methods is that it does not need a session to be completed in order to raise an alarm of that session being anomalous. Instead of an input given by session aggregated features, a FSA accepts as input the actions of a session one by one. This allows them to update the session score before it is over, giving the possibility to raise a preemptive alarm as soon as the session starts to be anomalous. This gives a clear margin of action to counter attacks such as DoS, with long sessions and high number of superfluous requests to a targeted host.

It is also worth noting that as our models are built on network analysis rather than content analysis they will not be affected by content encryption such as TLS.

In summary, we believe the current results are promising in terms of building a robust malware detector where interpretable behavioural models based on simple actions achieve similar performance, or even outperform, standard binary classifiers with more complex features.

3.3 On evaluating intelligibility

DARPA's XAI programme [16] has a goal of producing more explainable models while maintaining a high level of learning performance. As we have used fully interpretative models we have only focused on the second objective of maintaining a high level of learning performance. However, we end this section with some remarks on evaluating how intelligible, or explainable, a model may be.

The general problem of explaining results is a major challenge for machine learning, and AI more generally. A report by McKinsey from 2018 [8] considers explainability the third biggest challenge for AI adoption.[11]

Explanations may serve different purposes [29], and this will impact on how to measure the quality of the explanation. For example, it may be to *accept* the decision from the ML system, in our case to help convince the SOC analyst that traffic is indeed malicious; or it may be

[10] The experiments has been conducted on a single node in of two 2.1 GHz 18-core Intel Xeon E5-2695 processors with 256 GB of memory.

[11] The explainability challenge was only behind labelling and obtaining good datasets – both of these are challenges that also affect the work presented in this paper.

to *educate* the user, which in our case entails helping the analyst in the actual investigation of the cause of the alarm and respond accordingly. Explanations can also have many *modes* [16]: they may describe elements and contexts to support a choice made by the ML system; they may utilise visualisations to highlight portions of data that supports a choice; they may contain cases that show specific examples/stories that support a choice; and they may be used to communicate rejections of other alternative choices.

Weld et al [29] argues for the use of *counterfactuals*, i.e. the ability to answer "what-if" questions, in order to define and measure intelligibility. In other words, it relates to how a user can predict how changes to an input feature will change the result from the ML system. Intuitively, one would expect that a human will be able to "step through" how a particular session fits a FSA, and it is easy to visualise where and which edits will be required. Thus, by making changes to the session a human should be able to predict how that will change the number of edits required (as output). Thus, from a counterfactual point of view we would expect FSA to be considered intelligible.[12]

However, it is not obvious that the ability of answering counterfactuals necessary helps to reduce the semantic gap between an anomaly detector and actions to be taken by the SOC analyst. Here, a psychological model of explanation outlined in [16] may be more promising, where both changes to the analyst's mental model from explanations, and subsequent improved performance due to this, are measured. This study of the intelligbility of our detection methods remains future work.

4 Conclusions and future work

We have demonstrated that we can learn interpretable behavioural models from normal network traffic, and receive better scores than standard classifiers, even when they are trained using ground truths. Our results are promising but more experiments are required both to check that our findings are transferrable, and to use finer grained ground truths to measure the accuracy of identifying malicious sessions rather than malicious hosts. To achieve this, further modern curated data sources would be highly valuable.

Our actions, which in essence contains the relavant NetFlow feature are limited. We would like to measure which information from a NetFlow record is most important in an action, and more expressive ways to relate actions, which may require incorporating FSA with state (as in [27]). Using states in the models may also allow quicker detection of DoS attacks.

The experiments have shown that the structure/composition of NetFlows is important as the FSA outperformed the much simpler MC model, which is rather unsurprising. We also note that one-class SVM combining aggregate features with similarity measures from FSAs and MCs improves recall, but has lower scores on the other evaluation metrics.

Our approach to generate sessions, and extracting actions from NetFlows within sessions, are a first approximation that require further study. For sessions, we want to see what happens when we filter out irrelevant NetFlows and better group relevant ones.

An alternative to using interpretable models is to generate explanations from more powerful methods (see section 1). (Deep) neural networks are currently receiving considerable attention in many fields. Whilst there has been a considerable amount of work addressing explanation generation for neutral networks [15], this powerful representation is in general not considered to be intelligible [29, 22]. *Recurrent Neural Networks* (RNN) [19] is a type of neural network

[12] It is worth recalling that, due to the labelling of CTU-13, we are not predicting if a session is malicous based on the number of edits, but if a host is malicious based on the average number of edits for each session for the given host. The prediction is however still based on the number of edits.

that can capture behehaviour patterns. Explaination generation from RNN has been studied by Weis et al [28] in the form of FSA generation by active learning with the L* algorithm [1], where the FSA acts as the explanation.[13] It would be interesting to compare our results with what their approach will produce.

References

[1] Dana Angluin. Learning regular sets from queries and counterexamples. *Information and Computation*, 75(2):87–106, November 1987.

[2] A. W. Biermann and J. A. Feldman. On the Synthesis of Finite-State Machines from Samples of Their Behavior. *IEEE Transactions on Computers*, C-21(6):592–597, June 1972.

[3] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[4] Marco Caselli, Emmanuele Zambon, Johanna Amann, Robin Sommer, and Frank Kargl. Specification mining for intrusion detection in networked control systems. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 791–806. USENIX Association, 2016.

[5] Wei Chen, David Aspinall, Andrew D Gordon, Charles Sutton, and Igor Muttik. More semantics more robust: Improving Android malware classifiers. In *Proceedings of the 9th ACM Conference on Security & Privacy in Wireless and Mobile Networks*, pages 147–158. ACM, 2016.

[6] Wei Chen, David Aspinall, Andrew D Gordon, Charles Sutton, and Igor Muttik. On robust malware classifiers by verifying unwanted behaviours. In *International Conference on Integrated Formal Methods*, pages 326–341. Springer, 2016.

[7] Chia Yuan Cho, Eui Chul Richard Shin, Dawn Song, et al. Inference and analysis of formal models of botnet command and control protocols. In *Proceedings of the 17th ACM conference on Computer and communications security*, pages 426–439. ACM, 2010.

[8] Michael Chui, James Manyika, and Mehdi Miremadi. What ai can and can't do (yet) for your business. *McKinsey Quarterly*, 2018.

[9] Paolo Milani Comparetti, Gilbert Wondracek, Christopher Kruegel, and Engin Kirda. Prospex: Protocol Specification Extraction. pages 110–125. IEEE, May 2009.

[10] C. Damas, B. Lambeau, P. Dupont, and A. van Lamsweerde. Generating annotated behavior models from end-user scenarios. *IEEE Transactions on Software Engineering*, 31(12):1056–1073, December 2005.

[11] S. García, M. Grill, J. Stiborek, and A. Zunino. An empirical comparison of botnet detection methods. *Computers & Security*, 45(Supplement C):100–123, September 2014.

[12] Sebastian García and Michal Pechoucek. Detecting the behavioral relationships of malware connections. In *Proceedings of the 1st International Workshop on AI for Privacy and Security*, pages 8:1–8:5. ACM, 2016.

[13] Guofei Gu, Roberto Perdisci, Junjie Zhang, and Wenke Lee. Botminer: Clustering analysis of network traffic for protocol- and structure-independent botnet detection. In *Proceedings of the 17th Conference on Security Symposium*, pages 139–154. USENIX, 2008.

[14] Guofei Gu, Phillip Porras, Vinod Yegneswaran, Martin Fong, and Wenke Lee. Bothunter: Detecting malware infection through ids-driven dialog correlation. In *Proceedings of 16th USENIX Security Symposium*, pages 12:1–12:16. USENIX, 2007.

[15] Riccardo Guidotti, Anna Monreale, Salvatore Ruggieri, Franco Turini, Fosca Giannotti, and Dino Pedreschi. A survey of methods for explaining black box models. *ACM computing surveys (CSUR)*, 51(5):93, 2018.

[13] It is though worth noting that that there is no guarantee of behavioural equivalence between the RNN and the generated FSA, i.e. between the explanation and the underlying model.

[16] David Gunning. EXplainable Artificial Intelligence (XAI), 2017. Defense Advanced Research Projects Agency (DARPA),Program Update November 2017,Available from https://www.darpa.mil/attachments/XAIProgramUpdate.pdf [last visited: 16.08.2019].

[17] Anukool Lakhina, Mark Crovella, and Christophe Diot. Mining anomalies using traffic feature distributions. In *Proceedings of the 2005 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, pages 217–228. ACM, 2005.

[18] Kevin J. Lang, Barak A. Pearlmutter, and Rodney A. Price. Results of the Abbadingo one DFA learning competition and a new evidence-driven state merging algorithm. In *Grammatical Inference*, volume 1433, pages 1–12. Springer, Berlin, Heidelberg, 1998.

[19] Danilo P Mandic, Jonathon A Chambers, et al. *Recurrent neural networks for prediction: learning algorithms, architectures and stability*. Wiley Online Library, 2001.

[20] S. Miller and C. Busby-Earle. The role of machine learning in botnet detection. In *11th International Conference for Internet Technology and Secured Transactions (ICITST)*, pages 359–364, 2016.

[21] J. Oncina and P. Garcia. Inferring regular languages in polynomial update time. In N. Pérez de la Blanca, A. Sanfeliu, and E. Vidal, editors, *Pattern Recognition and Image Analysis*, volume 1 of *Series in Machine Perception and Artificial Intelligence*, pages 49–61. World Scientific, 1992.

[22] Alina Oprea, Zhou Li, Robin Norris, and Kevin Bowers. MADE: Security Analytics for Enterprise Threat Detection. In *Proceedings of the 34th Annual Computer Security Applications Conference*, pages 124–136. ACM, 2018.

[23] G. Pellegrino, Q. Lin, C. Hammerschmidt, and S. Verwer. Learning behavioral fingerprints from netflows using timed automata. In *2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, pages 308–316, May 2017.

[24] P. Rubin-Delanchy, D. J. Lawson, M. J. Turcotte, N. Heard, and N. M. Adams. Three statistical approaches to sessionizing network flow data. In *2014 IEEE Joint Intelligence and Security Informatics Conference*, pages 244–247, Sep. 2014.

[25] Bernhard Schölkopf, John C. Platt, John C. Shawe-Taylor, Alex J. Smola, and Robert C. Williamson. Estimating the support of a high-dimensional distribution. *Neural Comput.*, 13(7):1443–1471, July 2001.

[26] Robin Sommer and Vern Paxson. Outside the Closed World: On Using Machine Learning For Network Intrusion Detection. In *2010 IEEE symposium on security and privacy*, pages 305–316. IEEE, 2010.

[27] Neil Walkinshaw, Ramsay Taylor, and John Derrick. Inferring extended finite state machine models from software executions. *Empirical Software Engineering*, 21(3):811–853, June 2016.

[28] Gail Weiss, Yoav Goldberg, and Eran Yahav. Extracting Automata from Recurrent Neural Networks Using Queries and Counterexamples. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 5247–5256. PMLR, 2018.

[29] Daniel S Weld and Gagan Bansal. The challenge of crafting intelligible intelligence. *Communications of the ACM*, 62(6):70–79, 2019.

[30] Peter Wurzinger, Leyla Bilge, Thorsten Holz, Jan Goebel, Christopher Kruegel, and Engin Kirda. Automatically generating models for botnet detection. In Michael Backes and Peng Ning, editors, *Computer Security – ESORICS 2009*, pages 232–249. Springer, 2009.

A Details of the CTU-13 dataset

Here, we provide some additional information about the CTU-13 dataset [11]. The following table gives some details about the bots and type of malware for each scenario:

	Bot	#Bots	IRC	Spam	ClickFraud	PortScan	DDoS	P2P	HTTP
1	Neris	1	✓	✓	✓				
2	Neris	1	✓	✓	✓				
3	Rbot	1	✓		✓				
4	Rbot	1	✓			✓			
5	Virut	1		✓		✓			✓
6	Menti	1				✓			
7	Sogou	1							✓
8	Murlo	1				✓			
9	Neris	10	✓	✓	✓	✓			
10	Rbot	10	✓				✓		
11	Rbot	3	✓				✓		
12	NSIS.ay	3						✓	
13	Virut	1		✓		✓			✓

The following table gives additional information about both the size of scenarios of the dataset, and some infomration abouth the extracted sessions:

	Dur.(hrs)	#Botnet Flows	#Normal Flows	#Botnet sess.	#Normal sess.
1	6.15	39,933	30,387	449	2,773
2	4.21	18,839	9,120	527	1,422
3	66.85	26,759	116,887	1,485	23,460
4	4.21	1,719	25,268	237	1,869
5	11.63	695	4,679	32	243
6	2.18	4,431	7,494	639	843
7	0.38	37	1,677	5	151
8	19.5	5,052	72,822	1,690	7,444
9	5.18	179,880	43,340	347	2,192
10	4.75	106,315	15,847	462	1,955
11	0.26	8,161	2,718	11	95
12	1.21	2,143	7,628	133	468
13	16.36	38,791	31,939	363	5,780

For each session, the table contains the total duration (in hours) for the dataset, the number of malicious and benign NetFlows, and the number of malicious and normal sessions generated from these. Note that scenarios 3,4 and 10 are used for training in evaluation, while the remaining scenarious are used for testing.