On Robust Anomaly Detection with Behavioural Models

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

[1] EPCC, The University of Edinburgh, `m.sabate@epcc.ed.ac.uk`
[2] Norwegian Defence Research Establishment (FFI), `Gudmund.Grov@ffi.no`
[3] School of Informatics, The University of Edinburgh, `{David.Aspinall,wchen2}@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.
In this paper we investigate some new methods for building anomaly detectors using *behavioural models* which, we argue, can capture "normal" behaviours at a suitable level of abstraction to provide robustness. By behavioural model we mean one which captures a sequence of actions and reflects this in some internal structure. We consider three types of such model, based on Recurrent Neural Networks, Markov Chains and Finite State Automata. Analysing the CTU-13 dataset, we show how these models can be learned from normal network traffic. We find that they outperform common classifier methods and give comparable results to standard botnet detection methods. Moreover, behavioural models indeed show promise for improving 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. Suspicious network traffic can be used to trigger alarms for intrusion detection systems, to discover new malware or botnet infections, or perhaps even detect an advanced persistent threat.

If benign network traffic has very regular patterns and malware has very different traffic also in a constant form, the problem is not too challenging. With labelled training data representing ground truth, we can train a classifier which separates 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. 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.

Our idea to deal with this is to find *behavioural* models which capture higher-level invariants of network patterns with internal structure. 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.

Previous work shows that robustness of classifiers can be improved by capturing high-level specifications of behaviour from the source code of malware [7, 8]. This motivates our terminology:

Definition 1 (Behavioural model). *A* behavioural model *is a model that abstracts the behaviour of a system in terms of sequences of actions reflected in an internal structure.*

The behavioural model in [7, 8] was represented as a finite state automaton, derived using static analysis of the source code, and used as additional features within classifiers. In this paper we address the more challenging problem of improving robustness of an intrusion detection system when neither ground truth nor malware source code is available. Focusing on normal network traffic alone, our hypothesis is that

> *behavioural models derived from normal network traffic can improve the robustness of anomaly-based intrusion detection.*

Here, we provide some evidence towards this claim, testing one side of the picture, where the malware changes over time. We do this by treating a previously studied dataset in some new ways and applying three kinds of behavioural model.

The CTU-13 Dataset. Stratosphere Lab, Prague [13] has made a laboratory generated dataset available to study botnet detection. It consists of labelled NetFlow records captured on lab machines for 13 different botnet attack scenarios,which are summarised in Fig. 1. This dataset was built in 2011 and consists of more than ten million NetFlow entries. Some example raw data are given below:

```
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
```

Each NetFlow in the list is labelled based on the source IP address. In the experiments, certain hosts are infected with a botnet and any traffic arising

	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		✓		✓			✓

Fig. 1. Scenarios in the CTU-13 dataset [13].

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.

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. This is described further in § 2. Fig. 2 gives an overview of the CTU-13 scenarios in terms of the duration of the capture, the number of flows, and the our view of sessions, segmented by infected and normal host. The different scenarios are discussed further in §4.

Previous work and our methodology. 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 [16] and in [27]. More recently, CTU-13 has been used to extract probabilistic real-time automata for anomaly detection [22] and to build Markov Chains to model botnet behaviours [14]. In contrast to these studies, which focused on modelling malicious behaviours, we train our behavioural models only using *normal traffic* to model *benign behaviour*. In particular, we use normal traffic in scenarios 3, 4 and 10 of CTU-13 as a training set, where only one botnet attack *Rbot* was deployed. We test the trained models on the remaining scenarios to demonstrate that normal behavioural models help anomaly detectors identify new botnets. To compare our behavioural models with classifier methods we

	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

Fig. 2. Size of scenarios and extracted sessions of CTU-13.

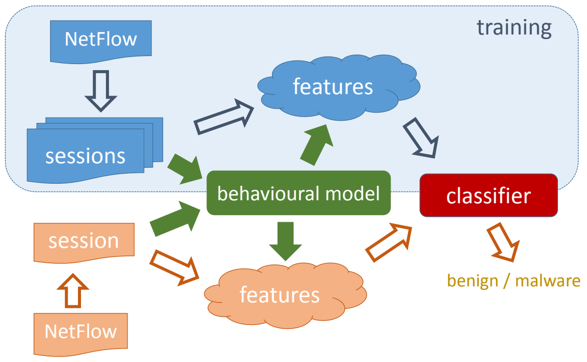


Fig. 3. Overall approach.

develop baseline experiments in §2 where classifiers are also trained on scenarios 3, 4 and 10 by taking common aggregates [20] as features for a session.

Fig. 3 illustrates our approach. We extract a behavioural model from the sessions in the training phase. We can compute a similarity score of a new session with respect to this model and use this score to detect anomalies. The baseline experiments correspond to omitting the (green) behavioural model from Fig. 3.

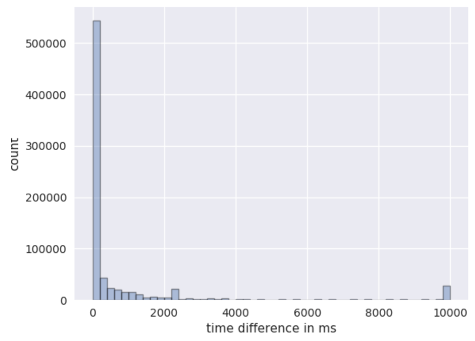
We train and evaluate three different type of behavioural models: a Recurrent Neural Network (RNN), a simple Markov Chain (MC) and a Finite State Automata (FSA). These are discussed in §3. Using three different model types helps explore our general hypothesis and make the results less susceptible to idiosyncrasies found in particular formalisms. As a secondary concern, we study the relative merits of the model types in this setting, in particular in terms of implicit models vs. explicit models (i.e., support for "explaining" the models) and simplicity vs. expressiveness of the models.

We evaluate our results in §4, discuss related work in §5, and conclude in §6.

Contributions. The main contributions of this paper are to show that (1) *it is possible to learn behavioural models from network traffic* and (2) that *these will improve robustness of anomaly detectors*, thus making an important first step towards validating our hypothesis. Additional contributions are: (3) we show novel applications of learning MC, FSA and RNN and (4) we introduce entirely new detection methods that give comparable results with existing botnet detectors [13] and outperform standard classifiers, even when they are trained using ground truths.

2 Experiment Setting

The raw input data, in the form of NetFlows, is first grouped by host and then organised into sessions based on their relative time difference. The relative time difference in the 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:

Definition 2 (Session). *If a NetFlow starts less than* 5 *seconds after the previous NetFlow for that host then they are in the same session; otherwise a new session is started.*

For each session, features are extracted and fed into the training process. Using the survey [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 try to determine *if a given session on a given host is benign or malicious*, the underlying aim of anomaly detection. But 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, NetFlow capture was 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.

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]. Classifiers are trained on normal and botnet traffic in scenarios 3, 4 and 10 of CTU-13. Note that our behavioural models from §3 are only trained on normal traffic in these scenarios. We test on the remaining scenarios.

All the three classifiers are shown to be reliable methods to detect malicious hosts, with precision approaching 100% and recall in the low to middle 80%. The computed F_β score, which favours precision, is in the low to middle 90% for all of them. Details are given in §4, after describing the behavioural models next.

3 Learning Behavioural Models from Network Traffic

In this section we will describe the behavioural models and how they are learned; §4 gives details on how they are used. We will introduce three different models: Recurrent Neural Networks (RNN), Markov Chains (MC) and Finite State Automata (FSA). Each of them have different characteristics and benefits. Common to all 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 over-fitted 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. In this setting, the NetFlows shown in §1 become the following sequence of actions used to train a behavioural model:

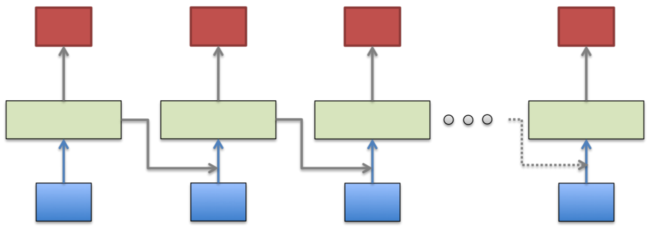
```
tcp; tcp; tcp; udp; udp
```

We refined this for the RNN model, by combining the protocol with port numbers, where infrequent destination port numbers are categorised. Initial experiments with using port number with FSA/MC 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.

3.1 Learning Recurrent Neural Networks

Recurrent Neural Networks (RNNs) are powerful models used in different domains such as signal processing applications, speech processing applications or language inductions [19]. RNNs can take random length sequences of vector inputs, providing much more flexibility than multi-layer perceptions or convolutional neural networks, which only accept fixed-sized vectors as input.

The following diagram illustrates how an RNN works for an input sequence:



When an input sequence of vectors is fed into a trained RNN, the elements of the sequence, which are depicted in blue, are processed through the network one at a time. For each "step", the RNN combines the input vector of the sequence with the current state vector. This is achieved by an activation function, depicted in green, which produces the new state vector. This will be carried through so that it can be combined with the next vector of the input sequence. In the current problem, the RNN will also predict the next element of the sequence, depicted in red, which is used to detect sessions that do not follow a normal behaviour.

As with any neural network, cells of the hidden layer corresponding to the new state vector are activated with a function ϱ of a linear combination of the activations of the network (h_j) in the previous instance of the input sequence, and inputs (x_k) at the current instant of the sequence:

$$h_i^{t+1} = \varrho\left(\sum_{j=1}^{N} a_{ij} h_j^t + \sum_{k=1}^{M} b_{ik} x_k^{t+1} + c_i\right), \quad i = 1, \dots N$$

The sigmoid function $\sigma(x) = \frac{1}{1+e^{-x}}$ is an example of an activation function. Any RNN with the activation function ϱ satisfying the properties

$$\lim_{x \to +\infty} \varrho(x) = t_+ \qquad \text{and} \qquad \lim_{x \to -\infty} \varrho(x) = t_-,$$

where t_+ and t_- exist for $t_+ \neq t_-$, can simulate a finite automaton [24]. The sigmoid function satisfies the above property. We use this result to build a RNN that is able to capture behaviour in the experimental data.

We follow the approach given in [25] to create and train a RNN. The overall architecture is shown in Fig. 4. This corresponds to a single step of the above

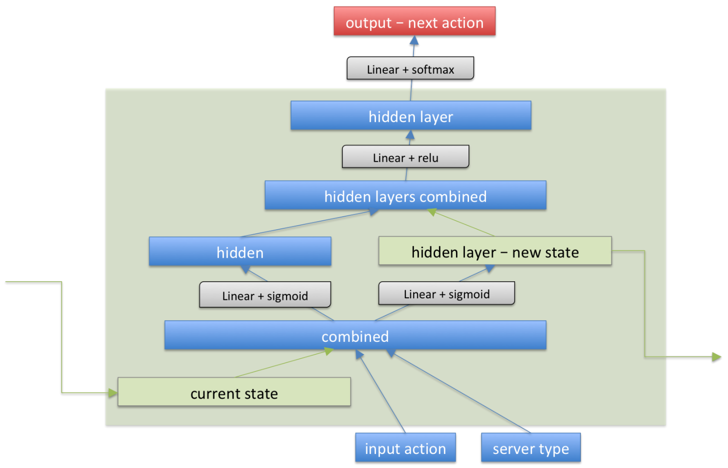


Fig. 4. Architecture of trained RNN [25].

diagram, with the green box "unfolded". It is represented as a directed graph that consists of a network of cells distributed over several layers.

The input sequence is a session, which is a sequence of actions as explained above. In addition, each host is categorised and the category of the host in question is given. There are four types of hosts: DNS servers, web servers, Matlab servers and normal servers. Actions of a session are fed into the network one at time, and an action corresponds to a blue box at the bottom of the diagrams.

The values of the activation cells of the hidden layer in the network are carried forward to be used with the next action of the session. The output of the network for each element of the input sequence is a vector of probabilities of the next action in the sequence.

Each grey box in Fig. 4 illustrates the operations performed between layers. These are activation functions[5] applied on a linear combination $\theta^l_{i0} + \theta^l_{i1} x^l_1 + \ldots + \theta^l_{im} x^l_m$ of the elements $x^l_1, \ldots, x^l_m$ of the l-th layer. Training the RNN requires finding the values of the set of parameters θ^l_{ij} that minimise the cost function evaluated on the training set. Since for each input vector of a session the RNN returns the most likely action that will follow out of a finite set of possible actions, we use as cost function the negative log-likelihood of the training set assuming it is drawn from a population following a multinomial distribution. The values of the parameters θ^l_{ij} that minimise this cost function will therefore be the most likely parameters of the population distribution from which our training set is assumed to be drawn.

Training is based on Stochastic Gradient Descent (SGD), a simple and efficient iterative approach to minimise a function. SGD randomly initialises the parameters θ^l_{ij}. In each iteration, the algorithm uses a mini-batch of the training set to update the values of θ^l_{ij} by taking a step towards the minimum of the cost function. The size of the steps are controlled by the learning rate. We use a

[5] This is either the sigmoid $\sigma(x)$, the linear rectifier $\mathrm{relu}(x) = \max(0, x)$ or the softmax used to obtain a multinomial probability distribution.

learning rate of 0.001, which is decayed by a factor of 2 each time that SGD does a pass through all the training set (also called *training epoch*). We also use a weight decay 5×10^{-4} to avoid overfitting, which occurs when the trained model not only fits the training set, but also captures the random noise in it, and hence is unable to generalise to new unseen sessions.

After training, when a new session is fed to the network, the output is a vector of probabilities. Each probability determines the probability of the various actions in the session given the previous actions. These probabilities are aggregated into a single score that will beis used to assess if the new session is normal or malicious.

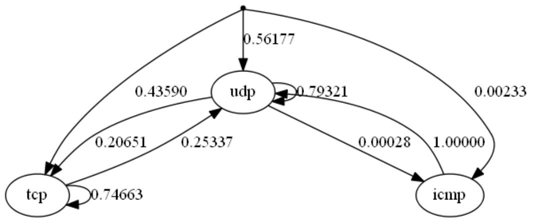
3.2 Learning Simple Markov Chains

While RNNs are very powerful, it is well known that one cannot explain the models so derived. Model intreptetation is useful in cyber anomaly detection, for example to explain why an alarm is raised to an operator in a Security Operations Center, or to help construct rules for an IDS automatically. To exmp also train two examples of *explicit* behaviour models.

The first and simplest explicit model is a simple Markov Chain (MC) which ignores the overall structure of a session, only considering 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 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 transition for the given state. The following MC captures one particular host in the training set:



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 of the type of action the next NetFlow is. For example, the probability that a TCP NetFlow is followed by another TCP is almost 75%. The unlabeled 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.

3.3 Learning Finite State Automata

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 learn a richer Finite State Automata (FSA) 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 [9]. 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 [12], 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, 26, 12]. 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. While 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 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.

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, 12].

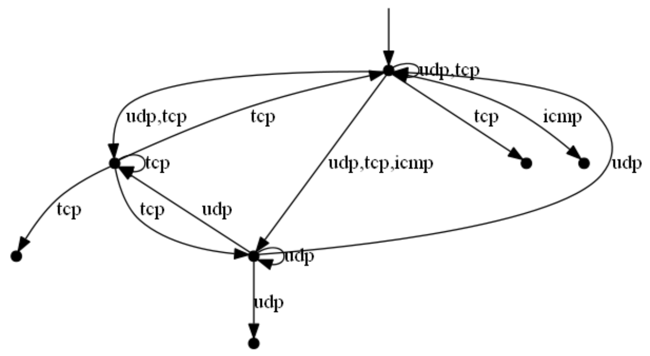
We follow a similar approach to [26] to compute a similarity measure to support (*ii*). This is achieved by comparing the similarity of outgoing transitions, increasing the measure for each edge labeled by the same action. This is applied recursively for the target states up to a given threshold. In our experiment, this threshold has been set to 3 steps. In addition, we subtract any disjoint transition thus reducing the similarity (this adjustment was not discussed in [26]). We can then set a threshold for this measure to decide how aggressive merging should be.

There several ways to configure the algorithm: choosing the number of steps of look-ahead, the treatment of disjoint transitions, and the threshold for the

[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.

overall score. In our experiments in §4 we looked 3 recursive steps ahead, reduced the similarity measure in the presence of disjoint transitions. We set a threshold of 0, 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 a valid for the automata. A distance 0 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 0 and 1.

These measures can be used directly, or as discussed next, as features for other anomaly detectors.

3.4 Integrating Behavioural Models with Aggregate Features

Besides using similarity measures from behavioural models directly as classifiers, we can use them as input features in standard machine learning algorithms. Because RNN uses slightly richer features than FSA and MC, we use the similarity measures as input together with aggregated features discussed in §2. This enables anomaly detection using *one-class SVM* [23], with only normal traffic used for training.

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.

4 Evaluation

We compare the baseline approaches described in §2 with the detectors using behavioural models described in §3, as well as analysing the relative merits of each behavioural model. First we briefly discuss the four phases of the evaluation: data preprocessing; training; prediction; and evaluation.

Data Preprocessing. For each host, the sessions are extracted as described in §2. The dataset is then split into a training set and a test set. The training set only includes scenarios 3, 4, and 10, where just one malware (*Rbot*) was deployed. The test set includes the 10 remaining scenarios. More details of each scenario can be found in Fig. 1. This setting aims to validate that behavioural models trained on benign sessions can generalise better to 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 (see §3) for both the training and test set. This is used to train and evaluate the behavioural models. For the SVM, Random Forest and Multilayer Perceptron models, the aggregated features per session is extracted for both sets.

Training. We only use normal hosts to train the behavioural models (RNN, FSA and MC), 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.

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

- The supervised machine learning models (SVM, RF and MLP) provide a probability (log-probability for SVM) of the session being malicious or not.
- RNN provides an aggregate probability of the NetFlows of a session taking into account the previous sequence of NetFlows within the session.
- 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.

	AUC	**F$_\beta$**	**Prec.**	**Recall**
RNN	0.979	0.957	0.964	0.949
FSA	0.965	0.918	0.937	0.867
MC	0.549	0.699	0.874	0.450
1CS*	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

*1CS = one-class SVM
+ features from MC
+ features from FSA

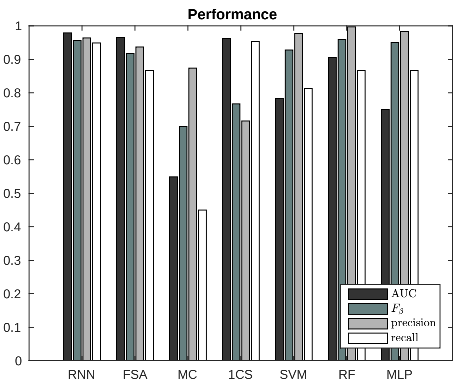


Fig. 5. Overall results.

Evaluation. Our goal is to correctly classify malicious hosts. Thus, a true positive (TP) is a 'correctly classified infected host', and 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 do have a high robustness at the expense of precision. We therefore choose $\beta = 0.6$ to avoid a high number of false positives.

4.1 Results of the experiments

We have performed classic anomaly detection by only training our behavioural models on benign behaviour, whilst the baseline classifiers were trained on labelled data. The overall performances are depicted in Fig. 5. The left hand side

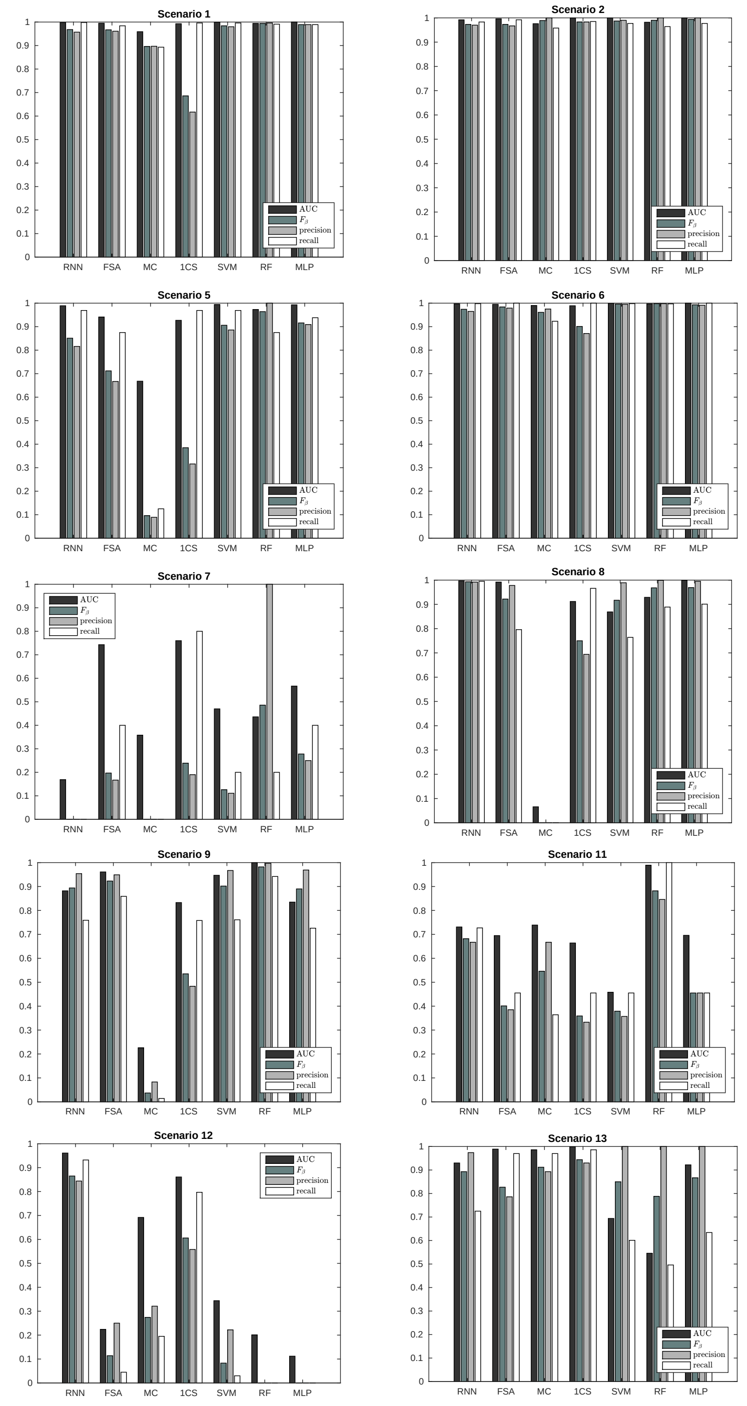


Fig. 6. Results for each scenario in the test set.

shows the respective percentages using the four classification metrics used, with the behavioural models above the line and the baseline classifiers below the line. This is shown graphically on the right hand side of the figure.

As the results show, all approaches, with the exception of the Markov Chain, are reliable methods to detect malicious sessions. The reason for the poor MC result is most likely that the MC features are too simple, and indicates that one either need more structure of the composition of NetFlows (as in RNN and 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 haven't investigated the degree to which the improvement is due to the MC.

The RNN had the best overall performance. The recall is increased by 8.2% to 94.9% with respect to the best classifier in the baseline experiment. With respect to robustness, recall is the most important measure as it identifies the degree in 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 both the FSA and the RNN in terms of precision, with a high number of false positives. This might be improved by using a search algorithm to tune the different hyperparameters of the model. RNN performs better than all classifiers for the AUC score, while the F_β 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 behavioural models. One reason may be that the ground truth used in training has more impact for precision.

The RNN has a better recall than the FSA, while the precision and AUC are similar. One reason for this may be that the RNN uses more detailed actions, which are needed to be able to detect new malware. Another reason may be due to over-generalisation of the learned FSA as configuring the correct level of similarity in order to merge states remains future work.

Fig. 6 breaks down the performance for each scenario in the test data. The training was conducted on only one specific type of malware behaviour; unsurprisingly, the classifiers have a lower recallas they are unable to detect other particular type of malicious behaviour. This is especially observed in scenarios 8, 12 and 13. In these scenarios the implemented classification models are vulnerable, being unable to detect malicious behaviour with the current features. FSA, and particularly the RNN, show that using simple actions such as protocol (FSA) or protocol and port (RNN) outperform the studied standard binary classifiers. They take advantage of observing the order of actions within a session, in contrast to the standard classifiers that make their predictions based on aggregated measures. Scenarios 5 and 7 where RNN and FSA have a worse performance than the binary classifiers are small in terms of botnet traffic (see Fig. 2) and therefore are not significant for the overall results.

Caveats with ground truth. To put this evaluation in context, it is important to note that these models were trained on a dataset that was the result of an experiment, where a set of hosts were infected by different malwares in different scenarios and network behaviour for different attacks was captured. As already discussed, the hosts were not necessarily infected at the beginning of the scenario and all network traffic from an infected host may not be driven by the malware. These bring two sources of discrepancy between the botnet labels in the data and whether the traffic is really malicious or not.

A clear example of this discrepancy is given in Scenario 9, where the hosts are not infected at the same time in order to mimic a real infection, even though during the whole experiment their traffic is labelled as *Botnet*. Furthermore, for scenarios 5 and 7 there is no direct evidence in the log files that an actual attack was conducted even if the hosts were infected, or for scenario 11 the attack only took 2 minutes of the whole scenario. Conversely, given the nature of the experiment, we can be sure that data labelled as *Normal* was benign behaviour. The behavioural models that we propose, RNN and FSA, are designed to be only trained on the normal traffic that we know is non-malicious. Therefore, some of the false negatives that we see in some test scenarios for RNN and FSA, meaning a recall lower than 90%, need further investigation.

On the other hand, in Fig. 6, we see that Random Forest gets high recall in scenarios where according to the experiments there is no direct evidence of attack or the attack took a small fraction of the whole scenario such as 7 or 11 (which have a recall of 1). The Random Forest was trained on a ground truth that in some cases does not correspond to reality, and with a training data that included a limited number of hosts. Perhaps it has actually overfitted the training set, learning the behaviour of each particular host in the training set (and therefore in the whole dataset) rather than discriminating between malicious and benign behaviour. The low recall of the RNN on scenario 11 is another suggestions that RNN may be able to correct the cases where the ground truth is wrong, classifying a host as infected only when this host starts showing a malicious behaviour.

Raising alarms early. Besides their performance, another advantage of the RNN and FSA models with respect to the other methods is that they do 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, RNN and FSA take 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 DoS, with long sessions and high number of superfluous requests to a targeted host.

Runtimes. We have not focused on the runtime of our model constructions, but give some indicative numbers for comparison. 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. Training the MC is a simple iteration over the data, which is of linear cost. The RNN took around 4 hours to train, where each epoch took around 20 minutes. Training the FSA depends on the size of the initial PTA, and varied from less than a minute to around 30 minutes. Computing the edit distance (FSA) and probabilities (MC) is instantaneous, while the output from the RNN took on average 1 second to measure a session.

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

5 Related Work

Our work falls into the general category of *collective anomaly detection* [6]. As far as we are aware, we are the first to learn behavioural models from normal network traffic to detect malicious activities. BotHunter [16] is built on abnormal behavioural models and complex anomaly detection rules. This deviates from our work as their model has to be provided and maintained, whilst we can automatically infer it. Moreover, their models capture abnormal behaviour while our approach capture normal behaviour. 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. As our models are built on network analysis rather than content analysis they will not be effected by content encryption such as TLS.

The CTU-13 dataset has been used to assess multiple botnet detection systems [13]. As we are using only normal traffic, sessions and anomaly detection, a direct comparison with this work is not possible. However, our evaluation on CTU-13 shows that automatically generated behavioural models can achieve similar precision and recall results to standard anomaly-based detectors and outperform classification methods in terms of robustness.

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 [15]. Models of normal or abnormal behaviour can be trained by exploring patterns between these features [16, 27]. Detection is usually by a threshold function on the deviation from, or the similarity to, these trained models. With the exception of one-class SVM, we only use behaviour and not aggregated features. However, for comparison we tried to follow these approaches with aggregate features in our baseline experiments. We obtain some better robustness results for behavioural models even though we only use normal traffic. Another advantage of the behavioural models is that they can flag an error on a partial session and do not require a complete session, as is the case when aggregates have to be computed.

In the BASTA system [22], a timed automata of malware behaviour is learned and used to detect anomalous behaviour. As for our work, it achieves a high recall on the CTU-13 data, albeit using a larger training set compared with ours. This indicates that behavoural models of both benign and malicious patterns show promise in detecting new malware.

Automata learning has been applied to learn botnet C&C protocols for botnets [9], and to learn behaviour from logs [10]. Learning of rules and signatures has been studied in the context of malware detection [5], however this general line does not fit with our notion of behavioural models, so is hard to compare.

RNNs have been trained and used in many applications [19]; our approach directly follows [25]. A related application to ours by Bontemps et al [4], uses a Long Short-Term Memory RNN to detect collective anomalies of multiple steps; like our approach, the crucial part is the composition of events. Bontemps et al also get promising results, but they evaluate on an older dataset no longer considered relevant (see [13]). Cordero et al [11] use a Replicator Neural Net for anomaly intrusion detection. This can be seen as finding a compression of benign traffic using aggregate features, and does not build a behavioural model.

6 Conclusions

We have demonstrated three different kinds of behavioural models learned from network traffic, and posed a challenge to show robustness of such models by considering detection against different botnet scenarios. The more expressive behavioural models (RNN and FSA) outperformed the simple MC model, which suggests that the structure/composition of NetFlows is important to detect malware. The RNN outperforms the FSA, but it has the advantage that it uses richer input features, taking port information into account. Also, we note that a one-class SVM combining aggregate features with similarity measures from FSAs and MCs has better recall than the RNN.

Our approach to generating 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, better group relevant ones, and measure which information from a NetFlow record is most important in an action. It would also be interesting to extend the FSA with state (as in [26]), which might allow quicker detection of DoS attacks.

For these experiments and others we would like to do, further modern curated data sources would be highly valuable. We need these both to check that findings are transferrable, and to use finer grained ground truth to measure the accuracy of identifying malicious sessions rather than malicious hosts.

Behaviour models abstract *how* a benign system behaves. A high-level specification captures *what* a benign system does and should thus in principle be able to reduce the false positive rate by capturing behaviours that “do the right thing but in a different way”. This is our original intuition and in previous work, we extracted specifications from code. We aim to carry forward our research programme by examining ways to learn such specifications.

References

1. Angluin, D.: Learning regular sets from queries and counterexamples. Information and Computation 75(2), 87–106 (Nov 1987)
2. Biermann, A.W., Feldman, J.A.: On the Synthesis of Finite-State Machines from Samples of Their Behavior. IEEE Transactions on Computers C-21(6), 592–597 (Jun 1972)
3. Bishop, C.M.: Pattern Recognition and Machine Learning. Springer (2006)

4. Bontemps, L., Cao, V.L., McDermott, J., Le-Khac, N.A.: Collective anomaly detection based on long short-term memory recurrent neural networks. In: Future Data and Security Engineering. pp. 141–152. Springer (2016)
5. Caselli, M., Zambon, E., Amann, J., Sommer, R., Kargl, F.: Specification mining for intrusion detection in networked control systems. In: 25th USENIX Security Symposium (USENIX Security 16). pp. 791–806. USENIX Association (2016)
6. Chandola, V., Banerjee, A., Kumar, V.: Anomaly detection: A survey. ACM Comput. Surv. 41(3), 15:1–15:58 (Jul 2009)
7. Chen, W., Aspinall, D., Gordon, A.D., Sutton, C., Muttik, I.: More semantics more robust: Improving android malware classifiers. In: Proceedings of the 9th ACM Conference on Security & Privacy in Wireless and Mobile Networks. pp. 147–158. ACM (2016)
8. Chen, W., Aspinall, D., Gordon, A.D., Sutton, C., Muttik, I.: On robust malware classifiers by verifying unwanted behaviours. In: International Conference on Integrated Formal Methods. pp. 326–341. Springer (2016)
9. Cho, C.Y., Shin, E.C.R., Song, D., 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. pp. 426–439. ACM (2010)
10. Comparetti, P.M., Wondracek, G., Kruegel, C., Kirda, E.: Prospex: Protocol Specification Extraction. pp. 110–125. IEEE (May 2009)
11. Cordero, C.G., Hauke, S., Mühlhäuser, M., Fischer, M.: Analyzing flow-based anomaly intrusion detection using replicator neural networks. In: Privacy, Security and Trust (PST) 2016. pp. 317–324. IEEE (2016)
12. Damas, C., Lambeau, B., Dupont, P., van Lamsweerde, A.: Generating annotated behavior models from end-user scenarios. IEEE Transactions on Software Engineering 31(12), 1056–1073 (Dec 2005)
13. García, S., Grill, M., Stiborek, J., Zunino, A.: An empirical comparison of botnet detection methods. Computers & Security 45(Supplement C), 100–123 (Sep 2014)
14. García, S., Pechoucek, M.: Detecting the behavioral relationships of malware connections. In: Proceedings of the 1st International Workshop on AI for Privacy and Security. pp. 8:1–8:5. ACM (2016)
15. Gu, G., Perdisci, R., Zhang, J., Lee, W.: Botminer: Clustering analysis of network traffic for protocol- and structure-independent botnet detection. In: Proceedings of the 17th Conference on Security Symposium. pp. 139–154. USENIX (2008)
16. Gu, G., Porras, P., Yegneswaran, V., Fong, M., Lee, W.: Bothunter: Detecting malware infection through ids-driven dialog correlation. In: Proceedings of 16th USENIX Security Symposium. pp. 12:1–12:16. USENIX (2007)
17. Lakhina, A., Crovella, M., Diot, C.: Mining anomalies using traffic feature distributions. In: Proceedings of the 2005 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications. pp. 217–228. ACM (2005)
18. Lang, K.J., Pearlmutter, B.A., Price, R.A.: Results of the Abbadingo one DFA learning competition and a new evidence-driven state merging algorithm. In: Grammatical Inference, vol. 1433, pp. 1–12. Springer, Berlin, Heidelberg (1998)
19. Mandic, D.P., Chambers, J.A., et al.: Recurrent neural networks for prediction: learning algorithms, architectures and stability. Wiley Online Library (2001)
20. Miller, S., Busby-Earle, C.: The role of machine learning in botnet detection. In: 2016 11th International Conference for Internet Technology and Secured Transactions (ICITST). pp. 359–364 (Dec 2016)

21. Oncina, J., Garcia, P.: Inferring regular languages in polynomial update time. In: de la Blanca, N.P., Sanfeliu, A., Vidal, E. (eds.) Pattern Recognition and Image Analysis, Series in Machine Perception and Artificial Intelligence, vol. 1, pp. 49–61. World Scientific, Singapore (1992)
22. Pellegrino, G., Lin, Q., Hammerschmidt, C., Verwer, S.: Learning behavioral fingerprints from netflows using timed automata. In: 2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM). pp. 308–316 (May 2017)
23. Schölkopf, B., Platt, J.C., Shawe-Taylor, J.C., Smola, A.J., Williamson, R.C.: Estimating the support of a high-dimensional distribution. Neural Comput. 13(7), 1443–1471 (Jul 2001)
24. Siegelmann, H.T.: Recurrent neural networks and finite automata. Computational intelligence 12(4), 567–574 (1996)
25. Siegelmann, H., Sontag, E.: On the computational power of neural nets. Journal of Computer and System Sciences 50(1), 132 – 150 (1995)
26. Walkinshaw, N., Taylor, R., Derrick, J.: Inferring extended finite state machine models from software executions. Empirical Software Engineering 21(3), 811–853 (Jun 2016)
27. Wurzinger, P., Bilge, L., Holz, T., Goebel, J., Kruegel, C., Kirda, E.: Automatically generating models for botnet detection. In: Backes, M., Ning, P. (eds.) Computer Security – ESORICS 2009. pp. 232–249. Springer (2009)