Botyacc: Unified P2P Botnet Detection Using Behavioural Analysis and Graph Analysis

Shishir Nagaraja

School of Computer Science,
University of Birmingham, UK
s.nagaraja@cs.bham.ac.uk

Abstract. The detection and isolation of peer-to-peer botnets is an ongoing problem. We propose a novel technique for detecting P2P botnets. Detection is based on unifying behavioural analysis with structured graph analysis. First, our inference technique exploits a fundamental property of botnet design. Modern botnets use peer-to-peer communication topologies which are fundamental to botnet resilience. Second, our technique extends conventional graph-based detection by incorporating behavioural analysis into structured graph analysis, thus unifying graph-theoretic detection with behavioural detection under a single algorithmic framework. We carried out evaluation over real-world P2P botnet traffic and show that the resulting algorithm can localise the majority of bots with low false-positive rate.

Keywords: Traffic analysis, botnet detection, behavioural analysis, graph theory.

1 Introduction

The detection and isolation of peer-to-peer (P2P) botnets is an ongoing problem. P2P architectures are attractive as they offer low end-to-end routing delays and provide robustness against botnet response mechanisms by decentralising importance throughout the network.

In response to the proliferation of P2P botnets, many researchers have proposed the use of machine learning techniques. Essentially, these are partitioning tools which convert a dataset into clusters of similar data points under some definition of similarity. However, the context of statistical botnet detection fundamentally differs from non-security applications: the context is adversarial and the attacker controls the data of interest.

Partitioning algorithms leveraging traffic similarity require special statistical properties. First, cluster boundaries must be *precise* — approximate boundaries are not sufficient. Otherwise, botnets can exploit this weakness to "blend-in" with legitimate traffic clusters. We also require that the cluster definition is *robust* — the property that resists the addition of botnet points to non-botnet clusters. Current botnet detection techniques do not offer these properties.

To enable *precise and robust* characterisation of the legitimate data subspace (clusters), one approach is to leverage a fundamental design characteristic of modern botnets: its P2P communication architecture — P2P botnets use structured communication networks which are highly resistant to churn and adversarial takedown. However,

M. Kutyłowski and J. Vaidya (Eds.): ESORICS 2014, Part II, LNCS 8713, pp. 439–456, 2014.

anonymous proxies and NATs can hide P2P topologies from network monitors and thereby adversely affect detection based on the structural differences in the communication graphs of the embedded botnets vis a vis the background Internet graph.

Our approach is to unify two well understood principles of botnet detection (P2P connectivity and traffic similarity) into a single algorithm underlying out detection technique. This results in high detection accuracy as well as evasion resistance properties. At the core of our technique is a novel Markovian diffusion process defined over input traffic traces, that leverages patterns in connectivity as well as flow statistics. Evading detection against our approach may be hard. First, detection is based on a fundamental property of botnet operation; structured P2P topologies are a pre-requisite for botnet robustness. Second, we exploit the attacker's lack of knowledge of the precise form and structure of legitimate traffic. To be clear, we are not proposing heuristics. This paper realises the following contributions

- A link between network behavioural analysis and graph-theoretic approaches to botnet detection.
- An algorithm that takes non-linearity of network traffic into account.
- A systematic approach to selection of network traffic features for capturing behavioural information.
- A single algorithm that works at different levels of scale, in both enterprise and ISP settings.

2 Architecture

The need to perform efficient accounting, traffic engineering and load balancing, detection of malicious and disallowed activity, and other factors has led network operators to pursue infrastructures to monitor traffic across multiple vantage points. Internally, enterprises run intrusion detection systems to collect more fine-grained information about protocols and bit patterns occurring in packets while ISPs run monitoring infrastructures to collect information about flow-level traffic volumes.

Our architecture consists of the following parts.

Monitor: First, *traffic monitors* are responsible for observing and sampling traffic information from the data-plane, and building a compact representation that is used for analysis and detection. These monitors may run at end-hosts, or on routers within the network using monitoring techniques such as Cisco IOS's NetFlow [9] or sFlow, the Openflow standard. By default, NetFlow and sFlow *sample* traffic by processing one out of every 200 to 500 packets. However, advances in counter architectures [20] enable efficient tracking of the entire traffic flows in ISP networks without need for sampling. For enterprises, several products under the name of Security and Information Event Management (SIEM) systems now seek to store full traffic trace information. The constant threat of attacks suffered by modern networks has led operators to pursue infrastructures to monitor for anomalous behaviour across multiple vantage points.

Aggregator: Second, an *aggregator* component periodically receives observed communication traces from individual monitors, and merges them together to compute a *network-wide* communication trace dataset. This dataset contains the overlay topology

corresponding to all pairs of intercommunicating hosts observable across the set of monitors. It runs an algorithm that analyses the communication traces. It then attempts to separate the dataset into two (possibly overlapping) subsets: the botnet trace, and the non-botnet communication traces. Bots (hosts that form the botnet communication graph) are then output as a set of *suspect* hosts. This list may then be sent to the set of clients that are *subscribers* to the service. The list may be used to install blacklists into routers, to configure intrusion detection, firewall systems, and traffic shapers; or as "hints" to human operators regarding which hosts should be investigated as being bots. The aggregator may optionally append a likelihood that each suspect IP address has engaged in a certain activity, so that clients can individually determine at what threshold to block traffic. Aggregators may be combined in a hierarchical fashion to further reduce control overhead. In other words, low-level aggregators can collect information from a subset of networks and hosts, and then in turn send their results to a higher-level aggregator.

Honeynet: Third, the network defender obtains botnet communication traces from a honeynet. Such traces are available from third-party sources or by building a small malware testbed. The honeynet is seeded with malware relevant to the defender. For instance, an oil and gas company, they might be particularly interested in targeted malware attacks. Once the relevant malware is seeded into the malware, it is allowed to connect to its control servers. The resulting traffic including C&C traffic is recorded and forms the seed input to the inference algorithm. IP communication headers and summary information for each traffic flow is recorded and used by the detection algorithm.

Inference Mechanism: Fourth, traffic traces from the aggregator and the honeynet is piped to our inference technique. The algorithms underlying our technique are able to partition the traffic into multiple botnet and non-botnet flow partitions.

3 The Problem

Our goal is to separate botnet C&C communication from legitimate network traffic. Consider a *communication graph* $G = (V, E)$ with V representing the set of hosts observed in traffic traces, and an edge $e \in E$ representing a *traffic flow*. Each edge e is a k-dimensional vector where k is a system parameter. Consider one or more *botnet graphs* $G_b = (H, M)$ embedded within the communication graph, where $H \subseteq V$ is the set of bots (infected hosts) and $M \subseteq H \times H \subseteq E$ are the corresponding edges (botnet flows). The objective of the inference algorithm is to detect subgraphs G_b whilst minimising false-negative rate and false-positive rate.

Graph techniques such as community detection algorithms, sybil detection algorithms, and other graph partitioning methods leverage the presence of a bottle-neck cut separating a subgraph of interest from the rest of the graph. This scenario is not applicable to botnets where there is no bottle-neck cut separating botnet edges and legitimate edges. From a graph-theoretic perspective, botnet detection is an edge-partitioning problem, an open research problem. Whereas conventional graph partitioning algorithms (community and sybil detection) are designed for vertex partitioning.

4 Inference Technique

4.1 The Methodology

Botnets create unique patterns in network traffic. These patterns manifest themselves in a number of ways which can be traced to the botnet's design. The use of structured P2P communication topologies increases resilience to bot-takedown, as well as the anonymity of messages on the botnet C&C channel when messages are routed via other bots.

A second source of patterns is statistical similarity of traffic patterns. Bots tend to have similar lifecycles of reconnaissance and initial compromise, followed by the establishment of a C&C (command and control) channel, which is in turn followed by attacks such as data-exfiltration or service denial attacks.

Botnet detection via the use of structured peer-to-peer topologies, similarity of traffic flow patterns, and collaboration involving a large number of infected hosts, have thus far been studied individually. In this paper, we propose a detection methodology that unifies these approaches. The intuition behind unifying graph-theoretic and statistical behavioural analysis, rather than their independent application, is to leverage feedback loops across these approaches.

The feedback loop is designed as a stochastic diffusion process over 'similar' traffic flows. It is based on a new type of random walk on graphs. Random walks allow us to reason about graph topology and have been heavily used in the development of graph partitioning techniques. To incorporate the notion of edge 'similarity', we apply theoretical tools from euclidean geometry. Each traffic flow is a vector whose scalar elements specify the Cartesian coordinates of a point with respect to a set of axes — one axis per element. The ensemble of points resulting from considering traffic flows constitutes a multi-dimensional geometric surface with a lot of structural information embedded within it.

Our inference algorithm constructs geometric surfaces whose structure depends on the communication graph as well as traffic-flow information. Our inference algorithm is a stochastic diffusion process over 'similar' edges. We start by representing traffic traces into a communication graph. We define a special random walk over this graph. The novelty of the walk is that state transition (choice of outgoing edge) depends on the incoming edge of a random walk step. This is done to incorporate the notion of edge 'similarity' — the walk has a bias towards similar flows. Flow similarity is defined using Euclidean distance in a high dimensional setting where each traffic flow is a vector whose scalar elements specify the Cartesian coordinates of a point with respect to a set of axes – one axis per element. The ensemble of points resulting from considering traffic flows constitutes a multi-dimensional geometric surface with a lot of structural information embedded within it.

4.2 Step1: Constructing the Dual Graph

From captured traffic traces, we construct a communication graph G where each edge $e \in E(G)$ is a traffic flow represented by a k-dimensional vector and whose nodes represent computers. This graph only contains topology information. We then construct a

new graph that which is influenced by communication topology and the geometry of traffic-flow vectors. We achieve this by creating a dual graph.

To find the dual of the communication graph, we convert edges (traffic flows) into nodes. We then connect pairs of nodes (traffic flows) whose which are *locally similar*: flows must transition adjacent IP addresses (share a common node in G) and demonstrate flow-similarity (flow vectors must be less than a threshold distance apart). The intuition behind this step is that random walks on the dual graph will achieve the equivalent of entering and exiting nodes over closely related flows in the original communication graph. Note that this would not be normally possible because random walks are memoryless. Whereas we wish to study diffusion effects of walks over similar (traffic) edges rather than different edges in the original communication graph — this is one of the primary design features that will reduce the false-positive rate problem we discussed in the motivation sub-section above.

The dual of G is a weighted graph $\mathcal{D}(\mathcal{G})$. Each edge in G is a node in $\mathcal{D}(\mathcal{G})$ therefore $|V(\mathcal{D}(\mathcal{G}))| = |E(G)|$. An edge between two nodes in $\mathcal{D}(\mathcal{G})$ is constructed as follows:

- *edge-adjacency:* For a edge e_{st} between nodes s,t in G, the set of adjacent edges is the set of all edges connecting s and x, or, t and y: $S_{e_{st}} = \{e_{st}, e_{xs}, e_{ty}\}$.
- *geometrical distance:* Each edge e in G is represented by a k-dimensional vector $(w_1, \ldots, w_k)$. The geometrical distance between a pair of edges e_i and e_j is given by:

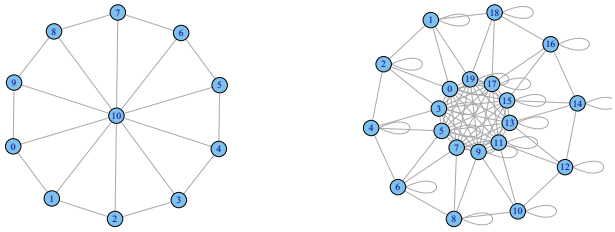
$$W_{ij} = \begin{cases} e^{-\frac{||e_i - e_j||}{t}} & \text{if} ||e_i - e_j||^2 \leq \varepsilon \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

  , where the norm is the Euclidean norm in $\mathcal{R}^k$.
- Each edge e in G is a node e in the dual of G, namely $\mathcal{D}(\mathcal{G})$. We place an edge with weight W_{ij} between two nodes e_i and e_j in $\mathcal{D}(\mathcal{G})$, if they satisfy the edge-adjacency property above and are geometrically close enough ($W_{ij} \neq 0$).

4.3 Step2: Partitioning

Now that we have a graph-geometric representation ($\mathcal{D}(\mathcal{G})$) of the traffic, our next task is to separate subgraphs corresponding to different traffic characteristics. The geometric space within which traffic points reside is represented as the graph, and we explore the local and global properties of surfaces using random walks (as explained earlier).

Constructing the dual allows us to partition a communication into subgraphs with similar traffic flow behaviour and subgraphs that have a different expansion properties than the background graph they are embedded within. Consider the toy example of edge-partitioning a graph consisting of a set of nodes connected using one set of edges within a ring structure and using a second set of edges as a star structure as shown in Fig. 1(a). Without taking geometry into account (flow-similarity), computing the dual graph gives us Fig. 1(b), where star-edges have been converted into a clique subgraph that is weakly connected to a subgraph containing nodes that were edges constituting a ring in the original graph. Now, using random walks over the dual-graph, we can



(a) graph G (b) Dual $\mathcal{D}(\mathcal{G})$

Fig. 1. A sample graph and its dual; vertex ids are reset in the dual

partition the ring-edges and the star-edges using the relative expansion properties of these two subgraphs in the dual.

Traffic data is represented as a graph with individual geometric surfaces represented as subgraph communities within a single connected component. A surface corresponds to a subset of $V(\mathcal{D}_C(\mathcal{G}))$ that is richly intra-connected but sparsely connected with the rest of the graph. To partition traffic by similarity, we consider the algebraic connectivity properties of the graph. of each surface and locate **gaps** between dense surfaces.

Partitioning Technique. To find gaps that naturally partition the data, we find the Laplacian over the graph dual. Laplacian operator is efficient at finding gaps between geometric clusters.

The standard technique for detecting gaps using the Laplacian operator consists of considering the adjacency matrix A and the graph edges, with weights $w_1, w_2, \ldots, w_n$ on the edges, the Laplacian matrix is defined as $L = AI_wA^T$. Here I_w is a diagonal matrix with the weights placed along the diagonal i.e. $I_{ii} = w_i = d_i$, where d_i is the degree of node i. We then find the eigenvalues of the Laplacian matrix. There are standard techniques for computing eigenvalues, for instance the well known Lanczos algorithm which scales as $O(n\ log\ n)$.

A partition for graph $G = (V,E)$ is defined as a partition of V into legitimate and botnet subgraphs L, and M, such that the number of edges across the gap $gap(L,M)/(|L||M|)$ is minimised. In such a scenario, the second smallest eigenvalue (λ_2) of L, yields a lower bound on the optimal cost of the gap is $(1-\lambda_2)$. The eigenvector (v_2) corresponding to the second eigenvalue, bisects the graph into only two clusters based on the sign of the corresponding vector entry. Division into a larger number of clusters is achieved by repeated bisection. To prevent repetitive bisection from using trivial gaps we use the well known conductance metric.

Quality Metric. The quality of a partition is measured by its conductance, *the ratio of the number of its external connections to the number of its total connections.*

We let $d(i)$ denote the degree of vertex i. For $S \subset V(\mathcal{D}_C(\mathcal{G}))$, we define $\delta(S) = \sum_{i \in S} d(i)$ as the volume of S. So,

$$\delta(V(\mathcal{D}_C(\mathcal{G}))) = 2|E(\mathcal{D}_C(\mathcal{G}))|$$

. Let $EE(S, V(\mathcal{D}_C(\mathcal{G})) \setminus S)$ be the set of edges connecting a vertex in S with a vertex in $V(\mathcal{D}_C(\mathcal{G})) \setminus S)$. We define the conductance of a set of vertices S, written $\phi(S)$ as

$$phi(S) = \frac{EE(S, V(\mathcal{D}_C(\mathcal{G})) \setminus S)}{\min(\delta(S), \delta(V(\mathcal{D}_C(\mathcal{G})) \setminus S))}$$

The conductance of $\mathcal{D}_C(\mathcal{G})$ is then given by:

$$\min_{S \in V(\mathcal{D}_C(\mathcal{G}))} \phi(S)$$

. To avoid obtaining trivial partitions, the conductance of a subgraph is normalised by the size of the partition.

Generalising the Approach. The Laplacian operator is applicable for linear contexts. However, since the botnet context is adversarial the attacker can architect bot traffic to behave in a non-linear manner. For instance, the attacker can engineer bot traffic to follow a non-linear geometric shape such as a sphere or a curved line. In such a case, a linear operator would make errorneous judgements. Since the adversary controls botnet traffic data, an assumption of linearity would be incorrect. The geometric interpretation of this assumption is that traffic feature vectors are points on a planar surface. However, a non-linear operator, being generic, (discards and) resists attacks based on exploiting this assumption. For instance, as a way of increasing the false-positive rate of detection, the botnet operator can alter botnet traffic vectors so they are a short euclidean distance from legitimate traffic vectors. In this case, a linear operator would be successfully misguided into accepting the botnet traffic as legitimate given their close proximity. However, a non-linear operator would be able to leverage the actual geometric structure of legitimate traffic (instead of assuming it is a planar surface) to resist botnet points being accepted into the legitimate cluster. Thus, we use the laplace-beltrami operator which allows valid distance measurements when the geometrical subspace is non-linear.

Let X denotes the set of functions defined on the vertices of $\mathcal{D}(\mathcal{G})$. The application of laplace-beltrami operator on a function results in another function in X, i.e $Lf \in X \forall f \in X$ since L is a function of functions: from X to X. An *eigenfunction* of the laplace-beltrami operator is a function f such that $Lf = \lambda f$ where λ is the eigenvalue of L and represents the scaling of f. By computing the eigenfunctions we obtain a compressed list of features that is expressed as a linear combination of the (larger number of) input features. This approach of using eigenfunctions for feature reduction prior to inference is well known within the machine learning community. We leverage this as a building block in our algorithm.

- To apply the laplace-beltrami operator, we compute $L = D - W$, where D is the diagonal matrix corresponding to W. L is a symmetric and positive matrix.

- Compute the eigenvectors and eigenvalues of $Lf = \lambda f$ ordered in the increasing order of the eigenvalues $\lambda_0 \leq \lambda_1 \leq \lambda_2 \ldots \lambda_{k-1}$.
- Ignore the first eigenvector $f_0 = (1, \ldots, 1)$ with eigenvalue $\lambda_0 = 0$.
- Each vertex i of $\mathcal{D}(\mathcal{G})$ is now expressed in terms of the m new features. The modified graph is referred to as $\mathcal{D}^{\mathcal{C}}(\mathcal{G})$. Vertices of the modified graph are now available as $V(\mathcal{D}(\mathcal{G}))_i = (f_1(i), \ldots, f_m(i))$.
- Each edge between a pair of vertices in $\mathcal{D}^{\mathcal{C}}(\mathcal{G})$ is refreshed with an edge weight that corresponds to the new set of features. Each edge between vertices i and j is refreshed as follows:

$$W'_{ij} = \begin{cases} e^{-\frac{||v_i - v_j||}{t}} & \text{if} ||v_i - v_j||^2 \leq \varepsilon \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

, where the norm is the Euclidean norm in $\mathcal{R}^m$.

5 Noise Tolerance

Our detection method leverages the attackers limited knowledge about the location and structure of the legitimate surface to bound statistical noise injected by an adaptive botmaster. Vectors corresponding to three categories – legitimate traffic, noise traffic introduced by the botmaster, and genuine botnet traffic – are represented in the dual graph. As described earlier, each traffic vector constitutes a vertex. An undirected edge exists between two vertices if they are within a ε threshold distance of each other.

An edge may exist between a noise vector and a legitimate vector if they are within close proximity. Such a *noise edge* allows "leakage" of walks between the legitimate and botnet surfaces contributing to false positives. Each noise edge connects a noise pack to vertices within a legitimate traffic surface.

To successfully evade detection, the attacker must succeed in placing a large number of noise points (*noise pack*) in close proximity of another surface, all points within the noise pack must be in close proximity of legitimate points rather than just a few. This requires knowledge of the location of a majority of the points in that surface. Our design leverages three important facts to limit the number and size of noise packs: *a*) legitimate surface graphs tend to have high *algebraic connectivity* (the smallest positive eigenvalue of its Laplacian matrix). *b*) large amounts of noise (compared to the number of noise edges) decreases algebraic connectivity.

Capping the number of noise edges: The evasion resistance property of our detection algorithm relies on limiting the number of noise edges (m). There are primarily two scenarios in which botnet malware might attempt to increase m:

- **No knowledge of legitimate subspace:** Botnet sprays noise randomly in the data space in the hope that some of these points will be near the legitimate surface. Spraying is achieved by the botnet altering behaviour by modifying traffic feature values. This is quite difficult to perform because the data space is large (theoretically infinite) whereas the legitimate surface is located on a relatively smaller geometric subspace.

– **Partial knowledge of legitimate subspace:** A second possibility is that the botnet has some awareness of legitimate surface points. For instance, this information may be gained by analysing legitimate traffic on the infected machine. It can add noise in the proximity of known legitimate points to create edges between the botnet surface and the legitimate surface. However, in doing so, the algebraic connectivity characteristics are disturbed by the small set of noise edges. The noise based on partial knowledge introduces a gap between the legitimate and botnet surface. This is true, unless the noise surface is in the proximity of a significant majority of points within the legitimate surface. This is very hard to accomplish unless the attacker has full knowledge of the legitimate traffic; high-level trends or summary information do not give information regarding structure and location of the legitimate surface, full traffic traces would be required.

6 Results

We evaluated our algorithm in two different experimental settings. We apply our algorithm on real botnet traces within an enterprise setting and measure its effectiveness.

Malware testbed: In order to obtain botnet traffic flows we created a testbed of 25 servers within a test network connected to the Internet. The computers was then seeded with samples from three peer-to-peer versions from well known botnet families: Zeus, Miner, and Spyeye. All three botnets have moved from centralised C&C servers to entirely P2P communication. Zeus and Spyeye are designed for stealing banking information while Miner steals Bitcoin credentials.

On each testbed computer, we instantiated 70 copies of a malware sample (at a time) within a hypervisor, i.e 1750 instances from each family, or a total of 5250 malware instances. The testbed allows the bot to connect with other bots in the wild which enables us to closely observe the actions of the bot and its interactions with other bots. The result is a lot of network traffic which will be *attack traffic by definition*. We exercised due diligence to prevent our testbed from being used as an attack platform. In particular, all probing, scanning, spam was blocked while DDOS attacks were rate-limited at the very least. However, command and control (C&C) incoming and outgoing traffic was allowed as this was essential for our study.

We set up traffic monitors on the backbone router at a university campus network using an electrical signal duplicator unit that works at up to 20Gbps. As opposed to port-mirroring, this approach allows us to capture packets at the line rate without inducing the effects of packet sampling. The traffic rate is typically 2.5–8Gbps.

We developed an efficient network flow capture tool that processes packet traces and generates flows. A flow is record of communication between a pair of hosts and is represented by a tuple containing a number of fields as given below. We consider UDP and TCP traffic. In the case of UDP traffic, the TCP fields listed are zeroed out as are optional TCP fields.

Each flow record is a tuple structure containing two parts; inter-flow values that are common to all flow records occurring within a ten-minute time period, and flow specific fields. The entire set of fields comprising a flow record are given below:

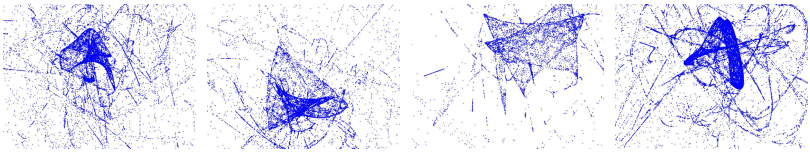
- Inter-flow statistics (fields) computed across the flows: *traffic volume, duration, distribution of packets per flow, distribution of flows per period, distribution of packets per flow, throughput distribution, distribution of inter-flow arrival times averaged distribution of inter-packet arrival times. Distributions are computed over a time interval of ten minutes.*
- Flow fields: *tcp/udp.source-port, tcp/udp.destination port, IP version, IP header length, ip.tos — precedence, ip.tos — delay, ip.tos — throughput, ip.tos — reliability, ip.tos — reserved, ip.tos — total length, ip.flags,*
  ip.fragmentoffset, ip.ttl , ip.protocol, Entropy of ip.id# distribution, Entropy of tcp.seq# distribution, Entropy of tcp.ack# distribution, tcp.offset, tcp.reserved, tcp.flags,
  tcp.maximum-segment-size, tcp.echotimedata

In the above list, each sample distribution is represented by the corresponding histogram. The first bin corresponds to $P(X < x) \leq 5\%$, the second bin corresponds to $P(X < x) \leq 15\%$ and so on.

Algorithm Application. We now consider the application of our algorithm on a real-world dataset. To access live traffic, we captured network traffic at a university gateway for a period of one month between March and April 2012. This dataset has 113,576 unique source IP addresses and 11,643,993 traffic flows. This includes 432,257 embedded botnet flows from seeded malware. This corresponds to a communication graph G_E containing both malware and non-malware edges.

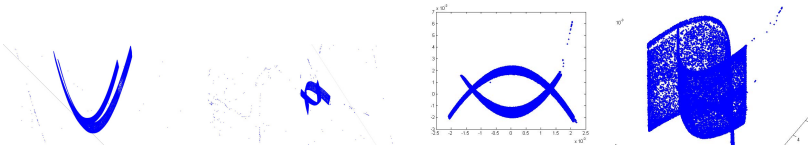
The first step of our algorithm is to create the dual of G_E, namely $D(G_E)$. At this stage each edge (flow) becomes a node and nodes become edges, therefore flow-vectors are now associated with each node. An edge is constructed between two nodes if the Euclidean distance between the corresponding flow-vectors is less than a certain threshold ε. We used $\varepsilon = .0025$. This value controls the runtime of the generation of the dual. Our inference algorithm is not sensitive to high values of ε (leading to a denser graph), since the diffusion effects of the subsequent random walk process is controlled by the non-linear kernel function: $e^{-\frac{||e_i - e_j||}{t}}$ see Eqn. 1. We chose $t = 1$, but higher values will produce a sharper decay. Our choice of ε leads to $O(logE(D(G_E)))$ edges per node in the dual. This step leads to the to embedding of information from the communication graph topology and the geometry of network traffic, within the dual graph. Figure 4(a) shows a rendering of the dual with vertices represented by blue points and edges represented by the distance between vertices; the number of edges is too high to be represented graphically. Figures 4(b) through 4(f) show the dual graphs corresponding to the other five weeks of enterprise traffic with embedded botnet traffic.

The second step of our algorithm is dimensionality reduction. This step prevents the botnet from altering traffic patterns over time in order to "throw-off" the detection system. Thus the compressed feature set selected by the algorithm can vary from across time. Feature selection is carried out in an unsupervised manner. The compressed feature list is given by the ordered eigenvalues of the laplacian of the dual graph computed at the end of the previous step. The first eigenvalue is zero by definition and this is ignored. The eigenvectors corresponding to the eigenvalues represent the new mapping



(a) (b) (c) (d)

Fig. 2. Visual representation (top three dimensions) of network traffic after dimensionality reduction, at the end of Step 2 of our algorithm



(a) (b) (c) (d)

Fig. 3. Visual representation of the results of our inference algorithm showing isolated botnet traffic

of traffic data points as a function of the compressed feature list. The embedding of the various geometrical surfaces within this compressed space is shown in figure 3.

Thus far, our algorithm has only partitioned traffic into different surfaces. The botnet flows are partitioned into respective surfaces. After three iterations of the partitioning algorithm, we obtain a subgraph (surface) of size 432,256 nodes, containing 423,906 nodes corresponding to botnet flows, and 8,350 nodes corresponding to non-botflows. At this stage, our validation metric indicates that the sub-graph has a graph conductance of about 0.9 (In all other scenarios, the graph conductance is less than 0.5, so we can safely set our threshold of the graph conductance test to be 0.5).

Table 1. Zeus in enterprise traffic – detection and error rates of inference

	#Malicious flows	#gateway-flows	Detection%	% FP
Week1	3368	1211736	99.98	0.019
Week2	8836	1392755	99.93	0.037
Week3	3231	1109264	97.95	0.082
Week4	8349	1312952	98.09	0.041
Week5	8217	1130120	98.21	0.030

We evaluated the performance of detection on a weekly basis through our dataset. Each week, we collected gateway traces and combined it with that week's botnet traces. The previous week's data was discarded from the graph and dual generation. The honeynet seed traces were also fresh, thus corresponding traces from each week were input to our detection algorithm.

To evaluate performance, we are concerned with the *false positive rate* (the fraction of non-bot nodes that are detected as bots) and the *false negative rate* (the fraction of bot nodes that are not detected). The results of botnet detection for Zeus, Spyeye, and Miner are shown in Table 1, Table 2, and Table 3 respectively.

Table 2. Spyeye in enterprise traffic – detection and error rates of inference

	#Malicious flows	#gateway-flows	Detection%	% FP
Week1	8021	1346235	98.11	0.041
Week2	6295	1327479	98.77	0.064
Week3	4213	1180134	99.86	0.074
Week4	3538	1396174	97.86	0.047
Week5	5388	1186480	98.70	0.023

Table 3. Miner in enterprise traffic — detection and error rates of inference

	#Malicious flows	#gateway-flows	Detection%	% FP
Week1	1050	1590306	97.50	0.018
Week2	2735	1186212	96.64	0.064
Week3	5341	1560028	94.89	0.048
Week4	3099	1186929	95.52	0.062
Week5	4566	1154067	97.76	0.072

Detection rates ranged between 97% and 99% for Zeus and Spyeye. For Miner, the detection rate was a bit lower at around 95% on the average. Importantly, for all three peer-to-peer botnets, the false-positive rate was well below 0.1%.

6.1 Effects of Botnet Topology and Size

In the next set of experiments, we seek to understand the effectiveness of deploying our algorithm in a setting where a majority of the botnet communication graph is embedded within the network traffic captured from our vantage points. This is the case of multiple ISPs cooperatively running our inference algorithm.

To study this, we constructed a dataset where traffic flows from the Zeus botnet were embedded it within ISP traffic using various peer-to-peer structures. To improve realism, we build the background traffic communication graph by using real packet-level traces collected by CAIDA on OC192 Internet backbone links [2]. Since packet level information is not available, we only used flow-level features for our experiments with ISP data.

Another aspect we need to consider is the different sizes of botnets. An inference algorithm must be able to effectively detect small botnets as well as large botnets. This is important in order to be able to track the evolution of the botnet throughout its life-cycle right from the early stages of deployment to large-scale botnets which may pose significant threat due to the possible scale of geographical spread as well as size. We perform this experiment by keeping the size of the background traffic graph constant, and generating synthetic botnet topologies of varying sizes (between 100 and 100,000 bots).

Finally, we must also consider the effects of partial visibility. Clearly, obtaining access to the Internet traffic of all ISPs is a fairly difficult proposition. However, it is certainly likely that a fraction of ISPs can be incentivised to cooperate via a combination of legal and economic incentives. We also understand from previous work that a subset of ISPs typically have access to a significant fraction of botnet traffic. A study [1] of 4,000 IP addresses belonging to the Storm botnet found that 60% of inter-bot paths traverse top six ISPs, and 89% of the inter-bot paths traversed top ten ISPs. More recently, reports from anti-virus companies indicates that India has the second highest number of spam bots. Interestingly, the whole country is served by two major ISPs. To

incorporate the effects of partial visibility, we construct the botnet graph, by selecting a random subset of nodes in the background communication graph (CAIDA) to be botnet nodes, and synthetically add bot flows between them corresponding to a particular structured overlay topology. We then simulate the effects of partial visibility by retaining only 55% of the total traffic flows in the combined graph and discarding 45% of the flows chosen uniformly at random.

We then pass the combined graph as input to our inference algorithm. By keeping track of which nodes are bots (this information is not passed to our algorithm), we can acquire "ground truth" to measure performance. To investigate sensitivity of our techniques to the particular overlay structure, we consider several alternative structured overlays, including (a) Chord [19], (b) de Bruijn [13], (c) Kademlia [14], and (d) the "robust ring" topology described in [11]. The remainder of this section contains results from running our algorithms over the joined botnet and Internet communication graphs, and measuring the ability to separate out the two from each other.

Table 4. CAIDA – results if only Tier-1 ISPs contribute views

Topology	$\|V_B\|$	% Detected	% FP
de Bruijn	1000	99.97	0.0011
	10000	99.98	0.0020
	100000	99.98	0.0170
Kademlia	1000	99.97	0.0040
	10000	99.97	0.0104
	100000	99.96	0.0350
Chord	1000	99.98	0.0017
	10000	99.97	0.0024
	100000	99.87	0.0202
LEET-Chord	1000	99.96	0.0040
	10000	99.65	0.0139
	100000	98.91	0.0613

Overall, we find performance to be fairly stable across multiple kinds of botnet topologies and sizes with detection rates higher than 98%. In addition, our algorithm is able to achieve a false positive rate of less than 0.06% on the harder-to-detect LEET-Chord topology. We find that as the size of the bot graph increases, performance degrades, but only by a small amount. For example, in Table 4, with the fully visible deBruijn topology, for 100 nodes the false positive rate is zero, while for 10,000 nodes the rate becomes 0.002%.

The high detection and low false-positive rates are better than state-of-the-art algorithms. It shows that the combination of traffic-flow features and graph-structure information holds good potential in designing reliable algorithms for botnet detection.

While our approach is not perfectly accurate, we envision it may be of use when coupled with other detection strategies (e.g., previous work on botnet detection [10,8], or if used to signal "hints" to network operators regarding which hosts may be infected.

7 Discussion

As we have demonstrated, starting with a certain definition of botnet behaviour — **traffic produced from the malware testbed**, graph theoretic analysis can help in identifying botnets in enterprise traffic. We now discuss the significance of our results and related insights.

The evaluation results indicates the usefulness of our approach. The main insight of our work is that both both legitimate and botnet traffic have specific geometry, i.e, traffic vectors lie on a low-dimensional geometric surface. The inference technique partitions the dataset into multiple botnet and legitimate surfaces.

The graph underpinning the partitioning process is constructed using both communication topology information and communication flow information. Since P2P topologies are a fundamental design requirement in order to maintain the botnet's resilience, the botmaster cannot evade detection without giving up resilience properties. Without P2P topologies underlying botnet communication, the C2 channel would not be robust enough to withstand take down attempts, thus forcing the attacker to choose between survivability and stealth.

At the same time, techniques to isolate the structured communication graphs induced by botnets depend on the integrity of communication links within traffic traces recorded by network monitoring systems. This can be a challenge when we consider the widespread use of NATs and other traffic aggregators. Aggregators hide the presence of communicating endpoints and appear as a few large nodes communicating with a large number of endpoints. This induces error into the inference process. If substantial parts of a P2P embedded botnet appear as leaf nodes connected to a few hubs, then the basis for isolating botnets from the background traffic by leveraging communication topology characteristics is substantially weakened.

Unifying both behavioural and structured graph approaches presents a credible approach to addressing the errors induced by traffic aggregators. When communication topology information is hidden using anonymous relays, or is otherwise incomplete, or mutilated in the dataset, the inference algorithm can recover from the errors. The construction of the dual graph driven by the statistical similarity of traffic flows still proceeds undisturbed. However the construction of the dual graph now involves a higher number of vector comparisons to dismantle the virtual high-degree node induced by the aggregator — instead of $O((log\ n)^2)$ vector comparisons in the normal case, we are required to carry out a significantly larger number of vector comparisons which is $O(n^2)$ in the worst case; when all the traffic is lumped into a single node. In practice, the computational effort to manage errors in topology is at least a large constant times $O((log\ n)^2)$.

Dynamic Feature Selection: Our inference algorithm incorporates dynamic feature selection instead of using a static heuristic-driven definition of which features are indicative of botnet traffic. The relevant feature set is derived as part of the dimensionality reduction step. This means that unlike static heuristics where the feature set has to be constantly updated by the network defender, we derive the feature set directly from the traffic traces. On the otherhand, this approach requires us to capture a large number of features beforehand which can increase the load on traffic monitoring. While the evaluation results are fairly positive, we have been unable to evaluate how dynamic feature selection behaves under botnet evolution. Dimensionality reduction simply selects a combination of features that capture most of the information contained in the dataset.

Scale: Our experiments show that the inference technique can scale to large traffic volumes, and in the presence of partial observations. This solves an number of practical problems concerning the use of different types of algorithms for enterprise detection

and ISP-level detection. Thus engineering and training efforts can be concentrated on developing and operating a single functional piece of equipment, as opposed to having different solutions each for ISPs and enterprises.

8 Related Work

Bots are unique amongst networked malware in that they collectively maintain communication structures across nodes to resiliently distribute commands from a *command and control* node. The ability to coordinate and upload new commands to bots gives the botnet owner vast power when performing criminal activities, including the ability to orchestrate surveillance attacks, perform DDoS extortion, sending spam for pay, and phishing. This problem has worsened to a point where modern botnets control hundreds of thousands of hosts and generate revenue of millions of dollars per year for their owners [4].

8.1 Non-signature Based Methods

We now describe related work in non-signature based detection methods. None of the techniques we discuss in this section, have any evasion resistance properties.

BotMiner [5] detects infected hosts without previous knowledge of botnets. In this system, bots are identified by clustering hosts that exhibit similar communication and (possible) malicious activities. The clustering allows hosts to be groups according to the botnet that they belong to as hosts within the same botnet will have similar communication patterns, and will usually perform the same activities at the same time (such as a DDoS attack).

There are also schemes that combine network- and host-based approaches. The work of Stinson et al. [18] attempts to discriminate between locally-initiated versus remotely-initiated actions by tracking data arriving over the network being used as system call arguments using taint tracking methods. Following a similar approach, Gummadi et al. [6] whitelist application traffic by identifying and attesting human-generated traffic from a host which allows an application server to selectively respond to service requests. Finally, John et al. [12] present a technique to defend against spam botnets by automating the generation of spam feeds by directing an incoming spam feed into a Honeynet, then downloading bots spreading through those messages and then using the outbound spam generated to create a better feed.

Server Detection: DNS. Several works provide a detection mechanism to identify domains associated with malware at using centralised C&C channels.

Paxson et al [16] attempt to provide a detection mechanism that leverages the amount of information transmitted over a DNS channel in order to detect suspicious flows. The system allows for a upper bound to be set, any DNS flow that exceeds this barrier is flagged for inspection. The upper bound can be circumvented by limiting flows, but this has an impact the amount of data exfiltration/command issuing that can occur. The system looks primarily at data included within domain names, but also looks at inter-query timings and DNS packet field values, both of which can provide low capacity channels.

Perdisci et al [17] apply clustering to domains so they are grouped according to overlap in the returned IP addresses. By then comparing the clusters to previously labelled data, they can then be classified as flux or non-flux, revealing domains that make use of the same network.

8.2 Graph-Based Approaches

Several works [3,8,7,21,10] have previously applied graph analysis to detect botnets. The technique of Collins and Reiter [3] detects anomalies induced in a graph of protocol specific flows by a botnet control traffic. They suggest that a botnet can be detected based on the observation that an attacker will increase the number of connected graph components due to a sudden growth of edges between unlikely neighbouring nodes. While it depends on being able to accurately model valid network growth, this is a powerful approach because it avoids depending on protocol semantics or packet statistics. However this work only makes minimal use of spatial relationship information. Additionally, the need for historical record keeping makes it challenging in scenarios where the victim network is already infected when it seeks help and hasn't stored past traffic data, while our scheme can be used to detect pre-existing botnets as well. Illiofotou et al. [8,7] also exploit dynamicity of traffic graphs to classify network flows in order to detect P2P networks. It uses static (spatial) and dynamic (temporal) metrics centered on node and edge level metrics in addition to the largest-connected-component-size as a graph level metric.

More recently, Botgrep [15] presented a scheme that searches for expander graphs to discover P2P graphs within ISP traffic. The theoretical component of the algorithm presented is our work is much more stronger. Botgrep does not consider traffic flow categorisation and therefore would end up with high false-positive rates when its core assumption is broken — high-degree nodes should not be infected and have incoming or outgoing botnet traffic flows. In the operational context of a NAT (Network Address Translator), the traffic of hundreds of computers would be aggregated into a single IP address. Such NAT installations are getting rather popular: mobile broadband ISPs use carrier-NATs where thousands of mobile consumers are behind a NAT run by the ISP, and each user is on a separate port. Our inference algorithm, will be able to operate in such deployment contexts very well since it combines flow clustering with structured graph analysis; even if graph structure is obscured by the NAT the inference algorithm can still leverage non-linear subspace analysis over traffic flow data to isolate botnet traffic.

Further, as compared with other graph-based and behaviour-analysis schemes, we have shown (see Fig. 3) that there is more to application traffic than mere clustering: there are intricate geometrical surfaces corresponding to application traffic characteristics. Indeed our algorithm is quite generic and we hope that our results will encourage other researchers to apply our technique to other traffic classification problems.

9 Conclusion

The ability to localise bot-infected hosts at Internet scales represents both a very challenging problem. In this work, we have approached the problem of botnet detection with a security-by-design approach: detection evasion is based on the attacker's detailed knowledge of legitimate traffic traces. Thus detection is based on the fundamental properties of botnets which enables evasion resistant detection. In future work, we will provide formal bounds for evasion resistance.

In this work we have tried to build a link between graph-theoretic botnet detection approaches with network behavioural analysis approaches. Our approach works by leveraging patterns within the communication graph as well as within network traffic

between Internet hosts from a set of traffic-monitoring vantage points, and then exploiting the intrinsic non-linear geometry of traffic in order to distinguish traffic flows that are part of the botnet. Behavioural analysis approaches (involving machine learning) are commonly criticised in the security community for assuming a static traffic profile of the botnet in the form of a feature list. As a first step towards being able to operate in an environment where the botnet evolves in response to the detection mechanism, we adopt the notion of dynamic feature selection.

Compared with results from previous work using graph-theoretic or behavioural analysis approaches, our techniques accomplish better results. This is not surprising since they exploit intuitions from both. However, our techniques do not achieve perfect accuracy, but they achieve a low enough false positive rate to be of substantial use, especially when combined with other complementary techniques. Finally, we do not attempt to address the challenging problem of botnet *response*. Future work may leverage our inferred botnet topologies by dropping crucial links to partition the botnet, based on the structure of the botnet graph.

References

1. Botlab: A real-time botnet monitoring platform, `botlab.cs.washington.edu`.
2. The Cooperative Association for Internet Data Analysis, `http://www.caida.org/`
3. Collins, M.P., Reiter, M.K.: Hit-list worm detection and bot identification in large networks using protocol graphs. In: Kruegel, C., Lippmann, R., Clark, A. (eds.) RAID 2007. LNCS, vol. 4637, pp. 276–295. Springer, Heidelberg (2007)
4. Franklin, J., Paxson, V., Perrig, A., Savage, S.: An inquiry into the nature and causes of the wealth of internet miscreants. In: ACM Conference on Computer and Communications Security, pp. 375–388. ACM, New York (2007)
5. Gu, G., Perdisci, R., Zhang, J., Lee, W.: BotMiner: Clustering Analysis of Network Traffic for Protocol- and Structure-Independent Botnet Detection. In: Proc. of the USENIX Security Symposium (2008)
6. Gummadi, R., Balakrishnan, H., Maniatis, P., Ratnasamy, S.: Not-a-Bot (NAB): Improving Service Availability in the Face of Botnet Attacks. In: NSDI 2009, Boston, MA (April 2009)
7. Iliofotou, M., Faloutsos, M., Mitzenmacher, M.: Exploiting dynamicity in graph-based traffic analysis: Techniques and applications. In: ACM CoNext (2009)
8. Iliofotou, M., Pappu, P., Faloutsos, M., Mitzenmacher, M., Varghese, G., Kim, H.: Graption: Automated detection of P2P applications using traffic dispersion graphs (TDGs). UC Riverside Technical Report, CS-2008-06080 (2008)
9. C. S. Inc. Cisco IOS Netflow, `http://www.cisco.com/en/US/products/ps6601/products_ios_protocol_group_home.html`
10. Jelasity, M., Bilicki, V.: Towards automated detection of peer-to-peer botnets: On the limits of local approaches. In: USENIX Workshop on Large-Scale Exploits and Emergent Threats, LEET (2009)
11. Jelasity, M., Billicki, V.: Towards automated detection of peer-to-peer botnets: On the limits of local approaches. In: USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET) (2009)
12. John, J.P., Moshchuk, A., Gribble, S.D., Krishnamurthy, A.: Studying spamming botnets using botlab. In: NSDI 2009: Proceedings of the 6th USENIX Symposium on Networked Systems Design and Implementation, pp. 291–306. USENIX Association, Berkeley (2009)
13. Kaashoek, M., Karger, D.: Koorde: A simple degree-optimal distributed hash table. In: Kaashoek, M.F., Stoica, I. (eds.) IPTPS 2003. LNCS, vol. 2735, pp. 98–107. Springer, Heidelberg (2003)

14. Maymounkov, P., Mazières, D.: Kademlia: A peer-to-peer information system based on the xor metric. In: Druschel, P., Kaashoek, M.F., Rowstron, A. (eds.) IPTPS 2002. LNCS, vol. 2429, pp. 53–65. Springer, Heidelberg (2002)
15. Nagaraja, S., Mittal, P., Hong, C.-Y., Caesar, M., Borisov, N.: BotGrep: Finding P2P bots with structured graph analysis. In: USENIX Security Symposium, pp. 95–110 (2010)
16. Paxson, V., Christodorescu, M., Javed, M., Rao, J., Sailer, R., Schales, D., Stoecklin, M.P., Thomas, K., Venema, W., Weaver, N.: Practical comprehensive bounds on surreptitious communication over dns. In: Proceedings of the 22Nd USENIX Conference on Security (2013)
17. Perdisci, R., Lee, W., Feamster, N.: Behavioral Clustering of HTTP-Based Malware and Signature Generation Using Malicious Network Traces. In: Proc. of the USENIX Symposium on Networked Systems Design & Implementation (2010)
18. Stinson, E., Mitchell, J.C.: Characterizing bots' remote control behavior. In: Lee, W., Wang, C., Dagon, D. (eds.) Botnet Detection. Advances in Information Security, vol. 36, pp. 45–64. Springer (2008)
19. Stoica, I., Morris, R., Karger, D., Kaashoek, M.F., Balakrishnan, H.: Chord: A scalable peer-to-peer lookup service for Internet applications. In: Proceedings of ACM SIGCOMM (August 2001)
20. Zhao, Q., Xu, J., Liu, Z.: Design of a novel statistics counter architecture with optimal space and time efficiency. In: ACM SIGMETRICS (June 2006)
21. Zhao, Y., Xie, Y., Yu, F., Ke, Q., Yu, Y., Chen, Y., Gillum, E.: Botgraph: Large scale spamming botnet detection. In: NSDI (2009)

A Appendix

In the following figure, we show a two-dimensional visual of botnet traffic. The long lines are an artifact of DNS fast-flux.

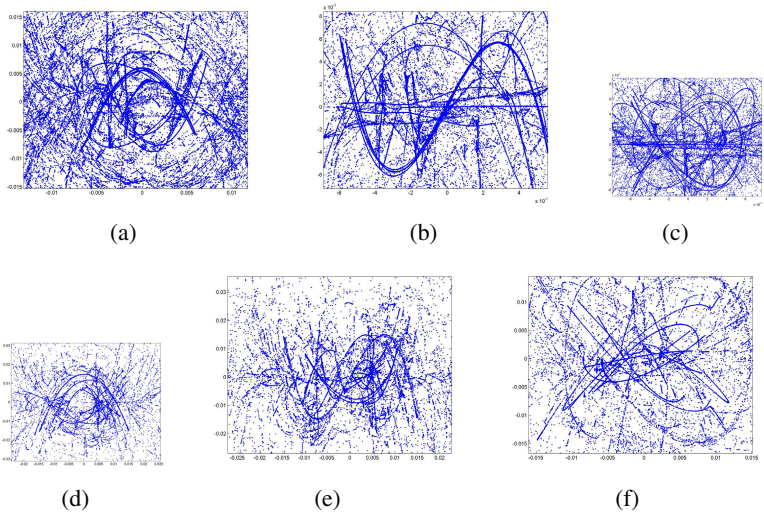


Fig. 4. Two dimensional representation of network traffic with embedded Zeus traffic, before feature selection. This figure shows the dual-graph $\mathcal{D}(\mathcal{G})$ of the traffic dataset.