BotGM: Unsupervised Graph Mining to Detect Botnets in Traffic Flows
Sofiane Lagraa, Jerome Francois, Abdelkader Lahmadi, Marine Minier, Christian Hammerschmidt, Radu State

To cite this version:

HAL Id: hal-01636480
https://inria.hal.science/hal-01636480
Submitted on 16 Nov 2017

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
BotGM: Unsupervised Graph Mining to Detect Botnets in Traffic Flows

Sofiane Lagraa*, Jérôme François†, Abdelkader Lahmadi‡, Marine Miner‡, Christian Hammerschmidt‡ and Radu State‡

*Inria Nancy Grand Est, France, email firstname.lastname@inria.fr
†SnT, University of Luxembourg, Luxembourg email: firstname.lastname@uni.lu
‡Université de Lorraine, France, firstname.lastname@loria.fr

Abstract—Botnets are one of the most dangerous and serious cybersecurity threats since they are a major vector of large-scale attack campaigns such as phishing, distributed denial-of-service (DDoS) attacks, trojans, spams, etc. A large body of research has been accomplished on botnet detection, but recent security incidents show that there are still several challenges remaining to be addressed, such as the ability to develop detectors which can cope with new types of botnets. In this paper, we propose BotGM, a new approach to detect botnet activities based on behavioral analysis of network traffic flow. BotGM identifies network traffic behavior using graph-based mining techniques to detect botnets behaviors and model the dependencies among flows to trace-back the root causes then. We applied BotGM on a publicly available large dataset of Botnet network flows, where it detects various botnet behaviors with a high accuracy without any prior knowledge of them.

I. INTRODUCTION

Botnet detection and tracking has been a major research topic in the last decade in the area of network security with numerous surveys revealing the large range of techniques to track and mitigate them [1], [2]. The evolving of botnets, and the inability of traditional protection mechanisms to stop them, has led security researchers and practitioners to develop behavior-based techniques [3]–[6]. Unlike statistical or signature-based techniques [7], [8] used for years to detect the presence of known bots, behavior-based techniques observe the actions of potential bots and attempt to match them against pre-defined specifications of bots’ behaviors, or try to assess how they differ from the normal behavior exhibited by other hosts. However, the continuously apparition of new botnet propagation and command-and-control (C&C) mechanisms shows the limitation of those detection techniques, which cannot cope with unknown and more and more complex behaviors.

Rather than analyzing individual host behavior, our approach, named BotGM, considers multiple host behaviors to represent dependencies among their flows which can be then traced back in a comprehensive manner to investigate root causes of the detected abnormal behaviors.

Our method relies on NetFlow because it is widely used in security and monitoring of computer networks [9], [10]. On one hand, it is considered privacy-preserving and scalable due to the little information which has to be collected and stored in comparison with full-packet capture. On the other hand, a single flow information is rather limited, only providing protocol, source and destination IP addresses, source and destination ports, number of packets, number of bytes, timestamp and duration. In order to work with this limited information, we used a specific modeling method to reveal meaningful knowledge, especially by merging multiple flow records in a unique representation. We propose a new graph-based behavioral model which represents causal relations among flows by automatically extracting frequent time-dependent relations among similar ones. Using this model, efficient detection is applied to extract malicious activities, especially those exhibited by botnets. Instead of simply raising alerts, BotGM identifies particular graphs whose the structure also eases the root cause analysis.

This paper has the following major contributions:

1) We develop a graph-based models of NetFlow records which, instead of focusing on spotting interactions among hosts, focuses on modeling dependencies among flows. The generic model can be finely tuned.
2) We apply an unsupervised graph mining approach to track botnets with the ability to trace root causes. Hence, no a-priori knowledge is assumed about botnet communication models.
3) We instantiate and evaluate our method on a recent dataset and assess its ability to detect individually infected hosts as well as groups of them. We rely on outlier detection and clustering techniques based off our models. We also discuss the impact of model parameter settings.
4) We compare our method detection accuracy with existing botnet detection techniques.

The rest of the paper is organized as follows. Section II reviews and discusses related works. We detail our mining approach in Section III. We then present the experimental results on real botnet datasets in Section IV. Section V concludes and gives some future research directions.

II. RELATED WORK

Unlike our method which relies on modeling dependencies among flows, statistical analysis or machine learning algorithms have been mainly applied on features available in flow
Our approach does not consider any particular type of botnet related traffic, the analysis of a specific traffic such as DNS activity has been investigated in several works. In particular, by detecting co-occurrence of DNS requests from numerous hosts [13], by tracking algorithmically generated domain names [14] or by using signal processing techniques over a large-scale and distributed dataset [15] for example. Botnets can be also detected by correlating other types of attacks (spam, denial-of-service, etc.) which can have been triggered from infected hosts as highlighted in BotHunter [16]. In [3], the authors employed fuzzy pattern recognition and combined DNS and TCP flow analysis by considering packet counts and packet sizes.

Graph-based methods have been used in many papers either to model connectivity among hosts in the network [9], [17], among flows [18], [19] or services [20], [21]. In [22], the authors proposed a two steps approach which first detects potential malicious traffic flows with methods mentioned before and then correlate those ones in order to build a graph whose topology is finally analyzed. In [23], the authors proposed malware classification method based on maximal common subgraph detection. Their behavioral graph is obtained by collecting system call traces. However, their approach differs from ours since we are focusing on the analysis of passively collected network flows. Recently, HERCULE [24] models heterogeneous log entries as vertices in the graph and generates connections based on defined correlation rules. The authors then applied graph analytics techniques to detect attack communities inside these graphs. Similarly, the authors in [25], mine network log data (e.g., syslog). They reconstruct causality of network events from a set of the time series of flow records which can be compared to track abnormal activities and their root causes. As a preliminary step, flow records are divided into time windows. The size of time window is controlled through a parameter $\omega$.

Then, the process consists of six components as illustrated in Figure 1.

1) **Event extraction.** To build causal models, NetFlow records are considered as events. Each individual flow is transformed into an attribute-based event where each attribute is directly derived from standard NetFlow features such as IP addresses or ports. In addition, selected attributes are divided in two categories: keys and values. Fixing a key, values are then used for building a graph-based fingerprint in the next processing steps. In the example of Figure 1, keys are the source and destination IP addresses while respective ports are the values.

2) **Sequence extraction.** Sets of sequences of states from events are extracted to model the full behavior of hosts. Events sharing the same key, such as source and destination IP address, are used for building individual sequences whose states are represented by the value attributes. In Figure 1, sequences of pairs of source and destination ports are built for each key (the couple source and destination IP address).

3) **Graph construction.** This step allows us to construct a graph from each sequence. This reduces the behavioral representation from a long sequence to a graph that aggregates redundant patterns while still keeping all existing interactions. It is a primordial step for scalability. As shown in Figure 1, the first sequence has been reduced to three vertices.

4) **Outlier detection.** Through pair-wise comparisons among graphs, the outliers are extracted. It is thus possible to track back to the associated keys. For example in Figure 1, the couple of source and destination IP addresses (147.32.84.138, 147.32.80.9) represents the outlier graph $G_1$. As a result, the communication flows between these hosts are considered as abnormal and e.g labeled as botnet communication.

### III. System Design

#### A. Overview

Our goal is to establish probable causal relationships among flow records which can be compared to track abnormal activities and their root causes. As a preliminary step, flow records are divided into time windows. The size of time window is controlled through a parameter $\omega$.

#### B. Event extraction

Let $F = f_0, \ldots, f_N$ be a set of flows where each $f_i$ is described by a set of attributes as illustrated in Table I. Each flow will be transformed into an event assuming a set
Table I: NetFlow attributes.

<table>
<thead>
<tr>
<th>NetFlow attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ts</td>
<td>Timestamp</td>
</tr>
<tr>
<td>dur</td>
<td>Flow duration</td>
</tr>
<tr>
<td>proto</td>
<td>Protocol</td>
</tr>
<tr>
<td>sip</td>
<td>Source IP Address</td>
</tr>
<tr>
<td>sport</td>
<td>Source Port</td>
</tr>
<tr>
<td>dip</td>
<td>Destination IP Address</td>
</tr>
<tr>
<td>dport</td>
<td>Destination Port</td>
</tr>
<tr>
<td>tos</td>
<td>Type of Service</td>
</tr>
<tr>
<td>totpkts</td>
<td>Number of packets</td>
</tr>
<tr>
<td>totbytes</td>
<td>Amount of bytes</td>
</tr>
</tbody>
</table>

**Definition**

For each flow $f_i$, the event $e_i$ is defined as $e_i = (t_{si}, s_{i}, v_{i})$ consists of the same timestamp $t_{si}$ of the flow $f_i$; and a compound key $s_{i}$ and value $v_{i}$ composed respectively from the set of attributes $K$ and $V$ of the same flow $f_i$.

As an example, having the flow:

$$f_0 = \{11:09:44, 0.42243, udp, 147.32.84.165, 1025, 1025.32.80.9, 53, 2, 182, 71\}$$

With $K=\{\text{sip, dip}\}$ and $V=\{\text{sport, dport}\}$, We extract an event $e_0$ as follows:

$$e_0 = \{147.32.84.165, 147.32.80.9\}, \{1025, 53\}$$

**C. Sequence extraction**

After the extraction of events, dependencies among events are built based on keys and timestamps to identify successive events which are part of some global action (e.g., an attacker performing multiple steps of an attack). To transform the set of events into sequences, events are first grouped using shared key attributes. For example, all events (i.e., events) with the same source and destination IP addresses are merged into a single sequence as previously illustrated in Figure 1. This sequence...
representation allows to characterize the order of events of a specific IP address connecting to a destination IP address.

**Definition** Assuming \( E = \{e_0, \ldots, e_n\} \) the set of events of the considered time windows, a unique sequence \( S_i \) is built for each unique \( k_i \), \( k_i \in \bigcup_{t_j \in E} k_j \) such that \( S_i = [v_{i0}^k, \ldots, v_{in}^k] \) where:

\[
\forall (e_i = v_k^i \in S_i, e_u = v_l^i \in S_i) k_i = k_u
\]

The first condition guarantees that a sequence is built from extracted events. The second one guarantees that all events of a sequence shared the same key and the last one ensure that they are ordered by time using their respective timestamp attribute \( ts \). It is worth mentioning that events sharing both keys and values will appear individually in the sequence.

**D. Construction of behavioral graphs**

A behavioral graph is constructed from a sequence \( S_i \), i.e. originally extracted from flow records sharing the same user-defined keys \( k_u \) that occur in a given time window. It is a directed graph that represents successive relationships between events of an event sequence. Therefore, each unique element of a sequence appears once while it can be observed multiple times in the sequence. Thus, a cycle in the graph indicates successive repetitive events.

**Definition** Assuming a given sequence \( S_i = [v_{i0}^0, \ldots, v_{im}^m] \), a behavioral graph over a set of event values \( V \) is a labeled directed graph \( G = (V_G, E_G) \):

- The vertex \( V_G \) is the set of vertex of unique events in the sequence \( S_i \) sharing the same values

\[
V = v_k^i \{i=1 \ldots m\}
\]

- The edges \( E_G \) are derived from dependency rules. Let \( v_l^i \) and \( v_k^i \) two vertices of \( G \), an edge exists, i.e. \( (v_l^i, v_k^i) \in E \), if and only if those events appear consecutively at least once in \( S_i \):

\[
\exists v_k^i \in S_i, \exists v_l^i \in S_i, k+1 = l
\]

By compressing sequence information, behavioral graphs ease interpretation of long sequences of events. Intuitively, considering a particular time window and \( K \) (as an example source IP address), an edge between two vertices assumes a causal dependency and a path between two nodes represents a causal path among events where each of them is representative of similar events (same values but at different time). Since the flows of an host or an application can be intertwined, our built causal dependency may seem unadapted. However, a good tuning of the key \( K \) to be used allows to reduce the flows used when building such relations and so limit this effect. In addition, BotGM leverages a graph structural analysis based on the edit distance (detailed in the next section) which actually helps to reduce the impact of noise, i.e. incorrect dependencies.

**E. Graph analysis**

BotGM constructs and maintains a behavioral graph for each unique keys (e.g. source and destination IP addresses in the previous example). The challenge is to mine the graphs in each time period in order to detect, find and quantify suspicious behavior which deviated to another behavior.

1) **Graph edit distance:** Graph edit distance (GED) is a flexible fault-tolerant similarity measure for graphs. The fault-tolerant similarity means that the measure accounts for noise, i.e. structural deformations. Exact approaches (subgraph isomorphism) may not be adopted. The edit distance between two graphs \( g_1 \) and \( g_2 \) is defined by the minimum set of edit operations that are necessary to transform the graph \( g_1 \) into \( g_2 \) using elementary edit operations [29], [30]. An edit operation is either an insertion or deletion for both vertices and edges as well as a relabelling of a vertex. A cost function \( c(o_i) \) associates a cost to each edit operation \( o_i \) and the edit distance between two graphs \( g_1 \) and \( g_2 \) is then the minimum cost related to the set of operations that transform \( g_1 \) into \( g_2 \).

\[
d(g_1, g_2) = \min_{\gamma (o_i)} \sum_{i=1}^n c(o_i)
\]

where \( \gamma (g_1, g_2) \) is the set of edit operations that transform \( g_1 \) into \( g_2 \). In our case, without prior knowledge, each has a cost of one, \( c(o_1) = 1 \)

**F. Outlier detection**

We analyzed the pair-wise distances between the constructed graphs by means of a statistical outlier detection technique. Let \( O \) denote all the average distances between a given graph and all others. Assuming \( \bar{R} \), a set of all graph at a particular time, the value of the average distance of the graph \( g_i \in \bar{R} \) is as follows:

\[
O_i = \frac{\sum_{g_j \in \bar{R}, g_j \neq g_i} d(g_i, g_j)}{|\bar{R}|-1}
\]

A well used and simple statistical technique for outlier detection is the inter-quartile method [31]. First the quartiles of \( O \) are computed, \( Q_1 \), \( Q_2 \) (median) and \( Q_3 \).

**Definition** Assuming the average distance distribution \( O \) computed at a given time \( t \), a graph \( g_i \) is related to a botnet activity if

\[
O_i > Q_3 + \alpha \times IQR
\]

Usually, we set \( \alpha = 1.5 \) but fine tuning of this parameter is addressed in section IV. As a result, the outlier behavioral graphs are thus considered as the abnormal behavioral graphs. They show the infrequent and the uncommon behaviors of flows belonging to the same keys (such as source and destination IP addresses) in comparison with others.
IV. EXPERIMENTAL RESULTS

A. Dataset

We used the publicly available CTU-13 dataset of different botnet families [12]. The NetFlow records include both normal flows, C&C communication and attack traffic such as spam or DDoS. The dataset contains 13 separate scenarios whose general characteristics are described in Table II.

<table>
<thead>
<tr>
<th>ID</th>
<th>#flows</th>
<th>Duration (hours)</th>
<th>#Bots</th>
<th>#Bot flows</th>
<th>#unique IP addresses</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2,824,637</td>
<td>6.15</td>
<td>1</td>
<td>41,270</td>
<td>607,534</td>
</tr>
<tr>
<td>2</td>
<td>1,808,123</td>
<td>6.21</td>
<td>2</td>
<td>21,022</td>
<td>442,448</td>
</tr>
<tr>
<td>3</td>
<td>4,710,639</td>
<td>66.65</td>
<td>1</td>
<td>32,031</td>
<td>434,947</td>
</tr>
<tr>
<td>4</td>
<td>1,721,077</td>
<td>6.21</td>
<td>1</td>
<td>3,350</td>
<td>186,229</td>
</tr>
<tr>
<td>5</td>
<td>129,833</td>
<td>11.63</td>
<td>1</td>
<td>912</td>
<td>41,642</td>
</tr>
<tr>
<td>6</td>
<td>558,920</td>
<td>2.18</td>
<td>1</td>
<td>4,714</td>
<td>107,325</td>
</tr>
<tr>
<td>7</td>
<td>114,078</td>
<td>0.38</td>
<td>1</td>
<td>67</td>
<td>38,192</td>
</tr>
<tr>
<td>8</td>
<td>2,954,231</td>
<td>19.5</td>
<td>1</td>
<td>6,506</td>
<td>383,764</td>
</tr>
<tr>
<td>9</td>
<td>2,087,509</td>
<td>5.18</td>
<td>10</td>
<td>185,716</td>
<td>367,239</td>
</tr>
<tr>
<td>10</td>
<td>1,309,792</td>
<td>4.75</td>
<td>10</td>
<td>113,737</td>
<td>197,806</td>
</tr>
<tr>
<td>11</td>
<td>107,252</td>
<td>0.26</td>
<td>3</td>
<td>8,865</td>
<td>41,910</td>
</tr>
<tr>
<td>12</td>
<td>352,472</td>
<td>1.21</td>
<td>3</td>
<td>6,391</td>
<td>94,419</td>
</tr>
<tr>
<td>13</td>
<td>1,925,150</td>
<td>16.36</td>
<td>1</td>
<td>40,162</td>
<td>315,749</td>
</tr>
</tbody>
</table>

TABLE II: Datasets overview.

B. Parameter settings

Our proposed framework depends on three main parameters which will vary over the next experiments in order to evaluate the accuracy, the robustness and runtime performance of our approach under their variations:

- $\omega$: The size of the time window is an essential parameter. When it increases, sequences will be longer, emphasizing the importance of transforming the latter in a reduced representation, i.e. behavioral graphs. In our reported experiments, this parameter ranges between 1, 5, 10, and 20 seconds.
- $K$: The set of flow record attributes defined as key guides how to build behavioral graphs. Source and destination IP addresses have been widely used in the intrusion detection domain as being usually representative of the attackers and the victims. For botnet detection [32], [33], the primary value comes from the possibility to quantify the number of distinct connections. In this paper, we consider two cases. First, the global behavior of an host can be investigated by only using the source IP address ($sip$). Second, a more fine-grained approach will assess the connection from one host to another one using both source and destination IP address ($dip$). Finally, the protocol (TCP, UDP), $proto$, is always used (event if not mentioned) to avoid mixing TCP and UDP ports in behavioral graphs.
- $V$: The set of flows record attributes defined as values to extract sequence and build graphs is primordial. They represents attributes of the graph vertices. In this work, we are particularly interested in dependencies of services leveraged by the hosts. Hence, we propose to use either both source ($sport$) and destination ($dport$) ports, a single port, or the minimum value ($minport = min(sport, dport)$). The rationale behind this second choice is to potentially retrieve the most meaningful port in most of cases, i.e. a reserved port for a well-known and/or standard application.

C. Extracting abnormal behavioral graphs

Based on the example of dataset 13 (Virut botnet), this section illustrates how an analyst can rely on outlier detection and clustering to extract abnormal graphs. To perform this extraction, we set $K = (sip, dip, proto)$, $V = (sport, dport)$, and $\omega = 5$s.

Figure 2 represents a box plot of outlier distances calculated from Equation 5 in Section III-F with $\alpha = 1.5$. In this figure, dots represents the outliers. The figure represents a sample of time windows (between 9 and 30) where the botnet is active and have been identified over multiple time windows. An expert can then analyze corresponding graphs and even sort them regarding the distance of those outliers to prioritize his analysis. Unlike using a fixed threshold, our outlier detection method adapts dynamically the threshold, which is necessary regarding the variation of the distance as depicted in Figure 2.

Fig. 2: Outlier graphs marked with a dot in a set of selected widows of size 5 seconds.

(a) Abnormal behavioral graph of the IP address botnet targeting 147.32.84.165 (TCP scan)

(b) Abnormal behavioral graph of the bot 147.32.84.155 using UDP.

Fig. 3: Abnormal behavioral graphs of a bot performing a vertical scan over TCP (a), and a bot DNS activity over UDP (b).
As an illustrative example, Figure 3 highlights two outliers which are true positive, i.e. represent botnet flows. In Figure 3(a) represents a bot performing a vertical scan on another host, i.e. trying to discover open ports and therefore accessible services. Although detecting such scans can be done using dedicated and probably less complex methods, this shows the generality of our method to be also applicable in this case. At a first glance in Figure 3(b), the abnormal activity is not so visible as it seems that a host is contacting DNS (e.g. to request domain names). However, the same source ports are re-used multiple times as highlighted by the loop in the graph, which is highly improbable within a 5 seconds period. Hence, the structure of the graph itself is discriminative in that case.

D. Automated detection of botnet

In one hand, the previous section shows the ability of BotGM to extract abnormal graph candidates on specific samples and show how an analyst can thus retrieve the graph allowing her to analyze root cause. In the other hand, this section is dedicated to assess how botnets can be automatically detected. To achieve that, we employed the outlier method and computed standard metrics (false positive rate, true positive rate and accuracy).

However, it is worth to mention that the ground-truth is composed of labeled NetFlow records while the output of our method are supposedly abnormal graphs, potentially mixing both normal and botnet flows. For labeling a graph, and for validation purposes only, we have thus to traceback botnet original flow records which have been used for sequence and graph construction.

For a given time window, BotGM leads to the construction of n graphs: \( \{g_1, \ldots, g_n\} \). For each \( g_i \), our method will assign a class \( c_i \) equal to 1 when the graph is an outlier (botnet) or 0 otherwise (benign). In addition, each \( g_i \) is built based on an original list of events created from flow records. If at least one of them has been labeled as botnet in the original dataset, we assign a label \( l_i = 1 \), otherwise \( l_i = 0 \). In fact, those labels constitutes our ground-truth.

Relying on classes and labels, standard detection performance metrics are computed:

- **Accuracy (average)**:

  \[
  Acc = \frac{\sum_{i=1}^{n} l_i = 1 + \sum_{i=1}^{n} l_i = 0}{N}
  \]

As highlighted by the formula, the accuracy indicates within a single value the ability to classify properly the graphs as normal or as benign. A bias may be however introduced with unbalanced dataset similar to the one we used. Indeed botnet is not the major activity that is observed as highlighted in Table II and is representative of the reality. That is why true and false positive rates are important complementary metrics to consider.

- **True Positive Rate (TPR)**:

  \[
  TPR = \frac{\sum_{i=1}^{n} l_i = 1}{\sum_{i=1}^{n} l_i = 1}
  \]

- **False Positive Rate (FPR)**:

  \[
  FPR = \frac{\sum_{i=1}^{n} l_i = 0 \land c_i = 1}{\sum_{i=1}^{n} l_i = 0}
  \]

These metrics are computed per time window for all datasets and then we compute their average.

![Detection performance of BotGM](image)

Figure 4(a) shows how accuracy changes with varying parameters: time window size \( \omega \) and attributes used to build the graphs (\( K \) and \( V \)). When the time window is too small, the captured activity is limited to a few sets of flow records merging into a single graph which is not representative of the activity of the IP addresses. For example, a botnet may open multiple connections for C&C and attacks, but this may not be observed at a very short time scale. However, even if observed, the related activities can last for several seconds and longer, partial representative behaviors can still be properly identified by our method with an accuracy around 90% using five or ten seconds. As an example, \( Acc = 0.923 \) when \( \omega = 10s \), \( K = (sip, dip) \) and \( V = (sport, dport) \).

Differences can be noticed among the cases. On one hand, using only the source IP addresses leads to better results. It allows to represent 1-to-n interactions of a host which are intuitively more significant to the behavior of a host or an application in a today’s Internet. Actually, a bot usually opens multiple connections to synchronize and perform malicious activities. On the other hand, using both ports as values in
events is beneficial since they allow better characterization of graphs.

We use a ROC (Receiver Operating Characteristic) curve as depicted in Figure 4(b) to jointly evaluate the variation of TPR and FPR according to the different configuration. The results show that for false positive rate of less than 10%, the detection rate is above 95% as the time window size increases. Again, using both ports in values $V$ provides better results. As highlighted in the plotted results, almost no false alarms are raised while a TPR rate of 83% can be achieved in the best cases.

\textbf{E. Runtime evaluation}

All experiments were conducted on a Intel(R) Xeon(R) CPU E5-2420 v2 @ 2.20GHz with 62 GB main memory running on Ubuntu 16.04.2 LTS (GNU/Linux 4.4.0-59-generic x86_64). As depicted in Figure 5, the processing time logically grows with increasing window width for two reasons. First, the number of candidates keys ($K$) for constructing sequences and graphs increases, leading to more pair-wise pattern matches. Second, even if the graph representation avoids the drawback of the expansion of dependency rules in the sequence, previously unseen values in values ($V$) create new vertices, thus leading to larger graphs to analyze. Also, when selected keys ($K$) or values ($V$) are more fine-grained, same phenomena occurs with same consequences.

Fig. 5: Processing time in seconds for graph and sequence construction while varying $\omega$. The y-axis is in log scale.

Assuming the particular cases of using the protocol and IP addresses for keys, i.e. $K = (\text{proto}, \text{sip}, \text{dip})$, and both ports as values, i.e. $K = (\text{sport}, \text{dport})$, or only destination port as value, i.e. $K = (\text{dport})$. Figure 6(b) shows the average number of graphs and vertices in graph per each time window while varying the window size, $\omega$. It confirms the two reasons mentioned above but highlights that the increase of vertices in graphs is limited since set of values remains stable when $\omega \geq 5$. Moreover, Figure 6(a) shows the number of individual flow records (original data). In comparison with Figure 6(b), our method greatly reduces the size of data to handle for analysis and so helps to improve scalability. Although the cost of preprocessing and comparing graphs has to be considered, Figure 5 demonstrates that the processing time is lower than the window size, $\omega$, when the latter is lower than 10 seconds ($\omega < 10$). Assuming the best configuration trade-off established in Section IV-D with $\omega = 5$, our approach will be practicable to return results, and so raise alerts, in near real-time.

\begin{figure}[h]
\centering
\includegraphics[width=0.8\textwidth]{fig6.png}
\caption{Volume of data while varying windows size $\omega$.}
\end{figure}

\textbf{F. Comparison with other detection methods}

To evaluate our contribution better, we compared \textit{BotGM} with BClus, CAMNEP, and BotHunter described in Section II. These algorithms are presented as the most promising ones in [12]. The comparison is based on different dataset scenarios (1, 2, 6, 8 and 9) whose the results have been reported in [12].

\begin{table}[h]
\centering
\begin{tabular}{|l|c|c|c|c|c|}
\hline
Algorithm & Scenario ID & 1 & 2 & 6 & 8 & 9 \\
\hline
BClus & 0.5 & 0.5 & 0.4 & 0.3 & 0.4 \\
CAMNEP & 0.5 & 0.4 & 0.4 & 0.5 & 0.5 \\
BotHunter & 0.4 & 0.3 & 0.38 & 0.42 & 0.4 \\
\textit{BotGM} & 0.91 & 0.78 & 0.95 & 0.89 & 0.83 \\
\hline
\end{tabular}
\caption{Accuracy of different algorithms evaluated in [12] and compared to \textit{BotGM}.}
\end{table}

As shown in Table III, the best accuracy among these algorithms achieves 50% while ours achieves up to 95%. For fairness, it is important to point out and emphasize that \textit{BotGM} classifies graphs rather than single flows like other algorithms and, as explained in Section IV-D, a single graph is considered as true positive while it has been inferred from
normal and botnet flows. Although fine-grained classification (i.e., per individual flow record) is more prone to classification errors (and so lower accuracy), results obtained with BotGM are still very competitive.

V. CONCLUSION

This paper introduced BotGM, a new approach for tracking botnet activities with passive network monitoring. It relies on a behavioral graph modeling of NetFlow records. However, rather than building a single massive graph being difficult to analyze, BotGM creates a collection of smaller graphs for comparison with each other and thus indirectly comparing the behavior of end-hosts. The proposed approach combines unsupervised techniques to detect outliers and to group similar behaviors. Hence, in addition to detect botnet activity, BotGM offers several advantages. Through our experiments, we illustrated how BotGM can raise alerts and extract relevant suspect behavioral graphs. We compared our method with recently proposed techniques on a public dataset and results underlined its competitiveness and usability in a real-time manner.

Our future work consists in building signatures, a set of single behavioral graph equivalent to a cluster of graphs, in order to reduce the knowledge database. As a result, the time complexity of BotGM will be reduced.

ACKNOWLEDGMENTS

This work was partially funded by HuMa, a project funded by Bpifrance and Region Lorraine under the FUI 19 framework. It is also supported by the High Security Lab hosted at Inria Nancy Grand Est (http://www.lhs.loria.fr).

REFERENCES