UNIVERSITY OF CALIFORNIA
RIVERSIDE

# Developing Profiles of Malware and User Behaviors Using Graph-Mining and Machine Learning Techniques

A Dissertation submitted in partial satisfaction
of the requirements for the degree of

Doctor of Philosophy

in

Computer Science

by

Huy Nhut Hang

December 2014

Dissertation Committee:

Professor Michalis Faloutsos, Co-Chairperson
Professor Eamonn Keogh, Co-Chairperson
Professor Vassilis Tsotras
Professor K. K. Ramakrishnan

The Dissertation of Huy Nhut Hang is approved:

---

---

---

Committee Co-Chairperson

---

Committee Co-Chairperson

University of California, Riverside

## Acknowledgments

I thank my committee and my collaborators, without whose help, I would not have been here. I would also like to extend my gratitude to my dissertation advisor, Professor Michalis Faloutsos, who has taught me how to conduct research, how to write technical papers effectively, how to succeed outside of the academic environment, and last but not least, how to endure the challenges of the Ph.D. itself.

To my parents, who sacrificed so much to get me here today.

# ABSTRACT OF THE DISSERTATION

Developing Profiles of Malware and User Behaviors Using Graph-Mining and Machine Learning Techniques

by

Huy Nhut Hang

Doctor of Philosophy, Graduate Program in Computer Science
University of California, Riverside, December 2014
Professor Michalis Faloutsos, Co-Chairperson
Professor Eamonn Keogh, Co-Chairperson

The current fight between security experts and malware authors is an arms race. In this race, malware authors devise new attacks and exploits new vulnerabilities while the experts can only deflect the attacks and patch up the vulnerability after damage has been inflicted. Defending against miscreants is a difficult task precisely because experts do not know what attacks may come in the future. The ultimate goal of our work is to utilizing graph-mining and Machine Learning techniques to (a) develop profiles of user and malware behaviors and (b) detect anomalies and identify malicious actors. In this dissertation, we present three pieces of work that are aimed toward achieve that goal. The first is a graph-based approach designed to leverage P2P bots' behaviors to detect them when they lay dormant in the local network and wait for instructions from the botmasters. The second is a probabilistic algorithm based on the Stochastic Block Model that is designed to infer the group structure of users from their web browsing behaviors and leverage the group structure to detect when users in the network visit malicious websites. The third is an in-depth study of users'

exposure to web-based malware from the point of view of the malicious websites and the users themselves, where we explore the methods with which web-based malware spread and investigate their characteristics and temporal behaviors.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

The current fight between security experts and malware authors is an arms race. In this race, malware authors devise new attacks and exploits new vulnerabilities while the experts can only deflect the attacks and patch up the vulnerability after damage has been inflicted. Defending against miscreants is a difficult task precisely because experts do not know what attacks may come in the future. The ultimate goal of our work is to utilizing graph-mining and Machine Learning techniques to (a) develop profiles of user and malware behaviors and (b) detect anomalies and identify malicious actors. In this dissertation, we present three pieces of work that are aimed toward achieve that goal.

The first work is Entelecheia (presented in Chapter 2), a light-weight graph-based approach designed to detect the presence of P2P bots lying dormant in the network. A *botnet* is a group of compromised computers collective controlled by a *botmaster* that often engages in malicious activities for financial gain such as facilitating spam campaigns or performing Distributed Denial of Service (DDoS) attacks. At the height of its growth in 2007, the

spamming *Storm botnet* [24, 50] controlled upwards of twenty million computers and gained enough computational capability (in instructions per second) to rival a supercomputer [52]. In March 2010, Microsoft obtained a restraining order to take down the servers of the Waledac botnet, which infected hundreds of thousands of computers and was capable of sending between 1 and 2 billion spam messages per day [54]

Entelecheia works by leveraging the characteristics that the P2P bots exhibit during what we call their Waiting Stage: (a) the tendency to send "keep-alive" messages to their other peers to let the botmasters know of their status and (b) the tendency to have overlapping peers due to their bootstrapping processes. Our approach produces a median F1 score of 91.8%, in a very challenging setting: (a) no signatures available, (b) no initial seeding information, (c) P2P botnets, and (d) during the Waiting stage.

The second work (detailed in Chapter 4) is focused on the the task of protecting the cyber security of an organization, which has grown increasingly more important as hackers constantly find ways to infiltrate organizations to steal valuable information. There are two common methods with which malicious entities seek to accomplish this objective, and both begin with guiding the users to websites under the control of the hackers. In the first method, the websites would either deceive the users into installing some malware into their computers or quietly execute the installation without the users knowing. In the second method, called "phishing", the users would see a (fake) website which looks legitimate and asks for the users' credentials to continue (think the login page of the organization's webmail system).

We seek to address the problem by asking a key question: *"Can we detect when users in an organization visit suspicious websites only by knowing their history of browsing activities?"* Put differently, given that we know that a set of **persons** $P$ have visited a set of **websites** $W$ during a presumably sufficiently long period of time $T$, can we know which of the set of websites $W'$, which is seen during a subsequent time period $T{+}1$ but *not* during $T$, is suspicious and should be investigated?

We make three main contributions in this work. The first contribution is an in-depth study of the browsing behaviors of users. In this study we show that there is some measure of stability in the way the users browse the web in that users tend to visit the same set of websites over and over again and that we can make predictions, with moderate levels of certainty, as to which website a user is likely to revisit given his or her browsing behaviors during the previous week. With this study, we show that there is reason to believe that there may exists a stable group structure in a network of users, as there is already stability in the way each user interacts with the Web.

The second contribution we make in this work is Group Profiler, a probabilistic co-clustering approach based on the Stochastic Block Model that can also be leveraged to assess the "goodness" / "phishiness" of a website that is never seen before based entirely on the web-browsing history of the users in a network. We have stress-tested Group Profiler on many different synthetically created data sets and found that Group Profiler retains high accuracy even under more adversarial conditions.

Our third contribution is a method to measure the "goodness" of a never-before-seen website that is visited by a group of users given their browsing history. This measurement

method relies on Group Profiler itself to expose the inherent group structure between a group of users and the Web and then leverages this structure to assess whether a new website visited by a subset of the users conforms to the structure, thereby determining how suspicious it is. Our work is meant to be used by network administrators who want an automatic means to rank new websites each day in terms of goodness so that they can investigate one by one and, in case they discover a malicious website, can know quickly who have visited it and take necessary actions to protect the users.

The last piece of work (presented in Chapter 3) that is a part of this dissertation is an in-depth study of users' exposure to web-based malware. Distributing malware indirectly via web-pages has become a very popular way for spreading malware in the last 8 years. First, the previous technique of spreading malware via email attachments lost its effectiveness due to increased awareness and the use of email-based anti-virus tools. Second, new technologies have enabled lay persons to create sophisticated and interactive websites despite the fact that the site owners have limited understanding of technology and security. In 2012, Google published a blog post on the current growth of malicious websites and stated that they found 9,500 of them each day [42]. However, not all of these are websites created to be malicious. In fact, there are two main types of websites that spread malware: (a) the **born-malicious**, which are registered and operated by the malicious entities, and (b) the **compromised**, legitimate websites infiltrated by hackers and used as launching pads for their malware to get around spam filters and black lists.

Our key contribution in this work is an extensive study of user exposure to web-based malware following both a site-centric and user-centric point of view. We use two data

sets: (a) $D_{\text{ODB}}$, with roughly 66K malicious URLs collected from four online databases between December 2013 to September 2014, and (b) $D_{\text{WINE}}$, which captures visits to malicious websites from more than 500K users from January 2011 to August 2011 collected by the Symantec's WINE Project.

Our study can be summarized by the following key points: (a) compromised websites play a significant role in malware dissemination (33.1% of all domains we collected from online databases are compromised instead of born-malicious), (b) a malicious URL often distributes many different malware binaries but most malicious binaries are distributed by one URL, (c) most malicious URLs are short-lived, but 10% of them are active for more than three months, (d) many URLs exhibit the same bursty temporal pattern (which we call the "Space Needle" pattern) aligned with a campaign-like behavior, and (e) the distribution of the visits to malicious sites per user is skewed and can be described by a power law of exponent $-\frac{1}{2}$

# Chapter 2

# Entelecheia: Detecting P2P botnets in their Waiting Stage

## 2.1 Introduction

The ability to detect botnets is a crucial component of a network's security system. A *botnet* is a group of compromised computers collective controlled by a *botmaster* that often engages in malicious activities for financial gain such as facilitating spam campaigns or performing Distributed Denial of Service (DDoS) attacks. At the height of its growth in 2007, the spamming *Storm botnet* [24, 50] controlled upwards of twenty million computers and gained enough computational capability (in instructions per second) to rival a supercomputer [52]. In March 2010, Microsoft obtained a restraining order to take down the servers of the Waledac botnet, which infected hundreds of thousands of computers and was capable of sending between 1 and 2 billion spam messages per day [54].

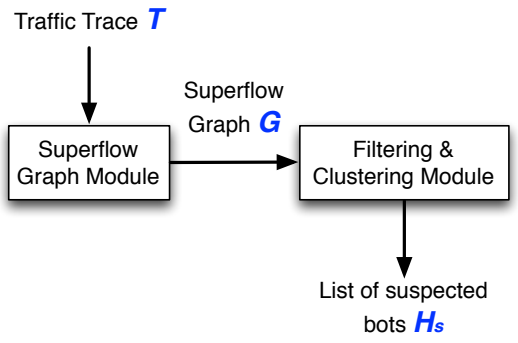



Figure 2.1: Entelecheia has 2 modules: *Superflow Graph* and *Filtering & Clustering*.

Botnets have two interesting dimensions that are relevant in our work: (a) botnet architecture and (b) botnet lifecycle.

**a. Botnet architecture:** There are two fundamental approaches in botnet architecture: (a) the centralized approach, which uses Command and Control (C&C) channels such as Internet Relay Chat (IRC) to receive instructions from a single source, (e.g. R-Bot, Spybot, or Gaobot), and (b) the decentralized approach, which utilizes a peer-to-peer protocol to coordinate its operation (e.g. Storm and Nugache). The decentralized (or P2P) approach offers higher resiliency and the botnets implementing it are harder to both detect and take down.

**b. Botnet lifecycle:** There are different stages in the life-cycle of a P2P bot [53]: (a) *Infection*, in which the bot spreads, via email attachments, drive-by-downloads, etc; (b) *Rally*, where the bot bootstraps itself by connecting with a peer list; (c) *Waiting*, where the bot waits for the botmaster's command; and (d) *Executing*, in which it actually carries out a command, such as a DoS attack. Each stage has its own distinct behavior, as such customized solutions for each stage can lead to more effective solutions.

We focus on the problem of detecting P2P botnets at the *Waiting* stage with two requirements: (a) we assume no signatures, or equivalently assume that the P2P bot has not been seen before, and (b) we assume no seed information through a blacklist of IPs. The *Waiting* stage is rather "quiet" compared to the other stages, where more aggressive

communications takes place. During this particular stage of the cycle, bots are known to maintain infrequent and low-intensity communications with their peers compared to other traffic [57]. It is this behavior that we quantify and exploit in our solution.

To the best of our knowledge, no previous work addresses the aforementioned problem in the challenging context described above. Very few efforts focus on detecting P2P botnets in their *Waiting* stage [21, 16]. The most relevant work is BotMiner [21], which uses signatures and statistical properties to detect botnets to in both Executing and Waiting stages. However, it needs signature analysis and deep-packet inspection, which reduces its effectiveness when signature-matching is not an option, as per one of our problem requirements. Other efforts detect botnets at the more visible *Executing* stage [59, 30, 51], which dovetails with the detection of DoS attacks [25]. Some earlier efforts focus on detecting centralized botnets [11, 38] and modeling the behavior of botnets at different stages [22, 21]. There has been much work on application classification, which is a more general problem. We provide a detailed discussion of previous work in Section 2.6.

We propose Entelecheia,[1] an approach for detecting peer-to-peer botnets during their *Waiting* stage by exploiting their "social" behavior. The driving insight for our work started with this question: *Can we detect botnets by focusing on long-lived and low-intensity flows?* Despite their intuitive appeal, traditional anomaly detection tools such as [46] introduce many false positives. To overcome this problem, we take advantage of the information hiding in the network-wide interactions of the nodes. Once we create the appropriate graph, we develop graph-mining techniques on it to detect bots. Our approach produces a median

[1]The name stands for **EN**trap **T**reacherous **ELE**ments through **C**lustering **H**osts **E**xhibiting **I**rregular **A**ctivities.

F1 score[2] of 91.8%, in a very challenging setting: (a) no signatures available, (b) no initial seeding information, (c) P2P botnets, and (d) during the Waiting stage.

Apart from being a stand-alone solution, we envision our approach as an essential component in a network administrator's toolset against botnets.

Our key contributions are summarized below.

**a. A novel graph-based behavior-focused approach.** We introduce Entelecheia, an effective approach for detecting bots, the power of which lies in synthesizing: (a) an intuitive observation on how bots operate, (b) the creation of graphs that captures long-lived low-intensity flows, through the definition of Superflows, and (c) two synergistic graph-mining steps to cluster and label botnet nodes. First, we introduce the concept of Superflows, one of the novelties of which is the focus on the interactions between IP addresses only, i.e., using a 2-tuple, instead of the 5-tuple of the common definition of flows. Second, once the Superflow graph is created, we exploit the homophily inherently present in the botnet traffic by identifying clusters of nodes which exhibit high communication persistence.

**b. Entelecheia detects botnets effectively**. We evaluate Entelecheia on real-world network traces that contained Storm and Nugache [24, 50]. Our approach delivers a median F1 score of 91.8% on a dataset of 20 distinct network graphs, as we explain in Section 2.4. In fact, the median precision rate that Entelecheia obtained is 98.1%, which shows how accurate Entelecheia is.

**c. Entelecheia is robust to variability and statistical "noise".** We show that Entelecheia is fairly robust for a wide range of parameter values, natural variability,

[2]The F1 score is the harmonic mean of Precision and Recall rates. We will show how to calculate all three metrics in Section 2.4.

and statistical noise, all of which are important for a practical deployment. Table 2.1 shows the recommended values and the ranges of the parameters used in our experiment

This work has been published in the IFIP Networking Conference in New York, May 2013 with Xuetao Wei (University of Cincinnati), Michalis Faloutsos (University of New Mexico, Albuquerque), and Tina Eliassi-Rad (Rutgers University).

## 2.2 Superflows: Definitions and Virtues

We introduce *Superflows,* which helps us hone in on the botnet behavior. Recall that a flow in the traditional sense is defined by a time-stamped 5-tuple: `<srcIP, dstIP, srcPort, dstPort, protocol>`. Intuitively, a Superflow is the maximal group of flows that satisfy the following two criteria: (a) they are between the same pair of nodes irrespective of port numbers or protocols and (b) they are *close in time.* Two flows are *close in time* if their starting and ending times are within $T_{gap}$, the **interflow gap** parameter.

Figure 2.2 demonstrates how Superflows are formed in an intuitive manner. There are four steps to how Superflows are constructed for a given trace:

1. We identify flows using the standard definition of a flow.
2. In the beginning, every flow is considered as a trivial Superflow.
3. We merge two Superflows into a larger Superflow when both of the following clauses are satisfied:
    - **Clause 1**: The two Superflows are between the same two nodes regardless of their protocols and port combinations.

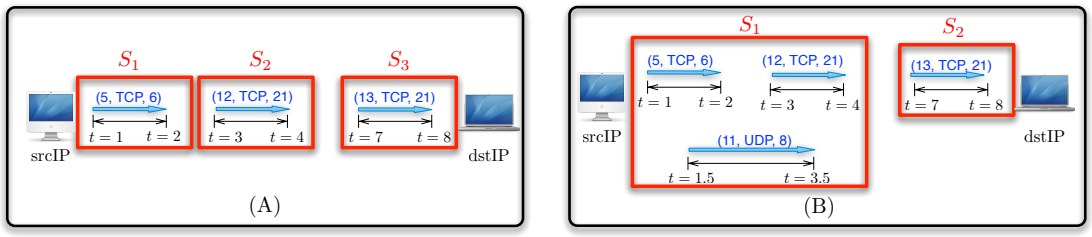
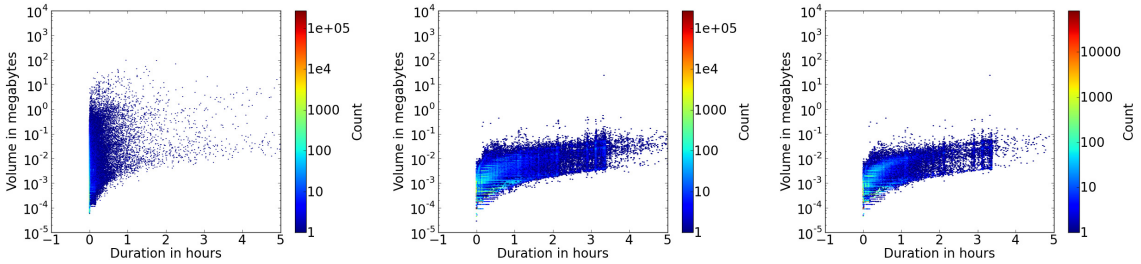



Figure 2.2: The 3-tuple above each flow indicates, in order, its source port, protocol, and destination port. **Given an allowable gap of 0.5 seconds between the start and end times of the flows**: (A) The three flows become trivial Superflows. (B) The first three flows can be merged even though their source ports, protocols, and destination ports are not identical.




(a) 5,000 hosts in WIDE (b) 13 Storm hosts (c) 82 Nugache hosts

Figure 2.3: Botnet behavior exhibits long-duration and low-intensity as evident in (b) and (c). These scatterplots show duration versus volume of Superflows in real data traces of backbone (a: WIDE) and botnet-only traffic (b: Storm) and (c: Nugache).

- **Clause 2**: The starting time of the second (later) Superflow is within the gapinterval, $T_{gap}$, from the end of the first (earlier) flow.

4. We repeat Step 3, until there are no more Superflows to merge.

Consider two Superflows, $S_1$ and $S_2$, between the same two nodes with starting ($s$) and ending ($e$) times: $s_1$, $e_1$, $s_2$, $e_2$. The duration of $S_1$ is $d(S_1) = e_1 - s_1$ and the duration of $S_2$ is defined likewise. For illustration purposes, let us assume that $S_1$ starts before $S_2$.

The second Clause shown above can be represented by the condition: $s_2 - e_1 \leq T_{gap}$ where $T_{gap} \geq 0$.

Similarly, we define $v(S_1)$ to be the volume (in megabytes) of the Superflow $S_1$. If $S_1$ can be merged with $S_2$, the volume $v(S)$ of the resulting Superflow $S$ is equal to $v(S_1) + v(S_2)$. Note that the starting time of the resulting Superflow $S$ is now $s_1$ and the ending time is now $e_2$. It also does not matter if $S_2$ starts before $S_1$ ends because in such a case $s_2 - e_1 \leq 0$ and by definition, $0 \leq T_{gap}$.

We next discuss why standard flows are less effective in detecting botnets and present the virtues of Superflows.

***a. The commonly-defined flows are less effective in detecting botnets.*** Previous work [57] showed that using only standard flows does not provide high detection rates (only 87.5% for Storm and 34.8% for Nugache). We will demonstrate in Section 2.4 that using Superflows raises detection accuracy significantly.

Another drawback of standard flows is that in practice flows are not uniquely defined. For example, different flow extractors–such as CAIDA's CoralReef, Wireshark's TShark, CERT's YAF–generate different sets of flows on the same trace because they have different definitions for a standard flow. By grouping flows into Superflows, we remove some of this variability and increase robustness.

***b. Superflows capture the behavior of botnet traffic.*** Figure 2.3 shows three heat maps. Each dot on a heat map is a Superflow, its X coordinate is its duration in hours and its Y coordinate is its volume in megabytes. Figure 2.3(a) shows the scatterplot of the

Superflows of a randomly selected 5000 hosts in the WIDE traffic traces. Figure 2.3(b) and 2.3(c), respectively, show the Superflows of the 13 Storm hosts and the 82 Nugache hosts.

Figure 2.3 illustrates that regular hosts tend to produce Superflows with short durations and high volumes while malicious hosts generate Superflows that live for hours and have very low volumes. Opportunistic P2P bots communicate with their peers as soon as the former comes online for as long as possible in anticipation of commands from the botmasters. Unsurprisingly, they would periodically send out keep-alive messages to inform their peers that they are online and awaiting instructions.

***c. Superflows circumvent a bot's effort to change its communication ports***. Upon inspection of the botnet traffic, we saw that unlike the Storm bots, which reused the same source-destination port combination, the Nugache ones changed theirs. So, if we simply grouped flows according to their classic 5-tuple, we would not be able to observe long-lived flows from the Nugache bots because their flows would be put into different groups due to their different source-destination port combinations. Using the full 5-tuple would not affect the detection rate of Storm bots, but it would hinder the effort of identifying the Nugache ones.

Using only the tuple `<srcIP, dstIP>` as the key to flows is a more natural and accurate manner to group the flows together because it bypasses a bot's attempt to change its ports—like Nugache does—and gets to the core of the communication itself: *who is talking to whom?* Furthermore, when a legitimate long-lived, low-volume flow occurs, it is likely to be the result of a *keep-alive* communication between a machine and a server (such as a MSN Live Messenger server). These are continuous TCP flows where the protocol and

the source and destination ports do not change. In such cases, source and destination IPs are enough to uniquely identify them.

***d. Automating the $T_{gap}$ parameter value selection.*** The value of the gap, $T_{gap}$, is crucial for using Superflows in practice. As a rule-of-thumb, the value for $T_{gap}$ should not be so large as to unreasonably consolidate every flow into a Superflow and not so small where flows are not merged into Superflows at all. We automate the selection of $T_{gap}$ by setting it to the median interflow time of the data trace. We chose the median because an inspection of the distribution of Interflow Times showed us that the existence of rare but extremely large values skewed the value of the mean and rendered it unrepresentative of the behavior of Interflow Times.

Specifically, given two flows $S_1$, $S_2$ in **Step 4**, we say that the Interflow Time between them is $IT(S_1, S_2) = s_2 - e_1$, where $s_2$ is the starting time of the second flow and $e_1$ is the ending time of the first. If $S_2$ starts before $S_1$ ends, $IT(S_1, S_2) = 0$.

To select a value for $T_{gap}$, we collect all Interflow Times between all pairs of successive flows between every pair of communicating hosts and use the **median** value for $T_{gap}$. This way, $T_{gap}$ is **not** a **free** parameter, as its value adapts to the data trace at hand.

## 2.3 Entelecheia: Detecting P2P botnets

Our approach for detecting botnets can be decomposed into two modules that work synergistically to (a) model the network-wide interactions of the hosts in the network as a graph, (b) focus on likely botnet activities, and (c) identify clusters of potentially infected machines. We provide an overview of the modules below.

**1. Superflow Graph Module:** This module is responsible for creating the Superflow Graph in which each node represents a host in the network and each edge is a vector of attributes that summarizes *all of* the Superflows sent and received between a pair of source and destination IPs. This Superflow Graph represents the *social communication behavior* between hosts.

**2. Filtering and Clustering Module:** This module first filters out high-volume edges since it is unlikely that they represent Waiting-stage botnet activities. As Figure 2.3 showed, the volumes of the Superflows from infected hosts tend to be much lower than those from benign ones. By design, P2P bots would form a web of peers among whom the connections are long-lived. If we examine the Superflow Graph and assign as edge weights the total duration of the Superflows ($\alpha_d$) between the two nodes then clustering nodes together by their connectivity would group infected hosts into communities with a high percentage of long-lived edges.

Below are the detailed descriptions for the aforementioned modules:

### 2.3.1 Superflow Graph Module

The input for this module is a network trace $N = \langle H, F \rangle$ where $H$ is the set of all hosts $h_i$, for $i = 1, 2, ..., |H|$ in the network and $F = \{f_1, f_2, ..., f_{|F|}\}$ is the set of all conventionally defined 5-tuple *flows*. The output is the Superflow Graph, which represents node-interactions with respect to certain attributes of interest. Algorithm 1 presents the pseudo code for this module. Below, we specify the attributes of interest and offer rationale for their selection.

**Algorithm 1** Superflow Graph Module.

**Require:** Trace $N = \langle H, F \rangle$, $T_{gap}$.

1: $E \leftarrow \{\}$

2: **for all** pairs of communicating hosts $h_i$ and $h_j$ **do**

3: Extract $F_{ij}$, the flows between $h_i$ and $h_j$

4: Construct $\mathcal{S}_{ij} = \{S_1^{ij}, S_2^{ij}, ..., S_l^{ij}\}$, the set of Superflows originating from $h_i$ to $h_j$, using the $T_{gap}$ parameter.

5: $\alpha_v \leftarrow \sum_{k=1}^{l} v(S_k^{ij})$

6: $\alpha_d \leftarrow \sum_{k=1}^{l} d(S_k^{ij})$

7: *edge* $e_{ij} \leftarrow < h_i, h_j, \alpha_v, \alpha_d >$

8: $E \leftarrow E \cup \{e_{ij}\}$

9: **end for**

10: **return** $G_S = \langle H, E \rangle$

**1) Identifying Superflows.** We parse the given network trace $N$ to aggregate flows into Superflows. This process involves: (a) extracting the set of flows $F$ from the data trace, and (b) merging them into Superflows using $T_{gap}$ as per the definitions given in the previous section.

**2) Reporting Superflow properties.** At the moment, we focus on two Superflow attributes that are sufficient to enable effective botnet detection: *Volume* and *Duration*. Intuitively, the Volume $\alpha_v$ of an edge between any pair of nodes is the sum of volumes (of the Superflows) between those two nodes throughout the traffic trace. The Duration $\alpha_d$ of an edge between any pair of nodes is the sum of all durations (of the Superflows) between

those two nodes throughout the traffic trace. Given that there are $n$ Superflows between hosts $A$ and $B$, the Volume and Duration attributes of the edge $AB$ are respectively:

$$\alpha_v = \sum_{i=1}^{n} v(S_i), \qquad \alpha_d = \sum_{i=1}^{n} d(S_i)$$

where $v(S_i)$ is the volume of the Superflow $S_i$ and $d(S_i)$ is its duration[3].

In addition, one could expand the number of attributes to more than Volume and Duration. In a more general form, Line 7 of the algorithm can be rewritten as $e_{ij} \leftarrow \langle h_i, h_j, A \rangle$, where $A$ is a vector of attributes of interest, which could include one attribute for the total number of packets, another for packet size variation, etc.

### 2.3.2 Filtering and Clustering Module

Algorithm 2 explains the steps for the Filtering and Clustering module. At a high level, the algorithm operates in three steps:

**a. Filtering out edges in the Superflow Graph that are unlikely to be botnet activity**. In this step, we want to focus on edges with the profile of P2P-bot behaviors. We only use the volume attribute $\alpha_v$ in this stage, although other attributes of interest can be utilized. In our initial experiments, we have examined other selection methods such as first selecting on duration and a combination of volume and duration, but those alternatives did not produce any performance gain.

Even when considering only volume, the selection process can be predicated on other attributes. For instance, one could envision different thresholds for TCP and UDP

[3]See Section 2.2

| Symbol | Name | Recommended value | Range |
|---|---|---|---|
| $T_v$ | Volume threshold | 0.1 MB | 0.01MB $\leq T_v \leq$ 0.1MB (Figure 2.3) |
| $T_d$ | Duration threshold | 1 hour | 1 sec $\leq T_d \leq$ 1 hour |

Table 2.1: The parameter ranges in our experiments. See text for selection procedure.

connections. More in-depth discussion on how to select a value for $T_d$ and how its effectiveness varies according to the selection can be found Section 2.4.

**b. Clustering.** The choice of which graph-clustering (a.k.a community discovery) algorithm to use is a parameter into our approach, given that many algorithms already exist in the literature [19]. We selected the Louvain algorithm [12] to cluster the Superflow Graph because (a) its notion of a community (with high intra-link density) is well-suited for botnet detection; (b) it is parameter-free; and (c) it is computationally efficient with runtime proportional to $|V| \log |V|$, where $V$ is the set of nodes in the graph. Louvain outputs a set of communities $C = \{c_1, c_2, ..., c_p\}$, where *community* $c_i = \langle H_{c_i}, E_{c_i} \rangle$. $H_{c_i}$ is the set of hosts and $E_{c_i}$ is the edges exclusively within the nodes of cluster $c_i$. In this work, we use the words *community* and *cluster* interchangeably.

**c. Identifying suspicious clusters and nodes.** At this step, the algorithm decides whether a cluster is likely to contain a group of infected hosts by examining the Duration attributes of the edges within the cluster. For each cluster $c$, we collect all edges $e$ whose $\alpha_d^e \geq T_d$ into the set $Suspect_c$. We next divide the cardinality of $Suspect_c$ by the cardinality of $E_c$, which is the set of all edges found exclusively inside the community $c$. This gives us a ratio $r$ that quantifies the presence of long-lived edges in the community.

We say that the cluster $c$ is suspicious if $r > R$. Recall that $R$ is the percentage of long-lived edges as determined by the parameter $T_d$ over the entire Superflow Graph $G_S$. In

**Algorithm 2** Filtering and Clustering Module.

**Require:** Superflow Graph $G_S = \langle H, E \rangle$, Volume threshold $T_v$, Duration threshold $T_d$.

{**Select low-volume edges**}

$E^{select} = \{\text{edge } e \in E\text{: Volume(e)} \leq T_v \}$

$R \leftarrow$ Ratio of edges $e$ in $G_S$ where $d(e) \geq T_d$

{**Cluster the selected graph using *Louvain***}

$C \leftarrow \textbf{cluster}(G(H, E^{select}), \text{Louvain})$

$H_s \leftarrow \{\}$

{**Find suspicious clusters**}

**for all** $c = \langle H_c, E_c \rangle \in C$ **do**

  $Suspect_c = \{e \in E_c : Duration(e) \geq T_d\}$

  {**Calculate percentage of long-lived edges**}

  $r \leftarrow \dfrac{|Suspect_c|}{|E_c|}$

  **if** $r > R$ **then**

    $H_s \leftarrow H_s \cup \{nodes\ of\ edges\ in\ Suspect_c\}$

  **end if**

**end for**

**return** $H_s$

our experiments, we varied the value for the threshold $R$ by multiplying it with a ***modulator*** value ranging from 1.5 to 3 to account for statistical variations. While increasing $R$ in this particular fashion improves the precision of our method by a very small amount, the recall

rate decreases gracefully as the value of the modulator rises. More details can be found in Section 2.4.

*Careful identification of bots within a cluster.* Our initial approach labeled all nodes in the suspicious clusters as malicious. This choice produced many false positives. So, we modified our approach to a two-step process for identifying likely P2P bots in a suspicious cluster. First, we identify suspicious clusters, and within those clusters, we label nodes as malicious only if they transmitted or received long-lived flows.

## 2.4 Experimental Evaluation

This section is divided into three parts where we introduce the setup of our experiments as well as the datasets we use, the two baseline approaches we evaluated to provide contrast with Entelecheia, and finally the effectiveness of Entelecheia itself. In a nutshell, we observe the following:

**(i)** Our approach is highly accurate (with a median F1-score of 91.8%) in detecting botnets during their *Waiting* stage.

**(ii)** Using the 2-tuple is critical for high detection accuracy.

**(iii)** Our approach is robust for a wide range of parameter values and statistical "noise".

### 2.4.1 Experimental Setup

**Baseline algorithm: 2-tuple and 5-tuple versions.** In our evaluation, we use two reference methods as baselines; and determine whether detecting bots is as easy as simply classifying all long-lived low-intensity connections as botnet traffic.

Our baseline algorithm works as follows. Given a duration threshold and a volume threshold, we label a host as a P2P bot if it produces a Superflow $S$ that is longer than the duration threshold and smaller in volume than the volume threshold. We use two variations of the baseline algorithm, which differ on the communication information used: 2-tuple vs. 5-tuple flows. The **2-tuple** Baseline approach operates on Superflows as defined in Section 2.2. The **5-tuple** Baseline approach operates on 5-tuple flows that are aggregated in the time dimension similar to Superflows (see Section 2.2 also).

In this study, we evaluate the following three methods: (a) the 5-tuple Baseline approach, (b) the 2-tuple Baseline approach, (c) and Entelecheia

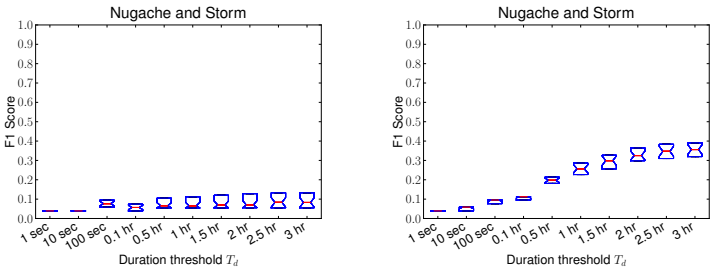
**Parameter selection.** Table I lists our parameters (namely, $T_v$ and $T_d$), their definitions, their recommended values, and the range of values tested for each parameter. To select appropriate values for our parameters, we conducted extensive experiments. We use $T_v = 0.1$ MB for defining volume-intense flows, which proved to be most effective in our experiments. With respect to the duration threshold $T_d$, we provide results for various values ranging from 1 second to 3 hours. $T_d = 1$ hour balances precision and recall rates, offering very high performance results.

**Datasets and Analysis Tool**

***WIDE:*** From the MAWI Traffic Archive [5], we downloaded a trace of 24 continuous hours from a Trans-Pacific backbone line between the U.S. and Japan on 03/03/2006 (sample point B). The IPs are anonymized and no packet payloads available. There are 3,528,849 unique IPs and 82,380,945 flows. 89% of all flows are TCP and the rest are UDP. We chose the WIDE dataset because (a) it is freely available online and (b) the length of the dataset spans an entire 24-hour period, which gives us the ease of integration with the Storm and Nugache botnet traces.

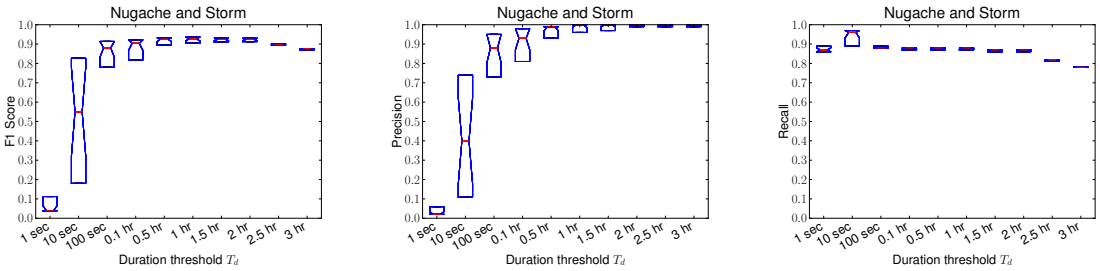
**Storm and Nugache traces:** The real-world malicious network traces that we used in our experiments contain observed data from 13 hosts infected with Storm and 82 hosts with Nugache during a period of 24 hours. The IPs in these traces, which are the same ones used in [21], are not anonymized. In these bot traces, all spamming activities have been blocked to avoid liability issues [57]. As a result, we only see the communication traffic between the bots as they exhibit social behaviors more closely associated with their *Waiting* phase than an active stage of propagation or attack.

**Analysis tool:** We utilize ARGUS [1] (with its default parameter settings) to process and aggregate packets into flows. We chose this tool because it is widely used and frequently maintained by the developers. Recall that we aggregate flows into Superflows, rendering our approach less vulnerable to the fact that different flow extractors may produce different amounts of flows due to their own definition of a flow and various parameter settings.




(a) F1 Score, Baseline, 5-tuple (b) F1 Score, Baseline, 2-tuple

Figure 2.4: F1-score versus $T_d$ (the threshold of what constitutes a long-lived communication) for the two Baseline approaches that use the 5-tuple and 2-tuple definitions. $T_v = 0.1$ MB throughout all experiments.




(a) F1 Score (b) Precision (c) Recall

Figure 2.5: Entelecheia's Precision, Recall, and F1 scores.

**Generating evaluation traces**

We created traces for our experiments by merging the botnet and the backbone traces. At a high level, the process may sound simple; but there are subtleties that must be addressed. In this section, we explain our process and assumptions to make the detection as unbiased and challenging as possible for stress-testing our method.

**Overview:** We generate 40 different graphs—each comprising of 100-200K nodes—and partition them into 2 datasets: $D_1$ and $D_2$, each with 20 graphs. Our goal is to

identify 91 malicious nodes[4] among 5000 benign nodes while making sure that we respect the bipartite property of the WIDE trace.

**a. Respecting the bipartite nature of the graphs.** In overlaying the malicious traces on the WIDE traces, we have to respect their bipartite nature since there are two clear sides in each dataset. A practical deployment of Entelecheia will most likely be on a network of the same property. We envision a detection algorithm to run at a firewall or a backbone link, so we have taken extra care to map all the nodes of each bipartite component to one bipartite component.

**b. Removing the "extra" connectivity from the bot traces.** Since the initial botnet traces contain both internal and external edges[5], the graph representing those traces is not bipartite. In our experiments, we remove the internal edges to ensure that the graph formed by overlaying the malicious traces on the WIDE ones remains bipartite. Note that by doing so, we make the detection more difficult for our algorithm, but more similar to an actual deployment of our solution.

**c. Sampling the initial trace.** Since the backbone trace is large, our first thought was to select a random subset or a time-based interval. Instead, we decided to ensure that the benign graph is connected. The rationale is simple: the more disconnected the graph of the benign nodes is, the easier it is for our algorithm to discover strongly connected communities of malicious nodes. It is for this reason that we need to sample a

[4]There is an overlap between the 13 Storm nodes and 82 Nugache nodes.

[5]Internal edges represent the connections among the infected nodes. External edges are the connections that go out to the rest of the Internet.

sufficiently large trace from the initial WIDE traces that would form a graph in which the nodes are connected.

To accomplish this, we start from a random node in the graph representing the full 24-hour WIDE traces and collect nodes via a breadth-first search until we obtained at least a set of 5000 distinct nodes on the same side of the graph to preserve the trace's bipartite nature. The resulting evaluation trace is comprised of all flows in the 24-hour period of the original trace that involve those 5000 nodes and their neighbors. This typically leads to the formation of a graph with a population of between 100K-200K nodes. Typically, the observed average node degree of a sampled graph is about 3 per node, but there is some variation.

This host selection process is repeated for a total of 40 times (starting with a different random node each time) to produce a set of 40 different traces. Each trace is then overlaid with the botnet traffic in such a way that the known malicious hosts are mapped randomly on the aforementioned 5000 nodes so that the bots are always on the same side of the resulting network graph. Half of those 40 traces ($D_1$) are used as a training set to help us fine-tune Entelecheia and select appropriate values for its parameters. The other half forms the dataset $D_2$ that is used to test the algorithm's performance. Each data point (or box) on any boxplots thus represents the median and 95% confidence interval of 20 runs unless otherwise stated. We use **Pr**ecision, **Re**call, and **F1**-scores as our evaluation measures where:

$$\mathrm{Pr} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathbf{FP}} \quad \mathrm{Re} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathbf{FN}} \quad \mathrm{F1} = 2 \times \frac{\mathrm{Pr} \times \mathrm{Re}}{\mathrm{Pr} + \mathrm{Re}}$$

TP stands for True Positives, FP for False Positives, and FN for False Negatives.

There are two assumptions that we made for the evaluation of Entelecheia:

**Assumption 1**: For our validation, we only count as True Positives the nodes that we identified as bots who are among the 13 Storm and 82 Nugache bots even though their direct neighbors in the botnet traces are likely to be malicious hosts themselves.

**Assumption 2**: Given Assumption 1, we further assume that all the nodes in WIDE are benign and report as a False Positive the classification of any such node as a P2P bot.

We revisit Assumption 2 later in Section 2.4.3.

### 2.4.2 Observations on the Baseline Approaches

Figure 2.4 shows the performance of the two baseline approaches on $D_1$. Each data point represents the median of 20 runs and the box indicates the median of the top half (upper quartile) and the median of the lower half (lower quartile). The slanted lines on the boxes indicate the 95% confidence interval around the median.

**Observation 1:** Using the 2-tuple level of aggregation for Superflows captures botnet behavior more accurately than the 5-tuple. One reason is that using the 5-tuple yields a much lower Recall rate since the Nugache worm changes its source and destination port combinations quite often.

**Observation 2:** The Baseline approaches perform poorly on $D_1$ for every value of the duration threshold. The F1-score for the 2-tuple Baseline approach never exceeds 40%. Compared with the excellent performance of Entelecheia (see Figure 5), this result yields two conclusions. First, there exists a large amount of long-lived communications in

the WIDE trace. Second, simply flagging long-lived low-volume Superflows is not sufficient to detect infected hosts.

### 2.4.3 Evaluating Entelecheia

**Running Entelecheia on $D_1$.** We first evaluate the performance of Entelecheia with various values for the parameter $T_d$ to obtain the best range of values for it. Figure 2.5 shows the classification performance versus the duration threshold $T_d$ that defines a long-lived Superflow.

We observe the expected tradeoff between Precision and Recall. Precision improves as the time threshold $T_d$ increases, while Recall decreases. This observation further confirms that P2P bots form communities with long-lived, low volume connections. We select $T_d = 1$ hour as the recommended value for $T_d$ since it is in the middle of a plateau where the performance is stable.

**Running Entelecheia on $D_2$.** Having picked the value for $T_d$ from our experiment on the dataset $D_1$, we now run Entelecheia on the second set of graphs, $D_2$ to evaluate its performance. The results are shown in Figure 2.6. Note that the median Precision rate is 98.1%, which shows that Entelecheia delivers highly accurate results.

**The *True* False Positives: Detecting existing botnet traffic.** We investigated Entelecheia's False Positives, namely the source-destination pairs flagged as suspicious, which did not correspond to injected botnet flows. We observed that the flows were directed toward two TCP ports of the destination IPs: 139 and 445. Interestingly, these two ports are associated with security risks on Windows systems. First, threats associated with port 130 include the IRC centralized Spybot, while worms, such as Sasser, compromised systems

via port 445 when it is left open. We are convinced that these FPs are of other previously unknown malicious activities, and argue that the reported accuracy would be higher if we had ground truth for the trace. In the experiment with the **largest** number of False Positives, we have 26 FPs initially, from which 6 (or 23%) are manually verified to be bots or engaging in malicious activities.

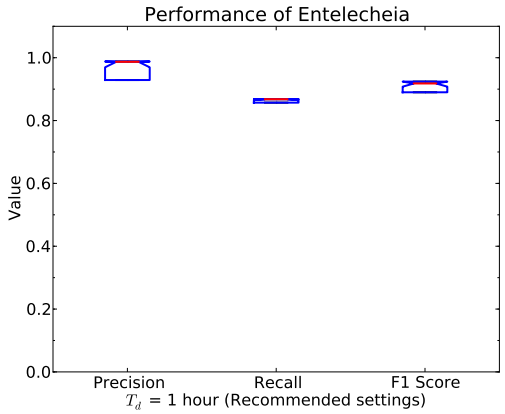



Figure 2.6: Performance of Entelecheia on the Test set ($D_2$): 20 new graphs. Note that the median Precision rate is 98.1%.

**The False Negatives.** Upon further inspection, we noticed that six of the bots, (6.6% of all bots), that escaped Entelecheia produced very few flows. This is due to two reasons. First, most of their peers were *not* online. Second, the ones that *were* online communicated with the other bots for less than an hour before they went offline. This in turn created short-lived Superflows, which is the reason why Entelecheia could not detect them.

We argue that these infected machines, due to their being disconnected from their peers, are not in the Waiting stage and their having eluded Entelecheia does not undermine Entelecheia's effectiveness because of two reasons: 1) Most of their peers were not online and

2) the ones that were online communicated with the bots for less than an hour before they went offline (the average volume for all of them during the day is 55 bytes). The bots, as a result, produced short-lived Superflows, which is the reason why Entelecheia couldn't detect them. We argue that these infected machines, due to their being disconnected from their peers, are not in the Waiting stage and their having eluded Entelecheia does not undermine Entelecheia's effectiveness.

## 2.5 Discussion

Here we discuss several deployment issues, limitations, and ways to improve our approach.

**a. Evading Entelecheia:** It is interesting to consider how a botmaster could avoid detection by Entelecheia. Such a case would mean a combination of the following three approaches: (a) non-collaborative operation, (b) short-lived connections, or (c) high-volume connections. First, abandoning the non-collaborative operation would be a triumph for Entelecheia, given that decentralized P2P botnet emerged as the answer to the problem posed by the methods designed to expose centralize botnets. Second, changing the last two properties increases the risk of their behavior being caught by an anomaly detection algorithm, which observes changes in the behavior of nodes. In addition, the collaborative operation depends strongly the long-lived connections between peers, therefore it is difficult to change this behavior.

**b. Deploying Entelecheia: practical tips and limitations.** Our approach has only two and quite intuitive parameters, such as the thresholds for duration and traffic

volume. We are confident that the proposed values provide a good starting point and some minor fine-tuning could be beneficial depending on the deployment location and properties of the network. The approach is lightweight as it operates at the packet header level and retains only high level information of the interaction (as opposed to signature based methods that require expensive deep packet inspection capabilities).

In a practical deployment, we envision our approach as a component in a comprehensive security system. For instance, it could serve as the seeding phase for other monitoring tools and approaches. More precisely, Entelecheia can be deployed to identify an initial set of bots, which will be used by another module to extract signatures.

While the anonymized and payload-free WIDE traces prevent us from knowing whether P2P traffic already exists there, we argue that legitimate P2P traffic are high-volume in nature and will be filtered out before the Clustering step. We believe our approach will still work with centralized botnets because their traffic will simply form easy-to-detect clusters with high-duration and low-volume flows.

## 2.6 Related Work

BotMiner [21] is the closest work to our research. It identifies botnets by looking at the bots' activities during different phases of its life cycle. BotMiner has two main components: 1) a C-Plane monitor that converts flows into combined, vectorized records (c-flows) and clusters the records to capture command-and-control traffic and 2) an A-Plane monitor built on the Snort engine[46] that clusters malicious activities (such as spamming and scanning) through Deep-Packet Inspection and signatures. Entelecheia has two advantages over

BotMiner. First, Entelecheia does not require signature examination. Second, Entelecheia captures temporal information in the flows, which BotMiner's c-flows do not. Third, Entelecheia does not rely on port information in the flows, which can be easily manipulated by botmasters.

In [36], Li et al. presented their analysis on the prevalence of scanning/probing patterns of bots and presented a statistical approach designed to detect malicious hosts when they are actively probing other hosts in order to discover vulnerabilities. This can be considered a detection method for the *Executing* phase. Entelecheia instead focuses on the *Waiting* one.

In [57], Yen and Reiter introduced the concepts of Traders (genuine P2P users) and Plotters (compromised machines). They presented a way to distinguish the two, which uses the classical 5-tuple definition of a flow. Their approach achieves a Storm-bot detection rate of 87.5% and Nugache-bot detection rate of 34.8%. We on average achieve a 100% detection rate for Storm and 87% for Nugache.

Zhang et. al. [58] proposes a scheme to separate stealthy P2P *attack* traffic from benign P2P ones by utilizing flow statistics to create statistical fingerprints for flow clusters. They do not use connectivity in their clustering step, presumably because it would complicate the process and makes it harder to separate P2P traffic from other types. Entelecheia utilizes *both* flow statistics and connectivity to deliver accurate detection results.

Nagaraja et al. [43] propose a probabilistic detection scheme that partitions the graph into subgraphs and designate clusters with more homogeneous probability values as malicious. Entelecheia uses the Superflow graph and finds communities based on the

standard notion of a community in social networks, which is more analogous to P2P botnets social in nature.

Graph-based approaches have successfully been used in problems where the goal is to identify the applications that generate traffic [28, 29, 26, 27]. Entelecheia focuses on a different problem, namely, detecting botnets in their Waiting stage.

## 2.7 Conclusion

We propose a novel, simple, intuitive, yet effective approach for detecting P2P botnets during their Waiting stage by identifying their "social" behavior. We operate under the following two requirements: (a) we assume no signatures or prior knowledge; and (b) we assume no seed information through a blacklist of IPs. Our key insight is to exploit the inherent behavior of botnets by examining long-lived and low-intensity flows. Identifying what constitutes as a long-lived and low-intensity flow is not a simple task, as we demonstrated through two baseline algorithms.

The algorithmic novelty of Entelecheia is two-folds: (a) the concept of **Superflow** (which filters out flows that are unlikely to be malicious) and (b) a graph-based behavioral approach (which employs clustering techniques). We stress-test our approach using real botnet traces from the Storm and Nugache botnets injected into real traffic traces. Our approach produces a median F1 score of 91.8%. It is robust to parameter values and variations of the setup. This detection performance is noteworthy since our IP network has less than 2% malicious nodes.

Our work can be seen as a first step in developing behavioral-based approach, that can lead to a series of techniques for detecting botnets without signatures and during their more quiet Waiting stage. Moreover, Entelecheia is complementary to other botnet detection approaches such as IP blacklists, deep packet inspection, and detection methods in other stages of their lifecycle.

# Chapter 3

# "Infect-me-not": A User-centric and Site-centric view of web-based malware exposure

## 3.1 Introduction

Distributing malware indirectly via web-pages has become a very popular way for spreading malware in the last 8 years. First, the previous technique of spreading malware via email attachments lost its effectiveness due to increased awareness and the use of email-based anti-virus tools. Second, new technologies have enabled lay persons to create sophisticated and interactive websites despite the fact that the site owners have limited understanding of technology and security. In 2012, Google published a blog post on the current growth of malicious websites and stated that they found 9,500 of them each day [42]. However, not all

of these are websites created to be malicious. In fact, there are two main types of websites that spread malware: (a) the **born-malicious**, which are registered and operated by the malicious entities, and (b) the **compromised**, legitimate websites infiltrated by hackers and used as launching pads for their malware to get around spam filters and black lists.

"How does web-based malware spread?" is the key question that motivates this work. We consider a site-centric and a user-centric point of view: (a) what is the behavior and the lifecycle of the website that spreads malware, and (b) what is the behavior of the users that visit such websites. Our goal is three-fold: (a) investigate the composition of the websites to find out how many of them are born-malicious and how many compromised, (b) understand the life of a malicious URL and its impact, and (c) identify patterns in the way users visit malicious URLs. In the rest of this document, we use the term malware to refer to web-based malware.

Unlike the focus of our work, most previous work has focused more on identifying malicious websites, and less on their propagation patterns. In more detail, we identify three areas of focus in the literature: (a) the identification of website vulnerability to infiltration, (b) the detection of websites actively distributing malware, and (c) the study of the ecosystem and the techniques used by hackers. We describe research efforts in these complementary areas in section 3.6.

Our key contribution is an extensive study of user exposure to web-based malware following both a site-centric and user-centric point of view. We use two data sets: (a) $D_{\text{ODB}}$, with roughly 66K malicious URLs collected from four online databases between December 2013 to September 2014, and (b) $D_{\text{WINE}}$, which captures visits to malicious websites from

more than 500K users from January 2011 to August 2011 collected by the Symantec's WINE Project. Note that Symantec's data captures the exposure of the users to malware as seen by its anti-virus products, as we explain in section 3.3.

Our work can be summarized into the following major observations.

**1) Compromised websites play a significant role in malware dissemination.** We find that among all the domains in our $D_{\text{ODB}}$ dataset, 33.1% of them belong to compromised (i.e. legitimate) websites. We consider this as a lower bound of the presence of compromised websites, as we believe that these websites are more likely to be cleaned-up by their owners before being added to the black lists compared to born-malicious websites.

For our study, we developed a Machine Learning-based method to distinguish compromised websites from born-malicious sites. Our approach exhibits a 95.3% accuracy. This is arguably the first technique focusing on this question and we will make it available to the community, as it could be of independent interest.

**2) A malicious URL often distributes many different malware binaries but most malicious binaries are distributed by one URL.** We find that 33% of the URLs with at least 5 visits in our data set distribute two or more different binaries (different MD5 hash values). This percentage increases to 46% among all websites with more than 20 visitors. These website are either: (a) distributing completely different malware, (b) using polymorphism to distribute the mutated versions of the same malware to escape detection. In contrast, most malicious binaries (94.6%) are distributed by one URL in our data set.

**3) Most malicious URLs are short-lived, but 10% of them are active for more than three months.** Although 71.6% of URLs appear for only one day during the

8-month period, roughly 10% of them stay active for at least three months and a much smaller number have been active for four years. This suggests that there may not be an efficient technical and/or legal process to either clean up or take down a malicious website.

**4) The "Space-needle" pattern: Many URLs exhibit the same bursty temporal pattern aligned with a campaign-like behavior.** Here, we focus on websites that are active for at least 30 days and have at least 100 visitors. We find that the time series of the visits to 45.6% of those URLs follow a bursty pattern, which we refer to as "space-needle" due to its shape. URLs following this pattern usually peak within the first two days of their life, and the maximum number of daily visits is at least an order of magnitude larger than the median, as we discuss in section 3.5.

**5) The distribution of the visits to malicious sites per user is skewed and can be described by a power law of exponent $-\frac{1}{2}$.** A small percentage of users are highly susceptible to visiting malicious URLs. For example, we find that 1.4% of users in our data set visited at least 10 malicious URLs.

**Data Archive and Acknowledgment.** The data is available for follow-on research as reference data set WINE-2014-002 in Symantec's WINE repository. We are grateful to Dr. Matthew Elder and Dr. Daniel Marino of Symantec Research Lab for their support and feedbacks.

This work has been submitted to the Passive and Active Measurement Conference (PAM) 2015, to be held in New York, March 2015 with Adnan Bashir (University of New Mexico, Albuquerque), Michalis Faloutsos (University of New Mexico, Albuquerque), Chris-

tos Faloutsos (Carnegie Mellon University), and Tudor Dumitras (University of Maryland, College Parks).

## 3.2 Terminology

A URL is comprised of four components: `scheme://hostname/URI?params`. `scheme` is often `http` or `https`, `URI` is the path to the resource (a webpage, an image, a video, etc.) that the browser is requesting, `params` are the (usually optional) parameters that are supplied to the remote website to (among many other things) specify language and display preferences, and `hostname` is the DNS name associated with the IP address of the server hosting the site. A domain in the context of this work is the `hostname` portion of the URL.

The Web of Trust (WoT) Reputation Score is a numerical value between 0 and 100, inclusively, given to a website by WoT [8], which relies on its user community to rate the websites the users came across. The higher score a website has, the more trustworthy it is. A poor reputation score does not imply that a website is malicious.

VirusTotal is a popular online service where a user can submit a binary or a URL or a domain so it can be scanned by a number of anti-virus engines (58 by our latest count). Once the scan concludes, the user can retrieve a report that shows, in the case of a domain, the number of AV engines that considered the domain to be of a malicious website. We call this value the VirusTotal Malicious Score of a domain.

Malware polymorphism is what we call the phenomenon wherein the malicious code of a binary undergoes mutation and causes the binary to have a different signature each time a mutation occurs. Malware authors have been employing rapid binary repack-

ing/obfuscation/encryption techniques to hide the malicious functionalities of the binaries and thwart anti-virus products.

## 3.3 Data

**A. Online databases** We collected malicious URLs from four different online databases: Cybercrime Tracker [2], Malc0de [3], Malware Domain List [4], and VX Vault [7]. These online databases are maintained by communities and publish new malicious URLs on a regular basis. We began collecting the URLs in March 2014 and continued to do so every day until September 2014. This data set, which we call $D_{\text{ODB}}$ from this point onward, will be used to build a classifier to distinguish born-malicious from compromised websites.

| | Duration | URL count | Domain count | Client count | MD5 count |
|---|---|---|---|---|---|
| O.D.B. | 2013/12/12 - 2014/09/12 | 71,542 | 8,724 | - | - |
| WINE | 2011/01/01 - 2011/08/31 | 626,472 | 106,026 | 530,061 | 504,324 |

Table 3.1: Data from Online Databases (O.D.B.) and WINE

**B. Symantec's WINE data.** Symantec's Worldwide Intelligence Network Environment (WINE) [18] is a massive corpus of telemetry data collected from more than 120 million machines, both enterprise and consumer, and made available to the research community. WINE does not contain all of the data produced by all Symantec Anti-virus product, but is a representative subset. Samples are added to WINE in such a way that if a client machine is selected by the sampling algorithm, all of the reports generated by the same machine are included in the database. If a client machine is not picked, none of its reports appears in WINE.

The WINE database is divided into multiple *schemas*, each containing data from different aspects of the data collection process. The data that we collected from WINE belong to two schemas:

1. Anti-virus Telemetry: this data was collected from all clients any time a Symantec AV product detected that a *malicious* binary **executable** was downloaded.

2. Binary Reputation: data collected on binary **executables** downloaded by users in Symantec's reputation-based security program. It contains information on both malicious and benign binaries.

**Machine IDs.** Each machine, on which a Symantec security product was installed, is identified by a unique `ID` which acts as a serial number. We used this ID to identify users instead of IP addresses to avoid the effect of Network Address Translation.

**Modeling the exposure to malware.** The WINE data we collected focuses exclusively on *visits to malicious URLs*. Every entry in the dataset represents a report any time a user downloaded a malicious binary from a URL. However, we do not make a claim as to whether the user was infected or not. In fact, we believe that the malicious binaries were detected and the users would have been protected, unless they explicitly overrode the antivirus warning.

Each data point in our data set, which we will call $D_{\text{WINE}}$, contains: (a) the timestamp of the receipt of the report at a Symantec server, (b) the URL from which the binary was downloaded, (c) the MD5 hash of the binary, and (d) the `ID` of the client machine.

We began with collecting the information about malicious URLs from from Anti-virus Telemetry from January to August 2011. We then correlated with Binary Reputation

to obtain the information about the URLs from *before* Symantec determined that the URLs were distributing malware so that we get the complete history about each URL (including which binaries they distributed and who downloaded from them).

## 3.4 Classifying born-malicious and compromised domains

We present our Machine Learning-based method that distinguishes born-malicious from compromised domains and we use it to study the domains that we collected from the online databases.

**A. Preprocessing.** Even though the $D_{\text{ODB}}$ dataset includes 8,724 domains in total, we will run the classifier on only 3,975 of them. We filter out the rest of the domains because they are:

1. Domains not resolving to IPs. Because we began crawling in March 2014 while a portion of them dates back to December 2013, 1,550 domains no longer possess valid DNS records. A close examination of such domains shows that most have poor reputation scores and created recently. It is likely these domains were deactivated because they hosted malware.

2. Domains that returned either `40X` HTTP codes or no content. We suspect that they are placeholders for malicious files.

3. Domains belonging to known Content Delivery Networks, file-sharing sites (e.g. `mediafire`) and websites hosting free software (e.g. `softpedia`).

**B. Training and Testing Data.** From the remaining 3,975 domains (which we call $D_{\text{classify}}$), we randomly selected 609 domains and split them into two smaller data sets used for training and then testing our binary classifier: (a) $D_{\textbf{train}}$ has 200 domains, with 139 of them are labeled as compromised and 61 born-malicious and (b) $D_{\textbf{test}}$ contains 409 domains (280 compromised and 129 malicious). Note we label the domains manually and carefully: we visit each domain in a browser in a virtual machine to see if it contains legitimate content.

**C. Feature selection.** The features we use to create our classifier are:

1. The number of URLs embedded in the home page
2. The number of images on the home page
3. The age (days) of domain since registration
4. Web of Trust's reputation score for domain
5. VirusTotal's malicious score for domain

We include features (1) and (2) because together, they capture how much effort that was put into creating the websites. Feature (3) captures the fact that a compromised website is more likely to have "lived" for a much longer period of time than a born-malicious one.

To obtain relevant statistics for each domain in $D_{\text{ODB}}$, we created an automatic web crawler using the Python distribution of Selenium [6]. We chose Selenium because it offers the ability to programmatically control a real browser that would fully render the content of the web page. Fetching only the HTML via applications like `curl` or `wget` would not achieve

this goal because they cannot trigger the dynamic, script-based elements commonly found on modern websites and therefore are unable to accurately record the necessary statistics.

**D. Training the classifier.** We use $D_{\text{train}}$ to train our classifier, and we select the Random Tree method, because it gives the best performance among all others included in the WEKA Machine Learning framework [55]. We tried to create single-feature classifiers to test the accuracy of each feature but none of the classifiers exceeded 80% in accuracy when applied on $D_{\text{train}}$, where we define accuracy as the ratio of the number of correctly labeled domains and the total number of domains. Applying a classifier built from all features on $D_{\text{train}}$ yields a perfect accuracy with no misclassified instances.

**E. Classification accuracy.** After running the classifier on $D_{\text{test}}$, we find that the number of correctly classified instances is 390 out of 409 (95.3%). Among the 19 misclassified domains, we find: 9 compromised that were labeled as malicious and 10 domains misclassified the other way around. Further investigation shows that some misclassifications happened due to several reasons: (a) the domains were hosted by dynamic DNS services, so the age values reported, which are very high, are of the DNS services themselves, (b) some compromised websites have extremely simple home pages with few images and embedded URLs.

**F. 33.1% of malicious domains are compromised.** With our classifier, we classify all the domains in $D_{\text{classify}}$. We find 2,885 compromised and 1,090 born-malicious. This means that roughly 33.1% of the domains from $D_{\text{ODB}}$ are in fact benign websites that were infiltrated by hackers and used as a launching pad for malicious software. We will

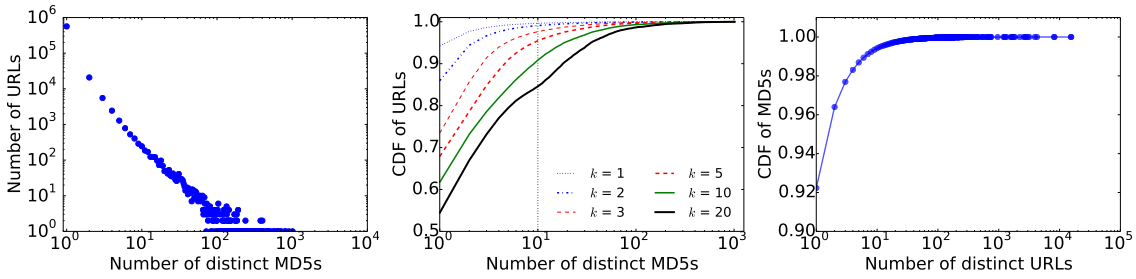
revisit the phenomenon of compromised domains again in the next section, when we provide our findings on the $D_{\text{WINE}}$ data set.

## 3.5 Profiling malicious URLs and their visitors

**A. A malicious URL often distributes many different malware binaries but most malicious binaries are distributed by one URL.** In the $D_{\text{WINE}}$ data set, we observed many instances where the same URL yielded many binaries with different MD5 hashes, often within the same hour and bearing similar malware designations by Symantec. Since we do not have the capacity to analyze the binaries themselves and ascertain that they are different "versions" of each other, we call this phenomenon *URL-centric polymorphism*. Polymorphism has a different meaning in *malware polymorphism* as discuss in section 3.2.

We looked at the malicious URLs seen in $D_{\text{WINE}}$ and counted the number of unique MD5 values associated with each URL. In Figure 3.1(a), we can see for 94% of URLs, each is associated with only one MD5. As we will show later, most of these low-visibility URLs are visited by only one or two users, which makes it unclear whether there is any automatic obfuscation process at work.

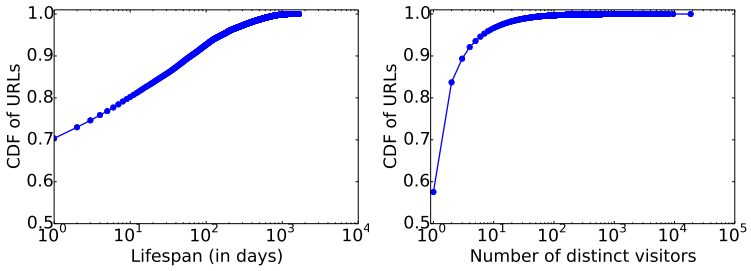
We focus instead on the high-visibility URLs, as URLs with low visitor counts do not give much insight into URL-centric polymorphism. In Figure 3.1(b), we show the distribution unique MD5 hash values distributed by each URL. Each curve, represented by a $k$ value, shows the distribution of MD5 count per URL where the URLs have at least $k$ visitors in our data set. The polymorphism becomes more evident for URLs with more visitors. With the minimum number of visitors $k = 5$, we have 33% of the URLs distributed




(a) (b) (c)

Figure 3.1: (a) Distribution of the number of unique MD5 hashes seen per URL. (b) URL-centric polymorphism can be observed more clearly on URLs with high number of visitors. (c) Distribution of the number of URLs associated with each MD5.

more than one binaries, but this number increases to 46% of the URLs when we require a minimum number of visitors $k = 20$.




(a) (b)

Figure 3.2: (a) Cumulative distribution of URLs according to their lifespan. (b) Cumulative distribution of URLs according to their visitor counts.

As we can see from Figure 3.1(c), 92.2% are *single-URL binaries*, meaning each of them was distributed by only one single URL, but there are some that appeared on more than a hundred, which we did not expect. To better understand this behavior, we randomly picked 600 of the multi-URL binaries and inspected the URLs they appeared on.

1. Only 87 (14.5%) appeared on multiple domains that belong to popular file-sharing websites or well-known software distributors.

2. 457 of them (76.2%) appeared on multiple domains that seem to have been created just to be malicious and disposable in that the domains are often sequences of letters that resemble English words (but not quite) followed by some sequence of digits, suggesting that the domains themselves were created via some *automatic* process.

3. 56 (9.3%) appeared on what seem like compromised sites, many of them active and containing legitimate content. These binaries were distributed by URLs that have similar structure. For example, one such binary was distributed by URLs of the form: `http://{D}/images/ facebook-pic-{X}.exe`, where `{D}` represents different domain names and `{X}` a sequence of random digits. This suggests that the sites were compromised: (a) through the use of the same hacking toolkit, and/or (b) by the same hacker.

**B. Most malicious URLs have short lifespan, but a small percentage live for more than three months.** In Figure 3.2.(a), we study the distribution of the lifespan of websites. We find that 70.6% of all malicious URLs are what we call *single-day URLs* as they appear for only one day in our dataset.

In Figure 3.2(a), we plot the distribution of lifespan for each URL and show that there are a total of 184,409 URLs (29.6%) in $D_{\text{WINE}}$ whose lifespan is longer than one day. In Figure 3.2(b), we plot the distribution of visitors per URL. We find that this distribution is highly skewed where 57.4% of the URLs have one visit, while 11.2% at have least three visits. Surprisingly, 10% managed to stay "alive" and actively distribute malware for more

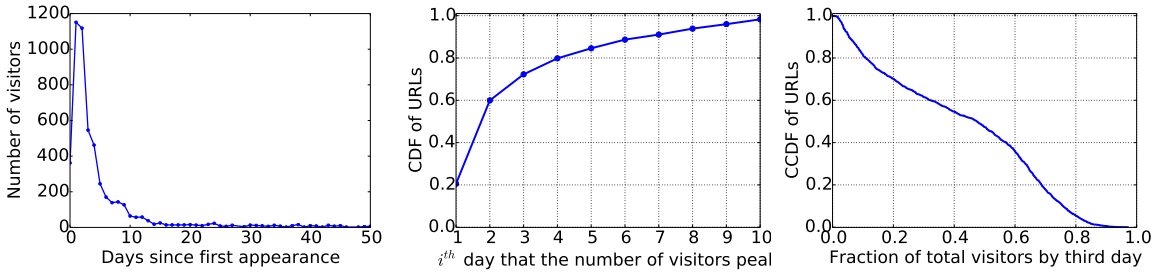
than three months and there are 194 malicious URLs that have been around for four years. Furthermore, a small percentage (2,427 URLs, making up 0.4% of all URLs) attracted at least a hundred users during their lifespan.

**C. A significant number of highly active URLs exhibit the same bursty temporal pattern that suggests a campaign-like behavior**. We discover that the visits to many URLs exhibit a bursty behavior, as can be seen in Figure 3.3(a). Temporal pattern, which we will call "space-needle", could be the result of an active campaign, staged by the hacker, to drive traffic to a newly infected or created site. Consequently, most of their visits take place during the first few days when the site appears, since after the first few days, the spam filters and black lists catch up and reduce the number of visitors.

We want to study the extent of the "space-needle" phenomenon in more detail. We start with focusing on URLs with a lifespan of 30 days or more and at least 100 visitors. We identify 2,402 URLs in $D_{\text{WINE}}$ that meet these criteria. We will refer to these URLs from this point on as **Highly Active URLs**. We then do the following analysis to jointly define the "space-needle" pattern and quantify its presence with a technique commonly used in data mining.

First, we select a representative time series that intuitively captures the essence of the "space-needle" shape (seen in Figure 3.3(a)), identify other behaviors that are similar to it. We use the following technique from time-series analysis: (a) we compute the Euclidean distance between the time series of the representative and all other behaviors, (b) we sort the time-series in ascending order in terms of distance to the representative, and (c) we check each time-series until we find ones who are visually different from the representative.

At the end of this process, we find 1,095 URLs or 45.6% of the 2,402 highly-active URLs, which we will call **Space-Needle URLs**.




(a) (b) (c)

Figure 3.3: (a) An example of the space-needle propagation pattern. (b) Distribution of which day the URLs gained the maximum number of visitors. (c) Distribution of the fraction of total number of visitors each URL accumulated by the third day.

The next step is to quantify the properties of the Space-Needle URLs, as shown in Figure 3.3. We can see from Figures 3.3(b) and 3.3(b) that 60% of these URLs achieved their peak number of daily visitors either on their first day of appearance, or the very next day. By the end of the third day, 50% of all the Space-Needle URLs have seen at least half of their total number of visitors. Moreover, 80% of the Space-Needle URLs, the peak number of daily visitors is at least one order of magnitude larger than the median value of daily visitors. Note that we only consider days with at least one visitor to compute the value of the median.

**D. Where do malicious domains end up?** When we performed DNS queries on all of the domains of the malicious URLs reported in $D_{\text{WINE}}$, we found that roughly a third of the malicious domains (35.9%) continue to be active. We were intrigued as to why these domains (presumably of malicious websites) are still active even though they were first

reported in 2011. To further investigate, we randomly selected 600 still-active domains and accessed them in a browser in a virtual machine and we identified the following categories.

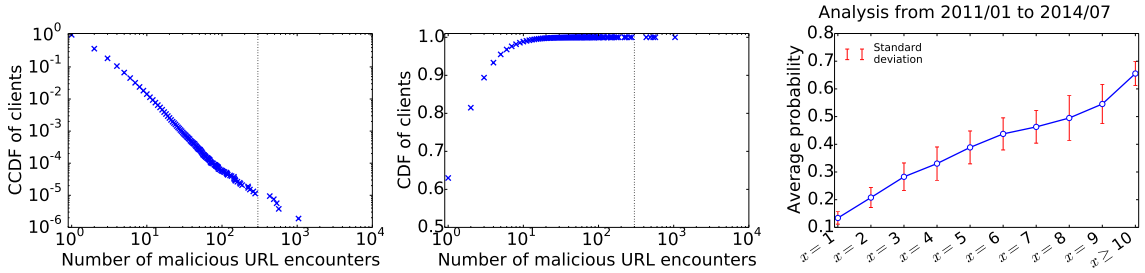
1. 121 domains (20.2%) are now under control of domain parkers and serving as advertisement space.

2. 138 (23.0%) are domains from file-sharing websites and software distributors.

3. 121 (20.1%) return `40X` HTTP error codes or blank pages.

4. 220 (36.7%) seem to be benign, bearing legitimate content. We believe they might have been compromised when Symantec detected malicious binaries being distributed by their servers. This suggests that compromised websites seem to have also played a significant role in malware delivery in 2011

**E. The distribution of the visits to malicious sites per user is skewed and can be described by a power law of exponent $-\frac{1}{2}$.** In Figure 3.4(a), we plot the CCDF of the number of visits to malicious sites for each user. We find that the distribution of the number of malicious URL encounter per person seems to follow a power law distribution with exponent $\alpha = -\frac{1}{2}$. Thus, the good news is that most users in $D_{\text{WINE}}$ encounter malicious URLs very infrequently. In Figure 3.4(b), we plot the CDF of the same distribution and show that 63% of all users visited a malicious URL only once during the entire eight months. However, 1.4% (close to 7,500 users) visited malicious URLs at least ten times during the same amount of time. This in general suggests that a small group of users were less cautious than others.




(a) (b) (c)

Figure 3.4: (a) CCDF of clients with respect to the number of times they encountered malicious URLs. (b) CDF version of (a). (c) Probability of a user visiting a malicious URL within a month given the number of such visits ($x$) the month before.

**Users with more than 400 visits to malicious URLs and an explanation.** In both Figures 3.4(a) and 3.4(b), we see that a few data points are well separated from the rest of the distribution (to the left of the dotted vertical lines). Each of these points represents users who visited at least 400 malicious URLs during the eight-month period, averaging to more than one URL a day. The most active user visited more than one thousand malicious URLs. This behavior seems unlikely for a human.

Upon investigation, we arrived at a possible explanation for these outlier points. Recall that users in the $D_{\text{WINE}}$ database are not identified by their IP address, as we discussed in section 3.3. Instead, each user is identified by a unique `ID`, associated with installation of a Symantec anti-virus product. The explanation for the outliers, as observed by one of the authors during his tenure at Symantec, is that a user can install the AV product in a Virtual Machine and clone it into multiple VMs, thereby allowing all VMs to report their download activities to Symantec as a single user.

**F. Some users are more prone to reckless surfing behavior.** We tried to estimate the probability of a user visiting a malicious URL given their history by executing the following steps.

1. Select one month from January to *July* of 2011.
2. Calculate how many URLs each client encountered during that month.
3. Let $C_x^i$ be the set of clients who visit $x$ URLs during month $i$, and let $V^{i+1}$ be the set of the visitors to malicious URLs during the following month. We compute the percentage of users in $C_x^i$ who will be **repeat offenders**: $P_x^{i \to i+1} = |C_x^i \cap V^{i+1}|/|C_x^i|$.
4. Repeat the steps above for every other months listed in step 1.
5. Compute the *average* $P_x^{i \to i+1}$ for each value of $x$ across all months.

In Figure 3.4(c), we plot the average probability of a user visiting a malicious URL within a month given the number of such visits ($x$) the month before. The average probability is computed across all users with the same number of visits to malicious sites for all pairs of consecutive months. We observe significant increase in the average probability as the number of URLs visited grows.

## 3.6 Related Work

The only work related to profiling the behaviors of binaries that we can find is [44], which presents a model called SHARKFIN that describes the propagation pattern of popular, benign software. There is also one recent work by Kuhrer et. al. [31] in which the authors

evaluated the completeness of black lists and presented a method to identify parked domains and sink holes.

Otherwise, there are three areas that touch on various aspects of web-based malware study. The first area is comprised of work on the landscape of web-based malware distribution, which then is split into two smaller areas: (a) how to actively seek out new malicious sites [49][35][47] and (b) the identification of malicious URLs [34][39] or drive-by-downloads website [15]. We do not use the approaches in (b) in our classification because URL-classification methods can only find malicious URLs, which may belong to either compromised or compromised sites. [15] is not applicable because not all born-malicious sites trigger drive-by-downloads.

The second area focuses more on the identification of websites that may be at risk of infiltration in the future [48][14]. The third area centers on the study of the ecosystem sustained by malware authors, providing insights on the attacks carried by malicious websites on the users [40] or on the infrastructure that supports the malware authors [20][13], enabling them to spread their malicious software for monetary gains.

## 3.7 Conclusion

In this work, we focus on modeling the user exposure to web-based malware by analyzing more than 500K users accessing roughly 600K URLs from a data set collected from Symantec's WINE Project. We find that: (a) the 10% of malicious URLs stay active for more than three months and a smaller percentage for more than 4 years, (b) the distribution of visits to malicious URLs among users follows a skewed distribution that is captured by

a power-law. Similarly, some users are more likely to be exposed to malware: close to 7,500 users visited at least 10 malicious URLs during our 8-month observation period and we show that past user exposure is an indication of future exposure for that user. We also present, arguably, the first method to distinguish born-malicious websites from compromised websites. We show that compromised websites play a major part in distributing malware: 33.1% of the websites are compromised and have legitimate content. Our study is a first step towards modeling web-based malware exposure and could help us understand malware distribution as a network-wide phenomenon.

# Chapter 4

# Group Profiler: "The reporting of this phishing site is not required"

## 4.1 Introduction

Protecting the cyber security of an organization has grown increasingly more important as hackers constantly find ways to infiltrate organizations to steal valuable information. There are two common methods with which malicious entities seek to accomplish this objective, and both begin with guiding the users to websites under the control of the hackers. In the first method, the websites would either deceive the users into installing some malware into their computers or quietly execute the installation without the users knowing. In the second method, called "phishing", the users would see a (fake) website which looks legitimate and asks for the users' credentials to continue (think the login page of the organization's webmail system).

In this work we seek to address the problem by asking a key question: *"Can we detect when users in an organization visit suspicious websites only by knowing their history of browsing activities?"* Put differently, given that we know that a set of **persons** $P$ have visited a set of **websites** $W$ during a presumably sufficiently long period of time $T$, can we know which of the set of websites $W'$, which is seen during a subsequent time period $T+1$ but *not* during $T$, is suspicious and should be investigated?

Finding the answer to this question depends on the assumption that there exists a group structure in the bipartite graph representing interactions between $P$ and $W$ during $T$. If there is a stable group structure, it means that there are groups of users sharing similar interests and visiting websites pertaining to their interests in *blocs* and that it would be feasible to detect when a subset of $P$ visit malicious websites. It is feasible because when hackers want users to visit malicious websites, they do so by orchestrating mass spam campaigns. These campaigns tend not to be targeted, meaning that only a random subset of the users of $P$ may see the links in the email. If this subset is randomly picked, then the presence of a new website would cause the group structure to be disturbed and this disturbance may be detectable.

We make three contributions with our work. The first contribution is an in-depth study of the browsing behaviors of users. In this study we show that there is some measure of stability in the way the users browse the web in that users tend to visit the same set of websites over and over again and that we can make predictions, with moderate levels of certainty, as to which website a user is likely to revisit given his or her browsing behaviors during the previous week. With this study, we show that there is reason to believe that

there may exists a stable group structure in a network of users, as there is already stability in the way each user interacts with the Web.

The second contribution is Group Profiler, a probabilistic co-clustering approach based on the Stochastic Block Model. Its input is the bipartite graph $G = \langle (P, W), E \rangle$ and a parameter $k$ whose value is the expected number of groups in the graph $G$. What the algorithm outputs is the affinity matrices $\theta$ and $\beta$ where $\theta$ represents the affinity of each **person** $p \in P$ to each of the $k$ groups and $\beta$ represents the affinity of each **website** $w \in W$.

Our third contribution is a method to measure $l_{w'}$, which is what we call the "goodness" of each $w' \in W'$, a set of websites never seen before. At a high level, $l_{w'}$ is a value that measures the likelihood of the configuration $E_{P \leftrightarrow w'}$, which is the set of edges between the people in $P$ and the website $w'$. The higher the value of $l_{w'}$, the more trustworthy $w'$ is and vice versa. Intuitively, a high value of $l_{w'}$ means that the people who visit $w'$ are generally from the same group and a low value means that the people are from many different group, which then implies that either (a) a large-scale event happens in the real world and attracts the attention of many or (b) some people in the organization receive unwanted emails and are guided to malicious websites.

Our work is arguably the first approach designed to solve the problem of detecting users' visits to malicious websites using nothing except from the users' historical browsing information. We have stress-tested Group Profiler on many different synthetically created data sets and found that Group Profiler retains high accuracy even under more adversarial conditions. Our work is meant to be used by network administrators who want an automatic means to rank new websites each day in terms of goodness so that they can investigate one

by one and, in case they discover a malicious website, can know quickly who have visited it and take necessary actions to protect the users.

This work is done in collaboration with Aric Hagberg (Los Alamos National Laboratory), Cristopher Moore (Santa Fe Institute), and Michalis Faloutsos (University of New Mexico, Albuquerque).

## 4.2 Stability in the browsing behaviors of users

The answer to the question of whether we can use the users' browsing histories to detect anomalies rests entirely on whether there exists a stability in the way users browse the web. If there is stability, it means that there exists a stable group structure in the interactions between the users and the Web and we can leverage this stable structure to detect "disturbances" to the structure whenever users visit malicious websites. In this section, we show that there indeed exists such a stability in the users' web-browsing activities by demonstrating that the users' behaviors are predictable with some measure of certainty.

### 4.2.1 Data and Pre-processing

Our data set is a collection of Web Proxy Logs collected from a Chinese university during a three-week period from a total of thirty users. Each of the user is a student who has a *fixed* IP address and each and every of their HTTP request is collected.

We, however, do not use this data as-is. The reason is that not all HTTP requests produced by a user's web browser are indicative of the user's *intention*. For example, when a person visits `cnn.com`, the HTTP request that their browser issues immediately spawns

hundreds of other automatic requests for resources (images, videos, cascading style sheets, advertisement banners, etc.) that are embedded on the landing page of CNN but are hosted *off-site* by content delivery networks. In this situation, the user is only interested in the landing page and only the HTTP request for the landing page is *intentional*. All other automatic requests are not so and should not be considered for the study.

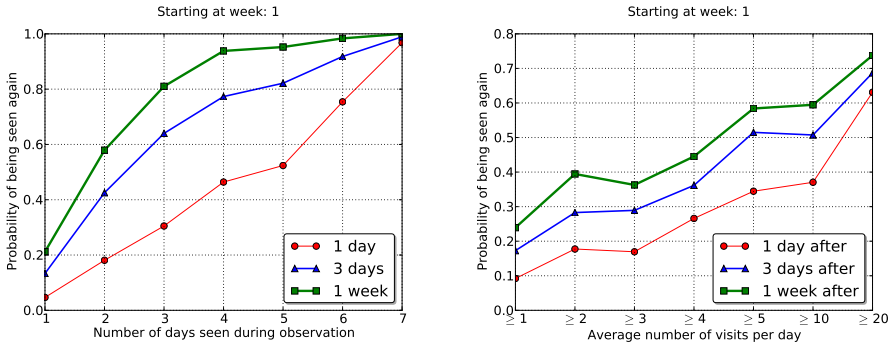
To filter out the non-intentional HTTP requests, we leverage the method Xie et. al. proposed in [56]. This method, called ReSurf, uncovers the intentional HTTP requests by combining two vital pieces of observations: (a) the referrer field of each HTTP request and (b) the time difference between each HTTP request and its parent where the parent HTTP request is the request for the URL that appears in the child request's referrer field. The intuition is simple: if an HTTP request is automatic, the time difference between itself and its parent request should be so small that they could not have been issued by a human in succession.

### 4.2.2 The predictability of users' browsing behaviors

**Goal:** We want to measure the *probability* of a website being revisited by the user during any specific week given either of the following metrics:

1. The number of different days (between 1 and 7 inclusively) that the website was visited during the previous week.

2. The average number of visits per day that the website had during the previous week.




(a) (b)

Figure 4.1: Given a week of observation, we show that (a) the higher the number of days during which a website is seen, the higher the probability that the same website would be revisited in the future, and (b) the same relationship applies with the average number of daily visits a website has during observation.

**Experiment:** We first split the original data set into three data sets $D_1$, $D_2$, and $D_3$ where $D_1$ is the set of all *intentional* HTTP requests (as uncovered by ReSurf [56]) that the thirty users made during the first week (out of three).

Let's say we begin with $D_1$. For each website $w$ visited by a user $u$ in $W_1$ , we count the number of different days that $w$ was visited and the average number of visits per day paid to $w$. Let $W_1^x$ be the set of websites seen in $D_1$ where each $w \in W_1^x$ was visited on exactly $x$ different days. Let $W_{1\to2}$ be the set of websites seen during both $D_1$ and $D_2$. We say that the probability of a website visited on $x$ different days during $D_1$ being revisited during $D_2$ is calculated by the formula:

$$\Pr_{1\to2}^{x} = \frac{|W_1^x|}{|W_{1\to2}|}, \qquad 1 \le x \le 7$$

Let $f$ be the average number of daily visits paid to a website $w$ by a user and $W_1^f$ be the set of websites seen during $D_1$ where each $w \in W_1^f$ received at least $f$ daily visits on

average. We say that the probability of a website visited on during $D_1$ with at least at least $f$ daily visits on average, being revisited during $D_2$ is calculated by the formula:

$$\Pr^{f}_{1\to 2} = \frac{|W^{f}_{1}|}{|W_{1\to 2}|}$$

We then measure both $\Pr^{x}_{1\to 2}$ and $\Pr^{f}_{1\to 2}$ and their relationships with $x$ and $f$ respectively.

In Figure 4.1, we show the result of the measurement. What the Figure illustrates is in fact the accuracy of predicting whether a website would be visited during a week using the number of days $w$ was visited (in 4.1(a) or its average number of daily visits (in 4.1(b)) during the previous week. As we can see in the Figure, both metrics possess some predictive power: (a) the higher the number of days a website is visited during a week, the more likely that it will be revisited during the following week and (b) the more often during the day a website is visited, the more likely that it will be visited during the next week.

When we repeat this experiment between $D_2$ and $D_3$, we obtain similar results. This, in turn, suggests that there indeed exists some stability in the way a user browses the Web.

### 4.2.3 Cores versus Ephemerals

Delving more deeply into the web-browsing behaviors of users, we observe that, in a temporal sense, there are two kinds of websites: (a) **Ephemerals** and (b) **Cores**. An ephemeral website is what we loosely defined to be one that a user visits only during a short

amount of time and then is eventually forgotten and never visited again. Cores, as opposed to ephemerals, are websites that a user visits frequently.

We conduct a small experiments to discover the prevalence of cores and ephemerals using the data set collected from the Chinese university. In this experiment, we look at each user and perform the following steps:

1. We pick a value for $n$, which we call the ephemeral threshold.

2. For every website in the user's history, we count the number of days (out of all three weeks $D_1 + D_2 + D_3$) that the website was visited by the user.

3. Given that a website was visited by the user $x$ times during the three weeks, we say that the website is an ephemeral if $x < n$.

4. We count the number of ephemerals according to the value of $n$ that we pick in the first step.

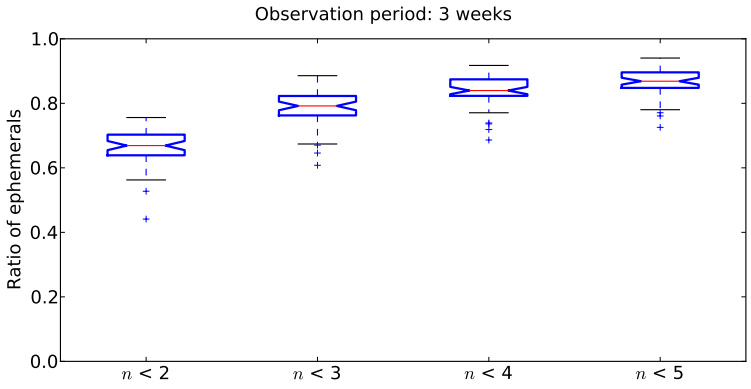



Figure 4.2: Most of the websites that a user visits is *ephemeral*

In Figure 4.2, we show the distribution of the ratio of ephemeral websites across all thirty users with various values of $n$. Interestingly enough, the median ratio of ephemeral websites per user (indicated by the red horizontal line) for $n = 2$ is close to 70%, meaning that 70% of the websites in a user's history is visited only on one day each and no more.

Now that we have seen the temporal behaviors of the ephemerals, we will move on to investigate the cores next. From Figure 4.2, we already know that for each user, no more than 10% of the websites were visited on at least five days during the entire three weeks. What we want to know more about is the behaviors of the users toward the websites that are frequently visited.

Our experiment is as follows. First we define $k$ to be a **promotion threshold** that decides whether a website belongs to a user's Core Set. We say that a website $w$ is a core website if during the last week (last 7 consecutive days), the total number of days on which the website was visited, which we now call $V(w)$ is at least $k$, or $V(w) >= k$. We also say that:

- an ephemeral website $w$ is **promoted** to be a core website if $V(w) = k$ by the end of day $d$.
- a core website $w$ is **demoted** to be an ephemeral website if $V(w) < k$ by the end of day $d$.

We perform promotion to and demotion from the core set as follows.

1. Each user's web browsing history is a collection of *vectors*, where each vector $w$ has 7 slots, where $w_i$ represents the *state* of the website on the $i^th$ day from the "present". $w_i = 1$ if and only if $w$ is visited by the user on day $i$ and 0 otherwise.

2. Then $V(w) = \sum_i w_i$

3. With the arrival of a new day $d$, if the website is visited by the user, a 1 is prepended at the head of the vector and 0 otherwise. Then the element at the tail of the vector is removed.

4. If $w$ is already in the core set:

   - By the end of $d$, if $V(w) = \sum_i w_i < k$ where $k$ is the promotion threshold, the website is demoted to an ephemeral.
   - Otherwise, the website remains in the core set.

5. If $w$ is *not* in the core set:

   - By the end of $d$, if $V(w) = \sum_i w_i \geq k$, the website is promoted to be a core.
   - Otherwise, the website remains an ephemeral.

We use the first week of data ($D_1$) as training, where we determine the membership of the core set and the ephemeral set. We then use the following two weeks to see how the memberships of the two sets *change*.

In Figure 4.3, we show that on a daily basis (during $D_2 + D_3$), the median number of promotions to the core set is small and the higher $k$ is (i.e. the stricter the requirement for a website to be a core), the lower the median number of promotion.

In Figure 4.4, we show the distribution of the ratio of ephemerals that have become cores by the end of $D_3$ over all users. What we can see from the Figure is that even when the promotion threshold $k$ is very relaxed ($k = 3$), only roughly 10% of all ephemeral websites

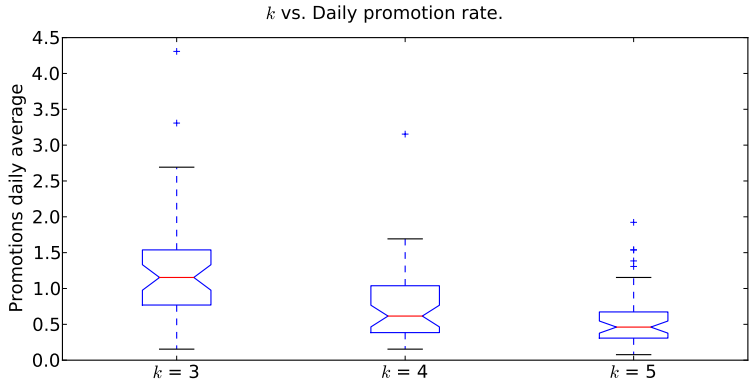



Figure 4.3: Promotion to the core set on a daily basis is low

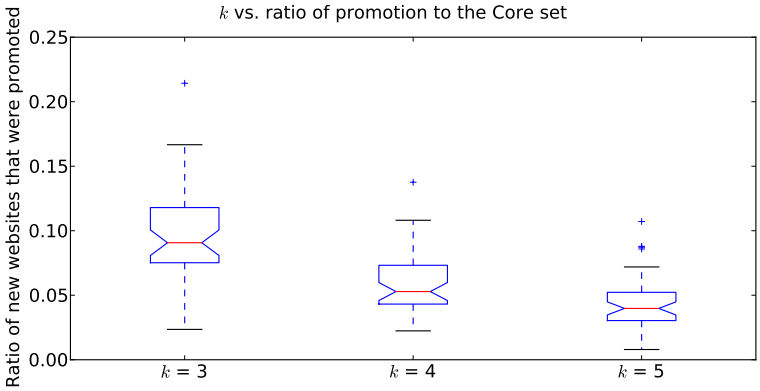



Figure 4.4: By the end of $D_3$, only a small number of ephemeral websites have become core.

(by the end of $D_1$) and all the new websites seen during $(D_2 + D_3)$ are promoted to the core set. And if we raise $k$ to be 5, the median ratio dips down to only 5%.

What both of these Figures show, intuitively, is that each user has a set of "Favorite" websites (which we call cores) that they frequently visit and that promotion to this core set is rare. This, in turns, suggests once again that there is a measure of *stability* in the way each user browses the web.

We will leverage this stability to detect visits to malicious websites in the following sections.

## 4.3 Terminology

A website $w$ is described uniquely by its Uniform Resource Locator (URL) and is one of the following type: HTML, XHTML, or XML.

In the context of our work, a group is simply a collection of persons and websites and we do not force any person $p$ or website $w$ to be in any single group. In fact, given $k$ groups where each group is denoted as $\{g_1, g_2, ..., g_k\}$, what our algorithm, Group Profiler, shows the affinity of each person $p$ and each website $w$ to each $g_i$ $(1 \leq i \leq k)$. We say that $p$ is more strongly affiliated with group $g_i$ than group $g_j$ if the affinity value of $p$ to $g_i$ is higher than that of $p$ to $g_j$.

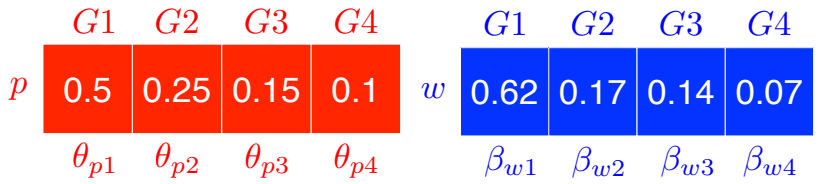



Figure 4.5: Affinity matrices of a person $p$ and a website $w$ given $k = 4$ groups.

In Figure 4.5, we show examples of **affinity vectors** for one $p$ and one $w$ where there are four groups. Note that the values of the vectors are not normalized (i.e. their values do not sum to one), the reason for which we will provide in a later section.

A Web Proxy Log is a log file that records the set of websites $W$ that were visited by a set of $P$ internal IP addresses[1] during some time period. A Web Proxy Log is so named because it is created by a Web Proxy, through which all HTTP requests originating from inside a network are routed before they reach the web servers hosting the websites.

## 4.4 Problems Definition

There are two problems that we tackle in this work. They are as follows.

1. **Inferring Affinity Matrices from the graph**: Given any bipartite graph that represents the browsing activities of a set of persons and a set of groups, we would like to infer the affinity of each person toward each group and then model the stability of the **group structure** over time. More details about the group structure will be given below.

2. **Detecting anomalies in group browsing behavior**: If the group structure of of the network stable over time, we would like to use them to detect changes in group browsing behaviors. For example, if there are websites never before visited by people in the same network, we would like to separate the ones that are visited by people from the same group from ones that are visited by people from *different* groups. The latter, as we explained before, can be a security concern.

### 4.4.1 Inferring Affinity Matrices from the graph

**Inputs:**

[1] We use IP addresses interchangeably with persons in this work

1. A weighted, undirected bipartite graph $G_{P,W,T}$ representing the browsing activities of $P$ persons, who visited $W$ websites during a time period $T$. An edge in this graph from $p$ to $w$ with weight $n$ means that $p$ visits $w$ $n$ times during the recorded period.

   - Let $n = |P|$, the number of persons (or IP addresses) seen in the Proxy Log.
   - Let $m = |W|$, the number of websites seen.

2. A parameter $k$ (positive and integer-valued) that is the expected number of **groups** the network is supposed to have.

**Outputs:**

1. $\theta$: the affinity matrix of size $n \times k$ where each row (of size $1 \times k$) is the affinity vector for some person $p$, which we will call $\theta_p$.

   Given $\theta_p = \{\theta_{pi}\}(1 \leq i \leq k)$, If $\theta_{pi} > \theta_{pj}$, then $p$ has a stronger affinity to group $i$ than $j$.

2. $\beta$: the affinity matrix of size $m \times k$ where each row (of size $1 \times k$) is the affinity vector for some website $w$, which we will call $\beta_w$.

   Given $\beta_w = \{\beta_{wi}\}(1 \leq i \leq k)$, If $\beta_{wi} > \beta_{wj}$, then $w$ has a stronger affinity to topic $i$ than topic $j$.

**Group structure:** We say that two persons $p_i$ and $p_j$ are in the same group if their most dominant interests are aligned. More specifically, $p_i$ and $p_j$ in the same group if and only if:

$$\text{argmax}(\theta_{p_i}) = \text{argmax}(\theta_{p_j})$$

What we want to show in our work is that the group structure in the network is stable over time, i.e. if $p_i$ and $p_j$ are in the same group during time period $T$, they will remain in the same group during time period $T'$ that follows $T$.

### 4.4.2 Detect anomalies in group browsing behaviors

**Inputs:**

1. $\theta$ and $\beta$ from Section 4.4.1

2. A weighted, undirected bipartite graph $G'_{P,W',T'}$ representing the browsing activities of the same $P$ persons, who visited $W'$ **new** websites during a time period $T'$, which follows $T$. Let $m' = |W'|$, the number of new websites seen during $T'$.

**Output:**

- A ranking $R = \{(w'_i, l_{w'_i})\}$, where $w'_i \in W'$, $1 \leq i \leq k$. $l_{w'_i}$ is called the **likelihood score** of website $w'_i$ such that given any two $l_{w'_i}$ and $l_{w'_j}$:

$$i < j \Longrightarrow l_{w'_i} < l_{w'_j}$$

In the context of our work, the higher the likelihood score a website has, the less suspicious it is. More details will be given in Section 4.5 on how to calculate the likelihood scores, but at a high level, a website has a high likelihood score if it is visited by people who are in the same group and a low one otherwise. The ranking $R$ is in fact sorted by the likelihood scores where the websites that are the most suspicious are at the beginning of the ranking.

The ranking, of course, is meant to advise a network operator as to how much scrutiny they should direct toward the website. The lower the likelihood score, the more scrutiny should be exercised.

## 4.5 Methodology

### 4.5.1 Inferring the affinity vectors for each node in the graph.

Let $n = |P|$, $m = |W|$, and $a$ be the weighted adjacency matrix representation of the bipartite graph $G_{P,W,T}$ where the value of $a_{pw}$ is the number of times the person $p \in P$ visits the website $w \in W$. The dimensions of $a$ is $n \times m$.

What we want to infer are the **affinity vectors** $\theta_p$ for each person $p$ and $\beta_w$ for each website $w$ where $\theta_{pi}$, $1 \leq i \leq k$, is indicative of how strongly a person $p$ is affiliated with the group $i$. Note that the vectors are not normalized, i.e. $\sum_i \theta_{pi}$ and $\sum_i \beta_{wi}$ are not necessarily 1. This is because we want to reflect the reality that some people spend more time online than others and different websites may have different popularity.

More precisely, we want to infer $\theta$ and $\beta$ given the adjacency matrix $a$ such that the log-likelihood function $\mathcal{L}(a|\theta, \beta)$ is maximized.

Our model is the bipartite version of Ball, Karrer, and Newman's mixed membership stochastic block model [9] where we assume that any $a_{pw}$ is Poisson-distributed with the mean being the inner product between $\theta_p$ and $\beta_w$:

$$a_{pw} \sim \text{Poi}\left(\sum_i \theta_{pi}\beta_{wi}\right)$$

In other words, the number of times $p$ visits $w$ depends on how well $\theta_p$ and $\beta_w$ line up in terms of group affinity distribution.

The model as-is has symmetries that make it non-identifiable, which means that there are no unique solutions $\theta$ and $\beta$ that can be inferred. The reason is that the model only cares about the inner product $\sum_i \theta_{pi}\beta_{wi}$. If for each $i \in \{1, 2..., k\}$ we choose a value $c_i$, multiply $\theta_{pi}$ by $c_i$, divide $\beta_{wi}$ by $c_i$ for all $p$ and $w$, the inner products are unchanged. To fix this, we impose a constraint:

$$\sum_p \theta_{pi} = n \tag{4.1}$$

which restrict the model to only rotation symmetry because applying an orthogonal linear operator to any two vectors $\theta_p$ and $\beta_w$ preserves their inner product. If we require that $\theta_p$ and $\beta_w$ be nonnegative, we confine the vectors to the positive orthant and therefore reduces the rotation symmetry to permutations of the $k$ groups.

The log-likelihood of the **affinity matrices** $\theta$ and $\beta$ given the weighted adjacency matrix $a$ is

$$\begin{aligned} l = \mathcal{L}(\theta, \beta|a) &= P(a|\theta, \beta) \\ &= \log \prod_{pw} \frac{e^{-\sum_i \theta_{pi}\beta_{wi}} \left(\sum_i \theta_{pi}\beta_{wi}\right)^{a_{pw}}}{a_{pw}!} \\ &= \sum_{pw} \left[ -\sum_i \theta_{pi}\beta_{wi} + a_{pw} \log \sum_i \theta_{pi}\beta_{wi} - \log a_{pw}! \right] \end{aligned}$$

We take advantage of Jensen's inequality $\log \sum_i x_i = \max \sum_i q_i(\log x_i - \log q_i)$ where the maximum is obtained by the probability distribution

$$q_i = \frac{x_i}{\sum_i x_i} \tag{4.2}$$

Applying the inequality, we obtain:

$$\log \sum_i \theta_{pi}\beta_{wi} = \sum_{pwi} q_{pwi}(\log \theta_{pi}\beta_{wi} - \log q_{pwi})$$
$$= \sum_{pwi} q_{pwi}(\log \theta_{pi} + \log \beta_{wi} - \log q_{pwi})$$

$q_{pwi}$ corresponds to the probability that $p$ visits $w$ because they belong to to the same group $i$ and:

$$\sum_i q_{pwi} = 1. \tag{4.3}$$

The log-likelihood is now

$$l = \sum_{pw} \Bigg[ - \sum_i \theta_{pi}\beta_{wi} + a_{pw} \sum_{pwi} q_{pwi}(\log \theta_{pi} + \log \beta_{wi} - \log q_{pwi}) - \log a_{pw}! \Bigg] \tag{4.4}$$

To infer $\theta$, $\beta$, and $q$ simultaneously, we apply the Maximum Likelihood Estimation method. Our task is to maximize the log-likelihood in Equation 4.4 with Constrains 4.3 and 4.1.

We derive $q$ by first dividing both sides of Equation 4.4 by the value $a_{pw}$ and taking the derivative of the log-likelihood with respect to $q_{pwi}$:

$$\frac{1}{a_{pw}} \frac{\partial l}{\partial q_{pwi}} = \log \theta_{pi} + \log \beta_{wi} - \log q_{pwi} - 1 = \lambda_{pw}$$

where $\lambda_{pw}$ is the Lagrange multiplier for Constraint 4.3.

We then have:

$$\begin{aligned}
\lambda_{pw} + 1 &= \log \theta_{pi} + \log \beta_{wi} - \log q_{pwi} \\
&= \log \frac{\theta_{pi}\beta_{wi}}{q_{pwi}} \\
\implies e^{\lambda_{pw}+1} &= \frac{\theta_{pi}\beta_{wi}}{q_{pwi}} \\
\implies q_{pwi} &= \frac{1}{e^{\lambda_{pw}+1}} \theta_{pi}\beta_{wi}
\end{aligned}$$

This implies that $q_{pwi} \propto \theta_{pi}\beta_{wi}$ as expected. Given Equation 4.2:

$$\boxed{q_{pwi} = \frac{\theta_{pi}\beta_{wi}}{\sum_j \theta_{pj}\beta_{wj}}}$$

For $\beta$ we have:

$$\begin{aligned}
\frac{\partial l}{\partial \beta_{wi}} &= -\sum_p \theta_{pi} + \frac{1}{\beta_{wi}} \sum_p a_{pw} q_{pwi} \\
&= -n + \frac{1}{\beta_{wi}} \sum_p a_{pw} q_{pwi} = 0
\end{aligned}$$

Therefore:

$$\boxed{\beta_{wi} = \frac{\sum_p a_{pw} q_{pwi}}{n}}$$

Finally, for $\theta$, we have

$$\begin{aligned}
\frac{\partial l}{\partial \theta_{pi}} &= -\sum_w \beta_{wi} + \frac{1}{\theta_{pi}} \sum_w a_{pw} q_{pwi} \\
&= \mu_i
\end{aligned}$$

where $\mu_i$ is the Lagrange multiplier for Constraint 4.1.

$$\mu_i = -\sum_w \beta_{wi} + \frac{1}{\theta_{pi}} \sum_w a_{pw} q_{pwi}$$

$$\theta_{pi}\mu_i = -\theta_{pi} \sum_w \beta_{wi} + \sum_w a_{pw} q_{pwi}$$

$$\sum_p \theta_{pi}\mu_i = -\sum_p (\theta_{pi} \sum_w \beta_{wi}) + \sum_p \sum_w a_{pw} q_{pwi}$$

$$\mu_i \sum_p \theta_{pi} = -\sum_w \beta_{wi} \sum_p \theta_{pi} + \sum_{pw} a_{pw} q_{pwi}$$

Applying Constraint 4.1, we have:

$$\mu_i = -\sum_w \beta_{wi} + \frac{1}{n} \sum_{pw} a_{pw} q_{pwi}$$

Therefore:

$$-\sum_w \beta_{wi} + \frac{1}{n} \sum_{pw} a_{pw} q_{pwi} = -\sum_w \beta_{wi} + \frac{1}{\theta_{pi}} \sum_w a_{pw} q_{pwi}$$

$$\implies \boxed{\theta_{pi} = n \frac{\sum_w a_{pw} q_{pwi}}{\sum_{pw} a_{pw} q_{pwi}}}$$

Thus we have derived all three iterative equations for the parameters.

To obtain $\theta$, $\beta$, and $q$, we begin with random values for any of them and run the algorithm for either a maximum number of iterations has been reached or until the difference between the log-likelihood values from any two successive iterations has fallen lower than a threshold $\varepsilon$

### 4.5.2 Ranking new websites in terms of likelihood scores

The question we want to ask is: if we have observed a network for a period of time and are able to infer the group affinity distribution for each machine in the network, i.e. $\theta_p$ for all $p \in P$, if a subset of $P$ starts visiting a new set of websites $W'$, what can we say about the occurrence likelihood of each $w' \in W'$? More specifically, if two new websites $w'_1$ and $w'_2$ appear and:

- $w'_1$ is visited by a subset of $P$ and most people in this subset are in the same group (meaning that they are interested in the same topics).
- $w'_2$ is visited by a different subset of $P$ and people belong to a diverse set of groups.

can we devise a method to easily distinguish the two?

The answer is simple: with every new $w' \in W'$, we can create a new adjacency matrix $a_{pw'}$ that is a single column. We then use the affinity matrix $\theta$ that we inferred previously to infer the group likelihood distribution $\theta_{w'}$ by maximizing:

$$l_{w'} = \sum_p \left[ -\sum_i \theta_{pi}\beta_{w'i} + a_{pw'} \log \sum_i \theta_{pi}\beta_{w'i} - \log a_{pw'}! \right] \tag{4.5}$$

The value $l_{w'}$ for each $w' \in W$ then is indicative of how "unusual" $w'$ is given what we know about the group structure $\theta_p$ of the people in the network. Intuitively, the more diverse the visitors of $w'$ are in terms of groups that they are interested in, the lower the value of $l_{w'}$ should be.

## 4.6 Experiment

### 4.6.1 Dense synthetic graphs

We took the following steps to create a single dense synthetic graph:

1. Choose values for $k$, $n$, $m$.

2. Choose a primary interest constant $\rho$ that determines the group that the node is most strongly affiliated with. The interest levels in the other $k-1$ groups will be randomly picked such that they sum to $1-\rho$. For each graph, $\rho$ is applied to every node.

   For example, if the node $p$ (or $w$) is most strongly affiliated with the second group out of three and $\rho = 0.6$, the value of $\theta_p$ (or $\beta_w$) may be $\begin{pmatrix} 0.1 & 0.6 & 0.3 \end{pmatrix}$. The dominant group of every node is selected randomly.

3. Normalize $\theta$ according to constraint 4.1 and create the weighted adjacency matrix $a$ where:

$$a_{pw} \sim \text{Poi}\left(\sum_i \theta_p \beta_w\right)$$

4. Record the index of the dominant group for each $p$ and $w$. The dominant group of any node $p \in P$ is

$$d_p = \underset{1 \le i \le k}{\text{argmax}}\, \theta_{pi}$$

5. We define the original group assignment $A$ of the graph $G$: $A(G) = \{A_i \; \forall i \in \{1, .., k\}\}$ where $A_i = \{x \in P \cup W \mid d_x = i\}$. In other words, the set $A_i$ only contains the nodes whose dominant group is $i$.

We pick four different values for $\rho$: 0.5, 0.6, 0.7, 0.8. For each $\rho$, we create a set of five different graphs where $n = 1000$, $m = 2000$, and $k = 5$. The average number of edges of the graphs is about 1.9 million edges.

We then run Group Profiler on each graph to infer the solution $\hat{\theta}$ and $\hat{\beta}$, from which we then construct the inferred group assignment $\hat{A}$ in the same way that we compute $A$ from $\theta$ and $\beta$. The stop condition is that either 300 iterations have been reached or the difference between the log-likelihood scores of any two successive iterations is smaller than 0.001.

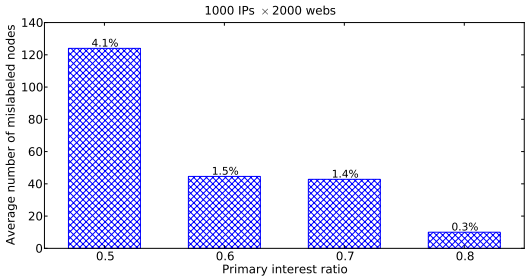



Figure 4.6: Performance of Group Profiler on dense graphs

We define the distance function between two group assignments $A$ and $\hat{A}$ as follows:

$$\delta(A, \hat{A}) = \frac{1}{2} \sum_i |A_i \triangle \hat{A}_i|$$

where $\triangle$ indicates the set symmetric difference operator[2].

Recall that the model retains a rotation symmetry that is the permutation of the $k$ groups. To assess the accuracy of Group Profiler, we first compute all possible permutations of the **labels** of $\hat{A}$. We say that the number of mislabeled nodes after we run Group Profiler on a graph $G$ is: $M(G) = \min_\pi(\delta(A, \hat{A}^\pi))$ where $\hat{A}^\pi$ is a label permutation of $\hat{A}$.

In Figure 4.6, we show the performance of Group Profiler. Recall that we have a set of 5 graphs for each value of $\rho$. We compute $M(G)$ for each such graph and for each value

[2]Applying the symmetric difference operation on two sets will yield a set of elements that only appear in one of the two sets but not both

of $\rho$, we average them. What's notable in the figure is the higher average of the number of mislabeled nodes for $\rho = 5.0$. This is because there are times that the randomization creates a $\theta_p$ or $\beta_w$ in which there is a secondary interest that has a value as high as the primary interest; for example: $\begin{pmatrix} 0.49 & 0.5 & 0.01 & 0 & 0 \end{pmatrix}$. As a result, the inferred $\hat{\theta}_p$ or $\hat{\beta}_w$ may return a vector that looks like $\begin{pmatrix} 0.5 & 0.5 & 0 & 0 & 0 \end{pmatrix}$ where the primary and secondary interest levels are indistinguishable.

### 4.6.2 Convergence of log-likelihood values.

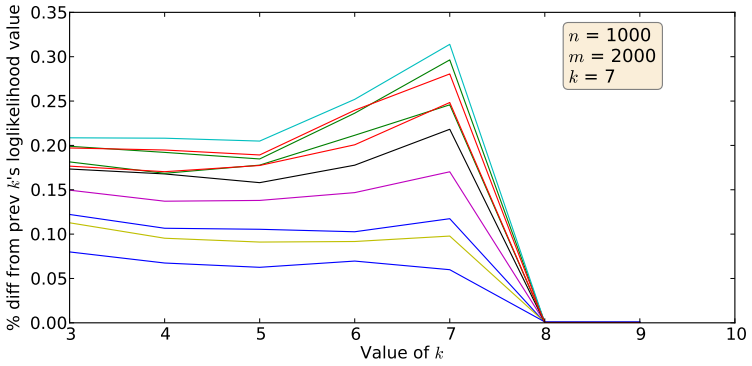



Figure 4.7: Convergence of log-likelihood values at the correct $k$

In this experiment, we investigated the effect of applying Group Profiler with different values of $k$ on graphs that were created by specific $k$s. More specifically, we created a set of 10 graphs with $n = 1000$, $m = 2000$, $k = 7$, and $\rho = 0.7$. We ran Group Profiler on each graph with $2 \leq k \leq 9$ and record the ratio of the change in log-likelihood values for every pair of successive $k$'s. Formally, if $\mathcal{L}_k^G$ and $\mathcal{L}_{k+1}^G$ are, respectively, the log-likelihood values computed by running Group Profiler on the graph $G$ with $k$ and $k+1$, what we record is:

$$\frac{\mathcal{L}_{k+1}^G - \mathcal{L}_k^G}{\mathcal{L}_k^G}$$

In Figure 4.7, we show the results of running Group Profiler with various $k$ values for each of the 10 graphs. We can see that the log-likelihood difference ratios all plummet after $k = 7$, which is the true $k$ value.

### 4.6.3 Sparse synthetic graphs

We first created a graph $G$ with primary interest level $\rho = 0.7$, $n = 1000$, $m = 2000, k = 5$. First we determined the maximum number of edges $E_{\max} = n \times m$, the number of intra-group edges $E_{\text{intra}}$, and the number of inter-group edges $E_{\text{inter}}$. Based on the original group assignment, an intra-group edge is an edge between two nodes whose dominant groups are the same. We then proceed to make the graph $G$ sparser through the removal of edges by taking the following steps:

- Pick a sparsity ratio $\sigma$, which is the ratio of the number of edges that are left after removal and $E_{\max}$.
- Calculate the number of edges that need to be removed, $E_{\text{rem}}$, according to $\sigma$. For instance, if $G$ has 1.9 million edges, $\sigma = 0.50$ means that we need to reduce $G$ to $0.5 \times 1000 \times 2000 = 10^6$ edges, which means we need to remove $0.9 \times 10^6$ edges from $G$.
- We **randomly** remove $E_{\text{rem}}$ in the same proportions between $E_{\text{intra}}$ and $E_{\text{inter}}$. If $E_{\text{intra}} = 2 \times E_{\text{inter}}$, we randomly remove twice as many intra-group edges as we would inter-group edges.

In Figure 4.8, we show the effect of increasing the sparsity of the graphs on the performance of Group Profiler. Although the algorithm still retains high accuracy even at $\sigma = 0.05$, which means that $G$ has only $10^5$ edges out of a maximum of $2 \times 10^6$ edges, the accuracy drops rather dramatically as the $\sigma$ decreases.

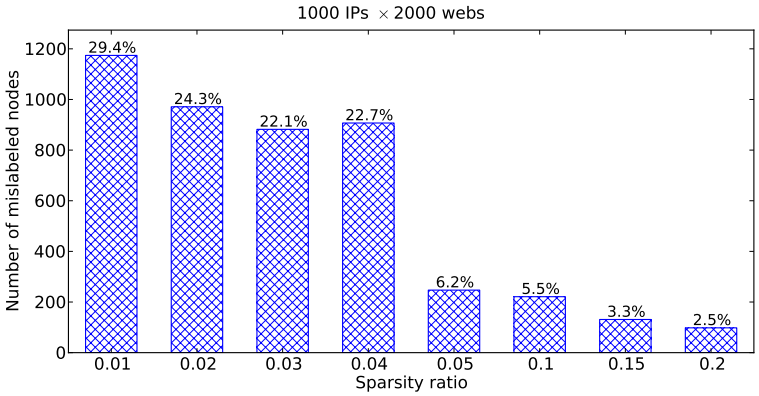



Figure 4.8: Performance of Group Profiler on sparse graphs

The reason is that as we keep removing edges from $G$, the group structure begins to break down as well. Removing enough edges and a single tightly connected group of people and websites would be split into smaller groups and we will have a situation where there are actually **more than** $k$ groups in the graph.

### 4.6.4 Anomaly detection on a sparse synthetic graph.

We created a synthetic sparse graph $G$ with the following parameters: $\rho = 0.7$, $n = 1000$, $m = 2000$, and $\sigma = 0.20$. We then created new websites that fall under two categories:

(I) Websites that are visited by people who have the same dominant group, as determined by the inferred $\hat{\theta}$.

(II) Websites that are visited by people randomly picked from the set $P$.

In essence, those new websites are new **columns** in the weighted adjacency matrix whose weights are randomly chosen. Given a fraction value $\phi$, we took the following steps:

1. Select a group that is sufficiently large. We call this group $g$ and the number of people in this group $|g_P|$.

2. Select $n' = \lfloor \phi |g_P| \rfloor$ people from $g$ and create a column $\mathbf{w_g}$ that represents a website that the selected people visit. This website falls under category (I).

3. Select the same $n'$ from the whole set of people $P$ and create a column $\mathbf{w_P}$ that represents a website that the selected people visit. This website falls under category (II).

4. Infer the log-likelihood value for both $w_g$ and $w_P$ according to equation 4.5.

In Figure 4.9, we show what happens when we gradually increase the value of $\phi$. The hypothesis is that a website that is visited by people in the same group should have a higher log-likelihood value than one that is visited by people who are affiliated with different groups. Furthermore, the more people who are interested in a never-before seen website, the lower the value of the website's log-likelihood should be. Indeed, as the Figure shows, this is the case.

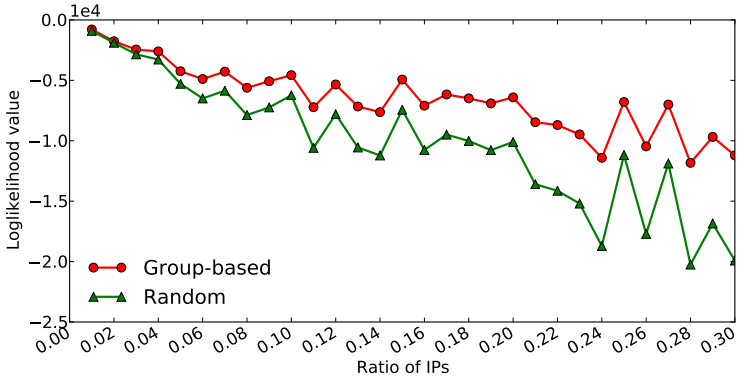



Figure 4.9: Anomaly detection on a sparse graph

## 4.7 Related Work

The algorithm presented in our work is based on the Stochastic Block Model[23] and developed from the work of Ball, Karrer, and Newman[9], where they presented an algorithm to infer the group structure and affinity matrices of uni-partite graphs. The topic of detecting group structure in uni-partite graphs was also explored at length by Decelle et. al. [17] who instead adapted the Belief Propagation algorithm to infer the group structure instead of the MLE technique. The most recent work that is similar to ours is by Larremore et. al. [33], who also adapted the work in [9] for bipartite graphs. However, unlike our approach, [33] presented a hard co-clustering approach where any given node in the graph belongs to one group only and therefore not suitable for the purpose of leveraging the group structure to detect anomalies.

Closely related to the field of data clustering and group detection is Collaborative Filtering (CF), which is commonly used for creating recommendation systems for e-commerce retailers. In Collaborative Filtering, there are three major directions of research:

Memory-based, Model-based, and the Hybrid of the first two. Memory-based CF techniques [45][37] are often deployed to online e-commerce systems. They can be easily and efficiently implemented but tend to suffer from the problem of data sparsity. Model-based CF techniques[10][32] are better designed to address the sparsity problem but can be expensive and there is no guarantee of convergence. The Hybrid techniques like [41] have improved prediction over the Model-based ones and can overcome data sparsity but can be even more expensive.

# Bibliography


[1] Argus: Auditing network activity. http://www.qosient.com/argus/.

[2] Cybercrime Tracker. http://cybercrime-tracker.net/.

[3] Malc0de Database. http://malc0de.com/database/.

[4] Malware Domain List. http://www.malwaredomainlist.com/.

[5] Mawi traffic archive. http://mawi.wide.ad.jp/mawi/samplepoint-B/20060303/.

[6] Selenium, Web Browser Automation. http://docs.seleniumhq.org/.

[7] Vx vault. http://vxvault.siri-urz.net/ViriList.php.

[8] Web of Trust Reputation API. https://www.mywot.com/wiki/API.

[9] Ball, B., Karrer, B., and Newman, M. Efficient and principled method for detecting communities in networks. *Physical Review E 84*, 3 (2011), 036103.

[10] Billsus, D., and Pazzani, M. J. Learning collaborative information filters. In *ICML* (1998), vol. 98, pp. 46–54.

[11] Binkley, J. R., and Singh, S. An algorithm for anomaly-based botnet detection. In *Proc. of USENIX SRUTI* (July 2006).

[12] Blondel, V., Guillaume, J., Lambiotte, R., and Lefebvre, E. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment* (2008).

[13] Caballero, J., Grier, C., Kreibich, C., and Paxson, V. Measuring pay-per-install: The commoditization of malware distribution. In *Usenix Security 2011*.

[14] Canali, D., Balzarotti, D., and Francillon, A. The role of web hosting providers in detecting compromised websites. In *WWW 2013*.

[15] Canali, D., Cova, M., Vigna, G., and Kruegel, C. Prophiler: a fast filter for the large-scale detection of malicious web pages. In *WWW 2011*, ACM.

[16] Coskun, B., Dietrich, S., and Memon, N. Friends of an enemy: identifying local members of peer-to-peer botnets using mutual contacts. In *Proc. of ACSAC 2010*, ACM, pp. 131–140.

[17] Decelle, A., Krzakala, F., Moore, C., and Zdeborová, L. Asymptotic analysis of the stochastic block model for modular networks and its algorithmic applications. *Physical Review E 84*, 6 (2011), 066106.

[18] Dumitras, T., and Shou, D. Toward a standard benchmark for computer security research: The worldwide intelligence network environment (wine). In *BADGER 2011*, ACM.

[19] Fortunato, S. Community detection in graphs. *Physics Reports 486*, 3 (2010).

[20] Grier, C., Ballard, L., Caballero, J., Chachra, N., Dietrich, C. J., Levchenko, K., Mavrommatis, P., McCoy, D., Nappa, A., Pitsillidis, A., et al. Manufacturing compromise: the emergence of exploit-as-a-service. In *CCS 2012*, ACM.

[21] Gu, G., Perdisci, R., Zhang, J., and Lee, W. Botminer: Clustering analysis of network traffic for protocol-and structure-independent botnet detection. In *Proc. of Usenix Security 2008*, USENIX Association, pp. 139–154.

[22] Gu, G., Porras, P., Yegneswaran, V., Fong, M., and Lee, W. Bothunter: Detecting malware infection through ids-driven dialog correlation. In *Proc. of 16th USENIX Security Symposium* (2007), USENIX Association, p. 12.

[23] Holland, P. W., Laskey, K. B., and Leinhardt, S. Stochastic blockmodels: First steps. *Social networks 5*, 2 (1983), 109–137.

[24] Holz, T., Steiner, M., Dahl, F., Biersack, E., and Freiling, F. Measurements and mitigation of peer-to-peer-based botnets: A case study on storm worm. In *Proc. of LEET 2008*.

[25] Hussain, A., Heidemann, J., and Papadopoulos, C. A framework for classifying denial of service attacks. In *Proc. of SIGCOMM* (August 2003).

[26] Iliofotou, M., Gallagher, B., Eliassi-Rad, T., G., X., and M., F. Profiling-by-association: A resilient traffic profiling solution for the internet backbone. In *Proc. of ACM CoNEXT* (Dec. 2010).

[27] Iliofotou, M., Kim, H., Faloutsos, M., Mitzenmacher, M., Pappu, P., and Varghese, G. Graption: A graph-based p2p traffic classification framework for the internet backbone. *Computer Networks* (2011).

[28] Iliofotou, M., Pappu, P., Faloutsos, M., Mitzenmacher, M., Singh, S., and Varghese, G. Network Monitoring using Traffic Dispersion Graphs. In *Proc. of IMC* (2007).

[29] Jin, Y., Sharafuddin, E., and Zhang, Z. Unveiling core network-wide communication patterns through application traffic activity graph decomposition. In *Proc. of SIGMETRICS 2009* (2009), ACM, pp. 49–60.

[30] John, J. P., Moshchuk, A., Gribble, S. D., and Krishnamurthy, A. Studying spamming botnets using botlab. In *Proc. of NSDI 2008*.

[31] Kührer, M., Rossow, C., and Holz, T. Paint it black: Evaluating the effectiveness of malware blacklists. In *Research in Attacks, Intrusions and Defenses*. Springer, 2014.

[32] Landauer, T. K., and Littman, M. L. Computerized cross-language document retrieval using latent semantic indexing, Apr. 5 1994. US Patent 5,301,109.

[33] Larremore, D. B., Clauset, A., and Jacobs, A. Z. Efficiently inferring community structure in bipartite networks. *arXiv preprint arXiv:1403.2933* (2014).

[34] Le, A., Markopoulou, A., and Faloutsos, M. Phishdef: Url names say it all. In *INFOCOM 2011*, IEEE.

[35] Li, Z., Alrwais, S., Xie, Y., Yu, F., and Wang, X. Finding the linchpins of the dark web: a study on topologically dedicated hosts on malicious web infrastructures. In *S&P 2013*, IEEE.

[36] Li, Z., Goyal, A., Chen, Y., and Paxson, V. Towards situational awareness of large-scale botnet probing events. In *IEEE Transactions on Information Forensics & Security* (March 2011).

[37] Linden, G., Smith, B., and York, J. Amazon. com recommendations: Item-to-item collaborative filtering. *Internet Computing, IEEE 7*, 1 (2003), 76–80.

[38] Livadas, C., Walsh, R., Lapsley, D., and Strayer, W. T. Using machine learning techniques to identify botnet traffic. In *Proc. of WoNS* (2006).

[39] Ma, J., Saul, L. K., Savage, S., and Voelker, G. M. Beyond blacklists: learning to detect malicious web sites from suspicious urls. In *SIGKDD 2009*, ACM.

[40] Mavrommatis, N. P. P., and Monrose, M. A. R. F. All your iframes point to us. In *Usenix Security 2008*.

[41] Melville, P., Mooney, R. J., and Nagarajan, R. Content-boosted collaborative filtering for improved recommendations. In *AAAI/IAAI* (2002), pp. 187–192.

[42] Mills, E. Google finds 9,500 new malicious Web sites a day. `www.cnet.com/news/google-finds-9500-new-malicious-web-sites-a-day/`, June 2012.

[43] Nagaraja, S., Mittal, P., Hong, C., Caesar, M., and Borisov, N. Botgrep: Finding p2p bots with structured graph analysis. In *Proc. of the 19th USENIX conference on Security* (2010), USENIX Association.

[44] Papalexakis, E. E., Dumitras, T., Chau, D. H. P., Prakash, B. A., and Faloutsos, C. Spatio-temporal mining of software adoption & penetration. In *ASONAM 2013*, ACM.

[45] Resnick, P., Iacovou, N., Suchak, M., Bergstrom, P., and Riedl, J. Grouplens: an open architecture for collaborative filtering of netnews. In *Proceedings of the 1994 ACM conference on Computer supported cooperative work* (1994), ACM, pp. 175–186.

[46] Roesch, M., et al. Snort: Lightweight intrusion detection for networks. In *Proc. of LISA* (1999), pp. 229–238.

[47] Seifert, C., Welch, I., Komisarczuk, P., Aval, C. U., and Endicott-Popovsky, B. Identification of malicious web pages through analysis of underlying dns and web server relationships. In *LCN 2008*.

[48] Soska, K., and Christin, N. Automatically detecting vulnerable websites before they turn malicious. In *Usenix Security 2014*.

[49] Stokes, J. W., Andersen, R., Seifert, C., and Chellapilla, K. Webcop: Locating neighborhoods of malware on the web. In *USENIX LEET 2010*.

[50] Stover, S., Dittrich, D., Hernandez, J., and Dietrich, S. Analysis of the storm and nugache trojans: P2p is here. *USENIX;login 32*, 6 (2007), 2007–12.

[51] Stringhini, G., Holz, T., Stone-Gross, B., Kruegel, C., and Vigna, G. Botmagnifier: Locating spambots on the internet. In *Proc. of Usenix Security 2011*.

[52] Tung, L. Storm worm: More powerful than blue gene? `http://www.zdnet.com/storm-worm-more-powerful-than-blue-gene-3039289226/`.

[53] Vacca, J. *Computer and Information Security Handbook*. Morgan Kaufmann, 2009.

[54] Whitney, L. With legal nod, microsoft ambushes waledac botnet. `http://news.cnet.com/8301-1009_3-10459558-83.html`.

[55] Witten, I. H., Frank, E., Trigg, L. E., Hall, M. A., Holmes, G., and Cunningham, S. J. WEKA: Practical machine learning tools and techniques with Java implementations.

[56] Xie, G., Iliofotou, M., Karagiannis, T., Faloutsos, M., and Jin, Y. Resurf: Reconstructing web-surfing activity from network traffic. In *IFIP Networking Conference, 2013* (2013), IEEE, pp. 1–9.

[57] Yen, T., and Reiter, M. Are your hosts trading or plotting? Telling p2p file-sharing and bots apart. In *Proc. of ICDCS 2010*, IEEE, pp. 241–252.

[58] Zhang, J., Perdisci, R., Lee, W., Sarfraz, U., and Luo, X. Detecting stealthy p2p botnets using statistical traffic fingerprints. In *Proc. of DSN* (2011), IEEE.

[59] Zhao, Y., Xie, Y., Yu, F., Ke, Q., Yu, Y., Chen, Y., and Gillum, E. Botgraph: Large scale spamming botnet detection. In *Proc. of NSDI 2009*.