DISSERTATION

ON DESIGNING LARGE, SECURE AND RESILIENT NETWORKED SYSTEMS

Submitted by

Dieudonné Mulamba Kadimbadimba

Department of Computer Science

In partial fulfillment of the requirements

For the Degree of Doctor of Philosophy

Colorado State University

Fort Collins, Colorado

Summer 2019

Doctoral Committee:

Advisor: Indrajit Ray

Indrakshi Ray
Ross McConnell
Leo Vijayasarathy

ABSTRACT

ON DESIGNING LARGE, SECURE AND RESILIENT NETWORKED SYSTEMS

Defending large networked systems against rapidly evolving cyber attacks is challenging. This is because of several factors. First, cyber defenders are always fighting an asymmetric warfare: While the attacker needs to find just a single security vulnerability that is unprotected to launch an attack, the defender needs to identify and protect against all possible avenues of attacks to the system. Various types of cost factors, such as, but not limited to, costs related to identifying and installing defenses, costs related to security management, costs related to manpower training and development, costs related to system availability, etc., make this asymmetric warfare even challenging. Second, newer and newer cyber threats are always emerging - the so called zero-day attacks. It is not possible for a cyber defender to defend against an attack for which defenses are yet unknown.

In this work, we investigate the problem of designing large and complex networks that are secure and resilient. There are two specific aspects of the problem that we look into. First is the problem of detecting anomalous activities in the network. While this problem has been variously investigated, we address the problem differently. We posit that anomalous activities are the result of mal-actors interacting with non mal-actors, and such anomalous activities are reflected in changes to the topological structure (in a mathematical sense) of the network. We formulate this problem as that of Sybil detection in networks. For our experimentation and hypothesis testing we instantiate the problem as that of Sybil detection in on-line social networks (OSNs). Sybil attacks involve one or more attackers creating and introducing several mal-actors (fake identities in on-line social networks), called Sybils, into a complex network. Depending on the nature of the network system, the goal of the mal-actors can be to unlawfully access data, to forge another user's identity and activity, or to influence and disrupt the normal behavior of the system.

The second aspect that we look into is that of building resiliency in a large network that consists of several machines that collectively provide a single service to the outside world. Such networks are particularly vulnerable to Sybil attacks. While our Sybil detection algorithms achieve very high levels of accuracy, they cannot guarantee that all Sybils will be detected. Thus, to protect against such "residual" Sybils (that is, those that remain potentially undetected and continue to attack the network services), we propose a novel Moving Target Defense (MTD) paradigm to build resilient networks. The core idea is that for large enterprise level networks, the survivability of the network's mission is more important than the security of one or more of the servers. We develop protocols to re-locate services from server to server in a random way such that before an attacker has an opportunity to target a specific server and disrupt it's services, the services will migrate to another non-malicious server. The continuity of the service of the large network is thus sustained.

We evaluate the effectiveness of our proposed protocols using theoretical analysis, simulations, and experimentation. For the Sybil detection problem we use both synthetic and real-world data sets. We evaluate the algorithms for accuracy of Sybil detection. For the moving target defense protocols we implement a proof-of-concept in the context of access control as a service, and run several large scale simulations. The proof-of-concept demonstrates the effectiveness of the MTD paradigm. We evaluate the computation and communication complexity of the protocols as we scale up to larger and larger networks.

ACKNOWLEDGEMENTS

This dissertation would not have been possible without the help of my advisor Dr Indrajit Ray. I would like to thank both Dr Indrajit Ray and Dr Indrakshi Ray for their mentor-ship and support during my studies. My gratitude also goes to my committee members Dr Ross McConnel and Dr Leo Vijayasarathy for their feedback. I'm thankful to all the faculty members and staffs of the Computer Science Department for all their supports and encouragements.

I would like to thank my fellow from the DBSec group for all the insightful discussions. I would like also to convey my gratitude to my former boss, Wayne Trzyna and all the members of the System and Network Administration team for all the memorable experiences.

I would like to thank all my collaborators here at Colorado State University. My warm thoughts go to all the wonderful people who have made my stay here in Fort Collins enjoyable and memorable.

DEDICATION

To my family

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

Defending large networked systems against rapidly evolving cyber attacks is challenging. This is because of several factors. First, cyber defenders are always fighting an asymmetric warfare: While the attacker needs to find just a single security vulnerability that is unprotected to launch an attack, the defender needs to identify and protect against all possible avenues of attacks to the system. Various types of cost factors, such as, but not limited to, costs related to identifying and installing defenses, costs related to security management, costs related to manpower training and development, costs related to system availability, etc., make this asymmetric warfare even challenging. Second, newer and newer cyber threats are always emerging - the so called zero-day attacks. It is not possible for a cyber defender to defend against an attack for which defenses are yet unknown.

Complex networks are networks representing complex systems and featuring patterns of connection between their elements that are neither purely regular nor purely random. A network is a powerful tool to represent a collection of objects and the relationship among them. An important problem over networked systems is anomalies detection. Anomalies detection is the analysis of the network in order to detect objects, or relationships that are unlike the rest.

One of the threats to complex networks is the Sybil attacks. Specific assumptions of identity are the basis on which many network applications and services are built upon. Not meeting these identity assumptions can leave these applications and services vulnerable to attacks. The consequence of these attacks can be a breach of the privacy offered by these applications and services to its users or rendering their results questionable or incorrect. An example of an attack on identity occurs when authenticating credentials are being stolen by a third party in an impersonation attack. Another attack on identity can involve an identity holder sharing purposely its credentials with multiple other entities. This kind of attacks have been given the denomination of Sybil Attacks, and the malicious nodes performing these attacks are called Sybil Nodes.

In this work, we investigate the problem of designing large and complex networks that are secure and resilient. There are two specific aspects of the problem that we look into. First is the problem of detecting anomalous activities in the network. While this problem has been variously investigated, we address the problem differently. We posit that anomalous activities are the result of mal-actors interacting with non mal-actors, and such anomalous activities are reflected in changes to the topological structure (in a mathematical sense) of the network. We formulate this problem as that of Sybil detection in networks.

A concrete example of systems that can be victims of Sybil attacks include but are not limited to online social networks, online voting systems, reputation systems, and peer-to-peer computing systems. In online voting systems, a single person can vote using many online identities. In peer-to-peer systems, a single entity with multiple identities can defeat the redundancy at the core of the peer-to-peer system by making its single entity to appear as many different entities. In online social network, a single entity can generate and control multiple identities that will be used to breach the privacy of other user's accounts. In addition, the single entity can also forge several of existing benign user's accounts. For our experimentation and hypothesis testing we instantiate the problem as that of Sybil detection in on-line social networks (OSNs).

An Online Social Network (OSN) is a web-based service that allows users to create a profile and view and interact with other users with whom they are connected [1]. OSNs create trust networks that can be used by members to more effectively communicate with each other, to process information, and to diffuse social influence. Among the more popular OSNs are Facebook, Twitter, LinkedIN, and Google+. Facebook, for instance, reported to have 1.415 billion active members as of March 2015, while Google+ reported 300 millions active users as of March 2015 [2]. Alexa [3] reported that Facebook and Twitter are respectively the second and the 9th most visited websites worldwide.

However, the success of OSNs have made them a lucrative target for attackers. Owing to their open nature, they are vulnerable to both classical and emerging threats to security and privacy [4]. These threats include Sybil attacks. Almost all OSNs assume that each participating entity controls

only a single entity – itself. In a Sybil attack, however, an adversary creates a large number of fake identities or forges a large number of existing identities that it can then use to target the trust underpining of an OSN and perform malicious activities benefiting itself [5]. These malicious activities include, but are not limited to, social spamming [6], malware distribution [7], and private data collection [8]. To give an idea of the scale of the problem, Facebook for instance, reported in 2014 that an estimated of up to 15 millions of its monthly active users represented fake accounts [9]. The malicious activities performed by these fake accounts not only affects the OSN users but also negatively impact the revenue of the victim OSN because they can lead to a loss of confidence by the users, investors, as well as advertisers.

Most existing Sybil detection mechanisms can be classified into two categories. The first one includes mechanisms that analyses graph-level structures while the second one includes mechanisms that analyses user-level attributes and activities. We call the first category structure-based approaches and the second, content-based approaches.

The structure-based approaches operate by modeling an OSN as a graph whose nodes and edges respectively represent user accounts and social relationships. The discrimination between Sybil accounts and benign accounts leverages graph-theoretic structural differences between the nodes representing these accounts. These approaches are based on the assumption that there is a strong trust relationship between users that makes it hard for Sybils to link with honest users. Therefore, there will be a small number of attack edges (which are links between Sybils and non-Sybils), and the graph would be partitioned into two distinct regions separating Sybil accounts from the benign ones [10]. However, several researches [11], [12], [13] have shown that structure-based approaches can be evaded by an attacker who succeeds in creating a large number of attack edges between the fake accounts and the benign ones. This happens specially in weak-trust OSNs.

In the content-based approaches, Sybil accounts are discriminated from legitimate ones by a classifier trained using machine-learning techniques. The classifier is built based on unique features extracted from recent user attribute and activities [14]. The problem with this approach is that it is difficult to obtain good and accurate data from OSNs. Users often keep their profiles

incomplete or frequently use misleading data in their profiles. Moreover, obtaining quality data about user attributes and activities may raise serious privacy issues. However, they can be evaded by a more sophisticated attacker. An attacker can evade a content-based approach by creating fake accounts whose features are similar to those of real accounts.

The second aspect that we look into is that of building resiliency in a large network that consists of several machines that collectively provide a single service to the outside world. Such networks are particularly vulnerable to Sybil attacks. While our Sybil detection algorithms achieve very high levels of accuracy, they cannot guarantee that all Sybils will be detected. Thus, to protect against such âĂIJresidualâĂİ Sybils (that is, those that remain potentially undetected and continue to attack the network services), we propose a novel Moving Target Defense (MTD) paradigm to build resilient networks.

Traditional security approaches have been criticized for presenting a static target for attackers. Critics argue that these security approaches put the defenders of computer networks and system in a disadvantaged position, by allowing attackers to reconnoiter a system and plan the attack at leisure. As a remedy, a new security paradigm for protecting computer networks and systems, called Moving Target Defense, has been proposed [15]. The core idea is that for large enterprise level networks, the survivability of the network's mission is more important than the security of one or more of the servers. We develop protocols to re-locate services from server to server in a random way such that before an attacker has an opportunity to target a specific server and disrupt it's services, the services will migrate to another non-malicious server. The continuity of the service of the large network is thus sustained. We implement a proof-of-concept in the context of access control as a service that needs to be provided in a mobile cloud system.

Cloud computing is a novel paradigm that allows on-demand network access to a shared pool of computing resources without requiring extensive management effort on behalf of the clients that require these service.This, in turn, allows efficient cyber foraging [16] by resource crunched mobile computing devices such as smartphones and cell phones, giving rise to the newer *mobile cloud* model. In this model, cloud computing and mobile devices and networks seamlessly interact with

each other to provide newer types of services that were previously not possible (such as location based services). Unfortunately, this model brings with it some unique challenges to access control that require re-visiting the traditional trusted computing base (TCB) approach [17].

In general, access control is implemented by the cooperation of four functional modules that are part of the trusted computing base:

1. *Policy Administration Point* (PAP): The PAP is a repository for the authorization policies that are expressed in terms of the actions that subjects (human users, devices, processes, organizations etc.) can take on various objects in the system. The authorization policies are essentially an instantiation of the access control model tailored towards the organization. It is the main component for the authorization portion of access control

2. *Policy Information Point* (PIP): The PIP is the module that gathers together all the information that are needed to evaluate an authorization policy. It is the main component responsible for achieving authentication.

3. *Policy Decision Point* (PDP): The PDP gets relevant information from the PIP and consults the PAP to arrive at a decision whether to grant or deny an access request.

4. *Policy Enforcement Point* (PEP): The PEP receives access requests from subjects in the external world, hands them to the PDP for evaluation, and after receiving the grant or deny response from the PDP, ensures the appropriate action is taken.

One of the requirements of a TCB is that it implements the concept of reference monitor [17]; that is, the TCB mediates all access to objects by subjects, it is tamperproof and cannot be bypassed, and it is small enough to be thoroughly tested and analyzed. In a mobile cloud environment, unfortunately, making the TCB small and tamperproof is very difficult to ensure. To understand why, let us consider the high level operational architecture of a mobile cloud system shown in figure 1.1.

The access control subsystem within the mobile cloud typically consists of three major components:

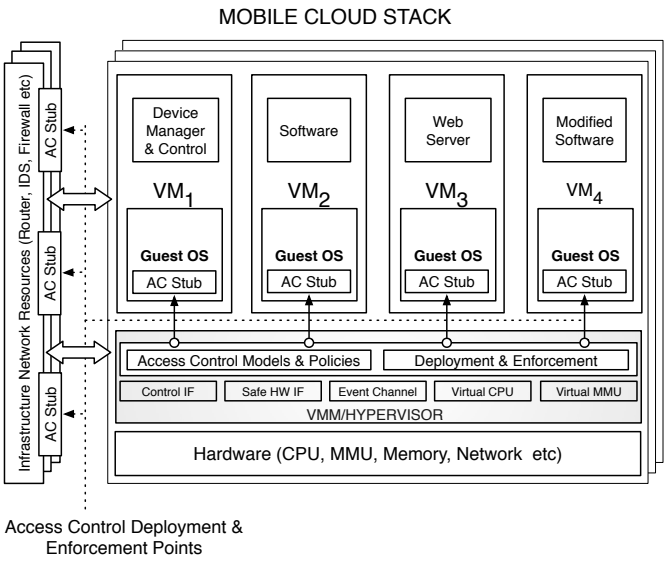


Figure 1.1: Operational architecture of mobile cloud system

1. access control model and policies at the hypervisor level,

2. deployment and enforcement at the hypervisor level, and

3. access control stub processes (represented by AC Stub in figure 1.1) that are distributed.

Components 1 & 2 reside in the hypervisor of the cloud stack while component 3 is linked to the access control subsystem of each VM and the mobile network infrastructure as well the mobile devices. Policies related to the cloud provider's infrastructure are effectuated in the hypervisor that can be distributed across several hardware platforms. Policies related to the tenants are present in the different virtual machines (VMs). When a VM is instantiated, an AC Stub process is also instantiated. It serves as a coordinator between the VMs access control subsystem and the hypervisor's. Different access control operations are effectuated by the AC stubs. At the mobile network level, AC stubs are implemented primarily within the smartphone operating system but can also be present at the network infrastructure. The deployment and enforcement of these policies are achieved by the deployment and enforcement module in the hypervisor is conjunction with the access control stubs that are present at various positions within the mobile cloud stack.

To remain within the confines of the TCB framework for access control not only all the three

components need to be made tamperproof but also all communications and coordinations among these components need to be ensured to be so. Thus, the trusted computing base needs to be enlarged which violates the principles of reference monitor. Moreover, the sheer size of a mobile cloud system, the degree of heterogeneity among the different devices and VMs, and the dynamicity of the whole system, compounds this problem many fold. We, therefore, posit that it is next to impossible to rely on a TCB to provide access control in a mobile cloud environment. The access control subsystem should try to satisfy as many properties of a TCB as possible but should also incorporate certain self-defending strategies to make it secure.

Modern computing environments are witnessing the proliferation of servers for a wider variety of critical services, in addition to the traditional email and file system applications. The computing power and versatility of the current day servers makes them indispensable assets for system administrators for providing essential services to the users. The range of user services offered by these servers has expanded towards security-critical services like access control systems, intrusion monitoring, anti-virus filtering and stateful firewalling, among others. However, these applications have ushered a new threat scenario in which the servers are the central targets for attackers instead of the end-users.

As an illustration, consider a server that is deployed for enforcing an attribute-based access control system and processes user access requests depending on the user attributes. Naturally, such a server is under constant threat of compromise by attackers who want to exploit the information on the server and disrupt the integrity of the service for their personal gains. For instance, in the access control setting, a possible manipulation is to *allow* illegitimate requests for resources by unauthorized users who are colluding with the attacker. Unlike the past trend of Denial-of-Service class attacks, modern day attacks are commercially motivated and are not necessarily focused on service disruption. Therefore, the security of servers providing security-critical services from commercially motivated attackers is an important and challenging problem. This problem can be stated as follows: Given a set of n servers: $S_1, S_2, \cdots, S_n$, some of which are compromised by

attackers, the objective is to design an algorithm to keep selecting one server to provide service for short periods. The algorithm constraint is that the period of server's service should be short enough to thwart manipulation attacks by a concerted attacker. Further, the selection needs to be a mutually agreed process. We cannot assume the presence of a central entity that will select the next server, as such an entity will be subjected to same attacks and will allow an attacker to manipulate the election.

1.1 Contributions

1.1.1 Sybil Detection using an Unsupervised Method

Given that extracting and selecting appropriate features from users attributes and activities for content-based Sybil detection is challenging, prone to inaccuracies, and often raises privacy issues, we propose SybilRadar, a Sybil detection mechanism that is solely based on graph-based structural properties of OSNs. SybilRadar is able to protect OSNs with weak trust relationships against Sybil attacks.

SybilRadar operates in three steps. The first step involves the computation of a similarity measure between every pair of nodes of the OSN social graph. The similarity between nodes is based on the notion of common friends. Honest nodes tend to have many common friends and can be a discriminator between honest nodes and Sybil nodes. The second step leverages the community structure of the social graph. A second similarity metric based on the community structure is computed and exploited to refine the result from the first step. This step produces a weighted graph whose edge weights are the similarity values of any given pair of nodes. In the third step, a node ranking is performed using a Modified Random Walk that uses the edge weights and provides the final list of Sybil nodes. The accuracy of SybiRadar is initially evaluated using synthetic data generated with the help of open-source graph-processing tools [18, 19]. The validity of our experimental results is further evaluated using real-world Facebook datasets. Finally, SybilRadar evaluation results are compared to the ones from SybilRank [20], which is currently the most well-known structure-based detection approach. Our analysis shows that, structural properties of an

Online Social Network with weak trust can be suitably utilized to build a robust Sybil detection mechanism that provides detection accuracy close to that obtained by user attribute and activity based mechanisms without raising privacy concerns.

1.1.2 Sybil Detection using a Supervised Method

To address the weakness of content-based classification methods, we adapt a classification mechanism that leverages the topological structure of the graph modeling the social network. An OSN user account (real or fake) is represented by a node in this graph and social relationships are represented by edges. The advantage of such an approach is that a malicious attacker cannot easily spoof the system since s/he does not have much control over the structure of the OSN graph. We compute a set of feature values for each node that constitute the feature vector for the node. The feature vector is then used to train classifiers using three machine learning techniques – KNN, Random Forests, and Adaboost. Our experimental evaluation using real world data shows that our method is very accurate in classifying Sybils, with the Random Forest and KNN approaches being able to predict Sybils with an Area Under the Curve (AUC) of 99%. To make sure that these results are not because of overfitting, we perform several tests that are well accepted by the community for this purpose. First we test the models on a held-out test set to see how well the models generalize to unseen data. Then, train our models after sampling our dataset to account for the imbalance of classes within the dataset. Finally, we plot the learning curve to detect an eventual case of overfitting.

We make the following major contributions in this work: First, we define a set of new graph centrality measures. These measures are:

- *Weighted degree-core centrality;*
- *Weighted degree-clustering centrality;*
- *Degree-intensity centrality;*
- *Degree-coherence centrality;*

- *Core-intensity centrality;*
- *Core-coherence centrality.*

It appears that these measures hold much promise in similar structure-based analysis of complex networks. Second, using feature selection techniques, we incorporate a set of 8 features – four from the newly proposed centrality measures and four other well known centrality measures that has been used in related works – to develop a novel Sybil detection method that achieves AUC levels up to 99%, much more than achieved by current techniques. Third, we perform several experiments using both synthetic as well as real-world data to evaluate our proposed approach as well ensure no over-fitting.

1.1.3 Protecting an Access Control Reference Monitor

In this work, we treat access control as a service that needs to be provided in a mobile cloud system. From a functional perspective, this service is achieved by the four functional – PAP, PIP, PDP and PEP – that are implemented by the AC stubs and the components at the hypervisor level. We assume that like any other service, the access control service can be attacked by a malicious intruder and hence needs to be protected. An attacker intent on damaging the access control service will launch reconnaissance efforts seeking exploitable vulnerabilities for this subsystem. We believe that allowing limited opportunity to the attacker to enumerate exploitable vulnerabilities can considerably enable protection of the access control subsystem. Towards this end we propose a Moving Target Defense (MTD) framework for protecting the access control subsystem in mobile clouds. In this framework, the four functional modules are effectuated by randomly materializing processes. As a result, the attacker does not know which processes can be targeted to compromise the system. Moreover, the window of opportunity for targeting processes is varied to further reduce opportunities for attack.

1.2 Organization

The rest of this dissertation is organized in the following manner. In Chapter 2 we introduce some necessary technical background relevant to the problem of Sybil detection as well as the Moving Target Defense. Chapter 3 introduces an extensive literature review on Sybil detection and Moving Target Defense. A presentation of our graph sampling algorithm is presented in Chapter 4. A protocol for Sybil detection in Online Social network using an unsupervised method is presented in Chapter 5. In Chapter 6 we present another Sybil detection mechanism using a supervised approach. Chapter 7 presents an architecture to protect an access control monitor through Moving Target Defense. In Chapter 8 we improve our proof of concept presented in Chapter 7, by proposing a byzantine fault-tolerant leader election protocol based on a consensus protocol which itself exploits a one-way secure hash commitment protocol. In Chapter 9 we conclude the dissertation and present some future directions.

Chapter 2

Background

The world in which we live comprises several complex systems. These complex systems include, but are not limited to :

- A society as a collection of individuals that are interacting with each other
- The organization of information and knowledge into a linked structure
- The interaction of genes as well as of neurons in the brain.

A network is a wiring diagram that represents the interaction between the components of a complex system. The study of complex systems involves, actually, the analysis of the network structures that represent them. As it turns out, graph theory is the study of network structures. So, in this chapter, we are going to present some basic graph theory concepts that are useful in the understanding of the rest of this work.

2.1 Graph

A graph is a structure that consists of a set of objects, called nodes, and a set of edges which are links connecting certain pairs of nodes. A graph G is usually noted as $G = (V, E)$ where V is the set of nodes, whereas E is the set of edges. Graphs are useful structures that help to model networks structures. The objects of the network structure are represented by the nodes, and the relationship between those objects are represented by the edges.

2.2 Centrality

A primary task of network analysis id to identify nodes that play a central role in the network. Centrality measures are metrics that are designed to quantify graph theoretic ideas about the prominence of a node within a network. There are several centrality measures, but in this work we describe degree centrality, closeness centrality, betweenness centrality, and Eigen Vector centrality.

2.2.1 Degree centrality

Given a graph $G = (V, E)$ and a node $u \in V$, the degree centrality of u measures the number of nodes that are directly connected to u. A node with a higher degree can interact and influence more nodes compared to a node with a lower degree.

2.2.2 Closeness centrality

Given a graph $G = (V, E)$ and a node $u \in V$, the closeness centrality of node u quantify the idea of how close node u is to other nodes of the graph G. A node with a higher closeness value can quickly interact with its peers, and only requires a small number of steps to reach them. The closeness centrality of node $u \in V$ is expressed by :

$$C_c(u) = \frac{1}{\sum_{v \in V} d(u, v)} \tag{2.1}$$

where $\sum_{v \in V} d(u, v)$ is the sum of the shortest distance from u to the other nodes of V.

2.2.3 Betweenness Centrality

Betweenness centrality quantify the idea that a node is prominent in a network if it occupies a position between several shortest paths connecting pairs of nodes in the graph. Such a node plays a role of broker for the interactions or transactions occurring between other pairs of nodes in the graph. Given a graph $G = (V, E)$, and a node $u \in V$, the betweenness of node u is expressed by :

$$C_b(u) = \sum_{j \neq k} \frac{\delta_{jk}(u)}{\delta_{jk}} \tag{2.2}$$

where $\delta_{jk}(u)$ is the number of shortest paths connecting a pair of nodes v, w and δ_{jk} is the total number of shortest paths connecting any pair of nodes v, w.

2.2.4 EigenVector Centrality

Degree centrality, even though it is the simplest centrality to compute, has the following drawback. Two nodes with the same degree are equally central in the graph because they have the

same number of connections. However, the quality of those connections is not taken into account. EigenVector centrality improves this situation by considering a node to be prominent when it is connected to several other prominent nodes. This centrality combines the ideas of the number of friends a node has with the idea of the quality of those friends. For a given graph $G = (V, E)$ with $A = (a_{u,v})$ being its adjacency matrix, the EigenVector centrality of a node $u \in V$ is expressed by :

$$C_e(u) = \frac{1}{\lambda} \sum_{t \in M(u)} C_e(t) \tag{2.3}$$

where $M(u)$ represents the set of nodes connected to u, and λ is a constant.

2.3 Similarity

Another aspect of network analysis involves the study of how an object relates to another one in the network. In this section, we present how the similarity between two nodes in a graph is computed. The computation of the similarity between connected nodes can be based either on the structure if the graph or on the attributes of these nodes. Several approaches for analyzing network networks use the concept of similarity of two nodes based on the graph structure as their building block. These approaches includes, but are not limited to link prediction, collaborative filtering, and community detection.

2.3.1 Structural similarity

Structural similarity measures are methods for computing the similarity between two nodes that are based on the topology if the graph. These similarity measures can be classified in two categories: local measures and global measures. Local similarity measures compute the similarity between two nodes based only on the information about their respective immediate neighbors. On the other hand, global similarity measures consider the global structure of the graph in the computation of the similarity between two nodes. Global similarity measures give better results

compared to local similarity measures, but this comes at the cost of the computation which is more expensive [21].

Local similarity measures

Local similarity measures compute the similarity between two nodes based only on the information about their respective immediate neighbors. In this section we present the Common neighbors metric, the Jaccard similarity metric, and the Adamic-adar similarity metric.

- Common neighbors metric

The common neighbors metric is the most common approach to compute the similarity between two nodes. This similarity measure works by computing how many neighbors the two nodes have in common. Let u and v be two nodes, an $N(u), N(v)$ be respectively the neighbors of the nodes u, and v [22]. Then the similarity of u and v is expressed by :

$$sim(u, v) = |N(u) \cap N(v)| \tag{2.4}$$

- Jaccard similarity metric

The common neighbors metric, even though is simple to compute, presents a drawback of favoring nodes with high degree over the ones with low degree. Jaccard similarity metric is proposed to correct this drawback. Let u, v be two nodes, and $N(u), N(v)$ be their respective set of neighbors, Jaccard metric works by dividing the intersection of $N(u)$ and $N(v)$ by the union of $N(u)$ and $N(v)$. It is expressed by :

$$sim(u, v) = \frac{|N(u) \cap N(v)|}{|N(u) \cup N(v)|} \tag{2.5}$$

- Adamic-adar similarity metric

The Adamic-adar metric is another solution proposed to overcome the problem posed by the Common neighbors metric. It is designed specifically to favor nodes with low degree. The adamic-adar metric of nodes u and v is expressed by :

$$sim(u,v) = \sum_{w \in N(u) \cap N(v)} \frac{1}{log|N(w)|} \tag{2.6}$$

Global similarity measures

Global similarity measures, unlike local similarity measures, are computed by considering the global topology of the graph. They allow the computation of the similarity of two nodes even when these nodes do not share any common neighbors. In this section we present the Katz similarity metric and the Simrank similarity metric.

- Kart Similarity metric

The Katz similarity metric is based on the concept of shortest path distance as used when defining the betweenness centrality. However, this metric considers the ensemble of all paths between two nodes when computing their similarity. Using this metric, the similarity of two nodes u and v is expressed by :

$$sim(u,v) = \sum_{l=1}^{\infty} \beta^l path^{<l>}(u,v) \tag{2.7}$$

where $path^{<l>}(u,v)$ represents the set of all paths of length l between u and v [22].

- Simrank similarity metric

The Simrank similarity metric is another global similarity measure, and is based on the concept that the similarity of two nodes depends of the similarity of their neighbors. For two given nodes u and v, first their similarity is set to 1. Then, their Simrank similarity is expressed by :

$$sim(u,v) = \lambda \frac{\sum_{a \in N(u)} \sum_{b \in N(v)} sim(u,v)}{|N(u)|.|N(v)|} \tag{2.8}$$

where $N(u)$, $N(v)$ are respectively the set of neighbors of u and v. $\lambda \in [0,1]$ is a parameter to be set depending of the problem [23].

2.4 Community detection

Community detection is an important topic in Social Network Analysis due to the clustering nature of these networks. The idea of using a clustering structure when designing a similarity metric was advanced by [24]. In an empirical study performed on synthetic and real-work networks, they have shown that link prediction measures based on structural similarity perform poorly for a network with a low clustering structure. This inspired [25] to first divide the network into communities, and use this clustering structure information in designing a similarity metric for the link prediction problem. Surprisingly, there is no consensus on the exact definition of a community but, in general, a good community is characterized by a subset of nodes that are well connected among themselves while being loosely connected to the rest of the graph. Most of algorithms for finding communities are built on a metric that measures the goodness of a cluster. They differ on the kind of techniques used to partition the network into clusters, and on how the goodness of a community structure is measured.

In order to measure the quality of a community structure, Newman et al [26] introduced a modularity function Q. Given a social graph $G = (V, E)$, the modularity function can be expressed as follows:

$$Q = \frac{1}{2m}\sum (A_{ij} - \frac{k_i k_j}{2m})\delta(C_i, C_j) \tag{2.9}$$

where k_i and k_j are respectively the degree of nodes i and j. A_{ij} represents an element of the adjacency matrix, and m is the size of E which is the set of edges of the given graph G. C_i and C_j are the respective communities to which i and j belong. The parameter δ is the Kronecker delta symbol whose value is 1 when both i and j belong to the same community, and is 0 when both nodes belong to different communities. Modularity is the most used function to measure the quality of a community structure. The goal of community detection is to divide a network into communities in a manner that maximizes the value of the modularity. However, there is a problem of resolution limit that arises when one tries to maximize the modularity. The problem refers to the situation where modularity maximization may fail to detect small but dense communities when they are in presence of very large communities. Moreover, modularity maximization may

also fail to detect communities in a network where the number of communities to be found is larger than the number of edges in the network [27], [28]. Several methods have been proposed to solve the problem resolution limit. Some of those methods modify the modularity function by adding tunable parameters [29], while others basically introduce different kind of modularity functions [30], [31].

Chapter 3

Related Work

3.1 Sybil Detection in Online Social Networks

Several studies have shown that OSNs are very vulnerable to Sybil attacks. Facebook [32], Twitter [33], [6], and Renren [11] have each experienced significant amount of spams whose origins were Sybil attacks. Several researchers have investigated approaches to defend against Sybil attacks on Online Social Network following studies that have been conducted to assess the severity of these attacks. Two bodies of works have been proposed in order to mitigate Sybils. The first body of works that we call content-based approaches leverages user behaviors and employs machine-learning techniques to learn and classify these behaviors. OSN nodes deviating significantly from these nodes are called Sybils. The second body of works that we call structure-based approaches leverages graph-theoretic proprieties of the social network. Nodes that exhibit significantly different properties than others are identified as Sybils.

Structure-based approaches model an OSN as a graph with user accounts and social relationships respectively represented by nodes and links. These approaches determine some graph-theoretic characteristics of nodes which are then used to discriminate Sybils from the real ones. Existing structure-approaches are based on two assumptions. The first is that the social graph will be partitioned into two distinct regions, one region with the Sybil nodes and the other one with benign nodes. The second assumption is that there will be only a small number of attack edges between the two regions, as a consequence of the strong trust relationship in the social graph. Several mechanisms use these approaches to detect Sybil communities, which are tight-knit communities that have a small quotient-cut from the honest region of the graph [34], [35], [36].

SybilRank [20] is one of the most well-known techniques. It uses graph-theoretic properties of the OSN social graph to compute the likelihood of users to be Sybils in order to perform the ranking. The detection starts with the administrator determining some known real users as initial

seed node. A short random walk is run with the known seeds. At the end of the random walk, all nodes are given trust values which are the landing probabilities for the random walk. SybilRank then ranks all the nodes based on their trust value. Nodes having higher trust value will be at the top, while the nodes with lower trust values will be lowly ranked. SybilRank performs almost linearly in the size of the social graph.

However, SybilRank is based on certain assumptions that several researches [37], [38] have proven not to be true in real life. In addition to these researches, Yang et al. show that Sybils on Renren blend into the social graph rather than forming tight communities [11]. Mohaisen et al. show that many social graphs are not fast-mixing, which is a necessary precondition for the structure-based Sybil detector of SybilRank to be effective [37]. SybilRadar, on the other hand, does not make any of these assumptions.

Integro [39] is an approach that extends SybilRank. It is developed without the two assumptions on which SybilRank is based on. Integro is a hybrid approach. It mixes content-based approach with a structure-based approach in order to detect Sybils. Integro first determines unique features for users which are used to build a feature-vector. The feature-vectors are used to train a classifier that predicts potential victims of Sybil attacks. After finding the potential victims, the edges in the social graph are given weights based on whether they are adjacent to the potential victims or not. The ranking is then performed by a modified random walk. Integro achieved a 95% precision in detecting Sybils. Our approach produces similar detection accuracy without using any content-based techniques.

SybilFrame [40] relaxes the assumptions that the social network can be partitioned into two distinct regions – Sybil and non-Sybil – and that there exists only a small number of attack edges between the two regions. SybilFrame is also a hybrid approach that leverages the attributes of an individual node along with a measure of correlation between connected nodes in order to classify nodes among benign and Sybils. SybilFrame operates in two steps. In the first step the initial network data are fed into the framework from which node unique features are extracted in order to compute node prior information. In Step 2, the node prior information are provided to the poste-

rior inference layer in order to compute the correlation between nodes. This nodes correlation is computed using Markov Random Field, and along with the Loopy Belief Propagation method, it provides the posterior information of nodes which is used to perform the ranking of nodes.

Content-based approaches aim to find Sybil accounts by using a classifier trained using machine-learning techniques. The most recent user activities are analyzed to extract some unique features that will serve as inputs on which a classifier is built. Machine-learning techniques such as clustering, support vector machines, and Bayesian networks are used to build the classifier. Some of these approaches are used for spam detection such as blacklisting, whitelisting, and URL filtering [6], [41], [42]. While many of these approaches have very high detection rates, the problem with these approaches is that they are only as good as the data that are used to train the classifiers. We believe that identifying proper features from user attributes and activities is challenging because these attributes often contain incomplete, inaccurate and sometimes purposefully misleading information. Additionally, a sophisticated attacker can create fake accounts presenting features similar to the ones one of real accounts, thus evading detection. We also believe creating such user profiles can lead to privacy breaches and are not supportive of such techniques. Consequently, We do not consider content-based approaches in our work any further.

The problem of detecting Sybils using machine learning classifiers is similar to the problem of detecting spammers in an OSN, a problem which many researchers have worked on to address. Some of the existing works leverage user profiles to build a classifier [43] while others extract user behavior information to built their classifier [44].

In [45], the authors built a classifier to detect spammers based on the users' attributes only. These attributes, collected from the users' profiles, constituted the content-based features and each account was labeled after a manual inspection. In [43], the authors first used content-based features derived from users' attributes to build an SVM classifier. Subsequently, they augmented content-based features with behavior-based features, which improved their accuracy rate from 84.6% to 87.6%. Another work using content-based features is [46]. The authors in this work aimed to

detect fake Twitter followers as part of the Fake Project. The dataset used was a collection of two sets. The first set is a set of Twitter users crawled from the Twitter network, while the second set is a set of fake Twitter accounts bought on the underground market. They designed a set of rules based on the content of user profiles. These rules were used to train a classifier that succeeded to accurately classify more than 95% of fake accounts.

Some researchers have shown that various means can be utilized by sophisticated attackers to evade being detected by classifiers built using only content-based features. For instance, [47] analyzed several evasion scenarios. These scenarios involve the attackers having full or partial knowledge of the features or the classifier training dataset. Based on these information, they have implemented attacks that resulted in the decrease of the classification accuracy.

The real possibility for an attacker to evade classifiers built using content-based features has lead researchers to augment users' attributes with either behavior-based features or structure-based features. One work that combines user and content-based features to detect spam users is [48]. Using these features, the authors built a number of classifiers that include SVM, Decorate, Simple logistic and Decision Trees. Their research showed that Decorate produced the highest accuracy rate of 88.98%. One instance of a work combining graph-based and content-based features to detect spam users is [41]. Using a dataset of 49 millions users collected from Twitter, they built several graph-based and content-based features which were used to build a series of classifiers. Among these classifiers, Bayesian classifier produced the highest result which was 91.7%. This result was even improved with the use of a combination of neural networks and SVM classifiers.

In [49] authors tackle the problem of detecting Sybil using a variety of features collected from user interactions, user profile, and the structure of the social graph. They use these features to build several classifiers, the Decision Tree (C4.5), Decision Tree (Random Forest), Support Vector Machine (SVM), and Multilayer Neural Network. The SVM classifier produced the highest precision, which is 97.1%. Building on this result, authors designed a browser plug-in capable of signaling the user if an account is malicious or not.

Several methods using the structure of social graph and a nodes ranking algorithm have been also proposed. In [50] authors propose SybilRadar, a robust Sybil detection framework based on graph-based structural properties of an OSN that does not rely on the traditional non-realistic assumptions that similar structure-based frameworks make. In this work, authors compute similarity values between nodes, and use those values as weights in a modified random walk. The modified random walk results in nodes ranking in which Sybils lie at the bottom.

While many of these approaches have very high detection rates, the problem with these approaches is that they are only as good as the data that are used to train the classifiers. We believe that identifying proper features from user attributes and activities is challenging. In addition a sophisticated attacker can create fake accounts presenting features similar to the ones one of real accounts, thus evading detection. Even though we have adopted a classification method to detect Sybils, our work differs from the works surveyed here by the fact that we use for the classification only information collected from the structure of the social graph without any input from the user profile information or the user activities information.

3.2 Protection of Networked Systems with MTD

In this section, we are presenting a survey of previous works relevant to the problem of protecting networked systems.

3.2.1 Network Resiliency

Network resilience is defined as the ability of the network to function in face of failures due to natural disaster or malicious attacks, which affect the proper operation of some of its components [51]. Techniques to increase a network resilience include segmentation [52], dynamic composition [53], diversity [54], deception [55], and dynamic reconstruction [56]. Segmentation [52] aims to limit the attack surface of a potential attack by logically or physically separating the network critical components. Dynamic composition [53] is the ability to dynamically provide new capabilities to the network. Diversity [54] is the action of using heterogeneous logical of physical

components in a network. The goal is to limit the attacks exploiting common vulnerabilities. Deception [55] is the action of misleading or confusing attackers in order to hide the critical assets of a network. Dynamic reconstitution [56] is the ability to reconfigure a network in order to render it resilient to ongoing and future attacks or faults while maintaining continuity of operations. One approach to implement a dynamic reconstitution of a network is through Moving Target Defense. In this work, we are interested in the problem of increasing the resilience of a distributed network that provides security-critical service such as access control. We take the Moving Target Defense approach in order to improve the resilience of a distributed network.

3.2.2 Access Control

One of the most important aspect of security is ensuring that users access only resources to which they are authorized. Research on designing and deploying access control in computers and networks can be traced back several decades [57]. Early standards of access control included discretionary and mandatory access control [58] [59] [60] [61]. However, Role based Access Control (RBAC) represented a major leap forward in term of flexibility. RBAC is built on the principle that users do not have discretionary access to enterprise objects. In a RBAC model, roles are created and users belong administratively to these roles, while permissions are administratively assigned to the roles. This arrangement provides more flexibility and simplicity to the management of authorization [61]. RBAC has been traditionally implemented for centralized systems. In recent years, several works have been done to provide the capabilities of this access control model to distributed systems and the cloud. For instance, in [62] authors present an access control tailored for distributed control systems. [63] explains how one can provide access control to anonymous users while verifying their authorization in a decentralized manner.

In several works, including recently in [64], researchers have worried that a malicious program may tamper with the operation of an access control system. The notion of a trusted computing base implementing the reference monitor concept was proposed by Anderson [17], in order to address this problem. Security kernels such as Scomp [65] and GEMSOS [66], included a reference

validation mechanism to satisfy the reference monitor concept of the TCB. Other operating systems such as Trusted Solaris [67], the Linux Security Modules (LSM) framework [68], TrustedBSD [69], Mac OS X, the Xen hypervisor provide some degree of support for reference validation so as to enable some shade of reference monitor. However, the major problem with these systems is that the tamperproof proof property that needs to be ensured for provably implementing a reference monitor, is hard to achieve. Tamperproofing requires the TCB to have a very small footprint. It can be shown that a general algorithm to prove that an arbitrary program behaves correctly reduces to solving the Halting problem. While current algorithms can prove correctness properties of specific programs, the variety of reference validation code and the complexity of correctness properties preclude verification for all but the smallest, most specialized systems. Unfortunately, for most of these systems, the TCB is too large to determine whether tampering is prevented. Moreover, for practicality and functionality, many systems allow user-level processes to modify the kernel. However, none of these user-level processes are immune to tampering thus becoming one of the weakest links. In this work, we are interested in the protection of the access control subsystem where ensuring the tamperproof property of a reference monitor cannot be ensured.

3.2.3 Moving Target Defense

Moving target defense (MTD) [70] is the concept of introducing controlled change across multiple components of the network in order to reduce the window of opportunity of attackers, and increase the cost of their attack efforts. The key concepts of MTD and their proprieties are presented in [71]. Different low level techniques of MTD and their effectiveness are presented in [72]. Those low-level techniques include Address Space Randomization, Instruction Set Randomization, and Data Randomization. In [73], two measures are designed that allow a defender to quantify its gain in security while deploying a MTD system. At the network level, [74] [75] and [76] present some network-based MTD approaches while [77] introduces a MTD approach for the cloud system. In order to chose a particular MTD technique, one needs to know its effectiveness. For that purpose, [78] proposes a comparison of different MTD techniques based on their effectiveness. In this

work we are interested in designing a MTD approach on a network whose servers are subjected to Byzantine failures. Among the work proposing the MTD approach in face of the Byzantine failures [79] proposes a moving target defense approach to switch among Byzantine fault tolerant protocol according to the existing system and network vulnerability.

3.2.4 Byzantine Fault Tolerance

A computer system can be affected by a type of failures that can cause it to behave in an arbitrary way. After being affected, the computer system can be lead either to process requests incorrectly, to corrupt their local state, and/or to produce incorrect or inconsistent outputs. This type of failures is named Byzantine Failures [80]. The problem for coping with this type of failure is known as the Byzantine Generals Problem [81]. The goal of Byzantine fault tolerance is to allow computer systems to be immune against Byzantine failures. It is a sub-field of fault tolerance researches.

Several works have been proposed to reach a consensus in the face of Byzantine failures. Building on Paxos, [82] has proposed an improvement that allows Paxos to support Byzantine fault tolerance with a modest latency. Castro and Liskov's proposed the Practical Byzantine Fault-tolerance protocol [83] that uses only four messages. [84] looked at improving the number of communication in the Byzantine Paxos protocols. In [85] authors proposed Tangaroa by improving Raft protocol [86]. They reach their goal by combining the ideas from the original Raft algorithm and from Practical Byzantine Fault-tolerance protocol [82]. For missions-critical applications, in [87] is described a practical asynchronous Byzantine fault tolerant protocol, which guarantees liveliness without making any network timing assumptions. In this work, we have designed a Byzantine fault tolerant consensus protocol for security-critical applications, which is new contribution in this problem space.

3.2.5 Consensus algorithms

One approach for building fault-tolerant applications is the Lamport's approach. The core of this approach involves two primitives : consensus and atomic broadcast [88]. Leader election protocols are generally used to solve the consensus problem.

Bully algorithm [89] and Ring algorithm [90] are among the most used algorithms for solving the consensus problem. Another hugely popular algorithm is the Paxos algorithm proposed by Lamport [91]. Another popular algorithm, considered even superior to Paxos due to its simplicity is Raft consensus algorithm [92]. Raft provides the capabilities for Leader election and log replication. [93] introduces Turtle Consensus (MPTC), an asynchronous consensus protocol for Byzantine-tolerant distributed system that uses MTD strategies to tolerate certain attacks.

ZooKeeper [94], an open-source replicated service for coordinating web applications and Chubby [95] are some practical systems exploiting these algorithms. Recently, GIRAFFE [96] has been proposed to provide a coordination service in scalable distributed system. Another practical system is the Apache Kafka [97] which allows the building of a replicated logging system. However, these algorithms and protocols do not take in account byzantine failures. In addition, their approach for electing a new leader dot not allow to prevent a malicious host from being elected as a leader

3.2.6 Leader Election

In a distributed system, leader election is a fundamental problem that requires that a unique leader be elected among a set of given nodes. The goal of a leader election algorithm is to elect a good processor as a leader in a setting where there are n processors of which a certain number m <n are bad while ensuring that no bad processor get elected as a leader [98].

A distributed computing system often requires that active nodes continue performing their task after a failure has occurred. This reorganization or reconfiguration necessitate that a coordinator be elected in the first place [99]. This is the reason of the wide interest the leader election problem [100] has received. Several works have been done on Leader Election [99] [101] [102] [103] [104].

3.2.7 Service Location Protocol

A zero configuration networking approach is a self-management networking approach that allows network devices to be automatically configured, to discover automatically services, and to access service automatically without the involvement of a network specialist [Intelligent Self-Management Home Multimedia Service System]. Three major Self-management technologies have been proposed. The Internet Engineering Task Force (IETF) promoted SLP [105] [106] as an intranet standard for automatic network resource discovery. Intel and Microsoft on their part proposed the Universal Plug and Play (UpnP) [107] as a standard for automatic communication between network devices using XML messages. Apple Inc proposed a protocol called Bonjour [108] as its Zero configuration networking standard. Recently, z2z [109] has been proposed for the discovery of network services beyond the local network.

Chapter 4

Graph Sampling

It is often necessary to sample smaller subgraphs from the bigger ones in order to facilitate the development and the testing of new algorithms for network analysis. This is necessary regardless of the availability of scalable analytical methods provided by the data mining community that are capable of handling very big graphs. However, in order to accurately infer the performance of new algorithms on big graphs from their sampled subgraphs, the sampled subgraphs need to be good representatives of the big graphs. A good sampled subgraph needs to preserve the structure of the big graph. Therefore, it is important to design a graph sampling method that produces a good representative subgraph of the larger graph. In this section we want to produce a subgraph representative of our twitter dataset.

The problem of graph sampling can be defined as follows: Given a graph $G = (V, E)$ where V is the set of nodes and E is the set of edges in the graph, find the sampling fraction $\wp$ that produces the sampled graph $G_s = (V_s, E_s)$ such that G_s preserves the structure of G.

4.1 The Original Sampling Method

We first describe the graph sampling method proposed in [110]. Given an original graph $G = (V, E)$, the sampling method performs the sampling in two steps: in the first step a simple edge sampling is performed on G. This means a set of edges are sampled from G and are added to E_s. The number of edges to be sampled depends on a threshold fixed by the experimenter. All nodes adjacent to the sampled edges are added to V_s. In the second step, a subgraph $G_s = (V_s, E_s)$ is induced by all the nodes in V_s. Here, extra edges between nodes in V_s that are present in E are added to E_s in addition to edges sampled in the first step. The result is a sampled subgraph $G_s = (V_s, E_s)$ that preserves the structure of $G = (V, E)$. The process is shown in algorithm 1.

Algorithm 1 Sample fraction ϕ, edge set E

Graph sampling with edge fraction ϕ

Input: Assume edges in E are stored in an array
Initialization: $V_s = \emptyset$, $E_s = \emptyset$
// Edge-based node sampling step
while $|V_s| < \phi \times |V|$ **do**
  $r = random(1, |E|)$ //uniformly random
  $(u, v) = e_r$
  $V_s = V_s \cup \{u, v\}$
end while
// Graph induction step
for k=1:$|E|$ **do**
  $(u, v) = e_k$
  if $u \in V_s \; AND \; v \in V_s$ **then**
    $E_s = E_s \cup \{e_k\}$
  end if
end for
return $G_s = (V_s, E_s)$

4.2 Our Proposed Sampling Method

Although the proposed graph sampling method is simple and produces a representative subgraph of the original graph, it is not suitable for our situation. In our case we want to predict malicious nodes. So, in addition to preserving the structure of the original graph we want the sampling method to preserve the proportion of honest nodes, sybil nodes, and attack edges. These requirements are not met by the Edge-based node sampling with graph induction. For this reason, we adapt this sampling method to meet our requirements.

The graph sampling method we propose still operates in two steps. In the first step, an edge sampling is performed. However, in our graph we have three types of edges: honest edges linking honest nodes, sybil edges linking sybil nodes, and attack edges linking sybil nodes to honest nodes. Therefore, from the original graph $G = (V, E)$, we split E into E_h (honest edges), E_m (sybil edges) and E_a (attack edges). Then edges are sampled from each of those subsets E_h, E_m, and E_a. This ensures that the proportion of those different type of edges is maintained. The sampled edges are added to E_s, and the nodes adjacent to these edges are added to V_s. In the second step we still induce a subgraph from all the nodes present in V_s. The result, provided by algorithm 2, is a

subgraph that preserves the proportions of different nodes and edges in addition to preserving the structure of the original graph.

Algorithm 2 : Sample fraction ϕ, edge set E

Input: Assume edges in E are stored in an array
Initialization: $V_s = \emptyset$, $E_s = \emptyset$
// Edge-based node sampling step
while $|V_s| < \phi \times |V|$ **do**
  // Sampling attack edges
  $r_a = random(1, |E_a|)$ //uniformly random
  $(u, v) = e_r$
  $V_s = V_s \cup \{u, v\}$
  // Sampling sybil edges
  $r_m = random(1, |E_m|)$ //uniformly random
  $(r, s) = e_m$
  $V_s = V_s \cup \{r, s\}$
  // Sampling honest edges
  $r_h = random(1, |E_h|)$ //uniformly random
  $(k, t) = e_h$
  $V_s = V_s \cup \{k, t\}$
end while
// Graph induction step
for k=1:$|E|$ **do**
  $(u, v) = e_k$
  if $u \in V_s \; AND \; v \in V_s$ **then**
    $E_s = E_s \cup \{e_k\}$
  end if
end for
return $G_s = (V_s, E_s)$

Figure 4.1 reveals that the node's degree of both the graph and the sampled subgraph have a similar cumulative distribution function (CDF). This indicates that our proposed graph sampling algorithm preserves the distribution of node's degrees from the original graph.

One goal of the proposed algorithm is to maintain the ratios of nodes in the original graph. Those nodes are the honest nodes and the sybil nodes. Figures 4.1 and 4.2 show that the proportion of honest nodes and sybil nodes is in accordance with their respective proportions in the original graph.

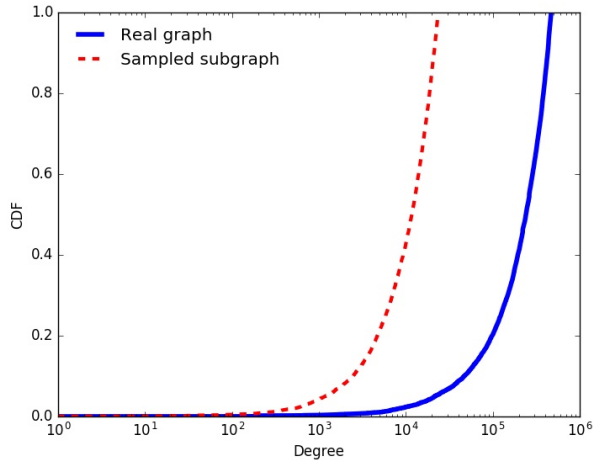


Figure 4.1: Degree distribution of the graph and the sampled subgraph

Table 4.1: Statistic of nodes in the graph and subgraph

Nodes	Graph	Subgraph
Honest nodes	372,251	30,840
Sybil nodes	97,253	16,111
Total nodes	469,504	46,951

Another goal of the proposed graph sampling algorithm is to sample the graph while maintaining the proportion of all types of edges in the graph. We have three types of edge: Honest edges which are the edges linking honest nodes, sybil edges which are the edges linking sybil nodes, and attack edges which are the edges connecting sybil nodes to honest nodes. Figures 4.3, and 4.4 show that the different edges has been sampled proportionally to their representation in the original graph.

Table 4.2: Ratio of nodes in the graph and subgraph

Nodes	Graph	Subgraph
Ratio honest nodes	0.793	0.657
Ratio sybil nodes	0.207	0.343

Table 4.3: Statistic of edges in the graph and subgraph

Edges	Graph	Subgraph
Honest edges	906,102	334,205
Sybil edges	1,147,939	903,190
Attack edges	99,385	91,918
Total edges	2,153,426	1,329,313

Table 4.4: Ratio of edges in the graph and subgraph

Edges	Graph	Subgraph
Ratio honest nodes	0.420	0.251
Ratio sybil nodes	0.533	0.679
Ratio attack edges	0.046	0.069

Chapter 5

SybilRadar: A Graph-Structure Based Framework for Sybil Detection in Online Social Networks

Online Social Networks (OSN) are increasingly becoming victims of Sybil attacks. These attacks involve creation of multiple colluding fake accounts (called Sybils) with the goal of compromising the trust underpinnings of the OSN, in turn, leading to security and the privacy violations. Existing mechanisms to detect Sybils are based either on analyzing user attributes and activities, which are often incomplete or inaccurate or raise privacy concerns, or on analyzing the topological structures of the OSN. Two major assumptions that the latter category of works make, namely, that the OSN can be partitioned into a Sybil and a non-Sybil region and that the so-called "attack edges" between Sybil nodes and non-Sybil nodes are only a handful, often do not hold in real life scenarios. Consequently, when attackers engineer Sybils to behave like real user accounts, these mechanisms perform poorly. In this chapter, we propose SybilRadar, a robust Sybil detection framework based on graph-based structural properties of an OSN that does not rely on the traditional non-realistic assumptions that similar structure-based frameworks make. We run SybilRadar on both synthetic as well as real-world OSN data. Our results demonstrate that SybilRadar has very high detection rate even when the network is not fast mixing and the so-called "attack edges" between Sybils and non-Sybils are in the tens of thousands.

The rest of this chapter is organized as follows. In Section 5.1 we present the system model for SybilRadar. We discuss why assumptions in existing structure-based detection mechanisms are invalid under real world settings. We end the section with a discussion on our attack model. The main design of SybilRadar is presented in Section 5.2. We discuss the major intuitions in our design and the different graph metrics that we used. Section 5.3 presents the experimental setup and evaluation of SybilRadar including comparison with SybilRank, which is the closest in design to SybilRadar. We conclude in Section 5.4 with a discussion of our results.

5.1 Preliminaries

We begin by presenting the system model for our work. We then introduce the notion of strong and weak trust relationships in OSNs. We explain why SybilRank does not perform well in a real-world OSN with weak trust. We end this section with a discussion of our attack model.

System Model: Trust relationship between two OSN users allows one to assess the information based upon which further information sharing can be performed or a service can be expected [111], and is the underpinning on which OSNs are built. Consider the social network topology as defined by a graph G = (V,E) comprising a set of vertices V, denoting users on the social network and E a set of edges, representing trust relationships (or friendship) between users. We assume trust relationships are mutual (bi-directional) and represent it with undirected edges between the users in the graph G. Two kind of nodes are considered here – an *honest* node and a Sybil node. A honest node that has accepted, or is susceptible to accepting a friend request from a Sybil node is considered to be a *victim* node. The subgraph of G containing all the honest nodes is considered to be the honest region of the OSN, while the Sybil region is the subgraph of G containing all sybil nodes.

We consider three kind of edges. *Attack* edges are those connecting victim nodes in an honest region and Sybil nodes. *Sybil* edges connect Sybil nodes to each other. Finally we have *honest* edges that connect honest nodes with each other. The system model is illustrated in figure 5.1.

OSNs with weak trust: In early studies [35], [112], OSNs were assumed to have strong trust relationships. OSNs with strong trust are those that possess the property of *fast-mixing*. For Sybil detection purposes, this boils down to a social network with a *small cut*, which is a set of edges whose removal will disconnect the graph into two distinct regions – the honest region and the Sybil region [113]. In other words, in a social network with strong trust we can distinguish the two distinct regions and there is a very limited number of attack edges between the regions (in the tens). OSN with weak trust, on the other hand, is a network that does not display the fast-mixing property. Indeed, it was demonstrated [37] that not many social networks are fast-mixing. In this work, we assume an OSN with weak trust, which is in contrast to SybilRank.

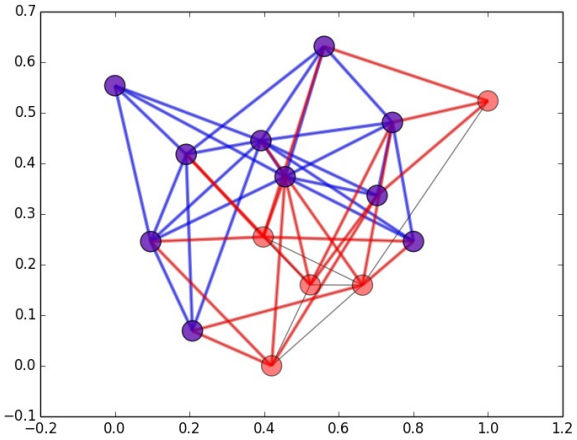


Figure 5.1: Social graph with Honest nodes in blue, Sybil nodes in red, and Attack edges in red

Attack Model: We assume that an attacker can create an unlimited number of Sybil nodes constituting a subgraph (the Sybil region) whose topology is beyond the control of the OSN provider. Attackers can create as many number of attack edges as they want, but they do not have control on how many of those attacks edges will be *successful* in establishing victims. Our Sybil defense mechanism is built around the assumption that we know at least one honest node. This assumption is reasonable since such information can be provided by the administrator of the OSN after a carefully designed process for that purpose. Same assumption is made by other works as well. In addition, we assume that the attacker does not have complete knowledge of the entire OSN topology, since this will require him to crawl the entire network. However, the attacker can acquire the knowledge about a subgraph of the OSN.

5.2 System Design

SybilRadar operates in three steps. The process starts with the network dataset (set of nodes and edges) being fed to the SybilRadar framework. The first step involves the computation of similarity values between a given pair of nodes. The chosen similarity metric is the Adamic-Adar metric [114], which is based on the notion of common friends between any given pair of nodes. The intuition for choosing this metric is that honest nodes will have more friends in common that Sybil

nodes. In the second step, the result from the first step is refined using another similarity metric which is the Within-Inter-Community metric (WIC) [25]. This metric leverages the underlying community structure of the given social graph. The Louvain method [115] is used to find the social graph community information that is fed to the WIC similarity metric computation. This step produces the prior information which is the similarity values of any given pair of nodes driven by the community they belong to. We end this step with a tuning of the nodes similarity values for those nodes with a similarity value greater than 1. We assign the resulting similarity values to the social graph edges as their weights. In the third step, we run a Modified Short Random Walk on the weighted social graph. This step produces trust values, which are the node's landing probabilities of the random walk. These values are assigned to each node as the posterior information in order to perform the ranking of nodes. The general framework of SybilRadar is shown in figure 5.2.

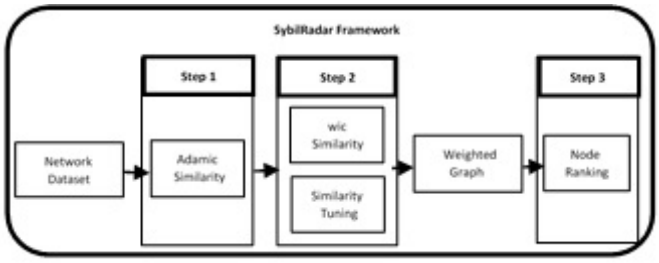


Figure 5.2: SybilRadar framework

5.2.1 Predicting Attack Edges

Similarity metrics have been extensively used in the field of link prediction in networks. The link prediction problem consists of predicting possible future links based on observing existing links in a given network. Sybils try to maliciously create trust relationships with honest nodes by creating attack edges. Our algorithm tries to predict these bad links. The prediction of future possible links can be based on observing unique and recent features of nodes present in the network, or can be based on structural properties of nodes present in the network. In the first case, feature similarity metrics are used, while structural similarity metrics are used in the latter case. Interested

readers are referred to [116], [117] for link prediction works using feature similarity metrics, and references [118], [119] for link prediction works based on structural similarity metrics. In OSNs node attributes are not always available. For example, users may not complete their profiles or provide inaccurate or misleading information to protect their sensitive information. Moreover, trying to learn user behavior, where complete information is available, may raise privacy concerns. This leads us to consider structural similarity metrics, which are based solely on the structure of the social graph induced by trust relationships between users [120].

We adopt the Adamic-Adar metric [114] to compute an initial similarity value of pairs of nodes. For a given OSN graph $G = (V, E)$, let x and y be two nodes and $\Gamma(x)$ and $\Gamma(y)$ be the sets of neighbors of x and y. The Adamic-Adar (or simply Adamic) similarity measure is given by

$$S_{x,y}^{AA} = \sum_{w \in \Gamma(x) \cap \Gamma(y)} \frac{1}{log \mid \Gamma(w) \mid} \tag{5.1}$$

Given the initial social graph, running the $Adamic$ similarity metric on each pair of nodes results in a weighted social graph with the weight on a link being the similarity value of nodes adjacent to that link. For a given social graph $G = (V, E)$ and for each edge $(u_1, u_2) \in E$, the similarity value $Adamic(u_1, u_2)$ becomes its weight $w(u_1, u_2)$. After computing the Adamic similarity metric we make the following observations :

1. We have three sets of edges: edges with weight $w(u_1, u_2) = 0$, those with weight $w(u_1, u_2) \in [0, 1]$, and the edges with weight $w(u_1, u_2) > 1$.

2. For the attack edges, at least 95% of them have their weight $w(u_1, u_2) = 0$, and less than 5% have their weight $w(u_1, u_2) \in [0, 1]$, while about zero to an infinitely small number of them have their weight $w(u_1, u_2) > 1$.

3. The situation for honest edges is quite different. At least 90% of them have their weight $w(u_1, u_2) > 1$, and about less than 5% have their weight $w(u_1, u_2) \in [0, 1]$, whereas those with weights $w(u_1, u_2) = 0$ are also less than 5%.

We were able to make these observation because the social graph used for simulation purpose is derived from a synthetic network whose attack edges, Sybil edges and honest edges are known beforehand. We were able to predict about 90% of existing attack edges. We made similar observation later with our real data. We observe that predicting attack edges can be very helpful. since it can reveal nodes that have potentially been victims of Sybil attacks. This can be a valuable information for a system administrator. Note, however, that not all edges that have their $w(u_1, u_2) = 0$ are all attack edges. In other words, some honest edges, as well as some Sybil edges, have their weight equal to 0. This is due to the fact that not all pairs of honest nodes or Sybil nodes have common friends, which is the criteria used in computing the similarity value using the Adamic metric.

5.2.2 Further Refinement of Attack Edge Detection

We next observed that there was an extreme case where our current Sybil detection algorithm completely looses its accuracy. This situation arises when the number of attack edges far exceeds the number of honest nodes.

This situation is not desirable because, at this level, any attacker that can succeed to create a huge number of attack edges compared to the number of benign accounts and get a high degree of certainty of having a significant number of his Sybil accounts evading the Sybil detection mechanism. We observe that among the attack edges that were not detected a significant number have their weights $w(u_1, u_2) \in [0, 1]$. These edges are mixed with a significant portion of other non attack edges which also have their weight $w(u_1, u_2) \in [0, 1]$. We want to filter out as many attack edges as we can in order to increase the number of detected attack edges. For this purpose, we leverage properties of communities (or clusters) in networks.

Community detection

OSNs typically display clustering characteristics. The idea of using a clustering structure when designing a similarity metric was advanced by [24] who showed that link prediction measures based on structural similarity perform poorly for a network with a low clustering structure. This

inspired [25] to first divide the network into communities, and use this clustering structure information in designing a similarity metric for the link prediction problem. In order to measure the quality of a community structure, Newman et al [26] introduced a modularity function Q. Given a social graph $G = (V, E)$, the modularity function can be expressed as follows:

$$Q = \frac{1}{2m}\sum (A_{ij} - \frac{k_i k_j}{2m})\delta(C_i, C_j) \tag{5.2}$$

where k_i and k_j are respectively the degree of nodes i and j. A_{ij} represents an element of the adjacency matrix, and m is the size of E which is the set of edges of the given graph G. C_i and C_j are the respective communities to which i and j belong. The parameter δ is the Kronecker delta symbol whose value is 1 when both i and j belong to the same community, and is 0 when both nodes belong to different communities. The goal of community detection is to divide a network into communities in a manner that maximizes the value of the modularity. In our Sybil detection algorithm we use a modularity optimization method called the Louvain Method [115].

The Louvain method [115] was developed at the University of Louvain to detect communities in a network in an efficient manner. The Louvain method operates by initially putting each node in its own community. Obviously, the modularity obtained at this steps is not optimal. The algorithm then proceeds to group together closely connected communities in a way that improves the modularity. A new graph is then constructed by converting the obtained communities to nodes and by adding links weighted by the inter-community connectivity. This iteration is then repeated until an optimal modularity is obtained. Louvain method is the most used algorithm for finding communities because the computation of each its iteration is linear in the number of edges. Moreover, it can be easily parallelized [121]. This algorithm can only find disjoint communities, which, however, is not an issue in our particular use case. Several other algorithm have been developed to detect overlapping communities in a network [122], [123]. The division of our social graph into communities can be illustrated in figure 5.3.

To identify clusters, we first collect all the edges with weight $w(u_1, u_2) \in [0, 1]$, and for each of these edges we compute the similarity value of its end nodes using the Within Inter Cluster (WIC)

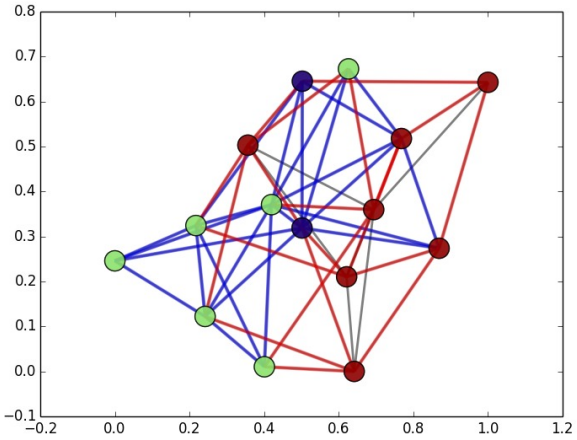


Figure 5.3: Finding communities using Louvain method. Each node color represents a community with Honest edges in blue, and Attack edges in red

similarity metric [25]. This metric is built based on the notion of *within-cluster common neighbors* and *inter-cluster common neighbors*. For a given graph $G = (V, E)$, and nodes $u, v, w \in V$, w is said to be a within-cluster common neighbor of u and v if w belongs to the same community as them. Otherwise, w is said to be an inter-cluster common neighbor of u and v. The WIC metric is defined to be the ratio between the size of the set of within- and inter-cluster common neighbors [25]. Formally, the WIC metric is expressed as:

$$S_{x,y}^{WIC} = \frac{|\bigwedge_{x,y}^{W}|}{|\bigwedge_{x,y}^{IC}| + \delta} \tag{5.3}$$

The numerator $|\bigwedge_{x,y}^{W}|$ represents the set of within-cluster (W) common neighbors of x and y and is defined by

$|\bigwedge_{x,y}^{W}| = \{z \in \bigwedge_{x,y} | x^C, y^C, z^C\}$ with x^C, y^C, z^C being the respective communities of x, y, and z. The denominator of the equation represents the set of inter-cluster (IC) common neighbors of x and y, which is defined as the complement $|\bigwedge_{x,y}^{IC}| = \bigwedge_{x,y} \setminus |\bigwedge_{x,y}^{W}|$ with $\bigwedge_{x,y}$ being the set of all common neighbors of x and y [25]. A close look at the equation above reveals that the information about the clustering structure of the given graph is necessary in order to compute this metric.

Running the WIC similarity metric on edges with weight $w(u_1, u_2) \in [0, 1]$ results in this set of edges being reduced in size. Some of its edges are converted to edges with weight $w(u_1, u_2) > 1$ while the remaining are converted to edges with weight $w(u_1, u_2) = 0$, thus increasing the size of the set of attack edges. We terminate this preprocessing with a tuning that aims to scale down all weights $w(u_1, u_2) > 1$ to $w(u_1, u_2) = 1$. The benefit of this transformation is a gain in the accuracy and the stability of the detection mechanism. We are now ready to proceed to the ranking of nodes in order to declare which ones are Sybil nodes, and which ones are benign nodes.

5.2.3 Trust Propagation

To rank the nodes, each node in the OSN is assigned a *degree-normalized landing probability* of a modified short random walk. The walk starts from a known non-Sybil node. Using this node, we compute the probability of a modified random walk to land on each node u_i after k steps. This landing probability is analogous to the strength of the trust relationship between the nodes, and each step of the walk's probability distribution is considered as a trust propagation process [20].

Early terminated walk: The modified random walk used by SybilRadar is called a short walk because it is an early terminated walk [124]. A random walk that is run long enough will end up with with all the nodes in the social graph having an uniform trust value. The uniform trust value is called the convergence value of the random walk [125]. The number of steps k required for a random walk to converge is called the mixing time of the social graph. Several researches [126], [13], [37] have shown that for various social networks, the mixing time is larger than $O(\log n)$ with n being the number of nodes in the social graph. To compute the trust values, SybilRadar adapts the Power Iteration method [127]. The power iteration method is an eigenvalue computation method that can compute a rank of individual nodes based on an averaging of ranks of all nodes in an efficient manner. It is a method used by other mechanisms such as PageRank, and TrustRank to compute rank values [128], [129]. In SybilRadar the modified power iteration is terminated after $O(logn)$ iterations.

Our modified power iteration method takes as input the transition matrix of social graph, where each element of the matrix is the probability of the random walk to transition from one node to another. The method is executed as a succession of transition matrix multiplications, and at each step the iteration computes the trust distribution over nodes. It works as follows. We define the trust value on a node v after i iterations as $T(v)$, and the total trust as the value $T \geq 1$. Given $s_1, \ldots, s_k$ the selected trust seeds, we initialize the power iteration by distributing the total trust among the trust seeds as follows:

$$T^{(0)}(v) = \begin{cases} T/k & \text{if } v \text{ is a trusted seed} \\ 0 & \text{otherwise} \end{cases} \tag{5.4}$$

After the initialization step, each node v_i is assigned a trust value $T(v_i)$. The process then proceeds with each node v_i evenly distributing its trust value $T(v_i)$ to each of its neighbor v_j during each round of power iteration. Each node v_i then updates its trust value in accordance with the trust values received from its neighbors. The trust distribution is done proportionally to $w(v_i, v_j) \div deg(v_j)$ which is the ratio of the weight on the edge between the node v_i and its neighbor v_j over the degree of the neighbor node v_j. The use of the weight ensures that a big fraction of the total trust will be distributed to benign accounts rather to Sybil accounts. This results in benign accounts having higher trust value than Sybil accounts. The entire process is summarized in equation 4.

$$T^{(k)}(v_i) = \sum_{(v_i, v_j) \in E} T^{(k-1)}(v_j) \frac{w(v_i, v_j)}{deg(v_j)} \tag{5.5}$$

After $O(logn)$ iterations, the resulting trust value $T(v_i)$ assigned to each node v_i is normalized according to v_i degree. The normalization process involves dividing each node trust value by its degree. This transformation is motivated by the fact that trust propagation is influenced by the node degree, and that this results in the trust propagation being biased toward node with higher degree

when the number of iterations grows larger. The normalization ensures that benign nodes get trust values that are close in value [20]. This is influential in identifying Sybil nodes after the ranking.

5.3 System Evaluation

We first evaluate SybilRadar using both a synthetic network and a real dataset collected from Facebook. For both evaluations we employ procedures that other researchers have used in this line of work. We compare SybilRadar against SybilRank which takes the same structure-based approach that is also based on the use of the power iteration method albeit on an unweighted graph unlike SybilRadar which uses a weighted graph.

Comparing SibilRadar to SybilRank will help highlight the role played by similarity metrics in detecting Sybil accounts. In addition, SybilRank has been demonstrated to outperform other previous structure-based methods [20]. Although Integro outperforms SybilRank, it is not a pure structure-based approach since it leverages account's feature information collected from recent users activities. We have indicated earlier our reservations for using user attributes or activities in Sybil detection. For this reason, we are not including it in our comparison.

Evaluation metric: To express SybilRadar's performance, we use the Area Under the Receiver Operating Characteristic Curve (AUC). AUC for our purpose is defined as the probability to have a randomly selected benign node ranked higher than a randomly selected Sybil node. The AUC is a tradeoff between the False Positive Rate and the True Positive Rate of the classifier. A perfect classifier has an AUC of 1 while a random classifier has an AUC of 0.5. Therefore, we expect our classifier to perform better than a random classifier, and to have an AUC as close as possible to 1.

5.3.1 Evaluation on Synthetic Networks

The synthetic network is generated using known social network models. First, the honest and the Sybil regions are generated by providing relevant parameters to the network model, like the number of nodes, and the average degree of nodes. Then, the attack edges are generated following

the scenario chosen in the experiment. They can be randomly generated or generated in a way to target some specific honest nodes.

Initial Evaluation: We generate the honest region and the Sybil region using the Powerlaw model. The honest region has a size 4000 nodes while the Sybil region has 400 nodes. Both regions have an average degree of 10. The attack scenario chosen simulates an attacker randomly generating 2000 attack edges. The weights on the edges are set to be the values resulting from the two similarity metrics previously described in this section 4.2. For this experiment, we select 20 trust seeds from the honest region. These are supposed to be some nodes that the OSN system administrator is absolutely certain to be honest nodes.

Results: Comparing the ranking quality of both SybilRank and SybilRadar under the chosen scenario, the results show that SybilRadar outperforms SybilRank. SybilRadar resulted in an AUC which is always greater than 0.95, an AUC that is higher than SybilRank's AUC of 0.90.

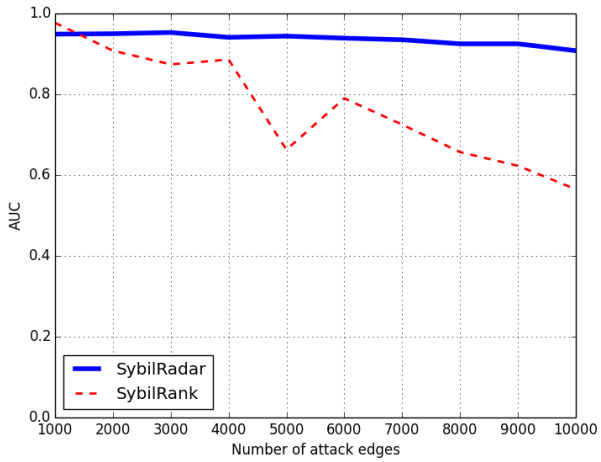
Varying the number of attack edges : In the next experiment, we keep the honest and the Sybil regions as set up in the previous Basic Evaluation. In order to stress-test the platforms being compared, we decide to successively vary the number of attack edges from 1000 to 10000. We want to investigate how the increase in number of attack edges affects the performance of both platforms.

Results: This result can be seen in Figure 5.4(a). As the number of attack edges increases, we notice that SybilRank is unable to keep its initial performance, with its AUC dropping from 0.97 to less than 0.6. Meanwhile, the increase in the number of attack edges affects the performance of SybilRadar only marginally. Its AUC still stays above 0.90. This highlights the effectiveness of using similarity metrics in detecting Sybil nodes in the case of social graphs with weak trust.

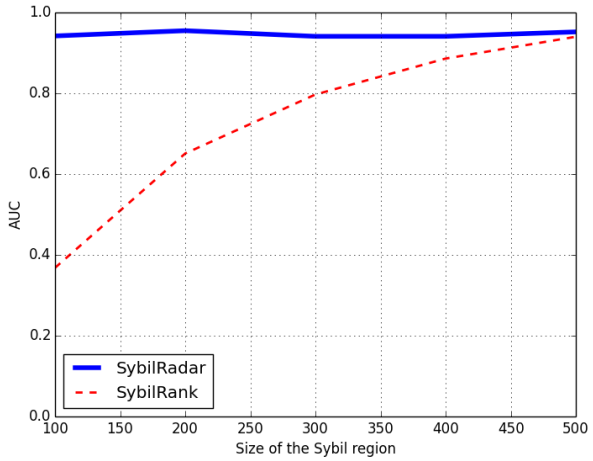
Varying the size of the Sybil region : In this experiment, we explore how the increase in the size of the Sybil region affects the performance of both platforms. For this purpose, we design a honest region with 4000 nodes, and an average degree of 10. The attacker is able to create randomly 4000

attack edges. We vary the size of the Sybil region from 100 to 500 nodes each with an average degree of 10.

Results: The experiment results (see figure 5.4(b)) show that SybilRadar and SybilRank react differently to the increase in the size of the Sybil region. When the size of the Sybil region is relatively small compared to the size of the honest region, SybilRank performs poorly. SybilRank performance improves when the size of the Sybil region get relatively bigger. However, as illustrated in figure 5.4b, SybilRadar displays a stable performance that is less sensitive to the size of the Sybil region.



(a) Varying number of attack edges



(b) Varying size of the Sybil region

Figure 5.4: Performance on synthetic data

5.3.2 Evaluation on Real-world Twitter Network

To study if our choice of data in the previous experiments biased our results, we also evaluated the performance of SybilRadar under larger datasets from a different OSN, namely, the Twitter network. The dataset we used is a combination of four datasets: The FakeProject dataset, the Elezioni2013 dataset, the TWT dataset, and the INT dataset [46]. The FakeProject dataset contained profiles of real users who received friend requests from @TheFakeProject, an account created for The FakeProject that was initiated in 2012 at IIT-CNR, in Pisa-Italy. The Elezioni2013 dataset was generated in 2013 for a sociological research undertaken by the University of Perugia and University of Rome, La Sapienza. The TWT dataset and the INT dataset were a set of fake accounts purchased respectively from the fake accounts providers http://twittertechnology.com and http://intertwitter.com. The first two datasets mentioned provided the honest nodes while the last two datasets provided the fake nodes [46].

Pre-processing: Since the Twitter network is directed, we considered only the set of bidirectional edges. This provided us with an initial network of 469,506 nodes and 2,153,427 edges. We further refined this network by removing all nodes with degree less than 1.
The resulting twitter network then comprised 135,942 nodes and 1,819,864 edges. The honest region comprised 100,276 nodes and 634,127 edges while the Sybil region was constituted of 35,666 nodes and 1,086,352 edges. The two regions were connected by 99,385 attack edges.

Results: We ran SybilRadar several times using the Twitter dataset described above. SybilRadar resulted in an AUC which was always greater than 0.95 as shown in figure 5.5.

5.4 Discussions

In this chapter, we presented a new framework for detecting Sybil attacks in an Online Social Network. In a Sybil attack, an adversary creates a large number of fake identities in an OSN or forges existing identities. The adversary then uses these fake identities to influence the underlying trust basis of the OSN and perform malicious activities such as social spamming, malware distribution and private data collection. Sybils are a significant threat to the OSN. While they cannot be

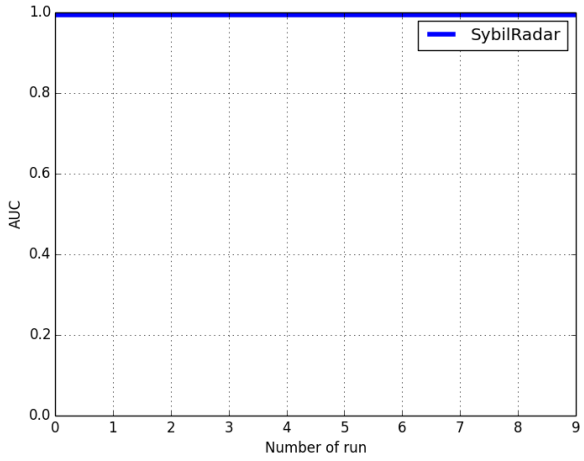


Figure 5.5: Performance on Twitter dataset

prevented in most OSNs because of their open nature, this work provides a solution by which the OSN operator can automatically, speedily and accurately detect such Sybils.

SybilRadar belongs to the class of Sybil detection techniques that rely on the graph structure of the OSN. This is in contrast to the alternate group of detection mechanisms that rely of identifying features related to user attributes and activities. We believe that while the second class of detection algorithms may provide good detection results on carefully cleaned up OSN data, in real life such data is difficult to obtain since OSN users frequently leave their profiles incomplete or use misleading information purposefully. Moreover, trying to obtain user activity related data may raise serious privacy concerns. As a result, SybilRadar relies on just the structural properties of the OSN graph. We used a variety of OSN test data – both synthetic as well as real-world – to evaluate the detection accuracy of SybilRadar. Our experimental results show that SybilRadar performs very well – much better than the most well known similar technique – even for OSNs that have the weak trust model and which have a very large number of attack edges between Sybil nodes and honest nodes.

Chapter 6

Sybil Classification in Online Social Networks Using Only Structural Features

Early Sybil account detection mechanisms involved classification of users into benign and malicious based on various attributes collected from the user profiles. One challenge affecting these classification methods is that user attributes can often be incomplete or inaccurate. In addition, these classification methods can be evaded by sophisticated attackers. More importantly, user profiles can often reveal sensitive user information that can potentially be misused causing privacy violation. In this chapter, we propose a Sybil detection method that is based on the classification of users into malicious and benign based on the inherent topology or structure of the underlining OSN graph. We propose a new set of structural features for a graph. Using this new feature set, we perform several experiments on both synthetic as well as real-world OSN data. Our results show that the proposed detection method is very effective in correctly classifying Sybil accounts without running the risk of being evaded by a sophisticated attacker and without compromising privacy of users.

The remainder of this chapter is organized in the following manner: Section 8.1 presents some background on the Sybil detection problem. We present our attacker model in section 6.1.1. In section 6.1.2 we describe the dataset used in this work. We then present a set of existing graph centrality measures that have been used elsewhere in Sybil detection and/or Spam detection. These existing features together with the new ones that we introduce in section 6.1.5 constitute the initial feature vector for our classification technique. On this initial feature vector, we employ feature selection techniques to reduce the set of features to eight. Section 6.3 introduces the classification methods used in this work. The results of our experiments are presented in section 6.4. A discussion of our results is presented in section 6.5.

6.1 Background

Most popular OSNs are being targeted by Sybils that can be designed to have realistic profiles, and to be well integrated in the social graph structure. This makes it challenging to identify Sybils. Our goal is to detect Sybils using their structural characteristics in the social graph. These characteristics are difficult for the attacker to fake because they depend on the structure of the rest of the OSN. In this section, we present some background notions necessary to the understanding of our study. We start by introducing the attack model that we use for this work. Next we present some graph theory concepts that we use throughout this work. This is followed by some important graph centrality measures that form the inspiration of our proposed new set of features. We finish with a description of what constitutes a sybil attack in OSN.

6.1.1 Attack Model

We do not make any assumption about the capability of the attacker to create Sybils. The attacker can create an unlimited number of Sybils that can be connected to each other in to constitute a subgraph. Attackers can use any method at hand to befriend benign accounts. However, we assume that attackers do not have control on how many of their friend requests will be accepted.

Most of OSNs make it near impossible to crawl the entire network. Thus, we assume that the attacker does not have a complete knowledge of the OSN topology. However, using freely available datasets along with de-anonymization techniques, an attacker can gain knowledge about any subgraph of the social network.

6.1.2 Dataset

In this section, we describe the different datasets we have used to conduct our experiments. The first dataset is a real world Twitter dataset with ground truth information. The second dataset is a real world Facebook dataset complemented with some synthetically generated Sybil nodes.

Twitter Dataset Description

We evaluated the performance of our classification on a real dataset which is a subset of the Twitter network. Our dataset is actually a collection of four different datasets: The FakeProject dataset, the Elezioni2013 dataset, the TWT dataset, and the INT dataset [46]. The FakeProject dataset was initially generated for the FakeProject, a project initiated at IIT-CNR in Pisa-Italy. This dataset comprises the profiles of real users who received and accepted a friend request from the @FakeProject account in 2012. The University of Perugia and the University of Rome La Sapienza initiated a sociological research in regard to the 2013 Italian election. During this research, the Elezioni2013 dataset was generated. This dataset comprises real users.

In order to complement the graph with fake users, two additional dataset of fake users were used. These are the TWT dataset and the INT dataset. These two datasets consists of fake accounts purchased on the black market. The TWT dataset is a set of fake accounts that were bought from the provider $twittertechnology.com$ while the INT dataset was acquired from the provider $intertwitter.com$.

All four datasets put together provide us with a network of 469,506 nodes and 2,153,427 edges. We pre-process this network by only considering the biggest connected component. The result is a graph of 469,504 nodes and 2,153,426 edges. The real users constitute the honest region with 372,251 nodes while the fake users constitute the Sybil region with 97,253 nodes. The two regions are linked by 99,385 attack edges that form the connection between Sybil and honest users.

Facebook Dataset Description

In order to illustrate that our classification method is not limited to the Twitter network, we have added to the analysis a Facebook dataset. This dataset is called the Ego-Facebook dataset and originated from the Stanford Network Analysis Project (SNAP) [130].

The social graph represented by this dataset is made of 4,039 nodes and 88,234 edges. Nodes and edges represent respectively Facebook accounts and friendship relationships. This is an undirected graph as was the case with the graph representing the previous dataset. This Facebook dataset constitutes our benign region. We generate the Sybil region by adopting the power-law

degree distribution [131], since social networks are known to have a long-tail degree distribution. The generated Sybil region has 4,000 nodes and 88,000 edges. We connect the two regions by randomly adding 60,000 attack edges between the two regions. The final network topology has 8,039 nodes and 236,234 edges. A similar setting combining the Facebook dataset with synthetically generated Sybils was also adopted by [40].

6.1.3 Graph Notation and Definition

We model an OSN as an undirected graph $G = (V, E)$, where V is the set of nodes representing the users in the network, and E is the set of edges. Each edge represents a bilateral social relationship between two users in the network. The resulting social graph G has $n = |V|$ nodes and $m = |E|$ edges. A given node v of the social graph is characterized by its degree $deg(v)$ which represents the number of its friends.

Definition 1. *Ego network [132]:* The Ego network of a node v is the network formed by v and all the nodes to whom it is directly connected to, together with the links among those nodes. The Ego network of a node v can be understood to be the neighborhood networks or the first order neighborhoods of v .

Definition 2. *Centrality measure [133]:* A centrality measure is a metric capturing a person's centrality, power, prestige, or influence in a given network. Centrality measures are used for the identification of key persons within a social network. The concept of centrality measure depends on the context, and different centrality measures have been designed based on the application.

6.1.4 Sybil Attacks

Sybil attacks are a form of attacks targeting OSN in which a malicious user creates multiple fake identities, known as sybils, which are used to breach the privacy of legitimate users. Researchers have identified four main types of sybil attacks in OSN: Sybils with a dense friendship subgraph, sybils with a sparse friendship subgraph, sybils with a normal friendship graph, and creepers [134].

In the first category, sybils with dense friendship subgraph, a malicious user creates multiple sybil accounts and a large number of connections among them [135], [20]. This results in the sybil account's friendship subgraph forming a tight knit cluster.

In the second category, sybils with a sparse friendship subgraph, the sybil account created by a malicious user either does not form social links among themselves or are loosely connected [10]. The sparsity of the sybil friendship subgraph is the result of the snowball sampling techniques used by sybils to identity potential victims. In addition, it was determined that those sybils tend to send friendship requests to popular users, because of their high likelihood to accept friend requests from strangers [136].

Sybils with normal friendship subgraph constitute the third type of sybil attacks. In this category, even though their friendship subgraph is normal, it is different from the one for legitimate users. The reason for this situation is that, in order to blend in, sybils create a few connections among themselves. Next, they send friend requests to legitimate users. However, it was shown in [135], [20] that only a small percentage of those friend requests get accepted by legitimate users [136].

Creepers constitute the fourth type of sybil attacks. Creepers are fake accounts created by non-malicious users with the purpose to perform activities like pranks, stalking, cyberbullying, etc, [14].

6.1.5 Graph Centrality Measures Previously Used in Sybil and/or Spam Detection

Previously, researchers have used graph centrality measures as features in Sybil detection. Graph centrality measures have also been used in OSN spam detection. To design and refine our classifiers, we start with these previously used centrality measures and augment them with a set of measures that we propose. Next, we use feature selection techniques to identify the feature set that gives the most accurate results. At this stage, we discuss the previously used centrality measures. These measures include the *average nearest neighbor degree*, the *average degree*, the

core number, the *clustering coefficient*, the *edge volume*, the *weighted vertex volume*, the *average core number* and the *average clustering coefficient*. We have run the experiments using other centrality measures such as the betweenness centrality, the closeness centrality, and the Page-rank centrality. These measures, besides being computationally expensive, did not contribute much to the improvement of the results.

Average nearest neighbor degree: The average nearest neighbor degree of a node of degree k is a measure of dependencies between degrees of neighbor nodes in a network [137]. This is also known as assortativity. Assortativity is the preference for a network's nodes to attach to others with similar degree [138]. It is expressed by:

$$K_{nn}(u) = \frac{1}{k_u} \sum k_j \tag{6.1}$$

where $K_{nn}(u)$ denotes the average nearest neighbor of node u, k_u is the degree of node u, and k_j is the degree of a node adjacent to node u.

Average degree: The average degree of a node is the average number of edges adjacent to that node [139]. For a given node u, and considering its Ego network, the average degree of node u is expressed as:

$$AD(u) = \frac{1}{|N(u)|} \sum_{j \in N(u)} k_j \tag{6.2}$$

where $AD(u)$ is the average number of edges adjacent to node u, $N(u)$ is the neighborhood of node u, and k_j is the degree of node j that belongs to the neighborhood of u.

Core number: Given a graph $G = (V, E)$, the $k\text{-}core$ or core of order k is the maximal subgraph $G' = (V', E')$ such that for all nodes $v \in V'$, $deg(v) \geq k$ where deg denotes the degree of node v. The order of the core with the highest order to which a node belongs determines the core-ness or core number of the node [140]. Previous research [141], [142] have shown the usefulness of $k\text{-}core$ in detecting cohesive groups of fraudsters [143].

Clustering coefficient: This centrality measures the propensity that the neighbors of a node v are linked to each other. It captures the friends-of-friends relationship between social network accounts. In a social network, the friends of a benign user are likely to be benign. This is not always true for malicious users since they establish friendships with benign users in a random manner [144]. The clustering coefficient of a node u, denoted by $CC(u)$, is intended to capture the situation described above, and is defined [145] as:

$$CC(u) = \frac{2tr(u)}{k_u(k_u - 1)} \tag{6.3}$$

where $tr(u)$ is the number of triangles formed by node u with two of its neighbors, and k_u is the degree of node u.

Edge volume: For a given node u, and its Ego network $N(u)$, the Edge Volume of node u is a measure of network properties in terms of the weights of edges. It represents the node strength [146], and is expressed by:

$$EV(u) = \sum_{v \in |N(u)|} w(u, v) \tag{6.4}$$

where $EV(u)$ is the Edge Volume of node u, $N(u)$ is the neighborhood of node u, and $w(u, v)$ is the weight between node u and all other nodes adjacent to it. If $deg(x)$ represents the degree of node x, then this weight is expressed by:

$$w(u, v) = \frac{deg(u).deg(v)}{\sum_{v \in V} deg(v)} \tag{6.5}$$

Weighted vertex volume: For a given node u, the Weighted Vertex Volume of u is the ratio of the degree of u over its Edge Volume [146]. It is expressed by:

$$WVV(u) = \frac{deg(u)}{EV(u)} = \frac{deg(u)}{\sum_{v \in |N(u)|} w(u, v)} \tag{6.6}$$

Average core number: This metric represents the average core number of nodes adjacent to the focus node. Let $C(v)$ be the core number of a node v. For a given node u and its Ego network $N(u)$, the average Core of u, $AC(u)$, is expressed by:

$$AC(u) = \frac{1}{|N(u)|} \sum_{v \in N(u)} C(v) \tag{6.7}$$

Average clustering coefficient: For a given node u, the average clustering coefficient of u is a measure of the average of clustering coefficients in the Ego network of u. Let $CC(v)$ be the clustering coefficient of node v as defined in equation 6.3. Then the average clustering coefficient $ACC(u)$ is expressed by:

$$ACC(u) = \frac{1}{|N(u)} \sum_{v \in N(u)} CC(v) \tag{6.8}$$

6.2 Proposed New Features for Sybil Detection

We now describe the set of new features for nodes that we have proposed. All features are extracted from the structure of the social graph. Each feature is based on some centrality metric, and represents the importance of the node in the graph. The philosophy for feature identification is to have features that an attacker cannot manipulate in order to bypass the detection mechanism. Some of the features – *weighted degree-core centrality* and *weighted degree-clustering centrality* – have been inspired by the *weighted degree-degree centrality* [147].

Weighted degree-degree centrality Let $EN(v)$ be the Ego network of node v, and w_{ij}, the *degree weight*, be the probability of two nodes i and j are connected based on their degrees. Degree weight is given as:

$$w_{ij} = \frac{deg(i).deg(j)}{\sum_{u \in V} deg(u)} \tag{6.9}$$

where $deg(u)$ is the degree of a node u. The Weighted Degree-degree centrality of a node v is the sum of weighted degrees of that node's friends and is expressed as:

$$DD_w(v) = \sum_{j \in EN(v)} [w(v,j).deg(v)] \tag{6.10}$$

A second set of features has been proposed based on the concepts of *subgraph intensity* and *subgraph coherence*. The clustering coefficient is a basic metric to study the clustering properties of a graph. By extending the concept of clustering to a subgraph, [148] introduced the concept of subgraph intensity and subgraph coherence. Let g be a subgraph and l_g is the set of its links whose respective weights are w_{ij}. The *subgraph intensity*, $I(g)$, is defined as the geometric mean of the weights on its links, and is expressed as:

$$I(g) = (\prod_{(i,j) \in l_g} w_{ij})^{\frac{1}{|l_g|}} \tag{6.11}$$

If one of the weights is low or all the weights are low, this can lead the subgraph intensity to be low without a way to distinguish which case is the cause of the low value.

The measure *subgraph coherence* was introduced to clarify this situation. It is defined as the ratio of the subgraph intensity to the arithmetic mean of its weights [148]. Let $\Phi(g)$ is the subgraph coherence of g, $I(g)$ is the intensity of the subgraph g, and l_g is the set of links, with each link having a weight w_{ij}. The subgraph coherence is expressed by:

$$\Phi(g) = \frac{I(g)|l_g|}{\sum_{ij \in l_g} w_{ij}} \tag{6.12}$$

The second set of new features includes the *degree-intensity centrality*, the *degree-coherence centrality*, the *core-intensity centrality*, and the *core-coherence centrality*. We call these centrality measures hybrid centrality measures. The hybrid centrality measure of a node is computed as the sum of basic centrality values of that node's friends. We now present our six new features.

Weighted degree-core centrality: We have extended the concept of weighted degree-degree centrality by proposing the *weighted degree-core centrality*. Let $EN(v)$ is the Ego network of node v, and w_{ij} is the probability of two nodes i and j to be connected based on their degrees, as

expressed by equation 6.9. The weighted degree-core centrality of a node is the sum of weighted core numbers of that node's friends, and it is expressed as:

$$DC_w(v) = \sum_{j \in EN(v)} [w(v,j).core(v)] \tag{6.13}$$

Weighted degree-clustering centrality: Let $EN(v)$ is the Ego network of node v, $Cl(v)$ the clustering coefficient of node v, and w_{ij} is the weight on the link (i,j) as expressed by equation 6.9. The *weighted degree-clustering centrality* of a given node v is the sum of weighted clustering values of the node v's friends, and it is expressed as:

$$DCl_w(v) = \sum_{j \in EN(v)} [w(v,j).Cl(v)] \tag{6.14}$$

Degree-intensity centrality: The *degree-intensity centrality* of a node is defined as the geometric mean of the degree weights on the links in its Ego network. If $EN(v)$ is the Ego network of node v, and w_{ij} is the degree weight of the link $(i,j) \in EN(v)$, the degree-intensity centrality, $DI(v)$ is expressed as:

$$DI(v) = (\prod_{(i,j) \in EN(v)} w_{ij})^{\frac{1}{|EN(v)|}} \tag{6.15}$$

Degree-coherence centrality: We define the *degree-coherence centrality* of a node v as the ratio of the degree-intensity of v to the arithmetic mean of the weights on the links in v's Ego network. This measure is expressed as:

$$DC(v) = \frac{DI(v)|EN(v)|}{\sum_{ij \in EN(v)} w_{ij}} \tag{6.16}$$

where w_{ij} is expressed by equation 6.9.

Core-intensity centrality: Modeled on the concept of subgraph intensity, we define the *core-intensity centrality* of a node v as the geometric mean of the weights on the links in v's Ego network. Let $EN(v)$ be the Ego network of node v, and w_{ij} is the core weight of the link $(i,j) \in EN(v)$ which represents the probability of two nodes i and j to connect based on their

core numbers. Let $core(u)$ represents the core number of a node u. The core weight is expressed as:

$$w_{ij} = \frac{core(i).core(j)}{\sum_{u \in V} core(u)} \tag{6.17}$$

The core-intensity centrality is then expressed as:

$$CI(v) = (\prod_{(i,j) \in EN(v)} w_{ij})^{\frac{1}{|EN(v)|}} \tag{6.18}$$

Core-coherence centrality: We define the *core-coherence centrality* of a node v as the ratio of the core-intensity of v to the arithmetic mean of the weights on the links in v's Ego network. Let $CI(v)$ be the core-intensity of node v and w_{ij} is the core weight expressed by equation 6.17. The Core-coherence centrality is expressed as:

$$CC(v) = \frac{CI(v)|EN(v)|}{\sum_{ij \in EN(v)} w_{ij}} \tag{6.19}$$

6.3 Classification Methods

Based on the graph centrality measures discussed earlier, we now describe the supervised machine learning methods we use to automatically detect Sybil accounts in Online Social Networks. The supervised machine learning methods are essentially trained classifiers that make the binary decision whether an account is benign or Sybil. The classifiers used in this works are: Random forest, Adaboost, and KNN. We have built our classifiers using scikit-learn [149], a machine learning toolkit that is supported by Google.

Adaboost [150] is the first practical boosting algorithm. Boosting is a machine learning approach based on the idea of combining several relatively weak and inaccurate prediction rules in order to reach a highly accurate prediction rule [151]. In practice, boosting involves repeatedly learning weak classifiers, each built using a different sample of the training data. The final classifier is obtained by combining the weak classifiers learned at each round of the classification process.

K-Nearest Neighbor (KNN) [152] is a non-parametric classification method that takes as input training examples and produces an output which is a class membership. Using KNN, the classification of an unlabeled object occurs when the object is assigned to a class based on a majority vote by its K nearest neighbors. The classification accuracy depends of two elements: the distance between the object to be classified and its nearest neighbors. The second element is the parameter K. This parameter can be chosen experimentally by taking the value K which gives the least classification errors.

Random Forests [153] is a classification method using several decision trees. In order to be classified, the input data point is assigned to each tree in the forest. Each tree is going to classify the given data point, and the final result is the majority vote from all participating trees. We have trained our Random Forests classifier using 100 estimators and the optimal parameters setting is obtained via cross validation.

Support Vector Machine (SVM) [154] performs a classification by finding a hyperplane that separates the training data points belonging to two different classes into two different regions of a N-dimensional space. The goal is to optimize this separation with a maximum margin. The classification of a data point is done by checking which side of the hyperplane it lies after its vector has been mapped into the N-dimensional space.

6.4 Experimentation and Results

We now present a description of the experimental results to demonstrate the effectiveness of the various classifiers used for the detection of Sybils. We compare the performance of the classifiers we have chosen, which include KNN, Random Forest, and Adaboost.

In this experiment, we have adopted the following settings. The Random Forests classifier was built using 100 trees. The number of estimators in Adaboost was also set to 100. The values of various performance evaluation metrics are obtained after applying a 10-fold cross validation. In the end, the mean value of each evaluation metric if collected.

The performance evaluation metrics considered in this work are: the Precision,the Recall, the F-measure and the AUC. These metrics are computed using the True Positive (TP), the False Positive (FP), the True Negative (TN), and the False Negative (FN) as illustrated by the confusion matrix in table 6.1.

These metrics are formally and respectively defined by the equations 6.20, 6.21, and 6.22. The Area Under the Curve (AUC) measures how the true positive rate (recall) and the false positive rate trade off.

$$Recall = \frac{TP}{TP + FN} \tag{6.20}$$

$$Precision = \frac{TP}{TP + FP} \tag{6.21}$$

$$F - measure = 2\frac{Precision \times Recall}{Precision + Recall} \tag{6.22}$$

Table 6.1: Example of a confusion matrix

	Predicted Class	
	Sybil	Benign
Sybil	TP	FN
Benign	FP	TN

6.4.1 Features Selection

We started with a set of fourteen features. We need to figure out the best possible features, namely, the ones that improve the classification accuracy and AUC. This is the purpose of features selection models.

Features selection models can be categorized into filter models [155], wrapper models [156] and embedded models [157]. Filter models select a set of features by ranking features based on the characteristics of the data, and without involving any classification algorithm. The wrapper models

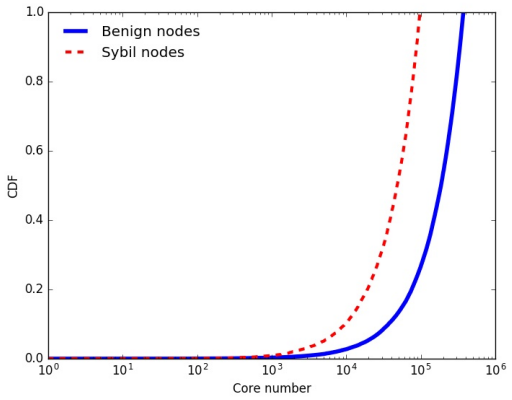


Figure 6.1: Distribution of Core number

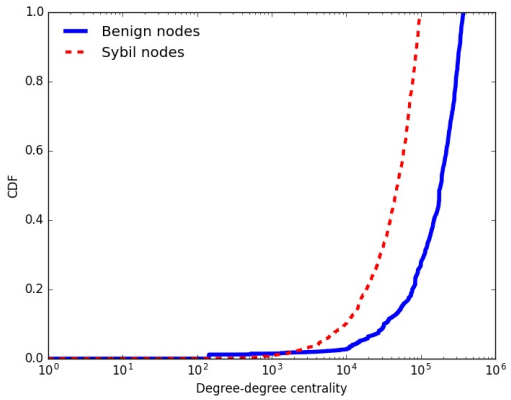


Figure 6.2: Distribution of Degree-degree

involve a classification algorithm in order to select a set of features which are going to improve the performance of the classification algorithm used in the process. The embedded models combine the advantages of the two previous features selection models.

In this work, we have chosen as our feature selection model the Recursive Feature Elimination (RFE) [158]. Since RFE is a wrapper model, we have used it along with a classification algorithm which is SVM [154]. It results from this process that the best features are: The core number, the weighted degree-degree centrality, the weighted core-degree centrality, the degree-coherence centrality, the core-coherence centrality, the edge volume centrality, the average degree centrality, and the average clustering centrality.

After selecting the eight best features, we want to identify how well each of the eight features is able to discriminate benign nodes from malicious nodes. For this purpose, we plot the Cumulative Distribution Function (CDF) of each feature respectively for benign nodes and Sybil nodes. It results from the observation of figures ranging from figure 6.1 to figure 6.8 that, for each feature, the distributions of benign nodes and sybil nodes are fairly distinctive. This explains why the combination of these features is able to provide us with a better classification performance.

6.4.2 Insight Behind the Features

In this section, we present the insight behind the choice of the features we have chosen to perform the classification of nodes into sybils and legitimate nodes. Our features are designed to

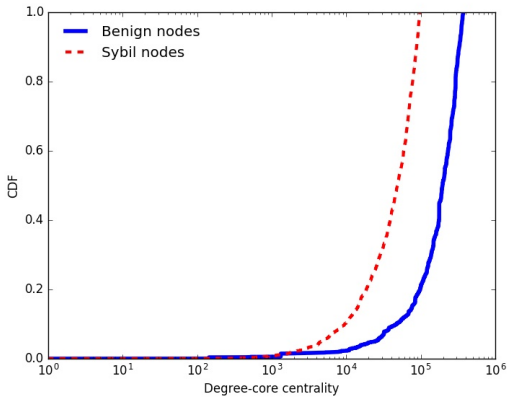


Figure 6.3: Distribution of Degree-core

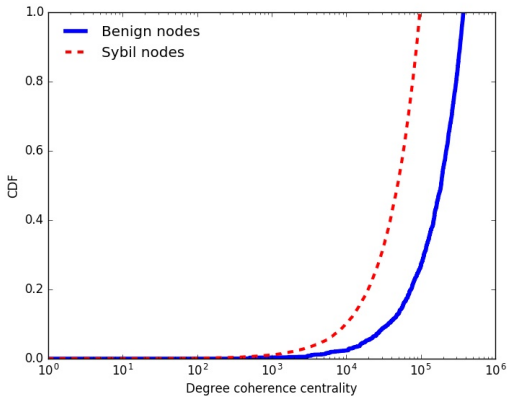


Figure 6.4: Distribution of Degree-coherence

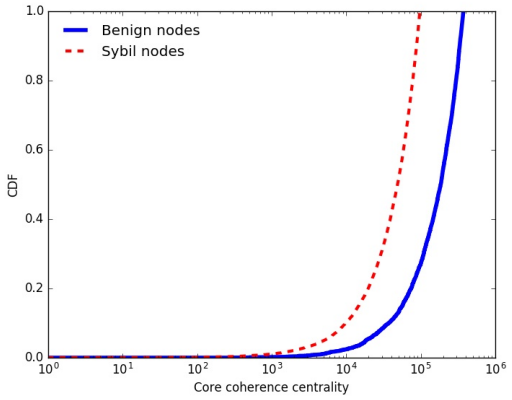


Figure 6.5: Distribution of Core-coherence

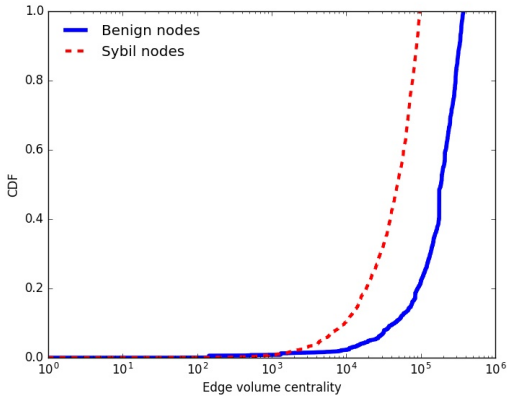


Figure 6.6: Distribution of Edge volume

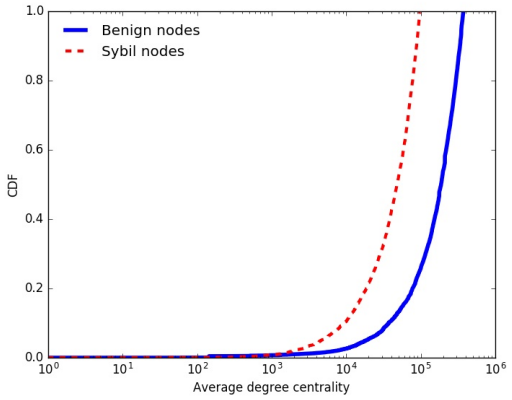


Figure 6.7: Distribution of Average degree

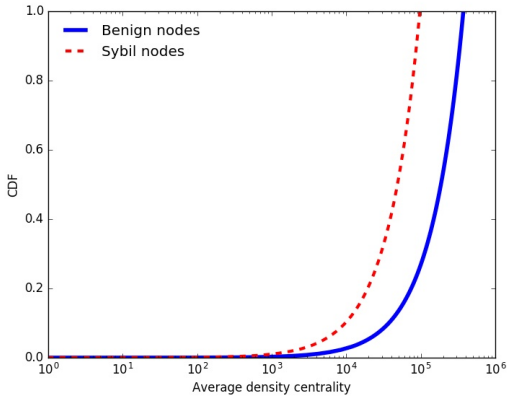


Figure 6.8: Distribution of Average clustering

capture specific topological patterns of sybil accounts that separate them from legitimate accounts. These patterns, as explained in section 6.1.4, are:

- *Sybils that form a dense friendship subgraph which is also said to be a near-clique*;
- *Sybils that form a sparse friendship subgraph resembling a star subgraph*;
- *Sybils tend to have friendship relationships with popular users. Therefore, they have friends with very high degree*.

In the following lines, we explain how the designed features help capture the topological patterns of sybil users.

In a study conducted in [159] and [142] that intended to distinguish fraudsters from normal auctioneers in a OSN, it was observed that fraudsters ofter appeared in the subgraph of the network with core number equal to two. This is the motivation behind using the core number as one of our features.

Since sybils tend to send a lot of friendship requests to legitimate users, the average degree centrality has been selected as one of the features to capture this pattern. However, for those sybils exhibiting a normal friendship subgraph, this feature is not sufficient to distinguish them from legitimate users. Since, this type of sybils tend to connect to popular users, we look at the patterns of their friends. The weighted degree-degree centrality and the the weighted core-degree centrality were designed to detect this type of sybils.

In a social network, the friends of a legitimate user are likely to be other legitimate users. This is not always true for sybil users since they establish friendship relationship in a random manner [144]. In [11], authors have observed that legitimate users have clustering coefficient values larger than sybil users. This is due to the fact that legitimate users tend to have a small number of well connected social cliques while sybil users are more likely to be connected to users with no mutual friendship relationship. For this reason, we have selected the average clustering coefficient as one of our features. In [148], authors have proposed the concept of subgraph intensity and subgraph coherence as an extension of the concept of clustering coefficient. In order to capture

the patterns of the clustering in a node neighborhood and in the neighborhood of that node's friends, we propose the use of degree-coherence centrality and the core-coherence centrality as our next features.

The edge-volume centrality, as explained in section 6.1.5, measures the strength of vertices in terms of the total weight of their connections [146]. In [160], authors have observed that the number of nodes and the number of edges in a node neighborhood follow a power law distribution. They also used this pattern in a scheme designed to detect anomalous nodes in a network since their pattern deviates from the normal one. That is why we have proposed to use the edge-volume centrality as one of our features.

6.4.3 Evaluating using the Best Features

The feature selection process has provided us with eight best features. In this section, we trained our three classifiers using only these eight best features. In order to perform the classification, we generate two dataset, one for the Twitter network and the other for the Facebook network. Each of these dataset has eight columns representing the eight best features. The rows in each dataset represent the nodes of the respective network.

On the Twitter dataset, the results show that Random Forest and KNN turn out to be the classifiers giving the best results with an AUC of 99%. These results are presented in table 6.2. Training the classifiers on the Facebook dataset reveals that all three classifiers perform equally with a prediction reaching a perfect AUC. These results can be seen in table 6.3.

Table 6.2: Classification performance on Twitter dataset

Classifier	Precision	Recall	F-measure	AUC
AdaBoost	0.95	0.94	0.94	0.94
KNN	0.99	0.99	0.99	0.99
Random Forests	0.99	0.99	0.99	0.99

Table 6.3: Classification performance on Facebook dataset

Classifier	Precision	Recall	F-measure	AUC
AdaBoost	1.00	1.00	1.00	1.00
KNN	1.00	1.00	1.00	1.00
Random Forests	1.00	1.00	1.00	1.00

6.4.4 Checking for Over-fitting

In this section, we present all the measures we have taken to avoid the issue of over-fitting our classifiers. We also present the analysis we have conducted to prove that we have not over-fit our classifiers. The problem of over-fitting and under-fitting are related to the concepts of model bias and variance. Bias expresses the ability of a model to approximate the data, while variance represents the stability of a model with regard to new training examples [161]. Over-fitting usually produces classifiers with very high predictive performance.

To prevent over-fitting, we have taken the following measures. During the pre-processing stage, the standardization of the data, was done using a pipe to avoid information leakage from the testing data to the training data. The hyper-parameters tuning for various classifiers was done using a nested k-fold cross-validation technique. Finally, since our dataset is heavily imbalanced toward the benign nodes, we have trained our classifiers after either oversampling the minority class (Sybils) or under-sampling the majority class (benign nodes). The sampling was done using a library called imblearn [162].

After all the measures exposed above, the crucial question is: In regard to the high predictive results we are getting, how would you tell that the model is not over-fitting? Andrew Ng [161] suggests doing two things: Use a held-out test set, and plot the learning curve. The held-out test set is considered as an unseen data. After training the classifiers using k-fold cross-validation, we test the models on the held-out test set to see how well the models generalize to unseen data, and can be helpful in determining how well the models generalize to new data. We run the experiment using the Twitter dataset, and the results show that the AUC of the models on the held-out test data is consistent with the one produced by the k-fold cross-validation.

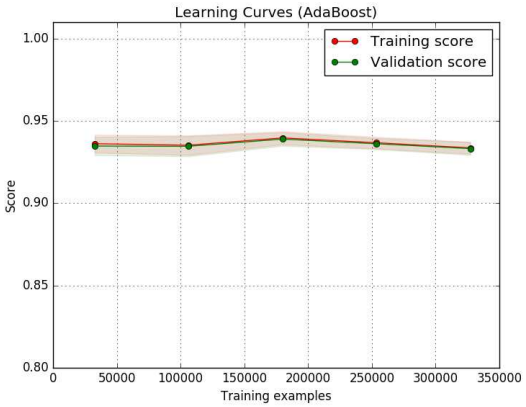


Figure 6.9: Learning curve (AdaBoost)

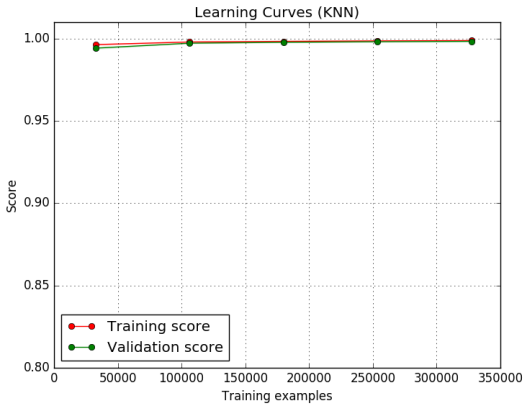


Figure 6.10: Learning curve (KNN)

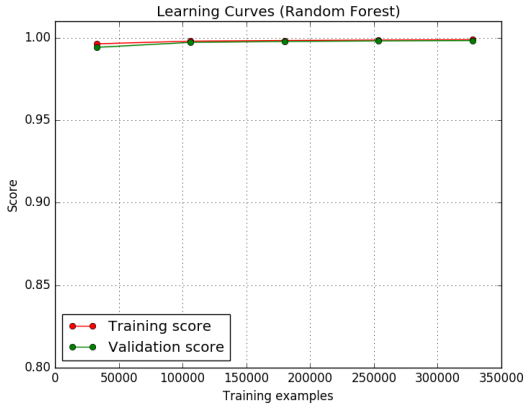


Figure 6.11: Learning curve (RF)

The learning curve is obtained by plotting the training set and the validation set error (or accuracy score) as a function of the training set size. The model is said to over-fit if the learning curve shows a stable gap at the right end of the plot. As shown on figures 6.9, 6.10, and 6.11, we do not have a gap between the training curve and the validation curve in any of our classifiers. Therefore, we can safely say that there is not over-fitting, and that the results are valid.

6.5 Discussion

In this paper, we presented a new framework for detecting Sybil attacks in an OSN. Sybil attacks occur when an adversary is able to create a large number of fake identities in an OSN, which are used to spread spams, malware or just to collect private information. The openness of OSN

platforms make it challenging to prevent such malicious activities from taking place. However, the platform we have designed can be a useful tool for accurately detecting sybil attacks.

Our framework relies on machine learning techniques to classify sybils. Several previously proposed sybil detection techniques also use machine learning techniques. Those previous techniques use user attributes and activities as source of information to build the features needed for the classification. User attributes, collected from the user profile, are often incomplete or contain misleading information. In addition, they can be forged by a sophisticated adversary. On the other hand, user activities are obtained by logging every operation a user performs when using his online account. This collection of activity data raises serious concerns about privacy. Because of these concerns, we have proposed a framework whose classification relies only on features engineered from information collected from the structure of the social graph. Those features present the advantage of being hard to forge by an adversary, unless one has the full knowledge of the entire social graph.

To validate our results, we used two datasets.The first dataset is a real world Twitter dataset with ground truth information. The second dataset is a real world Facebook dataset complemented with some synthetically generated sybil nodes. We have performed the classification using three machine learning techniques which are KNN, Random Forests, and Adaboost. The best result is provided by Random Forest, which is able to predict sybils with an AUC of 99%. This result is consistent with the one obtained from KNN. Adaboost reports the worse prediction with an AUC of 94%.

Chapter 7

Resilient Reference Monitor for Distributed Access Control via Moving Target Defense

Effective access control is dependent not only on the existence of strong policies but also on ensuring that the access control enforcement subsystem is adequately protected. Protecting this subsystem has not been adequately addressed in the literature. In general, it is assumed to be implemented as a reference monitor in a trusted computing base (TCB) that is tamper-proof. However, in distributed access control, ensuring TCB security kernel to be tamper proof is not always feasible. It needs to be implemented in software and on platforms that can potentially have vulnerabilities. We posit that allowing a very limited opportunity to the attacker to enumerate exploitable vulnerabilities in the access control subsystem can considerably facilitate its protection. Towards this end we propose a moving target defense framework for access control in a distributed environment. In this framework, access control is provided by cooperation of several distributed modules that materialize randomly, announce their services, enforce access control and then disappear to be replaced by another module randomly. As a result, the attacker does not know which process can be targeted to compromise the access control system.

The rest of the chapter is organized as follows: In section 7.1 we first present the reference architecture for access control in distributed environments. We then give an overview of our moving target defense approach for protecting the access control subsystem. Section 7.2 presents the moving target defense architecture. We present our implementation in section 7.3 as well as an analysis of the security of the proposed approach. Finally, we conclude this chapter with a discussion in section 7.4.

7.1 Architecture Overview

We assume that the access control model is Role-Based Access Control. We start with a high level operational architecture of access control (AC) in the distributed setup. The resources we are concerned about are the shared resources. The AC architecture is composed of four functional entities. Each entity has specific functions and participates in the communication as a client, as a server or both.

7.1.1 Access Control Architecture Components

The four entities are:

1. *A user's client*: It is an endpoint entity whose main objective is to access protected resources. It is responsible to initiate or terminate a session with the Resource Access Server. It resides on the individual devices running the applications that require access to shared resources.

2. *Resource Access Service(RAS)* : It is an entity that manages the various distributed resources and controls the access to it. It acts as both client and server when receiving or replying to access requests from the client. The decision to grant or deny the access to protected resources is received from the Authorization Control Server. The Resource Access Server is responsible for reinforcing that decision.

3. *Authorization Control Service (ACS)* : It is an endpoint entity that governs the access to each protected resource. It hosts the Access Control engine. The access control engine is based on RBAC model (other models are also possible) and is designed to prevent unauthorized access to protected resources. The policies defining the access to protected resources are also managed by the Authorization Control Service. This service receives any client request and replies with the decision to grant or deny the access to the needed resources.

4. *Discovery Service* : Since the protection of the Authorization Control Service requires this later to be moved to a different location in a non predictable manner, the clients need to

be able to discover the new location of the Authorization Control Service. The Discovery Service provides such capability.

Using these entities, the distributed access enforcement proceeds as follows. When a client needs to access a protected resource, it sends a request to the Authorization Control Service (ACS). The ACS verifies the policy governing the access to the needed resources, and replies with a decision to grant or deny access to the requested resource. If the decision was to grant access, the request is forwarded to the Resource Access Service along with a limited life-time token given to the client allowing it to access that particular resource. The occurrence of an election triggers the Authorization Control Service to be Switched to a different location. In this case, the ACS will register its new location with the Discovery Service. In addition, the client will need to consult the Discovery Service in order to find the new location of the ACS.

7.1.2 Threat Model

In our architecture, we consider access control (AC) as a service and we assume that the Access Control Service can be attacked and compromised. To mitigate the vulnerability of the open network in which an attacker passively listens to various communications, we make all access related communications go over secure channels. This makes the communication secure, but does not protect the endpoints, particularly the Resource Access Server and the Authorization Control Server where the AC engine components reside. We assume that an attacker can masquerade as one of these servers. Alternately, some numbers of these servers can themselves be compromised by malware and behave in a Byzantine manner. An attacker masquerading as a valid server or corrupting a server are treated similarly.

To protect these two entities, we propose the Moving Target Defense strategy. Our motivation for this approach follows from the observation that an attacker needs a reconnaissance window to explore the vulnerabilities in a system before attacking it. The moving target defense strategy reduces this window of opportunity. It requires both the Resource Access Server and the Authorization Control Server to be replicated. At each instance there is only one Resource Access Server

and one Authorization Control Server that is responsible to handle access requests. These are called respectively, RAS leader and ACS leader servers. In addition, both leader servers are periodically replaced by a pair of leader servers randomly chosen by following a Byzantine consensus process executed by the existing candidate RAS and ACS servers. The replacement is achieved through a migration process that relies on a secure service discovery process.

Using moving target defense in this manner to protect the Access Control Engine raises several challenges. Those challenges are as follows.

1. How does one avoid migrating a leader server to a malicious server during the migration process?

2. Which access control component is going to be migrated? And,

3. After the migration process, how does one discover which server is currently providing the services?

In the following sections we are going to address these challenges.

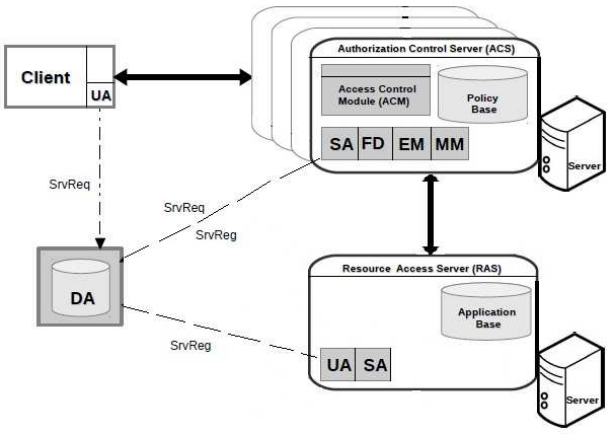


Figure 7.1: Moving Target Defense Architecture

7.2 Distributed Access Control Architecture

We present an architecture that provides access control services. In order to protect the access control service against certain attacks, we have proposed to implement a Moving Target Defense strategy on the access control service architecture. In this section, we present the different components that constitute our Moving Target Defense architecture.

7.2.1 The Client

It is an endpoint entity whose main objective is to access protected resources. It is responsible to initiate or terminate a session with the Resource Access Service. It resides on the individual devices running the applications that require access to shared resources.

7.2.2 The Authorization Control Service

In this section we present the different components that allow the Authorization Control Service to provide access control service, to be elected as a leader, and to announce its services once elected as a leader.

Access Control Engine

The access control engine is based on RBAC model (other models are also possible) and is designed to prevent unauthorized access to protected resources. It comprises the Policy Enforcement Point (PEP), the Policy Decision Point (PDP), the Policy Administration Point (PAP), and the Policy Information Point (PIP).

Fault Detector Module (FD)

The Fault Detector Module is designed to detect the byzantine faults occurring in the server providing the Access Control service. The failure and the compromise of the current leader server are reasons to trigger the Moving Target Defense. The Fault Detector is able to detect any kind of byzantine faults that are local to the leader server. The unavailability of the leader server is detected by the Fault Detector of any other server that probes the aliveness of the leader server.

The occurrence of either failure, compromise, or unavailability is used to trigger the election of a new leader server that will be responsible to provide access control services.

Election Module (EM)

The Election Module is responsible for processing the election of a new leader server. Several causes can trigger this election. Since the leader server is assuming this function for a limited time, called here a term, the expiration of this term is a cause that triggers the election of a new leader server. We add randomness to the duration of this term in order to prevent an attacker from correctly guessing the occurrence of the next leader election. At the expiration of his term, the current leader server proceeds to the election of a new leader. In other circumstances, the first server noticing the failure of the current leader server, is responsible to proceed to a new election.

Leader Election Protocol In our system, after having opted to replicate the Authorization Control Service among several servers, a single server is responsible to provide this service at any given time. We call this server the leader. At any instance, this leader may be the subject of attacks or of failures. Thus to realize the moving target defense, the leader is required to be periodically changed. This change can occur at the expiration of the current leader's lifetime or when the current leader fails. Moreover, the next server is not pre-determined but is elected by existing servers, each of which is a candidate. All this is done in an environment where we assume that some servers may be malicious attackers. Thus, we need a leader election algorithm that can ensure that a malicious server cannot be elected as leader. Once a server is elected, it sets a random lifetime for itself.

For the sake of maintaining our distributed system in a good functioning state, it becomes crucial to prevent faulty nodes from becoming leader. We adapt the algorithm from [163] in order to realize a leader election. The election process proceeds in the form of a distributed protocol as follows.

The election algorithm proceeds in rounds and in each round there is a node that is coordinating the consensus, called the coordinator. For each round r there is a coordinator c known a priori by

each participating node by computing $c \equiv (r \; mod \, n)+1$ with n being the total number of nodes. At each node, there are several local variables that are maintained, among which there is the estimate value which is an input value selected by the node, its current election round, its current coordinator c_p, and a timestamp ts_p. The consensus algorithm runs by exchanging messages between nodes participating in the distributed system. These messages include the types ESTIMATE, SELECT, CONFIRM, READY/NREADY, and SUSPECT. The algorithm runs in a sequence of five tasks that are concurrently executed.

The consensus algorithm, (see Algorithm 3), works as follows. The algorithm starts with each node p picking its estimate of the input value, and sending an ESTIMATE message to all nodes. The coordinator, after receiving $n - k$ ESTIMATE messages that it was waiting for, selects a value es based on all the estimate values received. It then sends a SELECT message carrying the es value to all nodes. Each node p, upon receiving SELECT message from the coordinator, sends a CONFIRM message carrying the es value to all other nodes. The es value should be the same for a given round r. After receiving a CONFIRM message from $\lfloor (n+k)/2 \rfloor + 1$ distinct nodes, each node p updates its local variables. It then sends a READY message or a NREADY message depending on whether it had received the same es value or not from $\lfloor (n + k)/2 \rfloor + 1$ CONFIRM messages. It should be noted that if the CONFIRM messages received by a node p did not contain the same es value, p will assume that the coordinator had deviated from the algorithm. The node p will therefore add the coordinator to its list of Suspect, and will send a SUSPECT message containing the id of the suspected coordinator to all. After a node p received the same es value as content of READY message from $\lfloor (n + k)/2 \rfloor + 1$ distinct nodes, it will decide on that value. A node q is confirmed to be malicious by a node p and added to $Output(D)_p$ if and only if node q have been reported malicious by at least $k + 1$ nodes. $Output(D)_p$ is the final list of malicious nodes. Any round in which the coordinator has not been reported with malicious nodes will end with a consensus on the input value and the coordinator being confirmed as the new leader. Otherwise, a new round will start with a new coordinator.

Algorithm 3 Leader Election

/* Each node p executes the following */
/* Initialization */
$e_p = V_P$ {Chosen value}
$r_p = 0$ {Initial round}
$ts_p = 0$ {Initial timestamps}
$Estimates_p = \emptyset$ {list of estimate msg}
$Confirms_p = \emptyset$ {list of comfirm msg}
$Suspected_p = \emptyset$ {list of suspected nodes}
$Output_p = \emptyset$ {blacklisted nodes}
for r in $listnode$ **do**
    $Suspecting_p[r] = \emptyset$
end for
COBEGIN {Concurrent tasks}
{Task 1:}
while true **do**
    $r_p \leftarrow r_p + 1$
    {Select a Coordinator c_p}
    $c_p \leftarrow (r_p \bmod n) + 1$
    {Task 1, Phase 1 : each node creates estimate msg}
    $estimate_p = (ESTIMATE, p, r_p, e_p, ts_p)$
    Send $estimate_p$ to all
    {Task 1, Phase 2: Coordinator counts received estimate msg}
    if [$p = c_p$] **then**
        [Wait until received $(n-k)$ distinct $estimate_q$ messages from q nodes]
        $Estimates_p \leftarrow estimate_q$
        ts = largest $ts_q : estimate_q \in Estimates_p$
        if [$ts = 0$ and (at least $(k+1)$ distinct $estimate_q \in Estimates_p$ have common value e)] **then**
            $es \leftarrow e$
        else
            $es \leftarrow e_p$
        end if
        {Coordinator creates select msg}
        $select_p = ((SELECT, p, r_p, e_s)_p)$
        Send $select_p$ to all
    end if
    {Task 1, Phase 3 : receiving confirm msg}
    [Wait until received $[(n+k)/2] + 1$ distinct $confirm_q$ messages or $c_p \in Output_p$]
    $confirm_p = ((SELECT, p, r_p, e_s)_p)$
    if [$[(n+k)/2] + 1$ distinct $confirm_q \in Confirms_p$ have common value e] **then**
        $ts_p \leftarrow r_p$
        $e_p \leftarrow e$
        $Confirm_p \leftarrow (CONFIRM, q, r_p, e)_q$
        $ready_p = (READY, p, r_p, e)_p)$
        Send $ready_p$ to all
    else
        $nready_p = (NREADY, p, r_p, e)_p$
        Send $nready_p$ to all
    end if
end while
{Task 2: }
while true **do**
    if [p received $select_p$ msg from $c = (r_p \bmod n) + 1$ and p has not sent $confirm_q$ msg] **then**
        $Selects_p \leftarrow ((SELECT, c, r, e, ts)_c)$
        $confirm_p = (CONFIRM, p, r, e)_p$
        send $confirm_p$ to all
    end if
end while
{Task 3:}
while true **do**
    [Wait until received $[(n+k)/2] + 1$ distinct $ready_q$ messages from q nodes with common r,e]
    decide(e)
end while
{Task 4:}
while true **do**
    {Send list of suspected nodes}
    $Suspected_p \leftarrow D_1$
    $suspect_p = (SUSPECT, p, Suspected_p)_p$
    Send $suspect_p$ to all
end while
{Task 5:}
for r in S **do**
    When p receives $Suspect_q$ from q
    if r in $Suspected_q$ **then**
        $Suspecting_p[r] \leftarrow Suspecting_p[r] \cup (q)$
    else
        $Suspecting_p[r] \leftarrow Suspecting_p[r] - (q)$
    end if
    if $|Suspecting_p[r]| \geqslant k+1$ **then**
        $Output_p = Output_p \cup (r)$
    else
        $Output_p = Output_p - (r)$
    end if
end for

Migration Module (MM)

The Migration Module is responsible for executing the migration protocol. The Migration Module receives a notification from the Election Module that a new leader has been elected. This notification contains the identity of the new leader server. Upon receiving the notification from the Election Module, this module executes the migration protocol, which transfers the Access Control service to the new elected leader server.

Service Migration Protocol In our architecture, replacing the current leader server by a new one necessitates the migration of the Authorization Control Service provided by the current leader server to the new one. For this purpose, we need to put in place a service migration protocol that can handle this task.

Process migration is the movement of a running process from one host to another. A process migration protocol can have several components like the transfer policy, the selection policy, and the location policy. Since we are interested with the migration of an access control service, we define these policies in terms of the requirements of the access control service.

1. *The Transfer Policy:* This policy determines when a host needs to send a process to another host. In our case, it determines when the current leader server needs to send the access control services to a newly elected leader server. In our architecture, this decision is triggered by the successful completion of the leader election protocol.

2. *The Location Policy:* This policy determines the destination host to which to transfer the process to be migrated. In our architecture, this information is provided by the leader election protocol that communicates the identity of the new elected leader server to all the hosts. This new Authorization Control Service is the destination host for the migration protocol.

3. *The Selection Policy:* Determines which resource to transfer. It is question here to determine which component of the access control service needs to be migrated. We have designed the Authorization Control Server to be fully replicated. We assume the existence of a replicated

protocol. Therefore, the discussion about the replication protocol is beyond the scope of this paper. Being fully replicated, each Authorization Control Service has the same PDP, PAP, and PEP. There is no need to migrate those entities. However, only the current leader server has the information about the granted access requests in addition to having the most up to date policy database. Granted access requests information is stored in the session history of the current Authorization Control Service. Therefore, the session history and the policy database need to be migrated to the new Authorization Control Service to allow users with granted access a continuous use of the allowed resources.

7.2.3 The Discovery Service

Implementing a Moving Target Defense for an Access Control service requires switching frequently but randomly the server that is responsible for offering the Access Control service. Once the Authorization Control service has been switched to a newly elected leader server, users need a way to rediscover the new server offering the service. In this section, we adapt the Service Location Protocol or SLP in order to enable users to discover the new location of the services they need.

SLP Module

The SLP Module is responsible for running the service discovery protocol. This service is provided to clients that need to consult the SLP Module in order to discover the new Authorization Control Service location.

SLP overview

Service Location Protocol (SLP) is a protocol designed by the Internet Engineering Task Force to eliminate the need of a manual configuration from users of communication networks in order to discover services, applications, and devices available in those networks. Since users, mainly mobile users, increasingly experience changing environments and the fact that the Internet has became more service oriented, service location is becoming more helpful in today's complex networks [106, 164].

SLP Architecture

The SLP framework includes three main components called "Agents" to process SLP information. These agents are: User Agents (UA), Service Agents (SA), and Directory Agents (DA).

- *User Agents (UA)*: They are responsible for requesting services on behalf of the users or applications.

- *Service Agents (SA)*: They are entities that advertise the location and description of services on behalf of services. To advertise services, the SAs embed the service information into an URL. These information include the IP address, the port number, the service type and the path. Each service type is characterized by specific attributes along with their default values. All of these are specified by the Service Templates.

- *Directory Agents (DA)*: They are central repositories that aggregate SLP information. Since service information are embedded in a URL, these URL are stored by the DA which provides them to any UA that have issued a request which matched some attributes in the URL.

  At the beginning of the protocol, any service provider needs to advertise its services. For that purpose, its SA registers the service with the DA. This step is known as the Service Registration. The DA acknowledges the registration by issuing a service ACK message to the SA. A user that needs to use the given service needs to have his UA issuing a query with the appropriate attributes to the DA. This is known as the Service Request step. The DA may reply back to the UA with the address and characteristics of the desired service (Service Replay). This is the general approach of the SLP protocol as illustrated on fig 7.2.

  There is a major issue with the general approach of the SLP protocol as explained above. No one from both the UA and the SA knows the address of the DA. Before registering a service with the DA, the SA needs to discover the existence of the DA. The same thing applies to the UA.

  Three different methods are used to discover the location of the DA: static, active, and passive. When using the static discovery method, both the UA and the SA learn the address of

the DA from a DHCP server. In case of an active discovery, SLP agents contact the DA by sending service requests to the SLP multicast address on which the DA is configured to listen to for incoming communications. Upon receiving a service request, DA responds directly to the requesting agent via the agent's unicast address. The passive discovery method involves DAs periodically advertising their existence through the SLP multicast address. The other SLP agents discover the DAs location after listening the multicast advertisements. They can then contact the DAs directly through their unicast addresses for other operations [165].

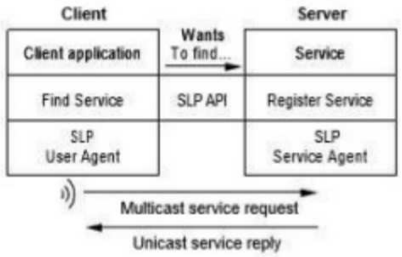


Figure 7.2: Service Discovery flow

Besides the basic SLP architecture involving SAs, DAs, and UA, it is possible to set a SLP architecture without DAs. In this case, UAs and SAs need to communicate directly to each other. In order to discover available services, UAs repeatedly send out their service requests to the SLP multicast address. On the other hand, SAs are listening for incoming requests on the SLP multicast address. Upon receiving a request corresponding to a service they are advertising, SAs reply through unicast address to UAs.

7.2.4 The Resource Access Service

It is a server that manages the various protected resources. It acts as both client and server when receiving or replying to access requests from clients. The decision to grant the access to those resources is received from the Authorization Control Service.

7.3 Implementation

In this section, we introduce the proof-of-concept implementation of our proposed architecture. We have designed a test case to exemplify the functioning of the protocol.

7.3.1 Clients and Resource Access Service

We assume that we have a set of users represented by client applications. Those users are grouped into a set of roles (Undergraduate, Graduate, and Faculty). We also have a set of resources, in this case files stored on a file server. This file server constitutes our Resource Access Service. We have also defined a set of actions to be performed on those files by users. We have chosen basic Linux actions: Read, Write, and Execute. The different permissions given to users over those files are represented on figure 7.3.

Role/Resource	syllabus.txt	research.txt	grades.txt	assignment.txt	certificate.txt
Undergrad	r		r	rwx	r
Graduate	rwx	rwx	r	rx	
Faculty	rwx	rwx	rwx	r	rwx

Figure 7.3: Roles-Permissions assignment

7.3.2 Authorization Control Server

Using socket programming, we have implemented five Authorization Control Services named $ACS1, ACS2, ACS3, ACS4$, and $ACS5$. Those have been implemented as Java client-servers. Each of those Authorization Control Services has an Access Control Module that is responsible for verifying user's authorization to resources that are stored on the Resource server. At each time, only one Authorization Control Service is responsible for providing the access control service.

Access Control

We start our process with the Authorization service being provided by, let us say, the Authorization Control Server ACS1. We consider that user Tom submits a request to read a file named certificate.txt. This request is intercepted by ACS1 which run the Access Control Module. The Ac-

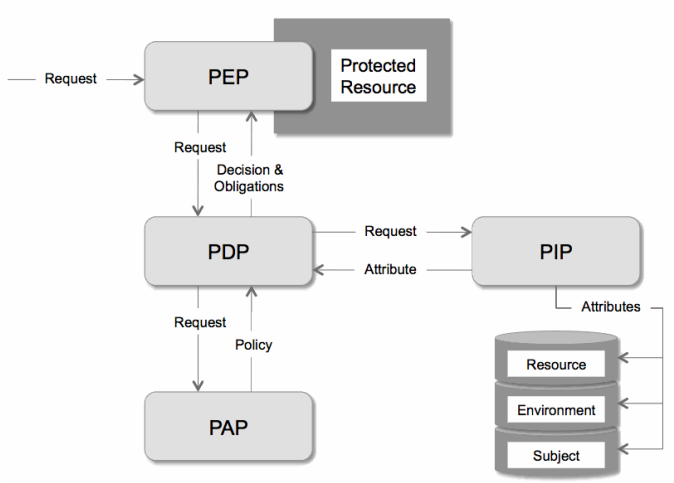


Figure 7.4: XACML architecture

cess Control Module has been implemented using the Balana [166] open source implementation of XACML. Tom's query will be a tuple user-id, action, resource-name where in this particular case user-id is Tom, action is read, and resource-name is the file certificate.txt requested by Tom.

Policy Enforcement Point

Tom's query is intercepted by the Policy Enforcement Point (PEP). In fact, the PEP intercepts all queries sent to the Resource Access Service [167]. We have developed a wrapper that converts the original query into a XACML request. The request is then sent to the PDP for verification. Figure 7.5 illustrates the form of the XACML request.

```
-<Request CombinedDecision="false" ReturnPolicyIdList="true">
  -<Attributes Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource">
    -<Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id" IncludeInResult="false">
       <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">resource</AttributeValue>
     </Attribute>
   </Attributes>
  -<Attributes Category="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject">
    -<Attribute AttributeId="http://wso2.org/claims/role" IncludeInResult="false">
       <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">Role</AttributeValue>
     </Attribute>
   </Attributes>
  -<Attributes Category="urn:oasis:names:tc:xacml:3.0:attribute-category:action">
    -<Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id" IncludeInResult="false">
       <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">operation</AttributeValue>
     </Attribute>
   </Attributes>
 </Request>
```

Figure 7.5: User Request sample

Policy Decision Point

The Policy Decision Point (PDP) receives Tom's request coming from the PEP. It needs to analyse if Tom fulfills the required conditions to read the file certificate.txt. The PDP will consult the Policy file to determine what actions Tom is allowed to perform on the file certificate.txt. Balana [166] provides us with an API call that allows us to create a PDP.

Policy Administration Point

To write policies, we have made use of the Simple Policy Editor. This policy editor is part of WSO2 Identity Server [168]. Simple Policy Editor allows anyone to create XACML 3.0 policies without an extensive knowledge of XACML language. However, an understanding of access control rules is required. Figure 7.6 is a sample of our policy file.

```
−<Rule Effect="Permit" RuleId="Rule-5">
  −<Target>
    −<AnyOf>
      −<AllOf>
        −<Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
            <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">certificate.txt</AttributeValue>
            <AttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id" Category="urn:oasis
          </Match>
        </AllOf>
      </AnyOf>
    </Target>
  −<Condition>
    −<Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-at-least-one-member-of">
      −<Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-bag">
          <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">read</AttributeValue>
          <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">write</AttributeValue>
          <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">execute</AttributeValue>
        </Apply>
        <AttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id" Category="urn:oasis:names:tc
      </Apply>
    </Condition>
  </Rule>
```

Figure 7.6: Policy file sample

In addition to the Policy file, the PDP also consults the user-role assignment table. After determining Tom's role, which is undergraduate, and consulting the Policy file, the PDP reaches the conclusion to authorize Tom to read the desired file. The PDP passes that decision back to the PEP. That response is represented as a XACML file. A sample of the response XACML file is exhibited on figure 7.7.

The PEP then replies to Tom with a response granting him access to the file. Tom can now access the file server.

```
−<Response>
 −<Result>
    <Decision>Deny</Decision>
   −<Status>
      <StatusCode Value="urn:oasis:names:tc:xacml:1.0:status:ok"/>
    </Status>
   −<PolicyIdentifierList>
      <PolicyIdReference>univ_undergrad_policy</PolicyIdReference>
    </PolicyIdentifierList>
  </Result>
</Response>
```

Figure 7.7: Response sample

Leader Election

Our objective is to protect the Access Control Module by regularly switching the Authorization Control Service providing the Access Control service at any given time. For the sake of demonstrating, we have chosen to switch the Authorization Control Service after every 10 minutes plus a random number of seconds. The random time is added to cancel the predictability of the time when the election takes place. An attacker knowing when the new leader is elected can schedule his attack accordingly.

An election is called after the end of term of the current Leader. In this instance, that term is set to ten minutes and some random seconds. The current leader being ACS1, it is the one responsible to call for an election. Using Java, the Leader Election is implemented according to the adaptation of the protocol presented in section 7.2.2. At the end of the protocol a new leader is elected. This leader is different from ACS1, for instance ACS3 has been selected a the new leader. This is the server that is going to be responsible of providing the Authorization Control Service until next election. We made all Authorization Control Services probe the leader after every ninety seconds by sending a IsAlive message to it. This is done in order to detect the failure of the current leader.

Tom want to request another file stored on the file server, but the Authorization Control Service has been moved from ACS1 to ACS3. Any attacker who was in the middle of preparing an attack against ACS1 will be attacking the wrong Authorization Control Service, which is the intended goal of our architecture. However, Tom will also be sending his authorization request to the wrong server.

Migration

We have implemented the Migration Module as a mechanism to simply transfer the session history file and the policy file from the previous leader ACS1 to the elected leader ACS3. As stated in section 7.2.2, the other access control modules are the same across all Authorization Control Services. The reason for transferring ACS1 Policy file to ACS3 is that while ACS1 was providing the Authorization service, policies, resources and users may have been updated. To avoid disruption in the access control service, ACS3 needs to have the most recent policy file.

Other alternatives to this migration can be envisioned. One option is to store the policies in a Policy Database, and replicate the database across all the Authorization Control Services accordingly. Another option would be to migrate the database from the current leader to the new leader at the end of an election. An additional option would be to use a single Policy database that would be shared by all the Authorization Control Services. This last option can create a potential issue by making that single policy database a single point of failure attractive for would be attacker.

7.3.3 Discovery Service

We need to let Tom know that the Authorization Control Server ACS3 has been elected as the new leader, and therefore he should send his request to ACS3. The Discovery Service allows us to achieve that goal through the adaptation of SLP protocol.

We have implemented the Discovery Service using a tool called OpenSLP [169] which is an open source implementation of the Service Location Protocol. OpenSLP can be used either in a three components mode or in a two components mode. In the first mode, we can have a User Agent (UA), a Service Agent (SA) and Directory Agent (DA). The User Agent is the Agent requesting services. The Service Agent is the Agent providing the services, while the Directory Agent is the repository of services. In a two component mode, we can have only the User Agent and the Service Agent. In this case, the Service Agent plays also the role of a Directory Agent (DA). For the sake of this demonstration, we have implemented the later option. We have installed OpenSLP and made sure that $slpd$, which is the OpenSLP daemon, is running.

Service Agent

Since our setting do not use a Directory Agent, the new leader will have to register its services with $slpd$ upon being elected. The old Authorization Control Service, previously registered, is unregistered to avoid confusing users. The new leader registers its access control service by issuing a query in the form of a ServiceURL. The ServiceURL has the following form: *service:ServiceName://IPAddress:PortNo*, where ServiceName is the Authorization Service, IPAddress is the IP address of the new leader, and PortNo is the port where the Authorization Service is running.

User Agent

In our setting, the User Agent is Tom who needs to find the location of the new leader which is providing the access control service. In order to discover the Authorization Service, Tom's client sends a multicast packet with a ServiceURL. The Service Agent, which is the new leader, will verifies if Tom's query matches the registered service. In case of a match, the Service Agent replies to Tom with the ServiceURL informing him how to access the access control service.

7.4 Discussion

In this chapter we have proposed a Moving Target Defense architecture aiming at defending an Access Control Reference Monitor. The design allows a master Resource Access Server and a master Authorization Control Server to be periodically and randomly switched to other ones. This mechanism allows the disruption of any ongoing attack on the Access Control Reference Monitor. This work opens a new direction in research on Moving Target Defense of an Access Control Reference Monitor. This architecture can benefit from some improvements. For instance, we do not believe that the election algorithm is an optimal one in term of computation and the number of messages exchanged during the election process.

Chapter 8

A Secure hash Commitment Approach for MTD

Protection of critical servers is an important requirement in the modern era of distributed systems and services. A centralized server offering security-critical services, like access control or anti-virus filtering, is an attractive target for attackers, as the compromise of a central server will allow attackers unlimited access to the network. In this work, we focus on this important problem of central point of vulnerability and implement an efficient solution based on the *moving "target" defense* strategy, to protect against such active adversaries.Specifically, we focus on the case study of a reference monitor service for access control, i.e., the server provides a reference monitor service to allow or deny user requests for resources based on the access policies. The main intuition in our approach is to increase the attack surface of an attacker by dynamically moving the security service among a group of designated servers. However, for this approach to be effective in practice, the movement of the service should be fast and unpredictable, i.e., the attacker should not have upfront information regarding the next server that might be hosting the service. Towards this, we describe an efficient Byzantine fault-tolerant leader election protocol that achieves the desired security objectives and validate it through theoretical and extensive experimental analysis on prototype implementation of fifty servers. We show that our approach is scalable and can tolerate Byzantine behavior among the nodes while ensuring that a server controlled by an adversary has no greater than a uniform probability for becoming the next active server.

The rest of this chapter is organized as follows. In Section 8.1 we describe the system model for the application domain under consideration as well as the threat model covering the capabilities of the adversary along with the nature of the attack. In Section 8.2 we outline the mechanics of our consensus protocol and describe the leader election protocol. The different phases of our election protocol are explained in Section 8.3. In Section 8.4 we discuss the security of our protocol considering, both theoretical as well as practical attack scenarios. We evaluate our protocol and

present the various results in Section 8.5. We conclude in Section 8.6 with a discussion of our results.

8.1 Background

In this section, we describe the system model for the application domain under consideration, the threat model covering the capabilities of the adversary along with the nature of the attack.

8.1.1 System Model

We consider the scenario where an online server is providing security-critical services, such as access control, to the end users. On the same network, there are other servers that can be chosen to become the point-of-service to the end users. To locate the current point of service, the end user uses a service discovery protocol like SLP [170] and is able to contact the server. We assume that there is a broadcast channel available on the local network and that the nodes are loosely synchronized, which is normal in LAN connected nodes. The pool of servers are considered to be reliable and available for service almost always, i.e., the churn of the network is negligible for all practical purposes. Our approach is to securely move the service from one server to another within a specific system specified time constraint to protect against service tampering attacks.

8.1.2 Threat Model

We consider a generic adaptive adversary who intends to compromise a live server and tamper with the integrity of the service being offered. The adversary stands to gain by the attacks and merely disrupting service is not profitable for the attacker. Now, since it requires the attacker to take complete control of the server, we assume that the attacker requires certain time-window to prepare and complete the attack. Furthermore, we assume that that attacker is in control of at most $\lfloor \frac{n-1}{3} \rfloor$ nodes, as a higher number of nodes puts the attacker nodes in a majority and there is no possibility of fair consensus [80, 171] in such a case. Also, we allow for the scenario where the network administrator detects a compromised server and fortifies it eventually. This ensures that,

on an average, the attacker time-window is always a non-trivial value. If this were not the case then, in the likely event of an attacker eventually compromising all the nodes, there would be no possibility of preventing tampering of the security-critical service.

8.2 One-way Hash Commitment for Secure Consensus

In this section, we outline the mechanics of our consensus protocol and describe the leader election protocol. The general approach for the existing consensus protocols [82, 172, 173] is for the nodes to declare a random value and then agree on the lowest value among those published. But, with this approach, there is a possibility of some nodes waiting for inputs from other nodes before declaring their own values. If the attacker controls a sufficient number of nodes then the attacker can successfully publish the best values that will help in winning the consensus. In the present network model, this problem is further exaggerated as the nodes are connected by a reliable broadcast channel and the attacker has instant access to inputs of all the nodes. To address this problem, we devise a commit and reveal approach, i.e., the nodes first commit a value, say $\mathcal{C}$, to the network and then reveal the value, $\mathcal{V}$. By using a suitable mathematical function, in our case a one-way hash function $\mathbb{H}$ such as SHA-256, to link $\mathcal{C}$ and $\mathcal{V}$, it is difficult for compromised nodes to deny their commitment values. The construction details are as follows.

Let $S_1, S_2, \cdots, S_n$ denote the set of n servers that are available to act as the next point-of-service. Let $\mathbb{H} : \{0,1\}^* \rightarrow \{0,1\}^t$ denote a strong pseudo-random one-way hash function that takes any length input and generates a fixed t-bit output. The main property of the pseudo-random one-way hash function we wish to exploit is that, given an input x it is easy to compute $\mathbb{H}(x)$, but the converse is hard, i.e., given $\mathbb{H}(x)$, it is difficult to obtain x. Let N denote a public parameter that is configured for all the nodes. Let P_i denote a public-value that is used in a particular consensus round i. There is no strict constraint that all the nodes participating in round i need to be aware of the up to date P_i value being used, they are allowed to use some older values of P_i as well. For each subsequent round, a node locally updates the P_i value, which it presently knows, as follows: $P_{i+1} = P_i + 1$. However, if a node is using an older P_i value that has not been incremented beyond

a certain system defined threshold, then the other nodes will consider the inputs of this node as faulty and ignore them. We use the notation P_i^j denote the local view of the P_i for a given node $S_j, \forall 1 \leq j \leq n$.

One-way Hash Commit Value $\mathcal{V}$. Now, in a consensus session i, each node S_j, $\forall 1 \leq j \leq n$, independently chooses a random value R_j and computes a commitment value $\mathcal{V}_j$ as follows: $[\mathbb{H}(R_j, P_i^j)\%N]$. Each node S_j broadcasts its commit value, $\mathcal{V}_j$, and the public value, P_i^j, used in the hash computation, i.e., the tuple $\{\mathcal{V}_j, P_i^j\}$, to the rest of the network. **Reveal Value $\mathcal{R}_j$.** Once the commit phase is complete, each node S_j broadcasts the secret value R_j to the rest of the network. Now, a given node compares all the commit $\mathcal{V}$ values received and verifies the integrity of these values by recomputing the hash values based on the respective random value, R_j, and the public value, P_i^j, pairs. For all the properly matched hash values, a node selects the smallest hash value among all the values and confirms the corresponding publishing node as the winner.

8.3 Leader Election to "Move" Target

The goal of our protocol is to keep *"moving"* the service, which is the attack target, from one server to another and reset the attacker's effort. We start with the assumption that one or more nodes, but no more than $\lfloor \frac{n-1}{3} \rfloor$ nodes, are compromised by the attacker already.

8.3.1 Protocol Overview

The protocol begins with each node selecting a random value independently and computing a one-way hash on the random value. Each node advertises its hash value to all the nodes in the network. We call this phase as the *Commit* phase and it is an implementation of the semantics of the *One-way hash commit* step of the commitment technique from Section 8.2. Once this round completes, each node reveals its chosen random value to the other nodes in the network. We call this phase as the *Estimate* phase and it is an implementation of the *Reveal* step of the commitment technique from Section 8.2.

Now, each node selects the smallest hash value from the set of hash values received from the other nodes, and its own hash value. At the end of the computation, each node broadcasts, the winning node identifier, the list of hash values and the corresponding sender identifiers, to the rest of the network. At the end of this communication round, each node will possibly have all the hash value lists received from the other nodes. Each node performs two steps at this point: confirms the leader of the election and detects faulty nodes. The leader of the election is the node that advertised the smallest value in the *Commit* phase and marked by at least $\lfloor \frac{n-1}{2} \rfloor$ other nodes.

Further, the node performs a fault detection step using the received lists to detect inconsistently advertised values, i.e., two different hash values being advertised by the same node. The inconsistent values are considered to be sent by Byzantine nodes that are compromised by the attacker. All nodes advertising such inconsistent values are put in a *suspect* list and, each node shares its local copy of the suspect list with the rest of the network. At the end of this communication round, each node compares its suspect list with the received suspect lists from other nodes and *detects* nodes that are declared as faulty by a majority, i.e., $\lfloor n - k \rfloor$ nodes where $k = \lfloor \frac{n-1}{3} \rfloor$. At the end of this round, the leader election is complete and the leader provides the service for a given time-period called as the *Election term*.

8.3.2 Protocol Constructs

We describe the building constructs used in our protocol like, cryptographic tools, public parameters, messages, and the time-outs, for different phases of the protocol. **Cryptographic Tools.** We assume that all the nodes are loaded with public-key certificates and digitally sign all messages sent into the network. The digital signatures are essential for ascertaining the identity of a sender and to prevent replay attacks. Also, we use the digital signatures to detect inconsistent messages sent by the same node to different receivers, which is an important step in our fault-detection protocol. For hash computations, we use a cryptographically strong pseudo-random one-way hash function $\mathbb{H}$ such as SHA-256. All protocol communication is in plain-text with digital signatures and no other confidentiality measures are in place.

Public Parameters. All the nodes are initialized with two public-parameters for the *Hash commitment* phase of our protocol.

The first public parameter is a large integer N, which is used for modulo reducing the one-way hash value at each node and is a fixed value throughout. The second public parameter is P. It is a critical component of our protocol and is one of the inputs to $\mathbb{H}$ along with the random value chosen by a node. A possible attack on the hash commitment approach in Section 8.2 is that a node might fix the value of P and brute-force over the space of random numbers to select the best candidates that yield the lowest hash values for several election terms. P plays the role of public randomness in our protocol, which is a difficult property in distributed systems [174]. The key property of this parameter is that it is incremented after each election session to provide the necessary randomness for the one-way hash computations. However, even with this safe-guard, a malicious node might not increment this parameter in order to fix one of the inputs to the one-way hash function and gain an unfair advantage. To prevent this, we define a certain *stale* threshold, which is the maximum difference of a node's copy of P from any other node's copy of P. This threshold reduces the attack surface of an attacker and, the brute-force attack will no longer be a feasible option. In our security analysis, we show that this parameter helps in achieving strong security guarantees in our protocol.

Messages and Formats. There are four messages in our protocol: $COMMIT$, $ESTIMATE$, $CONFIRM$ and $SUSPECT$.

We use NID to denote a node-identifier field, HCV to denote the hash commitment value field, P to denote the public-parameter field, and R to denote the random value field. We use σ_{NIDc} to denote the public-key signature of a message by the sending node $NIDc$ where c means "current sending node". The $COMMIT$ message has two fields, $\{HCV, P\}_{\sigma_{NIDc}}$. The $ESTIMATE$ message contains only one field: $\{R\}_{\sigma_{NIDc}}$. The $CONFIRM$ message is composed of two fields: the node identifier of the node that published the lowest HCV and, the list of all the HCV values and corresponding NIDs received by this node. The format is as follows:
$\{(NID_{min})_{\sigma_{NIDc}}, LIST[(HCV_1, NID_1)_{\sigma_{NID_1}}, \cdots, (HCV_{n-1},\ NID_{n-1})_{\sigma_{NID_{n-1}}}]\}$ where n is the

maximum number of nodes in the system. Finally, the $SUSPECT$ message contains the NIDs of suspected faulty nodes: $\{LIST[(1,\ NID_1), \cdots, (k,\ NID_k)]\}_{\sigma_{NID_c}}$ where k is the maximum number of allowed suspects in the system.

Time-outs. In our protocol, each distinct phase of the protocol is terminated by a properly chosen time-out value: T_p where p indicates the protocol phase[1]. For a particular protocol phase, the time-out value is chosen in such a way that it allows for just enough time for nodes to perform any local computations in this protocol phase and send out their results to the rest of the network. The time-out essentially binds the nodes to perform only necessary computations for the protocol and is an inhibitor for any node that is trying to manipulate the inputs to the protocol in an unfair manner. One important attack scenario where a time-out might prove beneficial is the case of a malicious node waiting for the inputs from all the nodes. The malicious node might try to utilize the best random value from the set of brute-forced values. However, if the node is unable to do so within the time-out value, then it is put in a list of faulty nodes. In our protocol, we choose the time-outs within a Δ-fraction of the estimated round-trip network delay, $RTDelay$, i.e., $T_p = (RTDelay + \Delta * RTDelay + f(n))$ where $0 < \Delta \leq 1$, to ensure that such "wait-and-see" behavior of nodes is detected. Also, f is an implementation dependent time function that accounts for the scale of the network, the network communication facility available, and the computations. Finally, we do not assume that the nodes are tightly synchronized and allow for the staggering of the timeouts across the network.

8.3.3 Commit Phase

At steady state or the end of a previous election term, each node computes a commit value $\mathcal{V}$ independently and sends the digitally signed $COMMIT$ message to the remaining nodes. The node starts a timer T and uses this value to wait for the next protocol phase to start. The node stores the $COMMIT$ values received from other nodes in this phase.

[1]We will drop p when it is clear from context and assume that all protocol phases use the same timeout.

8.3.4 Estimate Phase

At the end of the timeout of the $Commit$ phase, the node sends out an $ESTIMATE$ message containing the random value R used for generating the commit value $\mathcal{V}$ in the $Commit$ phase and starts a new timer T for this phase. Also, the node stores the random values received from other nodes along with their respective $\mathcal{V}$ values.

8.3.5 Confirm Phase

At the completion of the $Estimate$ phase, a node first needs to validate the respective $\{\mathcal{V}, R\}$ pairs from other nodes. Using Algorithm 4, each node verifies the digital signatures on the received messages and then, performs the validation step by checking if: $\{V_j == \mathbb{H}(R_j, P_i^j)\ mod\ N\}$, for the node NID_j in election session i. After validation, each node selects the minimum hash value from among the valid values and returns the NID of the corresponding $\mathcal{V}$.

Algorithm 4 : Minimum Commit Value at NID_c

Input: $I = \{I_1, \cdots, I_n\}$ where $I_m = (V_m, R_m, P_i^m, NID_m)$
Output: NID_i publishing minimum V_i

Let $Valid = \{\phi\}$
for $I_m \in \{1, ..., n\}$ **do**
  if $[V_m == \mathbb{H}(R_m, P_i^m)]$ **then**
    $Valid = Valid \bigcup I_m$
  end if
end for
Let $I_{min} = I_1$,
$\Rightarrow\ NID_{min} = NID_1\ and\ V_{min} = V_1$

for $I_t \in \{1, ..., |Valid|\}$ **do**
  if $V_{min} > V_t$ **then**
    $I_{min} = I_t$
    $V_{min} = V_t$
  end if
end for
Return NID_{min};

Finally, each node sends a $CONFIRM$ message containing the NID of the winning $\mathcal{V}$ and also, includes the list of all the $\mathcal{V}$ values received from other nodes. If each node receives a majority $\lfloor \frac{n-1}{2} \rfloor$ confirm messages asserting the same winning node identifier, then the leader is decided.

8.3.6 Fault-Detection Phase

Algorithm 5 : Fault Detection Algorithm

Input: $L_1, \cdots, L_n$ where $L_m = \{NID_a, \cdots, NID_m\}$
Output: $\{L_{faulty}$ where $\forall NID \in L_{faulty}$ each NID is in at least $|n-k|$ L_is

$L_{faulty} = \{\phi\}$
$k = \lfloor \frac{n-1}{3} \rfloor$
$THRESHOLD = \lfloor n - k \rfloor$

for $L_i \in L_1, ..., n\}$ **do**
  for $NID_j \in L_i$ **do**
    $COUNT = 1$

    if [$(NID_j \in L_p)$ AND $(p \neq i)$ AND $(COUNT \neq THRESHOLD)$] **then**
      $COUNT = COUNT + 1$

      if [$COUNT == THRESHOLD$ AND $NID_j \notin L_{faulty}$] **then**
        $L_{faulty} = L_{faulty} \bigcup NID_j$
        Break;
      end if
    end if
  end for
end for
Return L_{faulty}

Using the list of received hash values in the $Confirm$ phase, each node determines inconsistent messages by misbehaving nodes and creates a $Suspect$ list, which is broadcasted to the rest of the network in a $SUSPECT$ message. Next, each node compares all the received suspect nodes' lists, including its own, and identifies the nodes that are identified as suspect by a threshold majority of the nodes. The threshold majority is given by $\lfloor n - k \rfloor$ where $k = \lfloor \frac{n-1}{3} \rfloor$. The choice of these parameters are from the classic results [80] in the Byzantine fault tolerance domain. All suspect

nodes satisfying the threshold majority are placed in the faulty node list, as shown in Algorithm 5, and their inputs are ignored for a certain number of election sessions until the administrator patches up the server and fortifies them for service again. At the end of this phase, each node sets the time to the election term and waits until the current leader's term is complete.

8.4 Security Analysis

We discuss the security of our protocol considering, both theoretical as well as practical attack scenarios. For theoretical analysis, we consider an *Adaptive Byzantine Adversary* who has the ability to modify the inputs to the protocol by controlling a threshold number of servers, not necessarily the same, in any given session. For practical situations, we consider two kinds of attacks: brute-force and clone attacks.

8.4.1 Adaptive Byzantine Adversary

The definition of an adaptive adversary $\mathcal{A}$ in the context of our protocol is as follows:

Definition 1: An adaptive Byzantine adversary $\mathcal{A}$ is a Byzantine adversary who has access to all the inputs and results of the protocol for a polynomial q number of election sessions, i.e., the attacker knows all the hash commit values, the random values and the winning values. The adversary is also in control of some of the inputs, not more than $k = \lfloor \frac{n-1}{3} \rfloor$, which is required for a safe quorum to be achieved. We denote the adversary storage by: $S = \{S_1,\ S_2, \cdots,\ S_q\}$ where $S_i = \{(HCV_1^i,\ R_1^i,\ P_1^i,\ NID_1^i), \cdots,\ (HCV_n^i, R_n^i,\ NID_n^i)\}$ denotes the i^{th} protocol session's input-output pairs.

ϵ-advantage. The ϵ-advantage is the advantage probability of an adversary to win the election in any election session while being in control of k servers.

In practice, ϵ-advantage should be the uniform probability: $\frac{1}{n}$ and ideally, it should be $\frac{1}{n-k}$.

[ϵ-security]

Assuming the presence of a strong pseudo-random function, our protocol is ϵ-secure against Byzantine adversaries where $\epsilon \leq \frac{1}{k}$ and $k = \lfloor \frac{n-1}{3} \rfloor$, and the $stale$ threshold of the public parameter P is $\leq q$.

Proof. We base our proof on two important assertions and show that the adversary advantage does not increase based on the history of the interactions and the strategy used.

Assertion 1. The protocol is history independent as long as the public parameter P is incremented by the honest majority.

$\{(x_1, \mathbb{H}(x_1)), \cdots, (\ x_q, \mathbb{H}(x_q))\}$ The output of $\mathbb{H}(x_{q+1})$ is independent of all past outputs as per the definition of $\mathbb{H}$. In our protocol, there are two inputs to $\mathbb{H}$: R and P, of which R is random and P is monotonically increasing. Therefore, if an honest majority increment P, and even if the R space is controllable or predictable by the adversary, then the adversary gains no additional advantage from knowing the history of outputs of $\mathbb{H}$. This assertion states that the ϵ-advantage is not cumulative and forces the adversarial nodes to increment P after the q^{th} round to ensure that the adversarial inputs are acceptable. Although this result might seem to render our Byzantine adaptive adversary definition redundant, but our definition captures the real-life behavior of adversaries.

Assertion 2. Within a given session, the adversarial advantage is not increased due to attack strategies used.

There are two attack strategies an adversary might use: (a) Use all k compromised servers to select an input that is better than a particular target server, which the attacker feels might win the election, and (b) Use each of k servers to target all the honest servers with a divide-and-conquer approach, i.e., one compromised server targets a subset of the honest servers. For the first attack strategy, the advantage of the adversary is ϵ for selecting an input better than the target server. However, there are still $n - k - 1$ honest servers that will pick their inputs independently and at random and the probability of a winner from this coalition is $\frac{1}{n-k-1}$. Therefore, the final advantage of the adversary in winning the election becomes $\epsilon \times \frac{1}{n-k-1}$, which is in fact smaller than ϵ. For the second attack strategy, assume that the attacker divides the network into $B = \{B_1, \cdots,\ B_k\}$

where $|B| = \lceil \frac{n-k}{k} \rceil$ coalitions and assigns one server to each coalition B_i. Now, each compromised server has (ϵ/k) advantage of winning the election against a given coalition, B_i.

To compute the cumulative adversary advantage, we observe that, the adversary advantage within each coalition is independent of the results in the other coalitions. Therefore, the adversarial advantage is no better than $(\epsilon/|B|) = \epsilon/(\frac{n-k}{k}) = \epsilon \times \frac{k}{n-k}$, which is less than ϵ. While these attack strategies seem limiting, they are a good baseline for modeling real-world adversaries. Therefore, based on these two assertions, our protocol remains ϵ-secure under the conditions of monotonically increasing P, regardless of the adversarial strategies. □

8.4.2 Brute Force Attack

An adversary might try to win the election by brute-forcing the random number space and keeping the P fixed up to the $stale$ threshold. There are two reasons why this attack is likely to fail. First, while the attacker's servers do not update P, the other honest majority of nodes update P. As a result, the attacker's brute-force on the random numbers will result different $\mathbb{H}$ outputs from the honest majority's $\mathbb{H}$ outputs, even for the same random number choices. Therefore, the likelihood of the attacker's random numbers being the winning choice is as good as the honest majority's random numbers. Second, some of the honest majority do not update their P value due to various reasons like down time or lack of synchrony. Even in this scenario, brute-forcing is unlikely to be successful because it is more than likely that these honest nodes will still have a P that is different from the attacker's servers. Also, the honest nodes will go out of synchrony with respect to P for at most one election round. Once the election starts and then these nodes start receiving $COMMIT$ messages, these nodes will update their P values according to the higher values seen in the $COMMIT$ messages. Therefore, the attacker will have no additional advantage even if some of the honest nodes are not in synchrony with the remaining honest majority. Hence, based on the above two reasons, brute-force attack is not a practical option for an adversary against our protocol.

8.4.3 Clone Attacks

A more feasible attack on our protocol is a message "clone" attack, in other words, an attacker server "waits-and-sees" an honest servers inputs-outputs and "clones" these messages. This attack is as follows: an adversary waits for the commit messages from other servers and records the $\mathcal{V}$ values of other nodes and picks the smallest value. The adversary announces this value to the rest of the network. Now, for the next step, the attacker has two choices to complete the $Estimate$ phase: (a) first, either the attacker brute-forces the random number space and finds a suitable random number matching the $\mathcal{V}$ value it has advertised, or (b), the attacker repeats the "wait-and-see" step to announce the winning random number. The result of this attack is that, in both scenarios, there will be two winning nodes. First, we consider the brute-force scenario, where an attacker brute-forces a suitable random value and show that it can be addressed with properly chosen timeouts. If the attacker is successful then the attacker has essentially followed the protocol steps correctly and can be a winner. However, brute-forcing an input based on the output HF is a time-consuming operation as it involves computing a $\mathbb{H}$ output and reducing it modulo N, till the desired modulo value is found. As a result, there are no guarantees of finding a suitable random value within the timeout of the current protocol phase. Therefore, an attacker is unlikely to choose this option to be successful in the clone attack. Second, the scenario, where the attacker replays the same random value, can be addressed with a tie-breaking protocol in place. The remaining nodes can randomly choose one of the winning nodes as the leader and announce it to the rest of the network. The node that gets the majority votes will be the winner of the election. While this mitigation step is likely to allow an attacker server to be a leader, the probability is still bounded by: $\frac{1}{2(n-k)}$, for the remaining honest majority $n-k$ nodes and assuming that the honest majority chooses one of the nodes uniformly.

Timeouts can prevent this attack as the attacker has to wait for almost all the nodes in the network to send their values. In practice, this waiting time will timeout the attacker and render the cloning attack ineffective or put the attacker in a faulty node as desired. A modified version of this attack, that does not necessarily timeout, is that the attacker clones the input-output pairs of a

particular honest server in the network. But this attack has a far less likelihood of winning as the attacker's winning chances are only as good as the honest server being targeted. However, clone attack still remains one of the limitations of our protocol at present and there is a solution for this through log analysis. The attacker is assumed to be in control of k nodes. Now, if these nodes keep winning the election through the clone attack, a statistical analysis of the past logs will reveal this pattern and all the suspected nodes will be decommissioned and inspected. Also, any attempts by the attacker to remain under the radar of the statistical analysis is likely to reduce the impact of the attacker in the long run. One of our goals is to incorporate the tie-breaking protocol and the log analysis in our future work.

8.5 Performance Evaluation

8.5.1 Experimental Methodology

We implemented a prototype test-bed, shown in Figure 8.1, to measure and validate the performance of our protocol. The various modules are, PEP –Policy Enforcement Point, DSA ?-Digital Signature Algorithm, EM ?-Election Module and FD ?-Fault Detector. The programming language was Java with Apache Maven build environment.

We used fifty HP-Z440-XeonE5-1650v4 servers, running Fedora 25, each with 8 cores, 3.6Ghz clock, and 16GB RAM to act as the nodes communicating over a LAN network with 1Gbps capacity. The nodes communicated in unicast manner, i.e., the broadcast capabilities of the LAN were not utilized due to departmental restrictions. We used Apache Log4j logging service, which uses asynchronous loggers, to log the node activities. We used OpenSLP 2.0.0 implementation to register the leader that is servicing the access control request.

The service location protocol (SLP) is used to publish/broadcast the leader service to all the clients. For the service offered, we used Balana XACML implementation as the Policy Enforcement Point to provide access control service over the network. All nodes were equipped with public-key private-key pairs and used the Elliptic-Curve DSA signature algorithm with 512-bit keys. The secure one-way hash function chosen was SHA-256, the public-parameter was a 512-

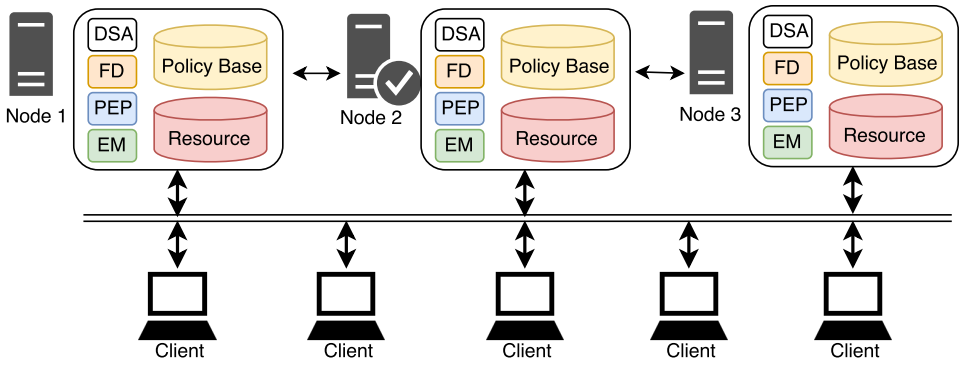


Figure 8.1: Implementation Architecture

bit integer and the modulo reducer N was a 256-bit integer. The configurations of the number of nodes per election term were: $5,\ 10, \cdots,\ 50$, and each experiment was averaged over 50 trials. The election term of server was chosen to be 120 seconds.

8.5.2 Timing Analysis

Our protocol is efficient and is suitable for practical deployment.

The summary of the performance of our protocol is shown in Figure 8.2 and Figure 8.3 .

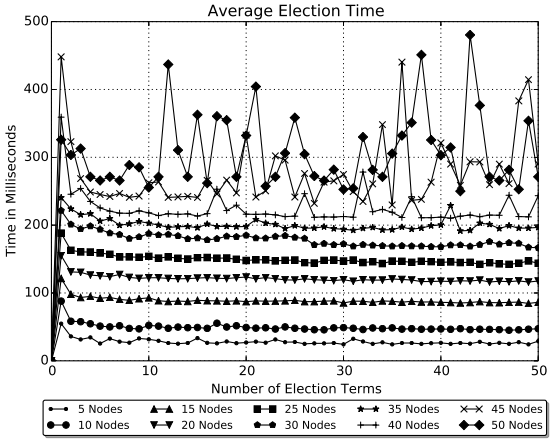


Figure 8.2: Avg. Election Time

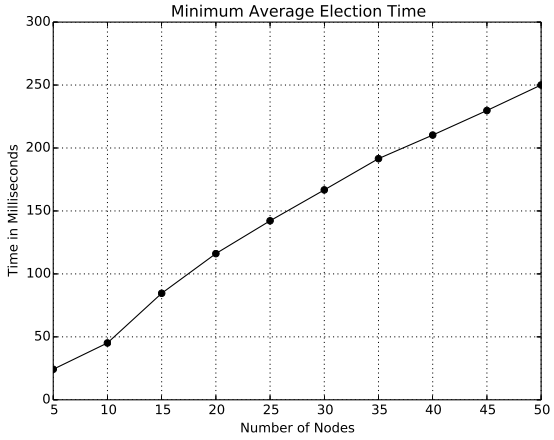


Figure 8.3: Min. Avg. Election Time

In Figure 8.2, we show the average election time for each network configuration, which varies from 10s of milliseconds for 5 nodes up to 400 ms for 50 nodes. In Figure 8.3, we show the minimum average time for leader election, which shows a minimum time of 25 ms for 5 nodes

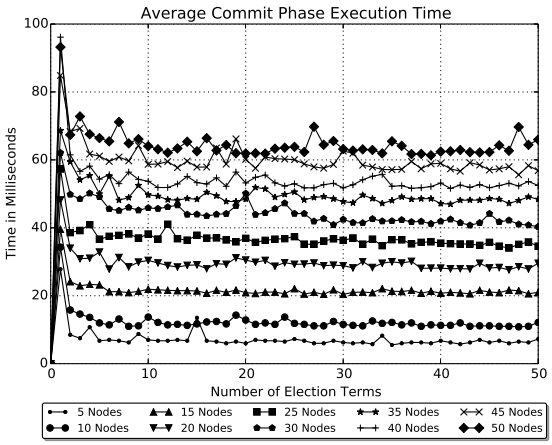


Figure 8.4: Commit

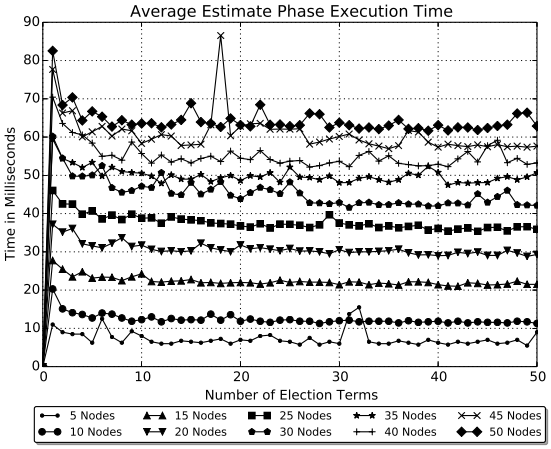


Figure 8.5: Estimate

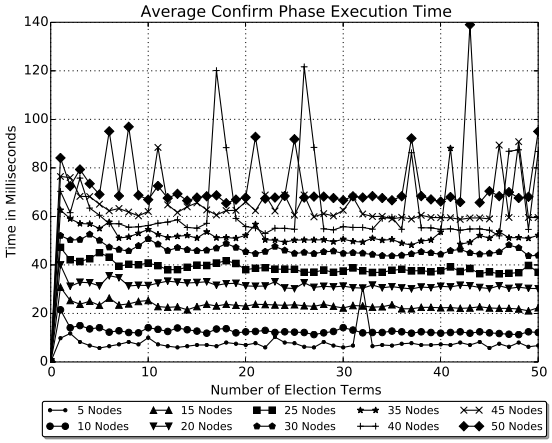


Figure 8.6: Confirm

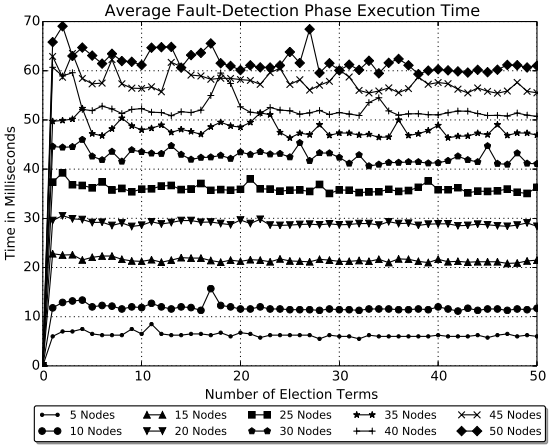


Figure 8.7: Fault-Detection

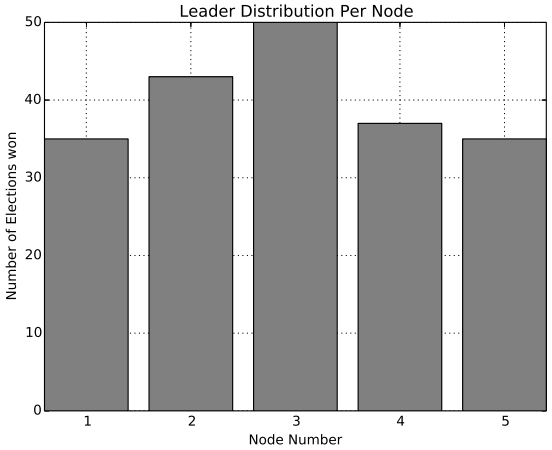


Figure 8.8: Leaders' Quantitative Distr: 5 nodes

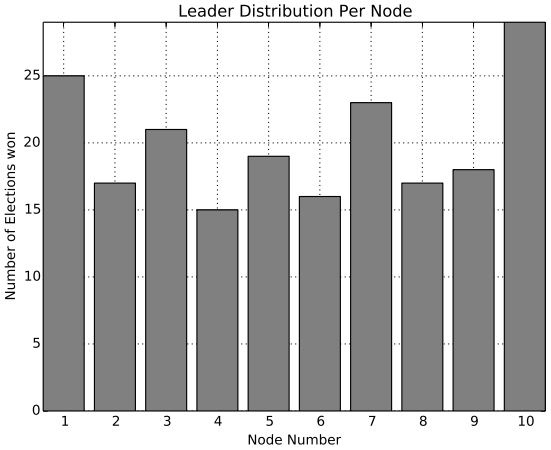


Figure 8.9: Leaders' Quantitative Distr: 10 nodes

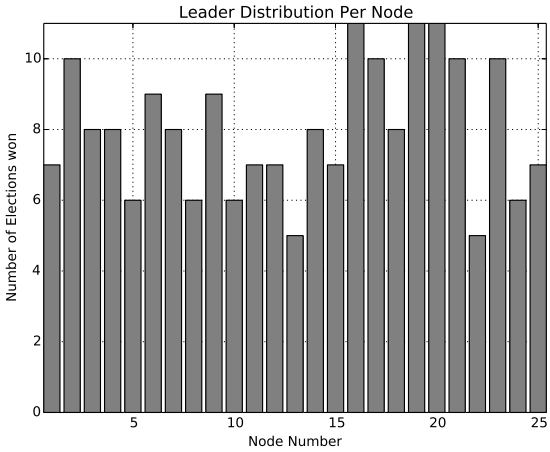


Figure 8.10: Leaders' Quantitative Distr: 25 nodes

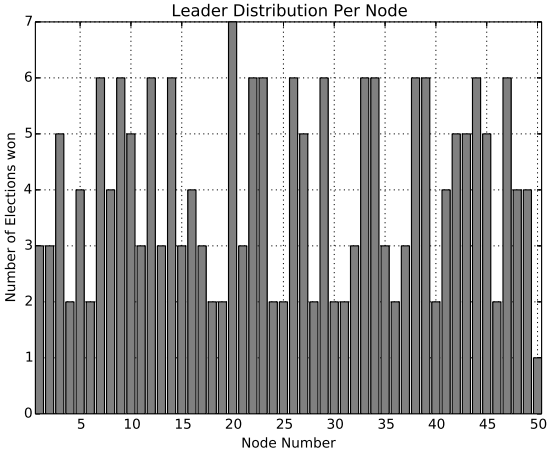


Figure 8.11: Leaders' Quantitative Distr: 50 nodes

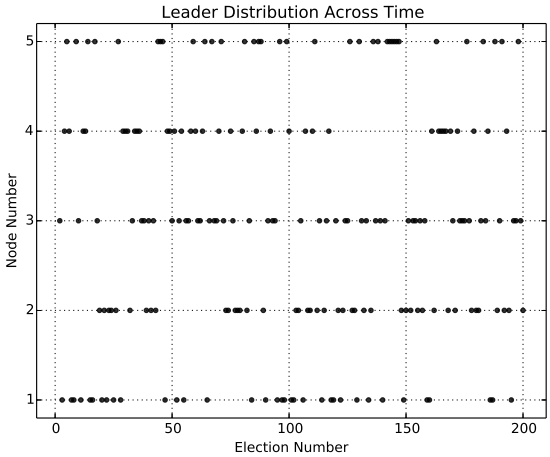


Figure 8.12: Leaders' Temporal Distr: 5 nodes

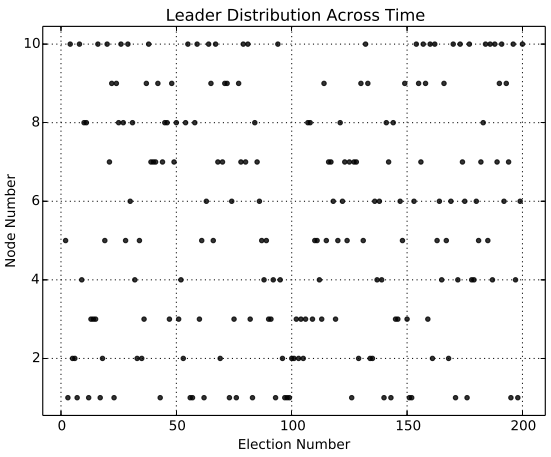


Figure 8.13: Leaders' Temporal Distr: 10 nodes

and about 250 ms for 50 nodes. Given that the election term is about 120 seconds, the protocol execution times constitute only a fraction, $0.008 - 0.3\$$ of the election term, which shows that our protocol is suitable in practice.

From Figure 8.4 to Figure 8.7, we show the execution times of the various phases of our protocol, which include the timeout values, the message transmission times and the computation required for each phase. The $Commit$, $Estimate$ and $Fault - detection$ phases averaged from 10ms up to 70ms for 5 up to 50 server configurations, respectively. However, the $Confirm$ phase required more time as it includes the validation of the committed $\mathcal{V}$ values from various nodes and then select the lowest among the valid values.

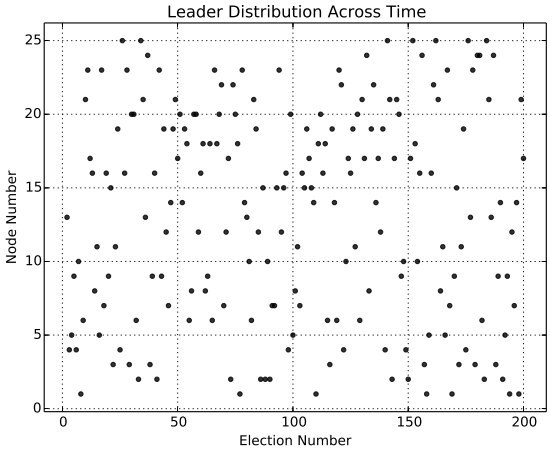


Figure 8.14: Leaders' Temporal Distr: 25 nodes

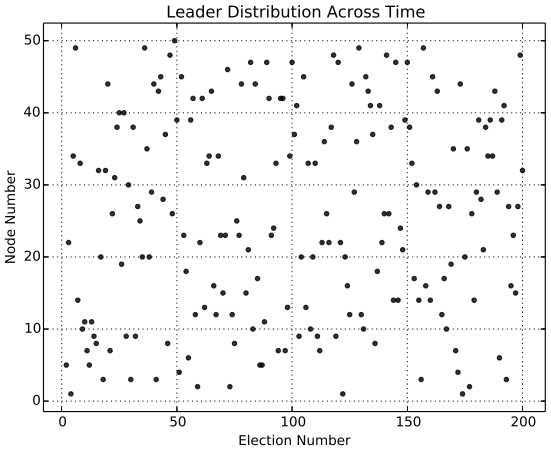


Figure 8.15: Leaders' Temporal Distr: 50 nodes

8.5.3 Leader Distribution

Our protocol exhibits uniformity in leader election and does not give the adversary additional advantage.

From Figure 8.8 to 8.11, we show the number of times a node is elected as leader across 200 election terms for server configuration of $5,\ 10,\ 25$ and $\ 50$ nodes. These results only count the total number of times a node is elected leader and not necessarily consecutively. For instance, for a 5 node network, the distribution in Figure 8.8 clearly shows that any given node is equally likely

to be the leader. Furthermore, for this 5 node network, the observed probabilities of the nodes being elected as leader are: $0.175,\ 0.225,\ 0.4,\ 0.18$ and 0.17, which compare very well with ideal probability of 0.2. For 50 nodes, over 200 election terms, a node has chance of getting elected 4 times 50/200. The graph shows that on an average this is true.

We also considered the temporal distribution of leaders, i.e., are there many instances of the same node being elected as the leader. The scatter plots in Figures 8.12, 8.13, 8.14, and 8.15 demonstrate this distribution, which shows that there are no regular patterns or clusters. This clearly demonstrates that our approach is successful in choosing leader uniformly at random.

8.5.4 Failure Scenarios

Our protocol successfully handled various failure scenarios like leader and nodes failing.

We tested our test-bed by considering various failure scenarios. In the event of a leader failure, we follow a simple policy of "no-service" to the clients. We use this policy due to the communication overhead of mechanisms like "heart-beat" or "alive" messages and also, of the additional complexity introduced by similar methods into our protocol. Also, the short timeouts and leader election times in our protocol make it a feasible option to elect a new leader. The different failure scenarios we considered are: *node failures during protocol phases,* and *leader failures*. For node failure scenarios, the nodes are detected by our fault detector phase and placed in faulty list until they recover and become active. For leader failure scenarios, the server node is also put in faulty node and the election resumed with the remaining active nodes.

8.6 Conclusion

In this chapter, we addressed the problem of securing online servers from concerted attacks by an Byzantine class adversary who is capable of compromising servers and manipulating the inputs to the protocol.We described an efficient moving target defense by which the next point of service node is decided in a uniform way with the help of randomly chosen hash commitment values. The movement of the point of service ensures that the adversary needs to keep looking for newer targets

to compromise on the network.Due to the nature of the pseudo-random one-way hash functions, the adversary has no better advantage of winning the election than any honest node in the network. We have implemented a prototype test bed to test the efficacy of our protocol in realistic workloads for an XACML access control monitor. Our protocol shows that it is possible to have election rounds within a few hundred milliseconds with an election term of 120 seconds for a network of 50 servers. We have shown the security of our protocol using formal as well as practical analysis. Our extensive experimental results show that the average probability of a node being the winner of an election is close to the ideal probability.

Chapter 9

Conclusion

In this work, we investigated the problem of designing large and complex networks that are secure and resilient. We looked at this problem from two different aspects. The first problem we looked at was about detecting anomalous activities in the network, in particular an Online Social Network under sybil attacks. Although this problem has been previously investigated, we addressed it using a different approach. Our approach is guided by the insight that anomalous activities are the result of mal-actors interacting with non mal-actors, and such anomalous activities are reflected in changes to the topological structure (in a mathematical sense) of the network. Using this insight, we have designed two sybil detection mechanisms. The first sybil detection mechanism uses topological properties of the social graph along with a ranking algorithm to accurately detect sybil accounts. The second approach leverages topological properties of the social graph to build relevant features that are fed to machine learning classifiers in order to accurately classify accounts respectively into benign and malicious.

While our Sybil detection algorithms achieve very high levels of accuracy, they cannot guarantee that all Sybils will be detected. Thus, to protect against such "residual" Sybils, the second aspect of our problem was how to build resiliency in a large network that consists of several machines that collectively provide a single service to the outside world. These networks are particularly vulnerable to Sybil attacks. To build resilient networks, we have designed two protocols based on the Moving Target Defense (MTD) paradigm. In this chapter, we present a summary of the results collected from all the experiments conducted as part of this dissertation.

9.1 The Results

The contributions of our dissertation can be described as follows:

- We present a new framework, SybilRadar, for detecting Sybil attacks in an Online Social Network. SybilRadar is an unsupervised approach that belongs to the class of Sybil detection techniques that rely on the graph structure of the OSN. This is in contrast to the alternate group of detection mechanisms that rely of identifying features related to user attributes and activities. We believe that while the second class of detection algorithms may provide good detection results on carefully cleaned up OSN data, in real life such data is difficult to obtain since OSN users frequently leave their profiles incomplete or use misleading information purposefully. Moreover, trying to obtain user activity related data may raise serious privacy concerns. As a result, SybilRadar relies on just the structural properties of the OSN graph. We used a variety of OSN test data - both synthetic as well as real-world – to evaluate the detection accuracy of SybilRadar. We show that SybilRadar performs very well - much better than the most well known similar technique – even for OSNs that have the weak trust model and which have a very large number of attack edges between Sybil nodes and honest nodes.

- We present a new framework for detecting Sybil attacks in an OSN. Our framework relies on machine learning techniques to classify sybils. Several previously proposed sybil detection techniques also use machine learning techniques. Those previous techniques use user attributes and activities as source of information to build the features needed for the classification. User attributes, collected from the user profile, are often incomplete or contain misleading information. In addition, they can be forged by a sophisticated adversary. On the other hand, user activities are obtained by logging every operation a user performs when using his online account. This collection of activity data raises serious concerns about privacy. Because of these concerns, we have proposed a framework whose classification relies only on features engineered from information collected from the structure of the social graph. Those features present the advantage of being hard to forge by an adversary, unless one has the full knowledge of the entire social graph.

To validate our results, we used two datasets. The first dataset is a real world Twitter dataset with ground truth information. The second dataset is a real world Facebook dataset complemented with some synthetically generated sybil nodes. We have performed the classification using three machine learning techniques which are KNN, Random Forests, and Adaboost. We show that the best result is provided by Random Forest, which is able to predict sybils with an AUC of 99%. This result is consistent with the one obtained from KNN. Adaboost reports the worse prediction with an AUC of 94%.

- We propose a Moving Target Defense architecture aiming at defending an Access Control Reference Monitor. The design allows a master Resource Access Server and a master Authorization Control Server to be periodically and randomly switched to other ones. This mechanism allows the disruption of any ongoing attack on the Access Control Reference Monitor. This work opens a new direction in research on Moving Target Defense of an Access Control Reference Monitor.

- We propose an efficient moving target defense by which the next point of service node is decided in a uniform way with the help of randomly chosen hash commitment values. This is done to address the problem of securing online servers from concerted attacks by an Byzantine class adversary who is capable of compromising servers and manipulating the inputs to the protocol. The movement of the point of service ensures that the adversary needs to keep looking for newer targets to compromise on the network. Due to the nature of the pseudo-random one-way hash functions, the adversary has no better advantage of winning the election than any honest node in the network. We have implemented a prototype test bed to test the efficacy of our protocol in realistic workloads for an XACML access control monitor. We show that it is possible to have election rounds within a few hundred milliseconds with an election term of 120 seconds for a network of 50 servers. We have shown the security of our protocol using formal as well as practical analysis. Our extensive experimental results show that the average probability of a node being the winner of an election is close to the ideal probability.

9.2 Future Work

During the design of SybilRadar, we observed an interesting behavior of our algorithm, which we have not yet been able to explain. We observed that as we increased the number of attacks edges, the accuracy degrades a little, which is expected, but that the accuracy fluctuates also. We observed the same behavior with SybilRank which was more dramatic than ours. Immediate future work involves exploring the reason behind this fluctuation. It might give us better insights into OSN behaviors that we had overlooked and guide us towards designing better detection algorithms.

We also plan to add a temporal dimension to our detection framework in the next round. Sybil behavior will most likely not be static but change with time. We expect to see major differences in how structures properties of honest nodes change over time and how that of Sybil nodes change. We would like to investigate how this can be modeled to detect Sybils. Although we are not a big supporter of using user attributes and activities in Sybil detection, we admit that these techniques can provide somewhat better results. We would like to investigate if and how these techniques can be integrated with SybilRadar so as to improve it but in a manner that does not raise any privacy issues related to OSN users.

For the detection framework using machine learning, we would like to perform the classification using a dynamic graph and an online classifier. This is motivated by the fact that the structure of OSNs is dynamic with accounts being added or deleted over time. This will necessitate the design of methods to compute our features on the fly from structural data streamed from the social graph.

Our future work regarding the protection of networked systems using Moving Target Defense is to consider different attack strategies possible and expand the protocol to work for larger networks. Another direction is to consider the possibility of message "clone" attacks in realistic computing scenarios and design safeguards against such attacks. Finally, we would like to explore the usage of the protocol for realistic workloads on different kinds of services besides access control and explore the service specific challenges in such deployments.

Bibliography

[1] Nicole B Ellison et al. Social network sites: Definition, history, and scholarship. *Journal of Computer-Mediated Communication*, 13(1):210–230, 2007.

[2] global social networks ranked by number of users. http://www.statista.com/statistics/272014/global-social-networks-ranked-by-number-of-users/. Accessed: 2015-06-20.

[3] The top 500 sites on the web. http://www.alexa.com/topsites. Accessed: 2015-06-20.

[4] Zhenqiang Gong. Towards Secure and Privacy-Preserving Online Social Networking Services. Technical Report UCB/EECS-2015-76, EECS Department, University of California, Berkeley, May 2015.

[5] Wei Chang and Jie Wu. A survey of sybil attacks in networks. Technical report, Department of Computer and Information Science, Temple University.

[6] Kurt Thomas, Chris Grier, Dawn Song, and Vern Paxson. Suspended accounts in retrospect: an analysis of twitter spam. In *Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference*, pages 243–258. ACM, 2011.

[7] Guanhua Yan, Guanling Chen, Stephan Eidenbenz, and Nan Li. Malware propagation in online social networks: nature, dynamics, and defense implications. In *Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security*, pages 196–206. ACM, 2011.

[8] Yazan Boshmaf, Ildar Muslukhov, Konstantin Beznosov, and Matei Ripeanu. The socialbot network: when bots socialize for fame and money. In *Proceedings of the 27th Annual Computer Security Applications Conference*, pages 93–102. ACM, 2011.

[9] Facebook, quarterly earning reports, april 2015. http://goo.gl/YujtO. Accessed: 2015-06-20.

[10] Lorenzo Alvisi, Allen Clement, Alessandro Epasto, Silvio Lattanzi, and Alessandro Panconesi. Sok: The evolution of sybil defense via social networks. In *Security and Privacy (SP), 2013 IEEE Symposium on*, pages 382–396. IEEE, 2013.

[11] Zhi Yang, Christo Wilson, Xiao Wang, Tingting Gao, Ben Y Zhao, and Yafei Dai. Uncovering social network sybils in the wild. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 8(1):2, 2014.

[12] Saptarshi Ghosh, Bimal Viswanath, Farshad Kooti, Naveen Kumar Sharma, Gautam Korlam, Fabricio Benevenuto, Niloy Ganguly, and Krishna Phani Gummadi. Understanding and combating link farming in the twitter social network. In *Proceedings of the 21st international conference on World Wide Web*, pages 61–70. ACM, 2012.

[13] Matteo Dellamico and Yves Roudier. A measurement of mixing time in social networks, 2009.

[14] Tao Stein, Erdong Chen, and Karan Mangla. Facebook immune system. In *Proceedings of the 4th Workshop on Social Network Systems*, page 8. ACM, 2011.

[15] Marco Carvalho and Richard Ford. Moving-target defenses for computer networks. *IEEE Security & Privacy*, 12(2):73–76, 2014.

[16] M. Satyanarayanan. Pervasive Computing: Visions and Challenges. *IEEE Personal Communications*, 8(4):10–17, August 2001.

[17] J. Anderson. Computer Security Technology Planning Study. Technical Report ESD-TR-73-51, Electronic Systems Division, Hanscom Air Force Base, Hanscom, MA, 1974.

[18] Sypy : Graph-based sybil detection. http://boshmaf.github.io/sypy/. Accessed: 2015-06-24.

[19] Networkx : High-productivity software for complex networks. https://networkx.github.io. Accessed: 2015-06-24.

[20] Qiang Cao, Michael Sirivianos, Xiaowei Yang, and Tiago Pregueiro. Aiding the detection of fake accounts in large scale social online services. In *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, pages 15–15. USENIX Association, 2012.

[21] Edith Cohen, Daniel Delling, Fabian Fuchs, Andrew V Goldberg, Moises Goldszmidt, and Renato F Werneck. Scalable similarity estimation in social networks: Closeness, node labels, and random edge lengths. In *Proceedings of the first ACM conference on Online social networks*, pages 131–142. ACM, 2013.

[22] Xiao Han. *Mining user similarity in online social networks: analysis, modeling and applications*. PhD thesis, Evry, Institut national des télécommunications, 2015.

[23] David Liben-Nowell and Jon Kleinberg. The link-prediction problem for social networks. *Journal of the American society for information science and technology*, 58(7):1019–1031, 2007.

[24] Xu Feng, JC Zhao, and Ke Xu. Link prediction in complex networks: a clustering perspective. *The European Physical Journal B*, 85(1):1–9, 2012.

[25] Jorge Carlos Valverde-Rebaza and Alneu de Andrade Lopes. Link prediction in complex networks based on cluster information. In *Advances in Artificial Intelligence-SBIA 2012*, pages 92–101. Springer, 2012.

[26] Mark EJ Newman and Michelle Girvan. Finding and evaluating community structure in networks. *Physical review E*, 69(2):026113, 2004.

[27] Santo Fortunato and Marc Barthélemy. Resolution limit in community detection. *Proceedings of the National Academy of Sciences*, 104(1):36–41, 2007.

[28] Andrea Lancichinetti and Santo Fortunato. Limits of modularity maximization in community detection. *Physical Review E*, 84(6):066122, 2011.

[29] Andrea Bettinelli, Pierre Hansen, and Leo Liberti. Algorithm for parametric community detection in networks. *Physical Review E*, 86(1):016107, 2012.

[30] Zhenping Li, Shihua Zhang, Rui-Sheng Wang, Xiang-Sun Zhang, and Luonan Chen. Quantitative function for community detection. *Physical review E*, 77(3):036109, 2008.

[31] Andrea Lancichinetti, Santo Fortunato, and János Kertész. Detecting the overlapping and hierarchical community structure in complex networks. *New Journal of Physics*, 11(3):033015, 2009.

[32] Hongyu Gao, Jun Hu, Christo Wilson, Zhichun Li, Yan Chen, and Ben Y Zhao. Detecting and characterizing social spam campaigns. In *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*, pages 35–47. ACM, 2010.

[33] Chris Grier, Kurt Thomas, Vern Paxson, and Michael Zhang. @ spam: the underground on 140 characters or less. In *Proceedings of the 17th ACM conference on Computer and communications security*, pages 27–37. ACM, 2010.

[34] George Danezis and Prateek Mittal. Sybilinfer: Detecting sybil nodes using social networks. In *NDSS*. San Diego, CA, 2009.

[35] Haifeng Yu, Phillip B Gibbons, Michael Kaminsky, and Feng Xiao. Sybillimit: A near-optimal social network defense against sybil attacks. In *Security and Privacy, 2008. SP 2008. IEEE Symposium on*, pages 3–17. IEEE, 2008.

[36] Haifeng Yu, Michael Kaminsky, Phillip B Gibbons, and Abraham D Flaxman. Sybilguard: defending against sybil attacks via social networks. *Networking, IEEE/ACM Transactions on*, 16(3):576–589, 2008.

[37] Abedelaziz Mohaisen, Aaram Yun, and Yongdae Kim. Measuring the mixing time of social graphs. In *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*, pages 383–389. ACM, 2010.

[38] Lu Shi, Shucheng Yu, Wenjing Lou, and Y Thomas Hou. Sybilshield: An agent-aided social network-based sybil defense among multiple communities. In *INFOCOM, 2013 Proceedings IEEE*, pages 1034–1042. IEEE, 2013.

[39] Yazan Boshmaf, Dionysios Logothetis, Georgos Siganos, Jorge Lería, Jose Lorenzo, Matei Ripeanu, and Konstantin Beznosov. Integro: Leveraging victim prediction for robust fake account detection in osns. In *Proc. of NDSS*, 2015.

[40] Peng Gao, Neil Zhenqiang Gong, Sanjeev Kulkarni, Kurt Thomas, and Prateek Mittal. Sybilframe: A defense-in-depth framework for structure-based sybil detection. *arXiv preprint arXiv:1503.02985*, 2015.

[41] Alex Hai Wang. Don't follow me: Spam detection in twitter. In *Security and Cryptography (SECRYPT), Proceedings of the 2010 International Conference on*, pages 1–10. IEEE, 2010.

[42] Gianluca Stringhini, Christopher Kruegel, and Giovanni Vigna. Detecting spammers on social networks. In *Proceedings of the 26th Annual Computer Security Applications Conference*, pages 1–9. ACM, 2010.

[43] Fabricio Benevenuto, Gabriel Magno, Tiago Rodrigues, and Virgilio Almeida. Detecting spammers on twitter. In *Proceedings of the Collaboration, electronic messaging, anti-abuse and spam conference (CEAS)*, volume 6, 2010.

[44] Gang Wang, Tristan Konolige, Christo Wilson, Xiao Wang, Haitao Zheng, and Ben Y Zhao. You are how you click: Clickstream analysis for sybil detection. In *Proceedings of the USENIX Security Symposium*, volume 9, 2013.

[45] Grace Gee and Hakson Teh. Twitter spammer profile detection, 2010.

[46] Stefano Cresci, Roberto Di Pietro, Marinella Petrocchi, Angelo Spognardi, and Maurizio Tesconi. Fame for sale: efficient detection of fake twitter followers. *Decision Support Systems*, 80:56–71, 2015.

[47] Pavel Laskov et al. Practical evasion of a learning-based classifier: A case study. In *Proceedings of the 2014 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2014.

[48] Kyumin Lee, James Caverlee, and Steve Webb. Uncovering social spammers: social honeypots+ machine learning. In *Proceedings of the 33rd international ACM SIGIR conference on Research and development in information retrieval*. ACM, 2010.

[49] Mansour Alsaleh, Abdulrahman Alarifi, Abdul Malik Al-Salman, Mohammed Alfayez, and Abdulmajeed Almuhaysin. Tsd: Detecting sybil accounts in twitter. In *Proceedings of the 13th International Conference on Machine Learning and Applications (ICMLA)*. IEEE, 2014.

[50] Dieudonne Mulamba, Indrajit Ray, and Indrakshi Ray. Sybilradar: A graph-structure based framework for sybil detection in on-line social networks. In *Proceedings of the IFIP International Information Security and Privacy Conference*. Springer, 2016.

[51] Luke Rodriguez, Darren Curtis, Sutanay Choudhury, Kiri Oler, Peter Nordquist, Pin-Yu Chen, and Indrajit Ray. Action recommendation for cyber resilience. In *Proc. of the 22nd ACM CCS*, pages 1620–1622. ACM, 2015.

[52] Neal Wagner, Cem Ş Şahin, Michael Winterrose, James Riordan, Jaime Pena, Diana Hanson, and William W Streilein. Towards automated cyber decision support: A case study on network segmentation for security. In *Computational Intelligence (SSCI), IEEE Sym. Series on*, pages 1–10. IEEE, 2016.

[53] Shangguang Wang, Ao Zhou, Mingzhe Yang, Lei Sun, Ching-Hsien Hsu, et al. Service composition in cyber-physical-social systems. *IEEE Transactions on Emerging Topics in Computing*, 2017.

[54] Michael L Winterrose, Kevin M Carter, Neal Wagner, and William W Streilein. Adaptive attacker strategy development against moving target cyber defenses. *arXiv preprint arXiv:1407.8540*, 2014.

[55] Stefan Achleitner, Thomas La Porta, Patrick McDaniel, Shridatt Sugrim, Srikanth V Krishnamurthy, and Ritu Chadha. Cyber deception: Virtual networks to defend insider reconnaissance. In *Int. Workshop on Managing Insider Security Threats*, pages 57–68. ACM, 2016.

[56] Pradeep Ramuhalli, Mahantesh Halappanavar, Jamie Coble, and Mukul Dixit. Towards a theory of autonomous reconstitution of compromised cyber-systems. In *Technologies for Homeland Security (HST),IEEE Intl. Conf.*, pages 577–583, 2013.

[57] Ravi S Sandhu and Pierangela Samarati. Access control: principle and practice. *IEEE communications magazine*, 32(9):40–48, 1994.

[58] Mr Dennis M Gilbert. An examination of federal and commercial access control policy needs. In *National Computer Security Conference, 1993 (16th) Proceedings: Information Systems Security: User Choices*, page 107. DIANE Publishing, 1995.

[59] Ravi S Sandhu, Edward J Coyne, Hal L Feinstein, and Charles E Youman. Role-based access control models. *Computer*, 29(2):38–47, 1996.

[60] Imtiaz Mohammed and David M Dilts. Design for dynamic user-role-based security. *Computers & Security*, 13(8):661–671, 1994.

[61] David Ferraiolo, Janet Cugini, and D Richard Kuhn. Role-based access control (rbac): Features and motivations. In *Proceedings of 11th annual computer security application conference*, pages 241–48, 1995.

[62] Jun Ho Huh, Rakesh B Bobba, Tom Markham, David M Nicol, Julie Hull, Alex Chernoguzov, Himanshu Khurana, Kevin Staggs, and Jingwei Huang. Next-generation access control for distributed control systems. *IEEE Internet Computing*, 20(5):28–37, 2016.

[63] Sushmita Ruj, Milos Stojmenovic, and Amiya Nayak. Decentralized access control with anonymous authentication of data stored in clouds. *IEEE transactions on parallel and distributed systems*, 25(2):384–394, 2014.

[64] Ana Ferreira, David Chadwick, Pedro Farinha, Ricardo Correia, Gansen Zao, Rui Chilro, and Luis Antunes. How to securely break into rbac: the btg-rbac model. In *Computer Security Applications Conference, 2009. ACSAC'09. Annual*, pages 23–31. IEEE, 2009.

[65] Lester J Fraim. Scomp: A solution to the multilevel security problem. *Computer*, (7):26–34, 1983.

[66] Roger Schell, Tien F Tao, and Mark Heckman. Designing the gemsos security kernel for security and performance. In *Proceedings of the 8th National Computer Security Conference*, volume 30, pages 108–119, 1985.

[67] ORACLE. Trusted Solaris Operating System. http://www.oracle.com/technetwork/server-storage/solaris/overview/index-136311.html.

[68] C. Wright, C. Cowan, S. Smalley, J. Morris, and G. Kroah-Hartman. Linux Security Modules: General Security Support for the Linux Kernel. In *Proceedings of the 11th USENIX Security Symposium*, San Francisco, CA, 2002.

[69] The TrustedBSD Project. Trustedbsd. http://www.trustedbsd.org.

[70] dhs.gov. U.S. Homeland Security Cyber Security R&D Center: Moving Target Defense (MTD) program. https://www.dhs.gov/science-and-technology/csd-mtd, 2017.

[71] Rui Zhuang, Scott A DeLoach, and Xinming Ou. Towards a theory of moving target defense. In *Proceedings of the First ACM Workshop on Moving Target Defense*, pages 31–40. ACM, 2014.

[72] David Evans, Anh Nguyen-Tuong, and John Knight. Effectiveness of moving target defenses. In *Moving Target Defense*, pages 29–48. Springer, 2011.

[73] Yujuan Han, Wenlian Lu, and Shouhuai Xu. Characterizing the power of moving target defense via cyber epidemic dynamics. In *Proceedings of the 2014 Symposium and Bootcamp on the Science of Security*, page 10. ACM, 2014.

[74] Matthew D Compton. Improving the quality of service and security of military networks with a network tasking order process. 2010.

[75] Ehab Al-Shaer. Toward network configuration randomization for moving target defense. In *Moving Target Defense*, pages 153–159. Springer, 2011.

[76] Spyros Antonatos, Periklis Akritidis, Evangelos P Markatos, and Kostas G Anagnostakis. Defending against hitlist worms using network address space randomization. *Computer Networks*, 51(12):3471–3490, 2007.

[77] Noor O Ahmed and Bharat Bhargava. Mayflies: A moving target defense framework for distributed systems. In *ACM Workshop on Moving Target Defense*, pages 59–64. ACM, 2016.

[78] Jun Xu, Pinyao Guo, Mingyi Zhao, Robert F Erbacher, Minghui Zhu, and Peng Liu. Comparing different moving target defense techniques. In *Proceedings of the First ACM Workshop on Moving Target Defense*, pages 97–107. ACM, 2014.

[79] Sisi Duan, Yun Li, and Karl Levitt. Cost sensitive moving target consensus. In *Network Computing and Applications (NCA), IEEE 15th Int. Sympo. on*, pages 272–281. IEEE, 2016.

[80] Leslie Lamport, Robert E. Shostak, and Marshall C. Pease. The byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, 1982.

[81] Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 4(3):382–401, 1982.

[82] Miguel Castro, Barbara Liskov, et al. Practical byzantine fault tolerance. In *OSDI*, volume 99, pages 173–186, 1999.

[83] Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance and proactive recovery. *ACM Transactions on Computer Systems (TOCS)*, 20(4):398–461, 2002.

[84] J-P Martin and Lorenzo Alvisi. Fast byzantine consensus. *IEEE Transactions on Dependable and Secure Computing*, 3(3):202–215, 2006.

[85] Christopher Copeland and Hongxia Zhong. Tangaroa: a byzantine fault tolerant raft.

[86] Diego Ongaro and John Ousterhout. Raft consensus algorithm, 2015.

[87] Andrew Miller, Yu Xia, Kyle Croman, Elaine Shi, and Dawn Song. The honey badger of bft protocols. In *Proc. of ACM CCS*, pages 31–42. ACM, 2016.

[88] Leslie Lamport. The part-time parliament. *ACM Transactions on Computer Systems (TOCS)*, 16(2):133–169, 1998.

[89] A Arghavani, E Ahmadi, and AT Haghighat. Improved bully election algorithm in distributed systems. In *Information Technology and Multimedia (ICIM), 2011 International Conference on*, pages 1–6. IEEE, 2011.

[90] P Beaulah Soundarabai, J Thriveni, HC Manjunatha, KR Venugopal, and LM Patnaik. Message efficient ring leader election in distributed systems. In *Computer Networks & Communications (NetCom)*, pages 835–843. Springer, 2013.

[91] Leslie Lamport et al. Paxos made simple. *ACM Sigact News*, 32(4):18–25, 2001.

[92] Heidi Howard, Malte Schwarzkopf, Anil Madhavapeddy, and Jon Crowcroft. Raft refloated: do we have consensus? *ACM SIGOPS Operating Systems Review*, 49(1):12–21, 2015.

[93] Stavros Nikolaou and Robbert Van Renesse. Turtle consensus: Moving target defense for consensus. In *Proc. of the 16th Annual Middleware Conference*, pages 185–196. ACM, 2015.

[94] Patrick Hunt, Mahadev Konar, Flavio Paiva Junqueira, and Benjamin Reed. Zookeeper: Wait-free coordination for internet-scale systems. In *USENIX annual technical conference*, volume 8, page 9, 2010.

[95] Mike Burrows. The chubby lock service for loosely-coupled distributed systems. In *Proceedings of the 7th symposium on Operating systems design and implementation*, pages 335–350. USENIX Association, 2006.

[96] Xuanhua Shi, Haohong Lin, Hai Jin, Bing Bing Zhou, Zuoning Yin, Sheng Di, and Song Wu. Giraffe: A scalable distributed coordination service for large-scale systems. In *Cluster Computing (CLUSTER), 2014 IEEE International Conference on*, pages 38–47. IEEE, 2014.

[97] Guozhang Wang, Joel Koshy, Sriram Subramanian, Kartik Paramasivam, Mammad Zadeh, Neha Narkhede, Jun Rao, Jay Kreps, and Joe Stein. Building a replicated logging system with apache kafka. *Proceedings of the VLDB Endowment*, 8(12):1654–1655, 2015.

[98] Valerie King, Jared Saia, Vishal Sanwalani, and Erik Vee. Scalable leader election. In *Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*, pages 990–999. Society for Industrial and Applied Mathematics, 2006.

[99] Hector Garcia-Molina. Elections in a distributed computing system. *IEEE Trans. Computers*, 31(1):48–59, 1982.

[100] Gérard Le Lann. Distributed systems-towards a formal approach. In *IFIP Congress*, volume 7, pages 155–160. Toronto, 1977.

[101] Hosame Abu-Amara and Jahnavi Lokre. Election in asynchronous complete networks with intermittent link failures. *IEEE Transactions on Computers*, 43(7):778–788, 1994.

[102] Hasan M Sayeed, Marwan Abu-Amara, and Hosame Abu-Amara. Optimal asynchronous agreement and leader election algorithm for complete networks with byzantine faulty links. *Distributed Computing*, 9(3):147–156, 1995.

[103] Jacob Brunekreef, Joost-Pieter Katoen, Ron Koymans, and Sjouke Mauw. Design and analysis of dynamic leader election protocols in broadcast networks. *Distributed Computing*, 9(4):157, 1996.

[104] Gurdip Singh. Leader election in the presence of link failures. *IEEE Transactions on Parallel and Distributed Systems*, 7(3):231–236, 1996.

[105] Charles E Perkins et al. Dhcp options for service location protocol. 1999.

[106] Erik Guttman. Service location protocol: Automatic discovery of ip network services. *IEEE Internet Computing*, 3(4):71–80, 1999.

[107] Alan Presser, Lee Farrell, Devon Kemp, and W Lupton. Upnp device architecture 1.1. In *UPnP Forum*, volume 22, 2008.

[108] Apple Inc. Bonjour. https://support.apple.com/bonjour. Accessed: 2017-02-26.

[109] Jae Woo Lee, Henning Schulzrinne, Wolfgang Kellerer, and Zoran Despotovic. z2z: Discovering zeroconf services beyond local link. In *Globecom Workshops, 2007 IEEE*, pages 1–7. IEEE, 2007.

[110] Nesreen Ahmed, Jennifer Neville, and Ramana Rao Kompella. Network sampling via edge-based node selection with graph induction. 2011.

[111] Jennifer Golbeck. Trust and nuanced profile similarity in online social networks. *ACM Transactions on the Web (TWEB)*, 3(4):12, 2009.

[112] Shishir Nagaraja. Anonymity in the wild: Mixes on unstructured networks. In *Privacy Enhancing Technologies*, pages 254–271. Springer, 2007.

[113] Xinxin Zhao, Lingjun Li, and Guoliang Xue. Authenticating strangers in fast mixing online social networks. In *Global Telecommunications Conference (GLOBECOM 2011), 2011 IEEE*, pages 1–5. IEEE, 2011.

[114] Lada Adamic and Eytan Adar. Friends and Neighbors on the Web. *Social Network*, 25:211–230, 2001.

[115] Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2008(10):P10008, 2008.

[116] Indika Kahanda and Jennifer Neville. Using transactional information to predict link strength in online social networks. *ICWSM*, 9:74–81, 2009.

[117] Ryan N Lichtenwalter, Jake T Lussier, and Nitesh V Chawla. New perspectives and methods in link prediction. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 243–252. ACM, 2010.

[118] FY Yao and L Chen. Similarity propagation based link prediction in bipartite networks. In *Network Security and Communication Engineering: Proceedings of the 2014 International Conference on Network Security and Communication Engineering (NSCE 2014), Hong Kong, December 25–26, 2014*, page 295. CRC Press, 2015.

[119] Michael Fire, Lena Tenenboim-Chekina, Rami Puzis, Ofrit Lesser, Lior Rokach, and Yuval Elovici. Computationally efficient link prediction in a variety of social networks. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 5(1):10, 2013.

[120] Seymour Geisser. *Predictive inference*, volume 55. CRC Press, 1993.

[121] Yuzhou Zhang, Jianyong Wang, Yi Wang, and Lizhu Zhou. Parallel community detection on large networks with propinquity dynamics. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 997–1006. ACM, 2009.

[122] Jaewon Yang and Jure Leskovec. Overlapping community detection at scale: a nonnegative matrix factorization approach. In *Proceedings of the sixth ACM international conference on Web search and data mining*, pages 587–596. ACM, 2013.

[123] Jierui Xie and Boleslaw K Szymanski. Towards linear time overlapping community detection in social networks. In *Advances in Knowledge Discovery and Data Mining*, pages 25–36. Springer, 2012.

[124] Haifeng Yu. Sybil defenses via social networks: a tutorial and survey. *ACM SIGACT News*, 42(3):80–101, 2011.

[125] E Behrends. *Introduction to Markov Chains with Special Emphasis on Rapid Mixing. 2000*. Advanced Lectures in Mathematics. Springer, 2000.

[126] Jure Leskovec, Kevin J Lang, Anirban Dasgupta, and Michael W Mahoney. Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters. *Internet Mathematics*, 6(1):29–123, 2009.

[127] Gene H Golub and Henk A Van der Vorst. Eigenvalue computation in the 20th century. *Journal of Computational and Applied Mathematics*, 123(1):35–65, 2000.

[128] George Karypis and Vipin Kumar. Parallel multilevel series k-way partitioning scheme for irregular graphs. *Siam Review*, 41(2):278–300, 1999.

[129] Daniel A Spielman and Shang-Hua Teng. Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. In *Proceedings of the thirty-sixth annual ACM symposium on Theory of computing*, pages 81–90. ACM, 2004.

[130] Jure Leskovec and Julian J Mcauley. Learning to discover social circles in ego networks. In *Proceedings of the Advances in neural information processing systems*, 2012.

[131] Aaron Clauset, Cosma Rohilla Shalizi, and Mark EJ Newman. Power-law distributions in empirical data. *SIAM review*, 51(4):661–703, 2009.

[132] Valerio Arnaboldi, Marco Conti, Massimiliano La Gala, Andrea Passarella, and Fabio Pezzoni. Ego network structure in online social networks and its impact on information diffusion. *Computer Communications*, 76:26–41, 2016.

[133] Francis Bloch, Matthew O. Jackson, and Pietro Tebaldi. Centrality measures in networks. *CoRR*, abs/1608.05845, 2016.

[134] Naeimeh Laleh, Barbara Carminati, and Elena Ferrari. Graph based local risk estimation in large scale online social networks. In *Proceedings of the 2015 IEEE International Conference on Smart City/SocialCom/SustainCom (SmartCity)*. IEEE, 2015.

[135] Yazan Boshmaf, Konstantin Beznosov, and Matei Ripeanu. Graph-based sybil detection in social and information systems. In *Proceedings of the 2013 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*. ACM, 2013.

[136] Carlos Freitas, Fabricio Benevenuto, Saptarshi Ghosh, and Adriano Veloso. Reverse engineering socialbot infiltration strategies in twitter. In *Proceedings of the 2015 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*. IEEE, 2015.

[137] Dong Yao, Pim van der Hoorn, and Nelly Litvak. Average nearest neighbor degrees in scale-free networks. *arXiv preprint arXiv:1704.05707*, 2017.

[138] Mark EJ Newman. Mixing patterns in networks. *Physical Review E*, 67(2):026126, 2003.

[139] Albert-László Barabási and Márton Pósfai. *Network science*. Cambridge University Press, Cambridge, 2016.

[140] Vladimir Batagelj and Matjaz Zaversnik. An o (m) algorithm for cores decomposition of networks. *arXiv preprint cs/0310049*, 2003.

[141] Jyun-Cheng Wang and CQ Chiu. Detecting online auction inflated-reputation behaviors using social network analysis. In *Proceedings of the Annual conference of the North American association for computational social and organizational science (NAACSOS 2005)*, 2005.

[142] Chaochang Chiu, Yungchang Ku, Ting Lie, and Yuchi Chen. Internet auction fraud detection using social network analysis and classification tree approaches. *International Journal of Electronic Commerce*, 15(3):123–147, 2011.

[143] Jun-Lin Lin and Laksamee Khomnotai. Using neighbor diversity to detect fraudsters in online auctions. *Entropy*, 16(5):2629–2641, 2014.

[144] Ho-Yu Lam and Dit-Yan Yeung. *A learning approach to spam detection based on social networks*. PhD thesis, Hong Kong University of Science and Technology, 2007.

[145] Duncan J Watts and Steven H Strogatz. Collective dynamics of small-world networks. *nature*, 393(6684):440, 1998.

[146] Alain Barrat, Marc Barthelemy, Romualdo Pastor-Satorras, and Alessandro Vespignani. The architecture of complex weighted networks. *Proceedings of the National Academy of Sciences of the United States of America*, 101(11):3747–3752, 2004.

[147] Alireza Abbasi and Liaquat Hossain. Hybrid centrality measures for binary and weighted networks. *Complex networks*, pages 1–7, 2013.

[148] Jukka-Pekka Onnela, Jari Saramäki, János Kertész, and Kimmo Kaski. Intensity and coherence of motifs in weighted complex networks. *Physical Review E*, 71(6):065103, 2005.

[149] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 12(Oct):2825–2830, 2011.

[150] Yoav Freund and Robert E Schapire. A desicion-theoretic generalization of on-line learning and an application to boosting. In *Proceedings of the European conference on computational learning theory*. Springer, 1995.

[151] Robert E Schapire. Explaining adaboost. In *Empirical inference*, pages 37–52. Springer, 2013.

[152] Peter Hall, Byeong U Park, and Richard J Samworth. Choice of neighbor order in nearest-neighbor classification. *The Annals of Statistics*, pages 2135–2152, 2008.

[153] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[154] Ingo Steinwart and Andreas Christmann. *Support Vector Machines*. Springer Publishing Company, Incorporated, 1st edition, 2008.

[155] Huan Liu and Lei Yu. Toward integrating feature selection algorithms for classification and clustering. *IEEE Transactions on knowledge and data engineering*, 17(4):491–502, 2005.

[156] Mohamed Amir Esseghir, Gilles Goncalves, and Yahya Slimani. Memetic feature selection: Benchmarking hybridization schemata. In *Proceedings of the International Conference on Hybrid Artificial Intelligence Systems*. Springer, 2010.

[157] Thomas Lal, Olivier Chapelle, Jason Weston, and André Elisseeff. Embedded methods. *Feature extraction*, pages 137–165, 2006.

[158] Isabelle Guyon and André Elisseeff. An introduction to variable and feature selection. *Journal of machine learning research*, 3(Mar):1157–1182, 2003.

[159] Jyun-Cheng Wang and Chui-Chen Chiu. Recommending trusted online auction sellers using social network analysis. *Expert Systems with Applications*, 34(3):1666–1679, 2008.

[160] Leman Akoglu, Mary McGlohon, and Christos Faloutsos. Oddball: Spotting anomalies in weighted graphs. In *Proceedings of the Pacific-Asia Conference on Knowledge Discovery and Data Mining*. Springer, 2010.

[161] Andrew Ng. Advice for applying machine learning, 2011.

[162] Guillaume Lemaître, Fernando Nogueira, and Christos K Aridas. Imbalanced-learn: A python toolbox to tackle the curse of imbalanced datasets in machine learning. *Journal of Machine Learning Research*, 18(17):1–5, 2017.

[163] V. King, J. Saia, V. Sanwalani, and E. Vee. Scalable Leader Election. In *Proceedings of the 17th Annual ACM-SIAM Symposium on Discrete Algorithm*, Miami, FL, USA, 2006.

[164] John Veizades and Charles E Perkins. Service location protocol. 1997.

[165] Charles Perkins and Scott Kaplan. Service location protocol. In *ACTS Mobile Networking Summit/MMITS Software Radio Workshop*, 1999.

[166] XACML Info. Balana. http://xacmlinfo.org/2012/12/18/getting-start-with-balana. Accessed: 2017-02-26.

[167] Erik Rissanen et al. extensible access control markup language (xacml) version 3.0, 2013.

[168] XACML Info. Wso2 identity server. http://xacmlinfo.org/category/wso2is/. Accessed: 2017-02-26.

[169] OpenSLP. Service location protocol. http://www.openslp.org/. Accessed: 2017-02-26.

[170] Openslp: Service location protocol. http://www.openslp.org/, accessed: 2017-02-26.

[171] Christopher Copeland and Hongxia Zhong. Tangaroa: a byzantine fault tolerant raft, 2016.

[172] Ittai Abraham, Srinivas Devadas, Danny Dolev, Kartik Nayak, and Ling Ren. Efficient synchronous byzantine consensus. *arXiv preprint arXiv:1704.02397*, 2017.

[173] Dieudonne Mulamba and Indrajit Ray. Resilient reference monitor for distributed access control via moving target defense. In *DBSec*, pages 20–40, 2017.

[174] Baruch Awerbuch and Christian Scheideler. Robust random number generation for peer-to-peer systems. *Theor. Comput. Sci.*, 410(6-7):453–466, 2009.