The Latent Community Model for Detecting Sybils in Social Networks

Zhuhua Cai
Rice University
zc7@rice.edu

Christopher Jermaine
Rice University
cmj4@rice.edu

Abstract

Collaborative and recommendation-based computer systems are plagued by attackers who create fake or malicious identities to gain more influence in the system—such attacks are often referred to as "Sybil attacks". We propose a new statistical model and associated learning algorithms for detecting Sybil attacks in a collaborative network using network topology, called the latent community *(LC) model. The LC model is hierarchical, and groups the nodes in a network into closely linked communities that are linked relatively loosely with the rest of the graph. Since the author of a Sybil attack will typically create many false identities and link them together in an attempt to gain influence in the network, a Sybil attack will often correspond to a learned community in the LC model. Evaluation of the LC model using both real-world and synthetic networks demonstrates the promise of the method.*

1 Introduction

In distributed systems that rely on user recommendations and collaborations to determine the importance or influence of users, malicious users try to create multiple identities with the aim of increasing their own influence in the system. This is often called a *Sybil attack*. Sybil attacks are found in various domains, from security and routing in peer-to-peer networks to collaborative voting and recommendation systems.

There are two complimentary tactics for dealing with Sybil attacks. The first is prevention: building defense mechanisms that make it impossible for attackers to gain access to the network in the first place, usually through identity verification schemes. This paper considers the second tactic: mitigation, or detecting Sybils by their abnormal characteristics. In this approach, the network administrator monitors network status, looking for attacks, so that the attack can be removed from the network [34, 33, 29, 8, 32]. Existing methods for detecting Sybil attacks [34, 33, 29, 8] assume that an attacker begins by creating a set of "bad" nodes, and then builds a network of arbitrary topology among them. A large component that is disconnected from the rest of the network would be an obvious sign of a Sybil attack, so the attacker also attempts to compromise (or to trick) a set of "good" nodes into linking with some of the bad nodes. Since one might assume that it is expensive or difficult for an attacker to link with a good node, the attack can be detected by finding a subnetwork that is connected to the main network via relatively few links—that is, a narrow "choke point".

In our opinion, the presence of such a choke point is probably a good indicator of an attack, but we are concerned that the methods that look only for choke points will miss many attacks, particularly more sophisticated ones. One could imagine circumstances where an attacker is able to induce more than a few connections between good and bad nodes, through mechanisms such as phishing [27, 14, 32], or through worms or viruses that take control of good nodes long enough to create connections with bad ones. Even if mechanisms such as phishing are not available to an attacker in a particular application domain, we believe that the fewer assumptions that a mitigation scheme makes (such as the existence of a choke point), the better.

As such, we propose a fundamentally different, machine learning-based approach to detecting Sybil attacks, called the *latent community* (LC) model. Not surprisingly, the latent community model relies on partitioning the network into *communities*, which are subsets of the network that have strong internal connections. Community detection in graphs is not a new problem; in fact, it is a widely-studied and mature field with long history. We will cover some of the related work from this area of research subsequently; the essential survey on the area is due to Fortunato [10]. But while community detection is widely-studied, community detection methods are not obviously applicable to the problem of automatic Sybil detection. To detect Sybils, it is not enough to partition the network into tightly-connected communities; these communities must simultaneously be analyzed by seeing whether they abnormally connect with the other ones.

The novelty of the LC-based approach is that as communities are learned, they are simultaneously positioned in a latent, Euclidean space so that communities with tight interconnections are positioned more closely than communities that are loosely connected. Euclidean space respects the transitivity one might expect in a network that subnetworks near to each other tend to have a large number of connections. In both a network and in Euclidean space, if communities A and B are closely related (connected), as are communities B and C, then A and C will usually to be closely related.

The advantage of this latent positioning is that in the LC model, attack communities might tend to be outliers, since if they are attached to the “good” portion of the network in a way that is inconsistent with other communities, they will tend to be pushed to the “outside” of the the latent space. For example, imagine community A (consisting mostly of attackers) connects tightly with community B, which it has compromised. B connects tightly with a central community C, but A connects sparingly with C. Then A will be forced away from C and to the “outside” of the model. We give what appears to be a real-world example of this in Section 5.2 of the paper.

Our Contributions The contributions of this paper are:

- We propose the latent community model for partitioning a graph into subnetworks.
- We apply the model to detect Sybil attacks in social networks. Although there are numerous models for generating social networks and graph partitions, they are not specifically designed for this problem.
- To ensure its acceptable practical performance, we propose a Bayesian inference approach for learning the LC model, as well as associated MCMC algorithms.
- We show experimentally that our LC-based Sybil detector competes well with the state of the art algorithms for the Sybil detection.

2 The LC Model

In the LC model, the nodes in a network are partitioned into *communities*, which are sets of nodes with (relatively) dense interconnections. Each community is associated with a latent position in a multi-dimensional Euclidean space (hence the name “latent community model”); the position of each community dictates how its nodes connect with other communities. Communities that are close have many links between them; far apart communities have few links.

We employ a standard, statistical machine learning approach. The LC model describes a process whereby the statistics describing node interconnections in a graph are generated. By reversing the process and figuring out exactly how a particular network could have been stochastically produced via “learning” or “inference”, we reveal the community structure of the network.

While it would be possible for a human being to directly examine the communities learned via the LC model to search for Sybils, in the next section we will extend the model to detect Sybils automatically.

2.1 The Generative Process

The generative process underlying the LC model is as follows. The nodes in a network are partitioned among a set of communities. The ith community has a latent position μ_i, where $\mu_i \sim F(\theta)$. “$\sim$” should be read as “is sampled from”. $F(\theta)$ is some multi-dimensional distribution used to position the various communities in space. Any appropriate F can be chosen (we will use a multi-variate normal or Gaussian distribution as well as a special-purpose “ring” distribution subsequently). We use $\mathbf{c}$ to denote the vector of community sizes, so that $\mathbf{c}_i$ is the number of nodes in the ith community. $\mathbf{E}$ is the upper-triangular matrix of edge counts, where $\mathbf{E}_{ij}$ is the number of edges between community i and community j.[1] Finally, δ_i is the probability that two nodes in the ith community are connected.

Given this setup, the following stochastic process underlies the LC Model:

1. For each community, $\mu_i \sim F(\theta)$.
2. For each community, the number of edges connecting internal nodes is generated as $\mathbf{E}_{ii} \sim$Binomial$(\binom{\mathbf{c}_i}{2}, \delta_i)$.

[1]This paper assumes an undirected graph; the extension to directed graphs is slight and straightforward.

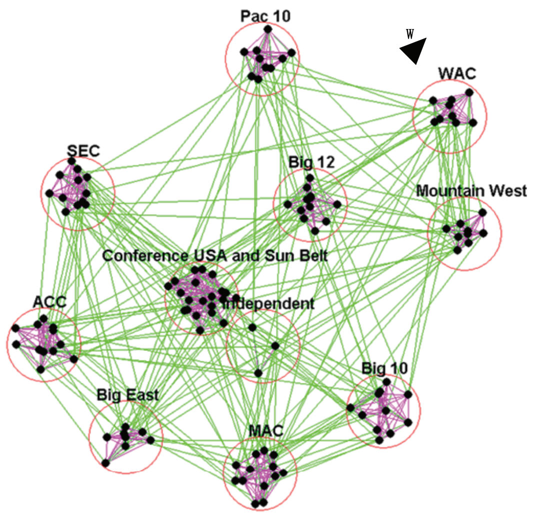

Figure 1: Learned model for the 2010 college football data set.

3. For each pair of distinct communities, the number of cross-community edges is generated as: $\mathbf{E}_{ij} \sim$Binomial$(\mathbf{c}_i \times \mathbf{c}_j, \mathbf{min}(\delta_i, \delta_j) \times \eta^{-\mathbf{ed}(\mu_i,\mu_j)})$.

The above generative process is quite simple. Step (1) positions each community in space by drawing its location from a random variable having distribution F. Step (2) links each pair of nodes in community i with probability δ_i. In step (3), pairs of nodes from different communities are linked depending upon the distance between the communities—$\mathbf{ed}(\mu_i, \mu_j)$ denotes the Euclidean distance between the latent positions associated with the communities. Thus, the probability that two nodes across communities are linked drops exponentially with increasing Euclidean distance between the communities; η is simply a scaling factor.

It is easily possible to make the LC model fully Bayesian by putting appropriate priors on all of the parameters. In our implementation, we give both δ_i and η Beta$(1, 1)$ priors. Letting $n = \sum_i \mathbf{c}_i$, we give $\mathbf{c}$ a Multinomial(π, n) prior. In this prior, π is a vector of probabilities, where the ith entry in π is the probability that a random node falls in $\mathbf{c}_i$. We give π a Dirichlet(1) prior.

2.2 Example

To make the LC model more concrete, as an illustrative example we apply it to the 2010 American FBS college football schedule [1].

The schedule for all 120 FBS football teams can be made into a 683-edge graph by placing an edge between two teams if and only if the two teams play one another. One would expect that there are twelve natural communities in this graph, because the 120 teams are organized into twelve so-called "conferences" (the conferences are called the Big Ten, the Pac Ten, the Big Twelve, etc.); the likelihood that two teams within the same conference play each other is much higher than the likelihood that two teams that are not in the same league will play each other. Thus, we assume that the graph was generated using the LC model and we attempt to learn the unseen parameters. In this example, the function $F(\mu_i|\theta)$ is a two-dimensional standard normal distribution.

The model learned is visualized in Figure 1. The black points represent the football teams, and the red circles are the communities inferred by the model. The center of a circle denotes the latent position of the corresponding community. The magenta lines denote interconnections within communities, and the edges among communities are green.

This example illustrates two key aspects of the LC model. First, it illustrates how the notion of a "conference" or a "league" in the sporting world—where teams within a conference play each other with high frequency, and play out-of-conference with less regularity—is almost identical to the notion of a "community" in the LC model. Given the option of identifying up to twelve communities in the graph, the learning process picked out eleven (the twelfth was left empty). The learned communities correspond perfectly to the twelve FBS conferences, with the one exception that the Conference USA and Sunbelt conferences are placed together in the same community.

Second, it illus rates how the learned, latent positions of communities can correspond to real-world phenomena. In the LC model, communities that are close to each other in the latent Euclidean space are more densely connected than communities that are far away from one another. Looking at Figure 1, it is clear that the learned latent positions actually correspond roughly to the geographic locations of the various conferences. In retrospect, this makes sense. In general, schools scheduling out-of-conference games will tend to play against schools that are physically close. This is why the "WAC" (or Western Athletic Conference) and Pac 10 are along the upper right edge of the latent space—these conferences consist of football teams from schools like UCLA, USC, and BYU that are along the west coast of the USA. The "ACC" (Atlantic Coast Conference) and Big East are on the opposite side of the latent space, and consist of schools like Duke, Rutgers, and Florida State on the US east coast.

3 Application to Sybil Detection

The generic LC model as described in the previous section can be used directly (along with human examination of the learned model) to detect Sybil attacks. We will examine this approach subsequently in the experimental section. In this section, however, we extend the LC model so that it is able to automatically detect Sybil attacks. Our basic tactic will be to assume that the latent community positions are generated via a mixture of two distributions: a Gaussian distribution that positions the benign communities close to the center of the space, and a spherical distribution of attackers that surrounds the Gaussian distribution (we will call this the "ring" distribution). Nodes that are found to likely belong to the mixture component associated with the attackers are then flagged as malicious.

3.1 Assumptions

We begin with a few assumptions:

1. A special set of size s of the graph's nodes is known to be benevolent; they are called the "seeds".
2. Nodes in the same community are either uniformly malicious or uniformly benign.

A bit of explanation of these assumptions is warranted.

The assumption that we have a set of seeds is required to break the symmetry between benevolent nodes and Sybils, who are free to create any topology among themselves, and can thus mimic the structure of the benevolent portion of the graph. In our model, we use a prior that tends to position communities with seeds near the origin of the Euclidean space. Seeds have been found to be trustworthy by a human expert, or else via automatic automatic methods—the longest-lived or most popular sites can be used. A similar idea is employed in existing Sybil defense systems [34, 8, 29], though those methods assume that only a single seed node is trustworthy.

The assumption of uniformity makes sense because in the LC model, nodes within communities are (by definition) connected with a uniform density. In a Sybil attack, it seems unlikely that a set of malicious nodes would be able to so thoroughly integrate themselves into a community of benign nodes that there is no real difference in the connection density between the benign nodes in the community and the attackers, though some attackers can possibly establish multiple links with the whole network [27]. Even if such an integration *did* occur, those benign nodes would be so thoroughly compromised that labeling them as attackers would not be an egregious error.

3.2 Applying the Model

Given this set of assumptions, the actual extension required to the LC model to allow for detection of Sybil attacks are as follows:

- The vector $\mathbf{c}$ of community sizes is now produced via a two-step process. First, the s "seed" nodes are assigned to communities using $\mathbf{s} \sim$ Multinomial(s, π); $\mathbf{s}_i$ is the number of seed nodes assigned to community i. Then the remainder of the nodes are assigned, so that $\mathbf{c} \sim$ Multinomial$(n - s, \pi) + \mathbf{s}$.
- Each community now has an additional variable ϕ_i associated with it. ϕ_i is 1 if the ith community is malicious, and 0 otherwise. If $\mathbf{s}_i$ is non-zero, then the ith community must be benevolent (since it has a seed node) and ϕ_i is zero. The status of the remaining communities is determined using $\phi_i \sim$ Bernoulli(β), with an appropriate Beta prior on β.
- Rather than having the latent position μ_i produced by a distribution F taking only θ as input, F now takes the form $F(\theta|\phi_i)$—that is, F is free to treat malicious and benevolent communities differently and now allow for latent positions to be generated via a mixture. For example, if ϕ_i is 1, then F can tend to scatter the community widely, but if it is 0, F will tend to locate the community centrally.

Given this extended version of the LC model, detecting a set of attackers is quite simple. Using a some sort of inference (such as an MCMC algorithm or a variational method), all of the unknown parameters and variables are estimated, including which nodes are in which communities, as well as whether each community is tagged as malicious. After learning completes, all of the nodes that likely belong to a community having an ϕ_i value of 1 are then returned as attacking nodes.

3.3 Generating Latent Positions

Thus far, we have been a bit coy as to exactly what form F should take. In principal, our model admits any distribution here, but we use the following variation for automatic Sybil detection. If ϕ_i is 0, then

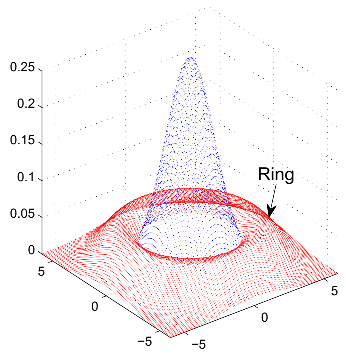

Figure 2: The probability density functions for a 2-dimensional normal distribution and a "ring" distribution.

$F(\mu_i|\theta, \phi_i)$ is simply a Normal$(\mu_i|\overrightarrow{0}, I)$ distribution, so that all of the benevolent communities are normally scattered around the origin. If ϕ_i is 1, then we use a special "ring" distribution. This distribution is parameterized by a mean distance ρ from the origin (ρ is given an InverseGamma$(1, 1)$ prior). To generate a data point, a distance d from the origin is obtained by sampling from a one-dimensional Normal(ρ, I) distribution. Then a random direction is chosen, and the latent position is placed distance d from the origin in this direction. This process forces Sybil communities far away from the benevolent communities. As will be shown in our experiment, this strategy works very well to identify Sybils.

Figure 2 depicts one example of the two-dimensional version of the resulting distribution when $\rho = 4$.

4 Learning Algorithm

4.1 Introduction

So far, we have described how a set of communities and their various interconnection statistics are generated by the LC model, given a set of parameters Θ. This section describes our algorithm for learning the variant of the LC model described in the previous section, which makes use of seed nodes as well as the special "ring" distribution (the simpler version of the model described in Section 2 can be learned via a straightforward and restricted version of the learning algorithms described in this section).

We begin by assuming we are given a graph $G = (V, S, E)$ where V is the set of nodes, S is a subset of V (S is the set of "seed" nodes), and E is the set of edges. Since we employ a Bayesian approach, given G, "learning" means determining the posterior distribution $P(\Theta|G)$ over the parameter set Θ; Θ contains all of the unseen variables described in the last two sections.

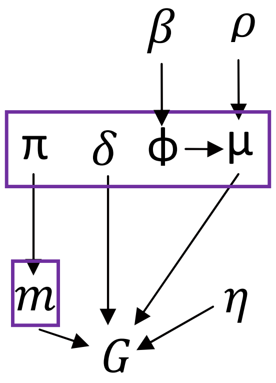

Figure 3: The Byesian network of our model with hyper-parameters removed. The boxes are "plates" representing replicates. The upper box represents communities, while the lower one represents nodes.

Before we describe our learning algorithms and exhaustively list the contents of Θ, it is important to point out that the generative process embodied by the LC model does not actually generate a graph; rather, it generates the matrix **E** of inter- and intra-community edge-counts, and the vectors **c** and **s** of community sizes (recall that **c** partitions all n nodes among the communities, and **s** partitions the size-s subset of "seed" nodes among communities). It is significant that none of these are directly observable given a graph G. Thus, to facilitate the learning of $P(\Theta|G)$, we add to Θ a membership vector **m**. $\mathbf{m}_i$ indicates which of the n communities the ith node in G belongs to; if $\mathbf{m}_i = j$, it means that ith node belongs to the jth community.

Given a particular value for m as well as the input graph $G = (V, S, E)$, it is possible to compute **E** as:

$$\mathbf{E}_{ij} = \begin{cases} \sum_{(a,b)\in E} I(\mathbf{m}_a = i)I(\mathbf{m}_b = j) & \text{If } i \neq j \\ \frac{1}{2} \times \sum_{(a,b)\in E} I(\mathbf{m}_a = i)I(\mathbf{m}_b = j) & \text{otherwise} \end{cases}$$

where I is the indicator function, returning one if the boolean argument is true (and zero otherwise). Likewise, **c** can be computed as:

$$\mathbf{c}_i = \sum_{v \in V} I(\mathbf{m}_v = i)$$

and

$$\mathbf{s}_i = \sum_{s \in S} I(\mathbf{m}_s = i)$$

Given this, $\Theta = \{\rho, \pi, \eta, \beta, \mathbf{m}, \delta, \phi, \mu\}$. A plate diagram showing the relationships between the variables is described in Figure 3.

4.2 Posterior Distribution

In this subsection, we tackle the problem of obtaining a formula for the desired posterior distribution, $P(\Theta|G)$.

From elementary probability, we know that:

$$P(\Theta|G) = \frac{P(G|\Theta)P(\Theta)}{P(G)}$$

Also, from the generative process described in Section 2, it follows that:

$$\begin{aligned} P(G|\Theta) = &\prod_i \text{Binomial}(\mathbf{E}_{ii}, \binom{\mathbf{c}_i}{2}, \delta_i)\times \\ &\prod_{i>j} \text{Binomial}(\mathbf{E}_{ij}, \mathbf{c}_i \times \mathbf{c}_j, \\ &\mathbf{min}(\delta_i, \delta_j) \times \eta^{-\mathbf{ed}(\mu_i,\mu_j)}) \end{aligned}$$

And from Figure 3 and the generative processes of Sections 2 and 3, (including the prior distributions listed there), it follows that:

$$\begin{aligned} P(\Theta) = &P(\rho)P(\pi)P(\eta)P(\beta)P(\mathbf{m}|\pi)\times \\ &\prod_i P(\delta_i)P(\phi_i|\beta, \mathbf{s}_i)P(\mu_i|\phi_i, \rho) \end{aligned}$$

where:

$$\begin{aligned} P(\rho) &= \text{InvGamma}(\rho, 1, 1) \\ P(\pi) &= \text{Dirichlet}(\pi, 1) \\ P(\eta) &= \text{Beta}(\eta, 1, 1) \\ P(\beta) &= \text{Beta}(\beta, 1, 1) \\ P(\mathbf{m}|\pi) &= \text{Multinomial}(\mathbf{m}, n, \pi) \\ P(\delta_i) &= \text{Beta}(\delta_i, 1, 1) \end{aligned}$$

$$P(\phi_i|\beta, \mathbf{s}_i) = \begin{cases} 0 & \text{If } \phi_i = 1 \text{ and } \mathbf{s}_i \geq 1 \\ 1 & \text{If } \phi_i = 0 \text{ and } \mathbf{s}_i \geq 1 \\ \text{Bernoulli}(\phi_i, \beta) & \text{otherwise} \end{cases}$$

$$P(\mu_i|\phi_i, \rho) = \begin{cases} \text{Normal}(\overleftarrow{0}, I) & \text{If } \phi_i = 0 \\ \text{Ring}(\mu_i, \rho, I) & \text{otherwise} \end{cases}$$

This gives us formulas for all of the components of $P(\Theta|G)$, except for $P(G)$. Unfortunately, obtaining an expression for this is very difficult, because it involves integrating out all of the variables in Θ from $P(G, \Theta)$. A common way around this problem (and several others associated with characterizing a complex posterior distribution such as ours) is to make use of an MCMC algorithm, such as a Gibbs sampler [4]. Some key advantages of using a Gibbs sampler are that (a) $P(G)$ is irrelevant when applying the Gibbs sampler, and (b) the Gibbs sampler actually obtains samples from $P(\Theta|G)$, which may be of more use than a closed form for the distribution itself. One can use those samples to estimate the mean and other interesting and/or useful statistics describing of each of the components of Θ; the samples can also be used to estimate joint statistics such as the covariances between variables. In the next subsection, we give a high-level overview of Gibbs sampling and how it can be applied to our problem.

4.3 Gibbs Sampling

Gibbs sampling works as follows:

1. Choose some initial values for all the parameters in Θ.

2. Iterate over the parameters in Θ, replacing the value of one parameter $\bar{\theta}$ by a value drawn from the distribution of this parameter conditioned on the graph G and other associated parameters, i.e., $\{P(\theta\ |\Theta_{\setminus\theta}, G)\}_{\theta\in\Theta}$.

3. Run step 2 for many iterations, which yields many samples for Θ, according to the distribution $P(\Theta|G)$, irrespective of the initial values of parameters in Θ.

In our case, obtaining a closed form for $P(\theta\ |\Theta_{\setminus\theta}, G)$ up to a constant multiplicative value for any given θ is easy; simply begin with $P(G) \times P(\Theta|G)$ from the previous section, and then remove any of the multiplicative terms that are constant with respect to θ (that is, remove all terms which do not include θ anywhere in the formula). Once the close form for $P(\theta\ |\Theta_{\setminus\theta}, G)$ (up to a constant multiplicative value) has been obtained, it is typically straightforward to obtain a pseudo-random sample from the resulting distribution using standard techniques such as rejection sampling [4].

Gibbs sampling is widely used, and details can be found in many references [4]. The "efficiency" of a Gibbs sampler is usually considered to be a function of how fast the above process drifts away from the initialization of Θ and produces samples from the actual posterior distribution. While describing anything more than the simplest Gibbs sampler applicable to our model is beyond the scope of the paper, one key benefit of applying Gibbs sampling to our problem is that as a well-studied and mature methodology, there are various standard "tricks" that can be used to improve the efficiency of the resulting learner.

5 Experimental Study

This section describes an experimental study of our models and associated learning algorithms. Our goals are twofold:

1. To illustrate how the simple LC model from Section 2 can be used to analyze a real, medium-to-large social network, and help a human expert identify potential Sybils in that network.

2. To see how our LC-based automatic detection scheme from Section 3 compares with existing, automatic Sybil detection methodologies.

5.1 Our System

We have implemented all of the algorithms described in the paper. The multi-threaded implementation consists of 5000 lines of C++ source code. We run our experiments on a Linux server machine with eight, 3.16 GHz cores sharing 33 GB of memory. In our experiments, all hyper-parameters are set as described in Section 4.2. The number of communities $n = 100$ in all experiments. As in our college football example, the prior $F(.)$ on the latent community positions corresponds to a two-dimensional normal distribution.

5.2 Utilizing the Simple LC Model

Rather than giving a proper "experiment", we begin with an illustrative example, where we show how the simple LC model of Section 2 (with the addition of the "seed" idea from Section 3) can be used to detect Sybil attacks on the Digg website, which is a popular social news website.

We obtain the Digg data from the SumUp [29] project. On Digg, people can submit or cast votes on articles. Based on these votes, articles are ranked. Users can "follow" and be "followed" by others, inducing a directed graph. Digg relies on the feedback (votes) of its users and who follows whom. This creates a strong motivation for potential Sybil attackers.

There are $594,426$ nodes and $5,066,998$ directed edges in the Digg graph. We also have the date/time at

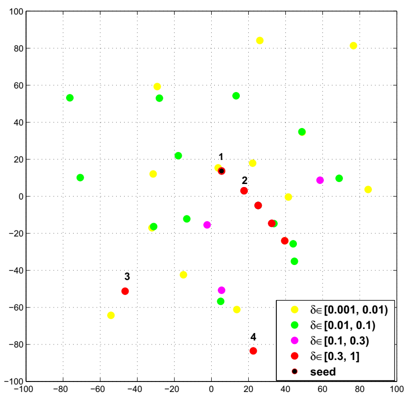

Figure 4: The positions of communities in Digg. δ denotes community density.

which edges are created; while our model as described cannot make use of this information, it serves as a way to help validate a discovered attack. Note that the LC model as described in this paper handles only undirected graphs. While it would be easy to extend the model to incorporate directionality, in order to keep things simple as create an edge between a pair of nodes if there exists an edge in either direction from one node to the other. After this preprocessing, there are $4,070,026$ undirected edges in the graph, with an average node degree of 13.7.

We use the node "Kevin Rose" as a single seed (Mr. Rose is the founder of Digg), and attempt to learn the LC model from the graph. We run our Gibbs sampler for a "burn in" of 1000 cycles. Estimates for parameter values are then obtained by taking the average over the next 100 iterations.

Figure 4 describes the latent positions of the communities we learn, with their relative density (δ). Looking at the plot, we immediately noticed that there are 7 communities with $\delta \geq 0.3$, but of those, only communities 3 and 4 in are distant from the center of the graph (their densities were 0.4 and 0.55, respectively). Furthermore, the communities are quite large (311 and 299 nodes, respectively).

It seems quite suspicious to us that these large communities with very high densities would be far from the center of the latent space. It is easy to explain why the learned process has placed the communities far from the center. Figure 5 depicts the so-called "relative edge densities" (abbr., "RED") for communities 1, 2, 3 and 4, as well as for a randomly selected community. If δ_i is the internal density of community i, and δ_{ij} is the probability that an arbitrary node in community i connects with an arbitrary node in community j, then

$\text{RED}_{ij} = \frac{\delta_{ij}}{\delta_i}$. The figure has five plots. In the plot associated with community i, if the line goes through point (x, y), it means that $\text{RED}_{ix} = y$. We note that the plots for communities 1, 2, and the random community all are strongly correlated with one another, suggesting that these communities all have strong connections with the same central communities. However, communities 3 and 4 both have a very strange set of connections—both have almost uniformly tiny RED values except for a single spike—which explains their positions as outliers in the latent space, and calls into question their legitimacy.

When searching for some final evidence of the malicious nature of these two communities, we decided to plot the histogram for the edge creation time of five communities of the previous figure (this is depicted in Figure 6). Here we see that in both communities 3 and 4, edges are created frequently within a short time interval. In particular, the vast majority of links involving community 4 are created within a time period of only 3 days, and then all activity stops! It is with high probability that this community is involved in a Sybil attack, despite that most activity in social networks is found to be temporally bursty.

We close the subsection by noting that the LC model is not the only way this attack could have been discovered. In particular, a temporal analysis could have uncovered this attack as well. However, it would have been easy for the attacker to more carefully spread the creation of the malicious nodes over time in an attempt to hide the attack. It would have been much more difficult to hide or alter the structure of the attack and the way in which its links to the rest of the graph were atypical.

5.3 Utilizing the Automatic Detector

We now compare our automatic LC-based sybil detector model of Section 3 with the state of the art Sybil defense schemes. There are various well-known specific Sybil-defense algorithms, i.e., SybilGuard, SybilLimit, SumUp, etc. We choose two representative algorithms: the first one is SybilInfer (SI), which has been shown to perform better than other methods on synthetic networks [30]. The second one is GateKeeper [28], an improved version of a previous popular algorithm (SumUp [29]), which accepts fewer Sybils per attack edge compared with SybilLimit [33] in real graph topologies.

We do not consider the set of community detection algorithms mentioned in [30] for the following reason: as pointed in [30], for those algorithms to work well, all the non-Sybil nodes need to form a single community that is distinguishable from the group of Sybil nodes. However, most of practical social networks have been shown to have multiple communities [18, 22, 31].

Experimental Setup. Our experiments will check the false positive and false negative rates of all three methods—LC, SI, and GK, over a variety of experimental settings. Our basic tactic is to take a small- to medium-sized, real network that is likely without an attack, add one or more attacks to the network, and to see how successful the methods are in discovering the attacks.

In our experiments, we systematically explore the effect of the following five variables:

- The real network to which an attack is added.
- The attack topology.
- The fraction of compromised nodes.
- The fraction of nodes in the graph that are attackers.
- The fraction of seeds.

We explain each of these variables now.

Real Network. We use three real data sets to which we add an attack. The first is the "Irvine Community" data set, created from a virtual community for students at University of California, Irvine, which consists of $1,899$ nodes and $13,820$ edges. When a new student joined the community, he is asked to create his profile with personal information. The second is "Wikipedia Vote", from the Stanford Large Network Dataset Collection [16]. It corresponds to vote relationships among Wikipedia users in elections for promoting individuals to be administrators [17]. The dataset consists of 7115 nodes and $100,762$ vote edges, of which half are voted by Wikipedia administrators, and half are from ordinary Wikipedia users. The third is the "Gnutella Peer-to-Peer Network" dataset, also from the Stanford Large Network Dataset Collection [16]. It corresponds to a sequence of snapshots of the Gnutella peer-to-peer network from August 2002. The dataset consists of 8717 nodes and 31525 edges. Both the "Wikipedia Vote" and "Gnutella" datasets have been shown to follow the fast mixing property [23]. We use the same strategy for Digg dataset to remove directionality from the graphs: we create an edge between a pair of nodes if there exists an edge in either direction from one node to another.

Attack Topology. We use three kinds of attack topologies. For each type of attack, a subset of the real nodes in the graph are chosen to be compromised nodes, and a set of new, attackers are added. For the "scale-free

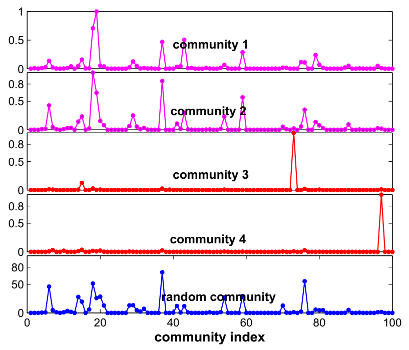

Figure 5: The relative edge density.

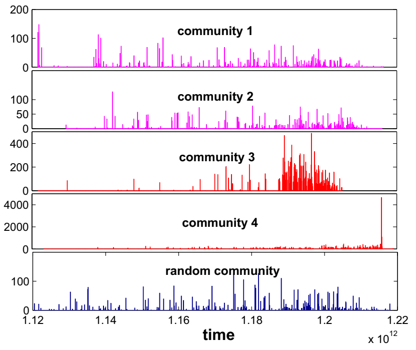

Figure 6: The creation time of edges.

attack" we delete all edges among compromised nodes, and then group them and the attackers together. We initially create a clique with m_0 nodes ($m_0 \leq 6$) and add the remaining nodes iteratively into the network by creating m new edges to the network. The probability for a node in the network to be selected is proportional to its degree. Finally, we add the deleted edges back in. For the "tree attack", a random tree is constructed among the set of attackers and compromised nodes. In the "football attack", the attacking nodes and the links between them are created by replicating the FBS football schedule data set from Section 2.2 repeatedly. To form connections between attackers and compromised nodes, the "scale-free" process is used.

Fraction of Compromised Nodes. This is the fraction of nodes in the data set that are victims of the attack. We consider three: 0.01, 0.1, and 0.25.

Fraction of Attackers. This is the fraction by which we increase the network size when we add an attack. We consider three: 0.01, 0.1, and 0.25.

Fraction of Seeds. This is the fraction of known, trusted nodes. We consider: 0.002, 0.005, and 0.01.

Given these variables, for each of the three Sybil detection methods we construct a suite of tests as follows. First we define default settings for the last four variables (that is, for the attack topology, the fraction of compromised nodes, the fraction of attackers, and the fraction of seeds). For the attack topology, the default is "scale-free attack". For the other three, the default settings are 0.1, 0.1, and 0.01, respectively. Then, for each data set, we consider each of the last four variables in order, and for a particular variable, we iterate through the three different settings, holding all other variables constant at the default values. This results in $(3$ data sets $) \times$ (four variables) $\times$ (three settings per variable) $=36$ tests for the LC model. The other two methods do not use seeds, and so they only have three variables to test, resulting in 27 tests for SI and GK. For each test, we report the observed false positive and false negative rate. The results are given in Figure 7.

All three methods give some sort of score to each node, where a high score means that the method is sure that the node is an attacker. For LC, this score is the faction of the last 100 Monte Carlo iterations that the node was in a malicious community. Thus, all must have some sort of threshold score that is used to flag a node as an attacker. For LC, we use the natural threshold of 50%. Both SI and GK also have similar thresholds to control the tradeoff between false positives and false negatives, and we also choose their values as 50%. In GK, we use f_{admit} [28], where a node is accepted by the admission controller if and only if the node is reachable from at least f_{admit} fraction of ticket sources. In SI, we use α, a threshold used to control whether a sampled cut is an attack cut [8]. In order to show how critical these settings are, in Figure 8 we show the false positive and false negative rates for all three methods as a function of the threshold chosen, under the default configurations for all four parameters, for the Irvine data set.

Discussion. There are a few key results. First and foremost, over all the experiments, the LC model always resulted in the lowest false positive rate. While it is not difficult to have a low false positive rate (after all, it is easy to simply return "no Sybils" every time), it is critical. In a real-world application environment, no user is going to accept false positives with any regularity. SI and GK both show 30% and higher false positive rates over most of the experiments. We worry that in practice, false positive rates higher than a few percent equates to a Sybil detection software being ignored.

Despite LC's low false positive rate, it also typically

Results Under Different Attack Topologies

	Irvine			Wikipedia			Gnutella		
Attack	LC	SI	GK	LC	SI	GK	LC	SI	GK
Tree	0.12/0.91	0.68/0.68	0.44/0.32	0.06/0.95	0.31/0.69	0.46/0.34	0.01/0.97	0.31/0.70	0.38/0.36
Scale-free	0.01/0.20	0.22/0.97	0.41/0.41	0/0.20	0.33/0.95	0.53/0.58	0.02/0.26	0.13/0.98	0.43/0.50
Football	0.01/0	0.24/0.99	0.49/0.43	0.23/0	0.32/0.94	0.51/0.58	0.06/0.33	0.17/0.99	0.55/0.69

Results Under Different Fractions of Compromised Nodes

	Irvine			Wikipedia			Gnutella		
Fraction	LC	SI	GK	LC	SI	GK	LC	SI	GK
0.01	0.00/0.06	0.21/1	0.42/0.07	0.00/0.06	0.31/0.99	0.46/0.11	0.0/0.02	0.13/1	0.37/0.08
0.1	0.02/0.20	0.22/0.97	0.41/0.41	0/0.20	0.33/0.95	0.53/0.58	0.02/0.26	0.13/0.99	0.42/0.50
0.25	0.03/0.96	0.20/0.92	0.56/0.62	0.10/0.99	0.33/0.87	0.61/0.75	0.02/0.95	0.14/0.96	0.47/0.61

Results Under Different Fractions of Sybils

	Irvine			Wikipedia			Gnutella		
Fraction	LC	SI	GK	LC	SI	GK	LC	SI	GK
0.01	0.1/0.99	0.2/0.81	0.45/0.53	0.14/0.96	0.30/0.76	0.55/0.81	0.03/0.28	0.12/0.91	0.45/0.75
0.1	0.02/0.20	0.22/0.97	0.41/0.41	0/0.20	0.33/0.95	0.53/0.58	0.02/0.26	0.13/0.99	0.43/0.50
0.25	0.02/0.12	0.24/1	0.49/0.49	0/0.16	0.35/0.98	0.52/0.53	0.01/0.08	0.15/0.99	0.39/0.37

Results for LC Under Different Fractions of Seeds

Fraction	Irvine	Wikipedia	Gnutella
0.002	0.04/0.07	0.29/0.07	0.02/0.15
0.005	0.14/0.04	0.04/0.13	0.01/0.44
0.01	0.02/0.20	0/0.20	0.02/0.26

Figure 7: False positive/false negative rates over the various experiments.

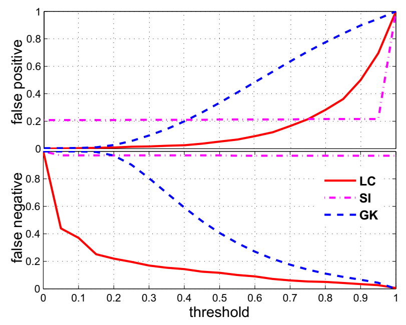

Figure 8: Sensitivity to threshold.

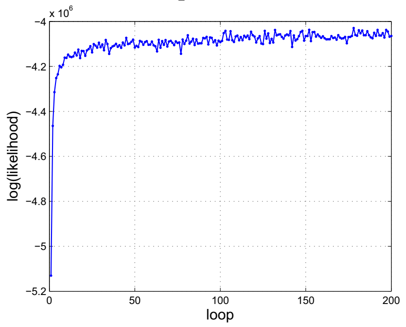

Figure 9: The convergence of our algorithm when applying to the Digg dataset.

had the lowest false negative rate. On both the scale-free and football attacks, and as long as 10% or less of the benign nodes have been compromised, it has excellent detection ability compared to the other two methods. We would go as far as to say that in those cases where at least one of the detection methods was in fact practical, LC had the lowest false negative rate. In those cases where LC did not have the lowest false negative rate (for example, on the tree attack topology) the other two methods have such a high false positive rate that they are simply not applicable. For example, consider the tree attack topology on the Wikipedia data—this attack is particularly interesting, because all methods were relatively powerless to detect it. LC has a 95% false negative rate, meaning it has almost no ability to detect the attack. But it also has a 6% false positive rate. SI, on the other hand, detects 31% of the attack nodes, while labeling 31% of the benign nodes as attackers. In other words, SI has no power to detect the attacks either, but while LC defaults to labeling very few nodes as attackers, SI labels 31% as attackers. This will result in missed attacks *and* an annoyed user who ends up investigating a very large number of benign nodes. However, it is worthy to point out that with more Sybils introduced in the tree attack, our approach regains its ability to detect Sybils while keeping a low false positive rate. When the ratio between Sybils and compromised nodes is 3, our approach can detect 60% of attackers. Due to the limited space, the figure is not displayed in the paper.

6 Performance Considerations

We consider two important performance considerations when handling "real-life" social networks: network size and handling changes in network topology efficiently. Performance is of key importance given the potential size of a social network. For example, the number of nodes in Facebook approaches one billion, and the number of active users who log on to Facebook in any given day can be 250 million [2].

The time complexity of our algorithm is $O(k^2 \times n \times l)$, where k is the number of communities, n is the number of nodes, and l is the number of iterations required by the Gibbs sampler. As should be clear from Figure 3, updating the parameter **m** takes the most time, since it iterates through the Cartesian product of n nodes with k communities. Note that the complexity for computing the posterior probability that a node belongs to a community is $O(k)$. Given the constant number for both k and l, the complexity of our algorithm is linear in the number of nodes.

Figure 9 shows the rate of convergence of our algorithm on Digg dataset. The purpose is to give the reader an idea of a reasonable value for l. In the figure, the X-axis shows the number of iterations, the Y-axis denotes the join probability between our model and data. Thus, the curve shows that the join probability converges with iterations. For a small number of l ($l < 100$), the curve begins to be stable. These results are typical: about 50 to 100 iterations of the Gibbs sampler are required in practice.

Finally, we note that an advantage of using a Gibbs sampler is that handling a dynamic network without needing to re-compute the model from scratch is easy. Node and edge deletions and additions can be batched, and then after enough changes have been observed, the Gibbs sampler is re-run using the previous model as a starting point. New users can be randomly or heuristically assigned to communities; after just an iteration or two of the Gibbs sampler they should be correctly integrated into the model.

7 Related Work

7.1 Sybil Attacks

Sybil attacks [9] are found widely. Various prevention schemes exist, such as computational games and CAPTCHAs. The first detection schemes were based on trust or reputation: Advogato [20], Appleseed [35] and SybilProof [7]. However, reputation-based systems are vulnerable to whitewashing attacks, where attackers initially behave honestly. IP or even IP cluster-based blacklists can be thwarted by techniques such as IP harvesting and Botnets.

There has been interest in leveraging network structure to thwart Sybil attacks, since it is generally assumed that it takes human efforts to establish connections among users in a reputation-based environment. Most topology-based Sybil defense algorithms [34, 33, 8, 29, 28, 19] are designed on this assumption, which seems to be questionable in practical online social networks [27, 14, 32]. SybilInfer [8] (tested in this paper) represents the first machine-learning-oriented scheme for solving this problem; however it assumes the existence of a bottleneck cut and assumes that the "fast mixing" property holds in the network, which contradicts with the measurement results from [23]. Viswanath et al. [30] have noticed that existing algorithms work well in synthetic networks but may show poor results in practical social networks. These findings are consistent with our own, though prior work excluded GateKeeper from the list of such poor-performing detection strategies (we tested GateKeeper in this paper). In addition, their work represents the first effort to introduce the mature community detection algorithms for Sybil detection. However, they also point out the limitations of current community detection algorithms for finding Sybils, which is analyzed in the next subsection.

Not all work in detecting Sybils has been based upon network topology. Most recently, researchers [32] use account-related statistics, like outgoing request accepted ratio, invitation frequency, clustering coefficient, etc, to detect Sybils in the Chinese Renren website [3].

7.2 Community Detection

Community Detection, unlike Sybil defense, has been a well- and long-studied topic in sociology, biology, mathematics, etc. Fortunato's recent survey [10] summaries hundreds of approaches for community detection with various ways to measure the quality of communities. In [18], Jure Leskovec et al. experimentally compare a range of community detection methods based on several common objective functions. In our opinion, there are two primary limitations when applying existing methods for community detection to the problem of Sybil detection. First, such approaches generally partition the graph into a number of communities, but there is not a natural way to distinguish benevolent communities from malicious ones. Viswanath and colleagues [30] intimate their opinion that for current approaches to work well, all the benevolent nodes need to form a single non-Sybil community, which is somewhat incompatible with the fact that social networks are generally found to have many local communities or groups.

It might be possible to post-process a standard community model to find Sybils. For example, hierarchical clustering [12] might be used to partition the graph into layers of communities, and finally lead to two regions, i.e., non-Sybils and Sybils. One worry is that Hierarchical clustering may return vertices with incorrectly labeled communities [24], since attackers can manipulate arbitrary number of communities with uncertain connections across themselves. Another idea is to first learn the communities, then position them into a Euclidean space, using a method such as multi-dimensional scaling [5]. But this has the obvious drawback of decoupling the community detection and embedding, which may influence one another. The other drawback with existing methods is that most have complexity greater than $O(n^2)$, which make them unsuitable for analyzing large scale online social networks.

Finally, we note that generative stochastic methods [13, 15, 25, 26] are used in analysis and modeling of social graphs. Using latent positions as part of a community detection framework is not new [21, 6, 31, 11]. However, such schemes tend to assign each node a latent position in a Euclidean space (in contrast, we assign each community a position in space). The drawback of a per-node assignment lies on its complexity of the learning algorithm, since the position for each node must be inferred which is generally very expensive. In contrast, the LC model needs only the aggregate statistics for each community when positioning the communities in space, which leads to much faster inference.

8 Concluding Remarks

We conclude the paper with a few high-level observations.

First, there is still work to do. A key weakness of our approach is that our inference algorithm assumes that all of the data is present at a centralized location. Existing protocols such as SybilGuard, SybilLimit and GateKeeper work in a distributed settings where Sybil defense is much more challenging. Extending our inference methods to a distributed environment is an important problem for future work. Also, we have not performed systematic, qualitative experiments on very large, real networks having tens- or hundreds of millions of nodes with millions of communities. Undoubtedly, our methods will need to be "tweaked" to work with real-life, web-scale graphs.

Second, we take care to point out that a topology-based approach such as the LC model is not appropriate for detecting every attack. For example, the LC method did not work well under a tree-topology attack. This was not surprising, given the fact that the model prefers high-density communities, and a tree has very low density. As an attack becomes less structured, the model's community-based approach makes less sense. Researchers have found that in some online social networks, such as the Chinese Renren network [32], almost 80% of their detected Sybils did not have edges among themselves at all. The LC model would not be appropriate for such a setting, and other methods must be used.

That said, we believe that when and if the Sybil problem is ever "solved", it will require a suite of methods that are all used together, and one important component of such a suite will be a topology-based approach. Our experiments indicate that the LC model deserves serious consideration as a network-topology-based method for automatic detection of Sybil attacks. Compared to the other, state-of-the-art methods we tested, it is the only one that does not consistently have an impractically high false positive rate. Plus, it seems to have the best ability to actually detect attacks.

9 Acknowledgments

This work has been supported by the US National Science Foundation under grant number 0964526. We thank the anonymous reviewers for their useful feedback and comments. We are grateful to other researchers who have shared their datasets and algorithms, either through web publication or personal communication: Alan Mislove, Bimal Viswanath, Prateek Mittal, Dinh Nguyen Tran, Jure Leskovec and Haifeng Yu. Finally, we thank Zografoula Vagena for her comments and help to the paper.

References

[1] http://www.collegefootballhelmetschedule.com/.
[2] http://www.facebook.com/press/info.php?statistics.
[3] http://www.renren.com/.
[4] C. M. Bishop. Pattern recognition and machine learning. In *Springer*, pages 542–546, 2006.
[5] I. Borg and P. Groenen. Modern multidimensional scaling: Theory and applications. In *Springer Series in Statistics*, pages 207–212, 2005.
[6] R. L. Breiger, S. A. Boorman, and P. Arabie. An algorithm for clustering relational data with application

to social network analysis and comparison with multidimensional scaling. In *J. Math. Psychol.*, page 328C383, 1975.

[7] A. CHENG and E. FRIEDMAN. Sybilproof reputation mechanisms. In *P2PECON '05: Proceedings of the 2005 ACM SIGCOMM workshop on Economics of peer-to-peer systems*, pages 128–132.

[8] G. Danezis and P. Mittal. Sybilinfer: Detecting sybil nodes using social networks. In *NDSS*, 2009.

[9] J. R. Douceur. The sybil attack. In *IPTPS*, pages 251–260, 2002.

[10] S. Fortunato. Community detection in graphs. *CoRR*, abs/0906.0612, 2009.

[11] M. S. Handcock, A. E. Raftery, and J. M. Tantrum. Model-based clustering for social networks. 2007.

[12] T. Hastie, R. Tibshirani, and J. H. Friedman. The elements of statistical learning. In *Springer, Berlin, Germany*, 2001.

[13] M. B. Hastings. Community detection as an inference problem. volume 74, page 035102. American Physical Society, Sep 2006.

[14] A. Jøsang, B. AlFayyadh, T. Grandison, M. A. Zomai, and J. McNamara. Security usability principles for vulnerability analysis and risk assessment. In *ACSAC*, pages 269–278, 2007.

[15] E. A. Leicht and M. E. J. Newman. Community structure in directed networks. volume 100, page 118703. American Physical Society, Mar 2008.

[16] J. Leskovec. Stanford large network dataset collection.

[17] J. Leskovec, D. P. Huttenlocher, and J. M. Kleinberg. Predicting positive and negative links in online social networks. In *WWW*, pages 641–650, 2010.

[18] J. Leskovec, K. J. Lang, and M. W. Mahoney. Empirical comparison of algorithms for network community detection. In *WWW*, pages 631–640, 2010.

[19] C. Lesniewski-Laas and M. F. Kaashoek. Whanau: A sybil-proof distributed hash table. In *NSDI*, pages 111–126, 2010.

[20] R. Levien and A. Aiken. Attack-resistant trust metrics for public key certification. 1998.

[21] D. D. McFarland and D. J. Brown. Social distance as metric: A systematic introduction to smallest space analysis. In *Bonds of Pluralism: the Form and Substance of Urban Social Networks (ed. E. O. Laumann)*, pages 213–253, 1973.

[22] A. Mislove, M. Marcon, P. K. Gummadi, P. Druschel, and B. Bhattacharjee. Measurement and analysis of online social networks. In *Internet Measurement Comference*, pages 29–42, 2007.

[23] A. Mohaisen, A. Yun, and Y. Kim. Measuring the mixing time of social graphs. In *Internet Measurement Conference*, pages 383–389, 2010.

[24] M. E. J. Newman. Detecting community structure in networks. In *Eur. Phys. J.B 38, 321-330*.

[25] J. J. Ramasco and M. Mungan. Inversion method for content-based networks. volume 77, page 036122. American Physical Society, Mar 2008.

[26] W. Ren, G. Yan, X. Liao, and L. Xiao. Simple probabilistic algorithm for detecting community structure. volume 79, page 036111. American Physical Society, Mar 2009.

[27] S. E. Schechter, R. Dhamija, A. Ozment, and I. Fischer. The emperor's new security indicators. In *IEEE Symposium on Security and Privacy*, pages 51–65, 2007.

[28] N. Tran, J. Li, L. Subramanian, and S. S. Chow. Optimal sybil-resilient node admission control. In *INFOCOM*, 2011.

[29] N. Tran, B. Min, J. Li, and L. Subramanian. Sybil-resilient online content voting. In *NSDI*, pages 15–28, 2009.

[30] B. Viswanath, A. Post, P. K. Gummadi, and A. Mislove. An analysis of social network-based sybil defenses. In *SIGCOMM*, pages 363–374, 2010.

[31] S. Wasserman and K. Faust. Social network analysis: Methods and applications. In *Cambridge: Cambridge University Press.*, 1994.

[32] Z. Yang, C. Wilson, X. Wang, T. Gao, B. Y. Zhao, and Y. Dai. Uncovering social network sybils in the wild. *CoRR*, abs/1106.5321, 2011.

[33] H. Yu, P. B. Gibbons, M. Kaminsky, and F. Xiao. Sybillimit: A near-optimal social network defense against sybil attacks. In *IEEE Symposium on Security and Privacy*, pages 3–17, 2008.

[34] H. Yu, M. Kaminsky, P. B. Gibbons, and A. Flaxman. Sybilguard: defending against sybil attacks via social networks. In *SIGCOMM*, 2006.

[35] C.-N. Ziegler and G. Lausen. Propagation models for trust and distrust in social networks. *Information Systems Frontiers*, 7(4-5):337–358, 2005.