Towards Traffic Anomaly Detection via Reinforcement Learning and Data Flow

Arturo Servin
Computer Science, University of York
aservin@cs.york.ac.uk

Abstract. Protection of computer networks against security attacks has been widely researched in the last years. Examples of these attacks are Flooding-Based Denial of Service (DoS) attacks and Flooding-Based Distributed Denial of Service (DDoS) attacks. This work presents our research results in the use of data flow information, Reinforcement Learning (RL) and tile coding to detect Flooding-Based Distributed Denial of Service Attacks. Using these approaches, RL agents analyse different parameters of the data flow between hosts and they learn to distinguish from legitimate and non-legitimate applications. We will show how the selection of characteristics taken from the data flow affects the accurate categorisation of normal and malicious activity and how different tile coding parameters such as the number of features affects the quality of the flow data generalisation. We will also show the advantages and limitations of our proposal.

1 Introduction

Intrusion Detection Systems (IDS) have been widely researched and implemented using a variety of techniques. Despite the advances in this field and the high accuracy to detect known intrusions by some of these systems, Intrusion Detection is still an area with many research opportunities. Open issues in Intrusion Detection are the automatic generation of signatures, intrusion response mechanisms, Distributed Denial of Service detection and containment and intrusion detection evaluation methodologies among others. Our main research is focused in the detection and containment of Flooding-Based Distributed Denial of Service Attacks. To fulfill this goal we have separated our research in two areas: Detection of Flooding-Based Distributed Denial of Service Attacks and Containment of Flooding-Based Denial of Service Attacks.

In previous work [1] we present research on the use of Reinforcement Learning (RL) along with a distributed architecture of agents to detect Distributed Denial of Service Attacks. In this paper we present our most recent work regarding the Detection of Flooding-Based Distributed Denial of Service Attacks. We have based the detection engine on the use of RL and data flow information on the network. Using this approach, RL agents analyse different parameters of the data flow between end hosts and they learn to distinguish whether the data flow belongs to a legitimate application or not.
This paper is divided into the following sections. The Intrusion Detection Systems and Reinforcement Learning sections give a brief technological and conceptual background. The section Towards traffic anomaly detection via Reinforcement Learning and data flow explains the rationale behind our proposal and it also explains in detail how our algorithm works. The section Experiments and Results describes and explains the set of experiments that we followed to test our proposal and its results. Finally we conclude with our general observations about the use of RL along with flow information and we point out to some future research in the area.

2 Intrusion Detection Systems

IDS are commonly separated in two functional categories: Anomaly Intrusion Detection and Misuse/Signature Intrusion Detection. Anomaly IDS states that intrusions are deviations from normal traffic. These systems create profiles of different variables or features over time to get a usage pattern. The difference between the pattern and current activity triggers an alarm. Anomaly Intrusion Detection Systems use a variety of schemes to detect normal user patterns from simple statistical to complex machine learning methods. These types of IDS are capable of detecting unknown attacks, however non-malicious activity that does not match normal behaviour can also trigger the intrusion mechanism. The result is that they usually have a high rate of false alarms. On the other hand misuse or signature intrusion detection systems use rule matching to detect intrusions. These systems compare system activity with specific intrusion rules that are generally hard coded. When the observed activity matches the intrusion pattern an intrusion is detected and an action is executed. Signature/Misuse IDS are very reliable and they generally have low rate of false positives. Nevertheless, they lack the ability to detect new types of attacks.

A Flooding-Base Distributed Denial of Service Attack is launched from several remote controlled sources trying to exhaust a target’s key resource. Generally the attack is a flood of UDP or ICMP packets but TCP could be used as well. Flooding-Base Distributed Denial of Service Attacks change the normal behaviour of the network in different ways. Spotting these differences could help us to detect the presence of attacks. Mirkovic et al [2] point out that the presence of spoofing source addresses, the low or non-response of the attacked host and the identification of an aggressive sending rate to the attacked host can lead to detection of a flooding attack. To identify these anomalies they observe data flow information. A variety of work on IDS have used flow information as the base to detect intrusion [2–5] among others. In general the approaches presented on those works present high detection rate and low false positive rates metrics. The drawback of these approaches is that they use fixed thresholds or values of the observed features. This added extra effort in the fine tuning of these systems make them complex to deploy in large environments or difficult to adapt to new applications.
3 Reinforcement Learning

In Reinforcement Learning agents or programs sense their environment in discrete time steps and they map those inputs to local state information. RL agents execute actions and observe the feedback from the environment or a trainer in the form of positive or negative rewards. After performing an action and receiving a reward, the agent observes any changes in the environment and it updates its policy in order to optimise the reward received for future actions [6]. The goal of the agent is to find the policy that maximises the rewards obtained over the time. This is called the optimal policy. One of the most widely used techniques to calculate the optimal policy is Q-learning [7]. In Q-learning as in other Temporal-Difference-Learning methods the agent iteratively tries to estimate the value function. The algorithm forms a Q table where rows are states and columns are actions. The agent in each state \( s \) chooses an action \( a \), observes the reward \( r \) and the next state \( s' \). Then it updates the estimated Q value denoted by \( \hat{Q} \) in (1). In this equation \( \alpha \) is the learning rate with a value \( 0 < \alpha < 1 \) and \( \gamma \) is a constant with value \( 0 < \gamma < 1 \) that represents the relative value of delayed versus immediate rewards.

\[
\hat{Q}(s,a) \leftarrow (1 - \alpha)\hat{Q}(s,a) + \alpha(r + \gamma \max_a \hat{Q}(s',a'))
\]  

(1)

In order to converge to the optimal policy Q-learning requires the agent to visit all the states an infinite number of times. To comply with this requirement and to discover which actions lead to the best rewards over time, the agent needs to explore and to exploit actions. When RL is used in real world applications it is not either practical or feasible to map all sensory information to individual states. To tackle this problem we use function approximation techniques. These techniques commonly used in supervised learning are adapted to RL in order to generalise the state representation. To learn which actions may lead to higher rewards the agent needs to estimate a value function for each action using a diversity of methods. One especially successful in real world applications is Tile Coding. Tile coding is a type of sparse coarse coding where the features or characteristics that we want to analyse are grouped in a set of partitions. These partitions form the input space and they are called tiling. Each tiling is divided into small pieces called tiles. To produce the state-action feature representation several tilings are overlapping. The value of the state-action \( V_{\pi}(s) \) is the sum of all the weights \( \overrightarrow{w} \) activated by the input.

4 Towards traffic anomaly detection via Reinforcement Learning and data flow

As we previously pointed, Flood-Based Denial of Service Attacks change the normal data flow of data in the attacked network. Analysing the changes on the features that are affected by these events can lead us to identify when the network is under attack. To detect those anomalies data flow information has been
used by different approaches intended to detect Flood-Based DoS and DDoS. These approaches can be very accurate to detect data pattern of abusive behaviour such as Flood-Based Denial of Service Attacks. The main drawback is that they usually required complex fine-tuning to provide the correct values of the features or thresholds monitored. To solve this problem we propose the use of a learning algorithm capable of interacting with the environment and learning when the flow data corresponds to a normal transmission of data or when it corresponds to an attack. We developed an agent that analyses flow data and triggers an action over the flow. There are different sets of actions that can we can choose to be executed. Examples are raising an alarm, let the traffic pass, throttle down the traffic or stop the traffic. To simplify our experiments in this stage of our research we limit the actions to only tag the flow as normal or abnormal.

Algorithm:

1. At time t=0 initialize $\vec{w}$ and $V_\pi(s)$ of each action
2. for i=0 to lastflow do; Agent analyse n flows at time t;
   
3. Get tiles according to number of features and $m$ divisions per tile
4. Sum weights $\vec{w}(t)$ to obtain the value function of the action
5. Follow greedy policy to select action
6. Receive reward $r$ Update $\vec{w}$ according to:
   
   $w_i = w_i + \alpha \left( \text{Target} - \sum \vec{w} \right)$  \hspace{1cm} (2)
   
   where:
   
   $\text{Target} = \sum \vec{w} + \alpha \left( \text{reward} - \sum \vec{w} \right)$  \hspace{1cm} (3)

7. Wait timestep seconds
8. Go back to 2 or continue if time = end
9. Store $V_\pi(s)$ of each action and $\vec{w}$

At the beginning of the algorithm the agent initialises the weight array $\vec{w}$ and the value function $V_\pi(s)$ of each action a to 0. For each flow in the agent, the agent takes the values of the $n$ monitored features. Some values come from the flow itself (e.g. protocol, port, average packet size) and some values come from the agent’s output interface of the flow. The features, the number of tiles and the divisions per tile (tilings) are processed by the subroutine Get_Tiles [9]. The output of the subroutine are the index of the weights $\vec{w}$ activated by the flow. Using this information the agent sums the activated weights in each action. Following a greedy policy (i.e no exploration, only exploitation) the agent selects the action with highest sum to execute. In our experiments we only used two actions. The first action is to send a communication signal tagged as Alarm when the flow is detected to belong to anomalous traffic patterns. The second action is to send a communication signal tagged as No-Alarm when the flow is detected to belong to a normal traffic pattern.
After triggering the communication signal, the agent receives a reward \( r \). The reward is positive if the action was correct, it is negative otherwise. Finally the agent updates the activated weights according to Eq. 2 and proceeds with the next flow. After all the flows are processed the agent waits \( \text{timestep} \) seconds and start sampling flows again. To finish it stores the values of the \( \overrightarrow{w} \) and the \( V\pi(s) \) of each action for future use.

5 Experiments and Results

A series of tests using the network simulator ns-2 [8], a specifically designed program and the Tile Coding Software [9] were run. The purpose was to find out whether the use of flow information along with RL and tile coding could enable an agent to learn to categorise normal and abnormal activity in the network. The network simulator was configured to use a simple network topology of 4 nodes as shown in Figure 1. Node 0 generates normal FTP-alike traffic while node 1 produces normal UDP traffic. Node 4 is an attacker producing a flood of UDP traffic. Node 2 is the Reinforcement Learning Agent (RLA) that must learn to differentiate the normal traffic from node 0 and 1 from the abnormal behaviour of node 4. Node 3 receives valid data from nodes 0 and 1 and it is the node under attack as well. The node parameters set up by this test are shown in Table 1

![Network Topology on Testing](image)

**Fig. 1.** Network Topology on Testing

**Table 1.** Node Parameters

<table>
<thead>
<tr>
<th>Node</th>
<th>Protocol</th>
<th>Port</th>
<th>PacketSize</th>
<th>Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Node 0</td>
<td>TCP</td>
<td>21 FTP</td>
<td>variable</td>
<td>variable</td>
</tr>
<tr>
<td>Node 1</td>
<td>UDP</td>
<td>5080</td>
<td>512 Bytes</td>
<td>1 Mbps</td>
</tr>
<tr>
<td>Node 4</td>
<td>UDP</td>
<td>1580</td>
<td>110 Bytes</td>
<td>16 Kbps</td>
</tr>
</tbody>
</table>
The test number one was our control test. It was used for the RLA for learning different characteristics of the flows generated by nodes 0, 1 and 4. We applied the learning algorithm and tile coding to this test using 2, 3 and 4 features or variables. If we had used a state-action table as the value function representation we would have needed 256, 4096 and 65536 states for a 2, 3 and 4 features respectively. This would have required thousands and even millions of iterations to learn, while with tile coding we only required a few hundred iterations. We used the numerical value of protocol and port as features 1 and 2, packet size as feature number 3 and the number of bytes transmitted by the flow between samples as feature number 4. We then set up 4 more tests (T2, T3, T4 and T5) to verify which features play an important in the learning process of the agent. These tests also helped us to identify the advantages and the limitations that our proposals have over other research work. A summary of the different tests is shown in table 2. For Test 2 we added more sources of FTP, UDP no-attack and UDP attack. We also changed the sending patterns of the applications. Using two, three and four features the RLA had no problem to identify and categorise the flows as shown in Figure 2. To simulate how a real world attacker will try to pass firewalls by changing information in their attacks in Test 3 we changed the port used by the attacker. As well as in the previous test the RLA had no problem to recognise the attack using three and four features. However with only using two features it was not possible to recognise the attack. In Test 4 we simulated how an attacker would try to masquerade its attack by using the same protocol and port of a valid application. For this test, the two-feature-learning-process was again unable to distinguish between the attack and the no-attack flows. The use of three and four features allowed the RLA to identify the attack throughout the packet size. This raises the question, what happens if the attacker has the ability to modify the application to use the same packet size, protocol and port than a valid application? We used Test 5 to simulate this scenario. Besides the features already used, we have the option to use bytes transmitted by the flow between samples, incoming interface of the flow and outgoing interface of the flow. We used bytes transmitted by the flow

\begin{table}[h]
\centering
\caption{Test Summary}
\begin{tabular}{|l|}
\hline
\textit{Test Changes} \\
\hline
Test 2 The layout and number of nodes in the simulation was changed. \\
Test 3 The attack port was changed to 6667. \\
Test 4 The attack port was changed to 1580. (same as the no attacker node) \\
Test 5 The attack port and packet size were changed to 1580 and 110 bytes respectively. \\
\hline
\end{tabular}
\end{table}
between samples as the fourth feature because we think that incoming/outgoing interface information is transitive data that can change even between slightly different network topologies. Unfortunately, even with 4 features (Protocol, port, packet size and bytes transmitted) the RLA could not discriminate between attack and no-attack flows as shown in the results of Test 5. We verified this in a separate test with three features. In this special test we use the feature "bytes transmitted" instead of "packet size". In this test the agent was not able to recognise attacks. We then conclude that "bytes transmitted" as feature does not provide the information needed to categorise events. Nevertheless we proved our point that feature selection plays an important role in function approximation techniques. It is important to point that this does not mean that flow information is not useful to detect attacks or that tile coding and RL are not suitable solutions either. It just exemplifies that as any other approach these techniques are not one hundred percent dependable for all sort of scenarios.

6 Conclusions

We have performed tests that have shown how function approximation in the form of tile coding could offer a suitable solution to apply RL in real world
scenarios by providing a high resolution and multivariable data without state-space explosion. The use of flow information, RL and tile coding has yielded some positive results along with some limitations in our specific domain of traffic anomaly detection. We showed that the RLA learned how to categorise flows of known applications. It also learned how to generalise some features to identify unknown activity. On the other hand it cannot categorise flows under complex attack scenarios.

In order to enhance the detection capabilities of our RL agents and to achieve our final goal of detecting and containing flooding-based DoS and DDoS we plan to add more capabilities to our approach. For instance we can analyse aggregate flow information to detect DDoS. In addition flow information is stateless. Adding stateful capabilities may add the possibility to include information about the bidirectional flow of data. Interesting information to follow is the round trip time of the flow, connection creation/termination (e.g. TCP three way handshake, SIP signalling). Finally, if the information required to increase the agent detection rate is not possible to be integrated in just one agent, a multi-agent system approach will be required.

References