APPROVAL SHEET

Title of Thesis: A Policy-based Framework for Privacy-respecting Deep Packet Inspection in TLS Implementations

Name of Candidate: Arya Renjan
MS, Computer Engineering
May 2019

Thesis and Abstract Approved: ______________________
Dr. Karuna Pande Joshi
Assistant Professor
Department of Information Systems

Date Approved: ____________________________

ABSTRACT

Title of Thesis: A Policy-based Framework for Privacy-respecting Deep Packet Inspection in TLS Implementations

Arya Renjan, MS Computer Engineering, May 2019

Thesis directed by: Dr. Karuna Pande Joshi
Assistant Professor
Department of Information Systems

Deep Packet Inspection (DPI) is instrumental in investigating the presence of malicious activity in network traffic, and most existing DPI tools work on unencrypted payloads. As the internet is moving towards fully encrypted data-transfer, there is a critical requirement for privacy-aware techniques to efficiently decrypt network payloads. With the introduction of TLS 1.3 standard that only supports protocols with Perfect Forward Secrecy (PFS), many existing techniques for decryption to do further DPI analysis will become ineffective. We have developed an ABAC (Attribute Based Access Control) framework that efficiently supports existing DPI tools while respecting user's privacy requirements and organizational policies. It gives the user the ability to accept or decline access decision based on his privileges. Our solution evaluates various observed and derived meta-characteristics of network connections against user access privileges using policies described with semantic technologies. Network meta-characteristics like IP intelligence is one of the many attributes that can be used in defining access control policies. We also present Dynamic Attribute based Reputation (DAbR), a Euclidean distance based technique, to generate reputation scores for IP addresses by assimilating meta-data from known bad IP addresses. This approach is based on our observation that many bad IP's share similar attributes and the requirement for a lightweight technique for reputation scoring. DAbR generates reputation scores for IP addresses on a 0-10 scale which represents its trustworthiness based on known bad IP address attributes. To evaluate DAbR, we calculated reputation scores on a dataset of 87k IP addresses and used them to classify IP addresses as good/bad based on a threshold. An F-1 score of 78% in this classification task demonstrates our technique's performance. The reputation scores when used in conjunction with the policy enforcement module, can provide high performance and non privacy-invasive malicious traffic filtering. In this thesis, we also describe our framework and demonstrate the efficacy of our technique with the help of use-case scenarios to identify network connections that are candidates for Deep Packet Inspection. Since our overall ABAC technique makes selective identification of connections based on policies, both processing and memory load at the gateway will be reduced significantly.

A Policy-based Framework for Privacy-respecting Deep Packet Inspection in TLS Implementations

by

Arya Renjan

Thesis submitted to the Faculty of the Graduate School
of the University of Maryland in partial fulfillment
of the requirements for the degree of
Master of Science
2019

Dedicated to my family

ACKNOWLEDGMENTS

I express my sincere gratitude towards my advisor, Dr. Karuna Joshi for her constant support and expert guidance to me throughout my research. Her advice and inputs have been invaluable in motivating during my life at UMBC. I would also like to thank my thesis committee Dr. Anupam Joshi, Dr. Tim Finin and Dr. Mohamed Younis for their expert advice and support. I also take this opportunity to thank my friends and lab mates in KnACC and Ebiquity lab for their valuable inputs. I am also immensely grateful to my loving husband Sandeep for his support and guidance in all my endeavors without which this journey would have been impossible. Lastly, I thankfully acknowledge my parents, sister, and in-laws for their constant love, support, and motivation.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

INTRODUCTION

With the unprecedented growth of the Internet, security and privacy have become critical concerns to its users. Web traffic encryption is a widely adopted technique to protect users' privacy. Adoption of security protocols like SSL (Secure Socket Layer) and TLS (Transport Layer Security) to provide secure network connections is also on the rise. Popular web browsers like Chrome and Firefox have made the requirement for encrypted connections a priority. Since 2018, Chrome deems a website as 'not secure' if it uses HTTP. As of 2019, around 80% of web traffic through Google Chrome is encrypted [1], and more than 50% of Alexa top one million websites are HTTPS [2]. However, it is reported that the number of attacks using encryption to conceal its malicious attack vectors [3] is also increasing at a fast rate. Zscaler [4], a cloud-based information security company, reports that there is a 30% increase in the encrypted malicious traffic and many new attack payloads are being delivered over encrypted communication channels. Gartner [5] expects that at least half of attacks caused by malware in 2019 will use some encryption. Hence, it is imperative to analyze encrypted data streams to detect potential security threats.

TLS version 1.2 was released in August 2008, and it supports many legacy cryptographic protocols like SHA1 and MD5 which are now considered insecure. They are susceptible to several known attacks like SLOTH [6], POODLE [7], etc. Its latest version, TLS 1.3 (released in 2018), addressed these issues by removing support for many such legacy protocols. Another significant change in TLS 1.3 makes Perfect Forward Secrecy (PFS) mandatory [8] unlike its predecessors, where it was optional. With PFS enabled, the session key (negotiated between the client and the server during TLS handshake) for data encryption is never transmitted across the network. Instead, it uses protocols like ephemeral Diffie-Hellman key exchange to generate the same session key in the client and the server. As a result, even a compromise of the server's private key will not affect the confidentiality of the previous sessions that used it.

It is evident that PFS improves the security and privacy of its users, but it has some adverse effects on several existing security solutions [9]. Many security solutions use TLS 1.2 features to decrypt sessions and use DPI (Deep Packet Inspection) tools like SolarWinds [1], Paessler Packet Sniffing [2], etc. for further analysis. For example, if RSA authentication is used, a unique pre-master secret (used for generating the master secret) is first encrypted with the server's public key and is sent to the server. The server's private key may not available to these security solutions also. However, they circumvent this by inserting a root certificate to their clients and use them during the handshake. Using this pre-master secret and other random numbers that can be extracted from the session traffic, the DPI tools decrypt the entire session and use it for malware detection, internet censoring, and so forth. With PFS, such solutions will not work because the actual session keys never leave the client or server machine and passive decryption becomes difficult.

One existing solution to overcome this problem is to use active proxying in which the gateway encrypts and decrypts every connection between all clients and servers. Considering the velocity and veracity of traffic through the gateway, this will become a big-data problem and will be highly resource intensive. Another solution is to use techniques like

[1]`https://www.solarwinds.com/topics/deep-packet-inspection`
[2]`https://www.paessler.com/manuals/prtg/packet_sniffer_header_sensor`

Cisco ETA (Encrypted Traffic Analysis)[3] that tries to identify malware without decryption. Anderson et al. [10] present a study on using TLS meta-characteristics for malware detection. Yet another competing technique is from ExtraHop Networks [8] where all session keys are retrieved from every clients for decryption. Decryption of such web traffic for inspection is not only expensive in cost and time, they also invade users' privacy. Therefore, there is a requirement for a system that injests network attributes and user privileges to generate decisions to be taken by the gateway on that network session.

In this research, we describe an ABAC (Attribute-Based Access Control) policy framework in which the various observed and derived attributes of the network connections are evaluated using policies defined with semantic technologies. It enables the support for existing DPI tools and respects the privacy of its users by giving them an option to choose in accordance with organizational policies. Since our solution selectively identifies connections based on policies for DPI analysis, the processing load and memory load at the gateway will be reduced significantly. This analysis could be taken offline as well. We demonstrate the flexibility and efficacy of our technique by describing handpicked use-case scenarios.

For generating access policy decisions, our system uses various direct and derived meta-features of network traffic. Network intelligence related information is an important attribute that could be used to define access control policies. However, due to the increasing encrypted traffic and more malicious websites moving to encrypted network to conceal their identity, there is a critical requirement to use unencrypted features of web traffic for identifying and filtering attacks. IP address is one such feature which could be utilized for filtering out malicious traffic in encrypted traffic. IP address based web traffic filtering is widely used as the first line of defense in many Intrusion Detection Systems (IDS) like

[3] `https://www.cisco.com/c/en/us/solutions/enterprise-networks/enterprise-network-security/eta.html`

Snort. Lists of bad or blacklisted IP addresses for such filtering are available from a plethora of sources like FireHOL[4], Cisco Talos[5], Spamhaus[6], etc. A major shortcoming of using such lists for filtering attacks is that many of them are based on known malicious incidents. However, attackers often take control of newer targets and use them for spreading malware or initiating attacks. Such cases are overlooked by these filtering systems.

This research also proposes a simple and computationally fast reputation scoring technique, DAbR (Dynamic Attribute-based Reputation), for scoring unknown IP addresses which are not present in existing bad IP address lists. This helps to generate a score for unknown IP addresses indicating thesir possible maliciousness nature. We envision that this IP reputation system works in conjunction with the policy framework to generate access decisions. The dynamically calculated IP reputation scores from DAbR module is inputted to the ABAC module discussed earlier as its attributes. DAbR generates an offline model for bad IP addresses by ingesting blacklists and their various meta-characteristics. When a real time IP address arrives, our proposed system tags a reputation score with it, by calculating the similarity of its meta-characteristics to the generated model. We purposefully chose simpler, but effective model generation and reputation scoring steps in DAbR to accommodate fast and high volume traffic, so that it can be easily integrated with the policy module to generate fast access decisions.

The rest of this paper is organized as follows: Chapter 2 describes a brief literature review, and Chapter 3 presents an overview of this work. Chapter 4 discusses the architecture of our policy-based system. In Chapter 5, we discuss our DAbR which constitutes a part of our overall system. Chapter 6 demonstrates the usefulness of our technique with the help of use-case scenarios followed by the conclusions and future work in Chapter 7.

[4]`http://iplists.firehol.org/`
[5]`https://www.talosintelligence.com/reputation_center`
[6]`https://www.spamhaus.org/`

Chapter 2

RELATED WORK

Policy-based access control is a much-researched topic that finds applications in a plethora of fields. In addition to the classical access control models like discretionary (DAC), mandatory (MAC) and role-based (RBAC) models, the attribute-based access control (ABAC) model also gained traction in recent years. In [11], Jin et al. investigate formal connections between these three classical models and ABAC models. ABAC techniques are used in several fields like cloud computing [12], web services [13], Internet-of-Things [14], [15], and grid computing [16].

Access control policies are also widely used in the field of network security to reduce the risk of unauthorized access. Some research in this field is discussed here. Berger et al. [17] propose a framework for dynamic ABAC configuration in firewalls where the temporary binding between a user and IP address is used to create policies to access resources over the internet. In another research, Burmester et al. [18] present an extended ABAC model called real-Time Attribute-Based Access Control model (T-ABAC), that can guarantee real-time availability for high priority IP packets. In [19], Basile et al. discuss an ontology-based policy translation approach that mimics IT administrators, to identify device configurations based on network topology and security policies.

The policy management techniques discussed above use Attribute-Based Access Control for traffic filtering, suspicious activities identification, and so forth. In this paper, we propose a system that uses ABAC to selectively identify network connections for Deep Packet Inspection (DPI) in encrypted traffic while respecting the user's privacy requirements and organizational policies. In another related research, Hu et al. [20] patented a policy enabled Deep Packet Inspection framework for telecommunications network. In contrast, our system extracts several direct and extended attributes and use ABAC using semantic technologies with a goal of specifically supporting DPI in perfect forward secrecy implementations like the new TLS 1.3 standard.

This literature review also discusses the existing work related to reputation scoring, as this is another main contribution in our research. Reputation scoring is studied and used in a variety of fields like search engine query [21], e-commerce [22], social network rankings for enterprise profitability calculation [23], network trust-worthiness [24] etc. Page et al. [21] devised "PageRank" algorithm that uses reputation scoring to rank the search results based on the backlinks from each webpage. Katkar et al. [22] developed a Trust Reputation System (TRS) for e-commerce applications and used data mining to perform a semantic analysis of feedback, for calculating the reputation score. In the field of online and social media reputation, Marrakhi et al. [23] devised a technique IRMS (Intelligent Reputation Measuring System) to rank a brand's presence in the social media. Their scoring is based on the number of citations of the brand, its reach, impact, and influence in social media.

Reputation scoring is also discussed in the field of networking [24] to ensure trust-worthiness of the participating nodes. Zhou et al. [25] devised a "Gossip" based reputation aggregation in peer-to-peer networks called 'GossipTrust' in which, each node receives reputation vectors from other nodes in the network and selectively integrates vectors to determine the trustworthiness of participating nodes. Mui et al. [26] used a bayesian probabilistic approach for formulating ratings in distributed networks. In the field of wireless sensor networks, Kim et al. [27] used a fuzzy logic based approach to score trust levels of each node, based on multiple degrees of trustworthiness in each node pair.

In network security, reputation scoring finds application in detecting malicious activities on Internet. Esquivel et al. [28] used reputation scoring of IP addresses for e-mail spam filtering by checking the SPF (Sender Policy Framework) resource records of SMTP senders. They were able to identify legitimate servers, spam servers and end hosts among the SMTP senders using their technique. In another research, Anderson et al. [10] used the meta-characteristics of TLS (Transport Layer Security) connections for TLS based malware detection. Studies have been done to detect malicious websites/domains using reputation scoring. Hegli et al. [29] used Maximum Entropy Discrimination (MED) classifier for reputation scoring of websites, based on data regarding domain registration, service hosting, IP address, domain creation date, popularity rank, number of hosts, etc. Chiba et al. [30] developed and evaluated a method to detect malicious websites using SVM based analysis of the octet-based, extended octet-based and bit string-based features of IP addresses. Another related work is from Antonakakis et al. [31] where they developed "Notos" reputation system that uses the unique DNS characteristics to filter out malicious domains based on their previous involvement with malicious or legitimate internet services. They used clustering analysis of network-based features, zone-based features and evidence-based features for each domain for reputation scoring. In many of these existing techniques, the scoring is done offline and hence can employ processing-intensive techniques. However, in this research, we focus on developing a lightweight technique which uses the similarity of an IP address to the existing bad IP's to determine reputation scores.

Chapter 3

OVERVIEW

The primary focus of this thesis is to enable efficient passive monitoring on encrypted traffic that also respects user's privacy requirements for implementations like TLS 1.3 where perfect forward secrecy is enforced. Consider a hypothetical situation of organizational network which has various types of users like high-privileged employee, low privileged employee, guests etc. Here, the organizational policies may define that all employees should be able to access confidential information whereas guest users must not. If a guest user is trying to access them, the system should block the connection. Instead, if a high-privileged employee or low privileged employee is accessing them, the gateway could inspect the connection to check if the user is trying to perform activities against the organizational policies, instead of blocking the connection. The organization may also typically assign more level of trustworthiness to the high-privileged employees. So, before inspecting their connections the gateway may also request permission from them to avoid false alarms.

In order to deep-inspect the connections for encrypted sessions, the gateway has to decrypt the packets. Existing techniques for deep-inspection may be harder to work in TLS 1.3 encrypted sessions owing to their default 'Perfect Forward Secrecy' feature. Our system uses ABAC policy framework to identify candidates for deep packet inspection on

TLS 1.3 encrypted connections as well as other security protocol versions like TLS 1.2.

3.1 Threat Model

We envision our system to work inside the gateway of an organizational network. Our system has three broad categories of users; organization administrators who create/update policies, internal entities who should comply with the organization's policies in order to access the resources, and external entities which are accessing the internal entities (or the external entities being used by the internal users). In our threat model, we consider the organization administrators to be trusted. The internal entities and external entities are not trusted by default. However, there may be some internal entities who are added as trusted by organizations administrators. However, we assume that the internal entity operating systems are trusted and we can run/manage some trusted user agents in the system.

We classify two categories of threat actors in our system. The first set of actors include all the malicious entities who are external to the organizations. Some of the expected threats from them include injection of threat vectors like malware, trojans, etc. into the system to perform various malicious activities like data exfiltration, denial-of-service (as source and destination), resource stealing (e.g. crypto-jacking), cyber-espionage, and so forth. The second category of threat actors is from the internal entities which may be results of intentional actions from its users (internal threats) or unintentional activities as a result of an already existing malware, software bug in the user's system.

Our proposed system detects these threats using two primary techniques. Firstly, it uses techniques like DAbR, white-lists, and blacklists to identify malicious, and dis-reputed sources. Secondly, it allows for efficient deep packet inspection which enables detecting various known attack vectors. Since we consider the internal entity operating system to be trusted, administrators will be able to run user agents them that allows secure communication between the gateway component. We scope ourselves out of those threats as a result of bugs in the operating system and untrusted operating system in the internal entities.

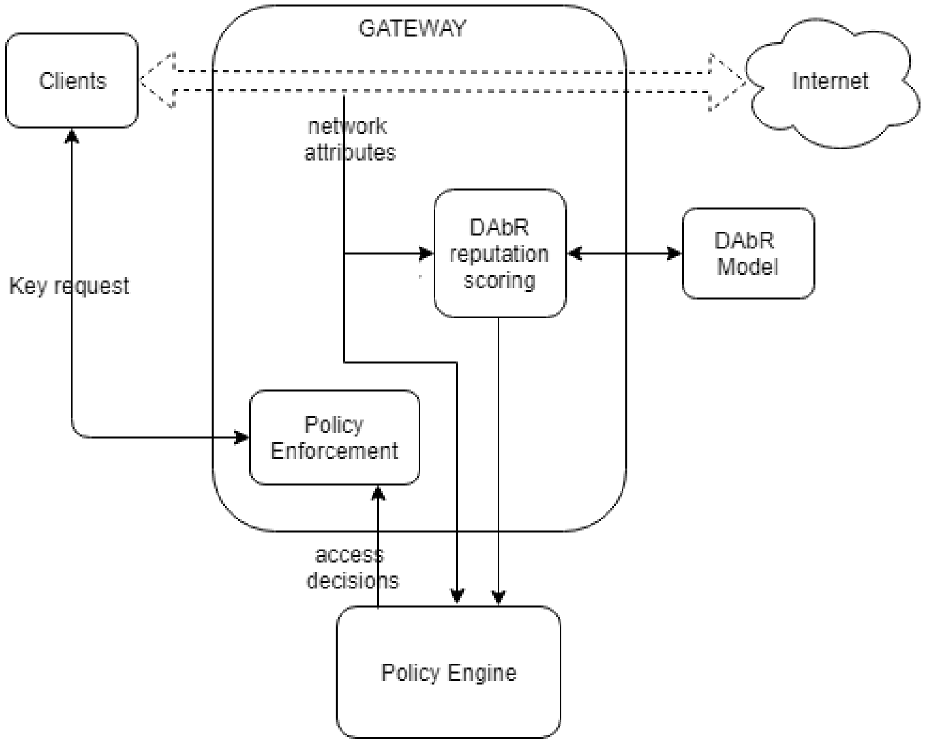

FIG. 3.1: System Overview

Figure 3.1 shows the overview of our system that uses network attributes and user privileges to generate access decisions via policy based approach. The network attributes like IP addresses are also given to the DAbR reputation scoring system to generate numeric reputation scores which are further used as attributes to generate access decisions. The policy based architecture is discussed in detail in Chapter 4 and DAbR in Chapter 5. DAbR helps to assign reputation scores for IP addresses by identifying its possible maliciousness even though it has never appeared in any existing blacklists. Integrating this in the policy based architecture helps to generate better access decisions whether to block, allow or inspect connections made to those IP addresses.

3.2 Thesis Statement

This research is aligned on the following statement:

A semantically-rich policy framework on network traffic meta-features and user privileges will allow us to enable privacy-respecting Deep Packet Inspection in Perfect Forward Secrecy enabled security protocols like TLS 1.3.

Here, a semantically rich policy framework implies the ABAC policy framework. The policies are defined in terms of network traffic meta-features (both direct and derived attributes). Direct attributes include network features that are directly accessed from real time network traffic, like source and destination IP addresses, SNI, Protocol etc. Derived attributes include the network intelligence information which is obtained from direct attributes using techniques like DAbR.

Chapter 4

SYSTEM ARCHITECTURE

The architecture of our system that enables efficient policy respecting packet inspection on encrypted traffic by factoring in the user access privileges and network traffic meta-features is discussed in this chapter. For this purpose, we envision a multi-agent system as presented in Figure 4.1. Unlike active monitoring techniques that are resource intensive, in our architecture, the client-user agents interact with the monitoring component based on organizational policies for achieving its goal. In this paper, we use an extended ABAC (Attribute Based Access Control) with a knowledge graph and reasoner to infer policy decisions. The three key modules in our architecture are Network Monitoring Engine, Client Agents, and Policy Engine.

The Attribute Extraction module in the Network Monitoring Engine resides at the edge node of the organizational boundary (typically external gateways). It extracts various attributes (observable attributes like IP address, protocol, etc. and extended attributes like IP intelligence) for every connection and requests the Policy Engine to make access control decisions. The policy engine uses organizational policies to make various access control decisions as described in section 4.1.2. The decisions are then sent to the Policy Enforcement module of the Network Monitoring Engine for further processing and enforcement. The Policy Enforcement module also interacts with the client-user agents and supports ef-

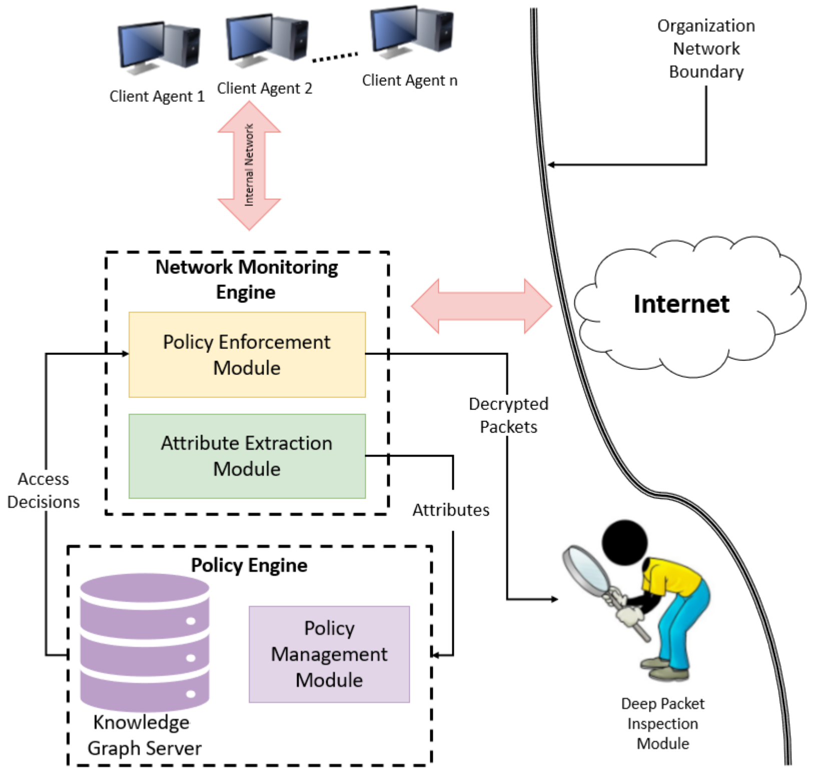

FIG. 4.1: System Architecture

ficient DPI only for selected connections, thus providing required levels of security and privacy to its users. The detailed description of each of these key components ensues in the subsequent sections.

4.1 Network Monitoring Engine

The Network Monitoring Engine monitors the traffic across the network boundary and enforces the decisions taken by the Policy Engine. It has two main modules: The first module is the Attribute Extraction module that fetches network attributes and security intelligence from traffic in real-time. The second sub-module is Policy Enforcement module which enforces the access decisions generated by the policy engine.

4.1.1 Attribute Extraction Module

The attribute extraction module performs real-time traffic monitoring of every connection traversing the gateway and extracts their attributes simultaneously. In the Attribute Extraction module, the system extracts various network and flow attributes of each connection. Network attributes are those attributes which give information on the network parameters of traffic like source and destination IP address, protocol, Server Name Indication(SNI) of the external user, etc. In addition to network attributes, the module also extracts flow attributes that provide information on traffic flow like time, packet size, count of packets, etc.

The attribute extraction module then uses this information to derive more attributes related to the connection. It takes inputs like external IP address and SNI to gather information about the IP intelligence, domain category, etc. in real-time. For extracting attributes on network intelligence, we can use reputation scoring services that give information on the malicious characteristics of external users. This reputation scores may be boolean values indicating the presence of external IP in blacklists, or numeric probability scores corresponding to its possible maliciousness. In addition to security intelligence, this module also gathers information regarding the category of service the user is trying to access. For example, *facebook.com* is a ‘social networking’ website, *youtube.com* delivers ‘media and video streaming’, etc. All these attributes are then sent to the policy engine for generating policy decisions.

4.1.2 Policy Enforcement Module

This module enforces the access decisions generated by the policy engine. The major access decisions made by the policy engine and their respective enforcement actions are discussed below:

1. *AllowConnection:* If a connection is as per the organizational policies, it will be allowed without any restrictions. This access decision will be used in scenarios where the user is trying to access legitimate websites like *google.com* or *stackoverflow.com*.

2. *BlockConnection:* This decision is generated if a connection is against the organization's policies. Network connections to known malicious hosts is a typical example of blocked connections. In such cases, the policy enforcement module will block the connection, thus avoiding any potential spurious network activity.

3. *MandatoryInspection:* This decision is used to give certain privileges to specific sets of users, but with some restrictions. Consider the case of a software company which does not want its developers to upload proprietary source code or resources to external file servers. Since blocking all connection to file servers is too restrictive, a more appropriate access decision will be to perform mandatory deep packet inspection and allow the connection. *MandatoryInspection* access decision informs the Policy Enforcement module to perform deep inspection of data packets. In such a case, it will start capturing the data packets and will interact with the client-agents in user-clients to retrieve the required session key. Once the session key is retrieved, the packets are decrypted and are transferred to the Deep Packet Inspection module to inspect further for suspicious content. If the client-agents fails to deliver the keys, the connection will be blocked.

4. *OptionalInspection:* This access decision will allow flexibility to privileged users who are more responsible or are experts. Consider a case where the security team wants to download malware samples from some known malicious external source. If the contents of the connections are deep inspected, it will raise false alarms and hence they can choose not to allow inspection. However, if they want to download some resources which are not malicious but from a suspicious source, they can opt for

inspection, giving flexibility to the users. If this access decision is received, Policy enforcement module interacts with the client user-agent and requests permission. If the user responds with the key, it will decrypt the contents with the session key and send them to perform deep packet inspection. However, if the response from the user is negative, the enforcement module will take action as if the access decision was *AllowConnection*.

4.2 Client Agents

Each user-client in the internal network will run a client-agent module. It has two main functions. Interaction with the Network Monitoring Engine and retrieving the required session keys from its user-client. The first task of the client agent is to extract session keys for different connections. Many applications provide inbuilt facilities to extract session keys from TLS connections they make with external clients. For example, Chrome and Mozilla web browsers have an option, when enabled, extracts and stores specific session keys. The user client agent will maintain this list of session keys and responds to requests from the Policy Enforcement module as required. The client user-agents can receive two types of requests from the Policy enforcement module. If the request contains a *MandatoryInspection* decision, it will retrieve the key and return it to the user. On the other hand, if it is an *OptionalInspection* decision, the agent will ask the user for a decision. The existence of this module enables the user with the ability to choose.

4.3 Policy Engine

This module is the core of our system and is responsible for making decisions by using a knowledge graph and ABAC policies specified for the organization. It accepts attributes extracted by the Attribute Extraction module and use them in conjunction with policies to generate access decisions. The policy engine can generate four access decisions: *AllowConnection*, *BlockConnection*, *MandatoryInspection*, and *OptionalInspection* as described in Section 4.1.2. The policy engine uses an extended ABAC (Attribute Based Access control) model using semantic technologies to make a policy decision per connection. In ABAC, the different attributes of each entity are used to define policies for access control. There are two major modules in the policy engine, knowledge graph server and policy management module which are described in the following sections.

4.3.1 Knowledge Graph Server

Knowledge graph server houses the knowledge-graph that encapsulates the domain knowledge, ABAC access control policies defined using semantic web technologies, and a reasoner that infers the access decisions. A major contribution in this paper is the development of a knowledge graph that abstract the attributes and knowledge to make policy decisions. Figure 4.2 presents a relevant part of the knowledge graph designed for this purpose.

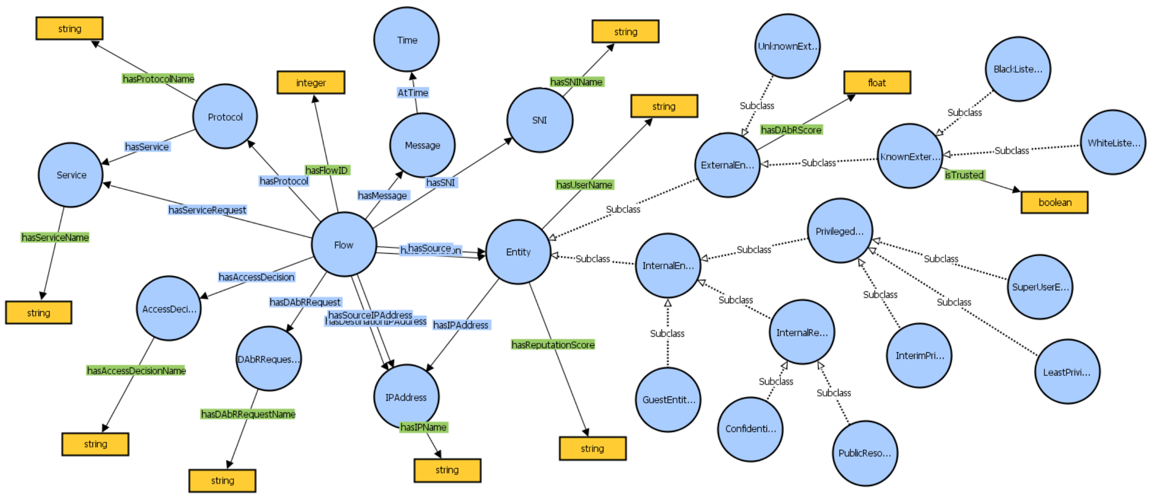

FIG. 4.2: Relevant part of the knowledge graph used
]

Of the many classes in the knowledge graph, *Entity* class plays an important role while developing policies. The *Entity* class has several object and data properties like IP addresses, associated roles, reputation scores, etc. depending on the different user categories. In our knowledge graph design, we define many types of users and are categorized hierarchically by their roles and functions. The first level of sub-classes consists of *InternalEntity* (users who are part of the organizational network) and *ExternalEntity* (users who are external to the network). The *ExternalEntity* class is further classified into sub-classes *BlackListedEntity*, *WhiteListedEntity*, *GreyListedEntity*, etc. This hierarchical arrangement helps our knowledge graph to incorporate various external attributes like IP addresses, intelligence about external users like their presence in blacklists, reputation scores, etc. and use them to define policies for decision making. Other classes include *Flow* class that define the information about the flow and messages for which we need to make access decisions, *Protocol* which may be extracted from the messages, *Service* class that define the type of resource or application the connection is trying to access, *Category*, *Time*, etc.

Another major component in the Knowledge Graph server consists of ABAC policies which are implemented using semantic technologies. We can define policies using SWRL[1] (Semantic Web Rule Language) or using other policy specification frameworks like Rein [32]. The advantage of using semantic technologies for specifying the policies is their ability to reason over the knowledge graph and infer complex relationships. For example, they give the administrators the ability to define simple rules like *if the external IP is an IP address corresponding to a BlackListedUser, then issue BlockConnection for all requesting users*. The reasoner will automatically infer which users should get access to it, what kind of services to that IP address need to be blocked, etc. Detailed examples of their usefulness are described in Chapter 6. However, different policies may result in conflicting

[1]https://www.w3.org/Submission/SWRL/

access control decisions for the same connection. We address this issue by prioritizing the access decision in the order *OptionalInspection*, *BlockConnection*, *MandatoryInspection*, and *AllowConnection*. For example, if the policies result in two decisions *BlockConnection* and *AllowConnection*, the priority decision *BlockConnection* will be the final decision.

4.3.2 Policy Management Module

This module is used to create, modify, and delete access policies on the entities defined in the knowledge graph. Additionally, system administrators use the policy management module to add more knowledge to the knowledge graph by adding new instances to it, updating the knowledge graph, etc. For example, this module can be used to add new user clients, define his/her privileges, create new classes, etc. The policy management module can also periodically update the knowledge graph with the latest blacklists and whitelists of IP addresses to keep in par with the dynamic nature of network intelligence.

4.4 System Implementation

This section describes the tools and API's used in the development of our prototype implementation. In our prototype, the Network Monitoring Engine is implemented using *mitmproxy*[2] and python scripts. *mitmproxy* is an open source HTTPS proxy that supports a python API to manipulate different connections flowing through it. We developed separate python scripts on top of *mitmproxy* to implement the functions of attribute extraction module and policy enforcement module. In addition to extracting flow and network related attributes as described in section 4.1.1, our prototype interfaces this module to the reputation scoring engine *DAbR* [33] described in Chapter 5. *DAbR* will extract several attributes about the external IP addresses and generates a reputation score. All these information are

[2]`https://mitmproxy.org/`

then sent to the policy engine.

The policy enforcement module is also implemented in python using the API's from the *mitmproxy*. If the access decision is *BlockConnection*, the user is redirected to a generic error page, and when the access decision is *AllowConnection*, the connection details are just logged. If the access decision is *MandatoryInspection* or *OptionalInspection*, the script will interact with the respective client-agents over secure sockets to retrieve sessions keys. Once the session keys are retrieved, the scripts use *tshark* to decrypt the encrypted packets and forward them to DPI tools.

Our prototype uses the RDF triple store, *Apache Fuseki*[3], as the knowledge graph server. The policy rules can then be defined using SWRL rules. *Fuseki* is configured with a generic reasoner and standard OWL ruleset to provide inferencing. The policy engine uses the *SPARQLWrapper*[4] API to insert information about different live flows into Fuseki and query it to generate access control decisions. This wrapper is also used by the policy management module to add and manipulate more knowledge into the knowledge graph server.

[3]`https://jena.apache.org/documentation/fuseki2/`
[4]`https://rdflib.github.io/sparqlwrapper/`

Chapter 5

DABR ARCHITECTURE FOR IP REPUTATION SCORING

Reputation scoring is an effective technique to identify attacks from known attack sources like botnets, spammers, virus infected systems, etc. However, most traditional IP reputation services rely on manually updated lists of bad/malicious IP addresses and cannot detect attackers using IP's which are not yet reported as bad or malicious. In this part of our research, we associate any IP address with a reputation score which is indicative of its malicious behavior. This reputation score of IP address is used to define access control policies in ABAC architecture discussed in Chapter 4.

In the literature, Elkhannoubi et al. [34] point out several interesting observations regarding malicious IP addresses and factors which potentially helps their proliferation. Their research points towards different social, economic, and political factors which affect malware spread. Our preliminary analysis of bad IP address lists and their attributes (eg. IP address origin country, ASN, ISP, etc.) also showed that many bad IP's share similar attributes. Hence in this research, we develop DAbR, a lightweight technique (computationally fast) for IP reputation scoring, by extracting and assimilating the similarity of IP address attributes from existing bad lists. Each DAbR score (the calculated reputation score) is a measure of trustworthiness of that IP address calculated from its attributes and

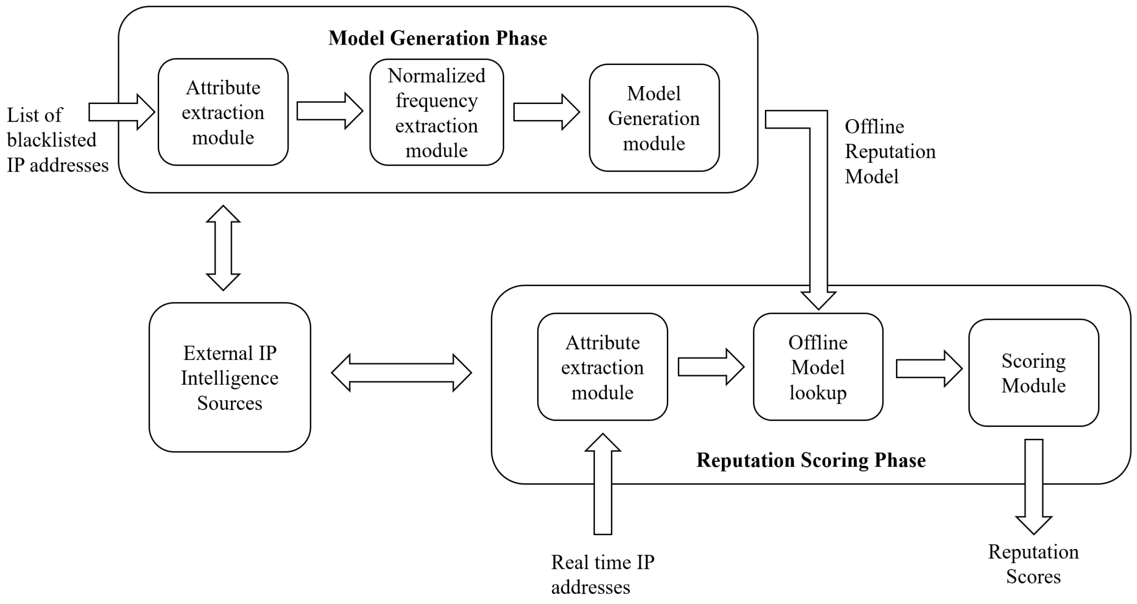

FIG. 5.1: DAbR Architecture

their presence in existing bad IP address lists. DAbR creates a vector space for bad IP addresses and when a new IP address is available, it is projected to this vector space for calculating its DAbR score.

5.1 System Architecture

The general architecture of DAbR is described in Figure 5.1. It operates in broadly 2 phases: a model generation phase that generates a model for bad IP addresses and a reputation scoring phase which uses the generated model for scoring new and unobserved IP addresses. Sections 5.1.1 and 5.1.2 delve into their specifics.

5.1.1 Model Generation Phase

This is an offline phase which generates a model for bad IP addresses based on existing bad IP address attributes. It uses a lightweight technique for model generation to accom-

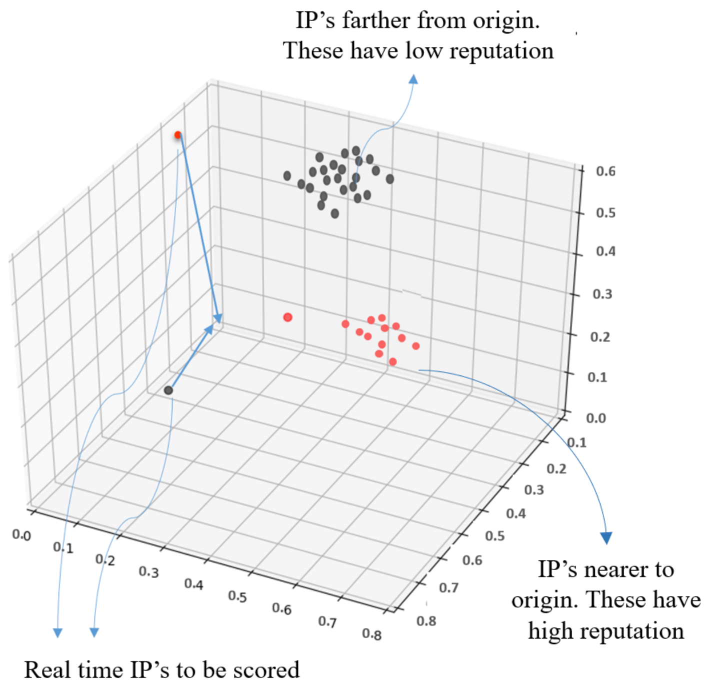

FIG. 5.2: A Sample IP Vector Space

modate the ever-changing threat landscape. This phase should be repeated and the model should be updated whenever the known lists of bad IP addresses are altered significantly.

The main inputs to this phase are the lists of known bad IP addresses and their corresponding attributes from intelligence sources. As shown in Figure 5.1, this phase has 3 major tasks, each performed by a separate module in our implementation. The first task is attribute extraction which gathers intelligence about each IP address in the ingested list of bad IP addresses. Let $AttributeList = \{attr_1, attr2, ..., attr_i, ..., attr_n\}$ be the list of n attributes we extract for each IP address. Current DAbR implementation extracts the following attributes about each IP address.

1. Autonomous System Number (ASN)

2. Internet Service Provider (ISP)

3. Country where the IP is registered

4. Usertype (residential or commercial user)

5. Country where the IP is located

6. Subdivision in the country where the IP is located

7. City in the country where the IP is located

We represent each IP address $X = \langle x_1, x_2, x_3, .., x_i, .., x_n \rangle$ as a vector of these n attributes, where x_i corresponds to the value of i^{th} attribute in the $AttributeList$. Most of the IP address attributes, x_i's, are nominal in nature and does not have a linear relationship between them. For example, the attribute '$CountryLocation$' takes values like '$country_a$', '$country_b$', etc. If we represent these values in a one-dimensional space, their relative positioning does not provide any valuable information due to their nominal nature. Hence, our second task in the model generation phase is to project such nominal values to a new n dimensional space such that their relative positioning is directly related to their reputation. We use normalized frequency to generate such a reputation scale for each nominal IP address attribute. Let $UX_i = ux_{i,1}, ux_{i,2}, .., ux_{i,j}, .., ux_{i,m}$ be the m unique nominal values possible for an attribute $attr_i$. The $NormalizedFrequency(NF)$ for each unique attribute $ux_{i,j}$ is defined as described in Eq: 5.1.

$$\begin{aligned} &NF(ux_{i,j}) = N_{ux_{i,j}} / N_{total} \\ where,& \\ &N_{ux_{i,j}}\text{: \textit{number of IP addresses having} } ux_{i,j} \\ &\qquad\qquad \text{\textit{as an attribute value}} \\ &N_{total}\text{: \textit{total number of IP addresses}} \end{aligned} \tag{5.1}$$

The normalized frequency, $NF(ux_{i,j})$, always lies in the range $[0, 1]$ and their relative positioning has the following semantics: the attributes which occur more frequently in the input list have a higher value of normalized frequency (closer to 1) compared to the ones that occurred less frequently (closer to 0). This implies that higher the value of $ux_{i,j}$, the more frequently it is present among the attributes of bad IP addresses. Conversely, if the value is closer to 0, we have not seen this attribute frequently among that list of known bad IP attributes. We hypothesize that larger the value of $NF(ux_{i,j})$, the larger is its tendency to indicate a malicious activity.

Algorithm 1 Model Generation Phase

```
    Input:
        $BlackListedIPList$
    Output:
        $ReputationModel$

1: badIPVectorArray = getAttributes(BlackListedIPList)
2: for i in badIPVectorArray do
3:     allUniqueAttributes = getUniqueAttributes[i]
4:     for j in allUniqueAttributes do
5:         NF = GetNormalizedFrequency[j] using Eq: 5.1
6:         append NF to ReputationModel
7: return ReputationModel
```

The final task in this phase is model generation. A DAbR $ReputationModel$ is an aggregation of the normalized frequencies for all nominal attributes present among the bad IP attributes. This module now generates $NF(ux_{i,j})$ for each nominal attribute $attr_i \in AttributeList$ and packs them into the model. The generated $ReputationModel$ is used by the reputation scoring phase for real-time scoring of unknown IP addresses. An algorithmic representation of this phase is presented in Algorithm 1.

5.1.2 Reputation Scoring Phase

The objective of the reputation scoring phase is to associate a numeric reputation score with an IP address based on the generated model. Whenever an IP address is available, DAbR associates a reputation to it in 3 steps as shown in Figure 5.1. First, DAbR looks up external IP intelligence sources and collects attributes about the IP address. It is represented as a vector $X' = \langle x'_1, x'_2, x'_3, ..., x'_n \rangle$ where x'_i corresponds to the value of $attr_i \in AttributeList$. The second step is offline model lookup in which X' is projected to the new n dimensional space and generate a new vector $NX' = \langle nx'_1, nx'_2, ..., nx'_i, ..., nx'_n \rangle$ using the $ReputationModel$ generated in the Model Generation Phase. Each nx_i is the normalized frequency of x'_i corresponding to $attr_i$ from $ReputationModel$. If the value x'_i is not present in the $ReputationModel$, the value is taken as 0 because we have never seen that attribute among the reported bad IP address attributes.

This new vector NX' is a point on the new n dimensional vector space. Since we use the normalized frequency value, the new vector space has the property that *closer the point is to the origin, the lesser we have seen it in the bad IP address; or conversely, farther the point is from the origin, the vector is more likely similar to some of the known bad IP address*. We use this property in the scoring module to generate a reputation score for each IP address. The '$DAbRScore$' or the reputation score generated by DAbR for an IP address is the inverse of the euclidean distance of NX' from the origin in the new n dimensional space. It is calculated using Eq: 5.2. The operation of this module is presented in Algorithm 2.

$$ED = \sqrt{ux_{1,j}^2 + ux_{2,j}^2 + ... + ux_{n,j}^2}$$

$$DAbRScore = (1.0 - (ED/ED_{max})) * 10 \tag{5.2}$$

$where,$

ED : *Euclidean Distance from origin*

ED_{max}: *Maximum value of ED computed*

Algorithm 2 Reputation Scoring Phase

```
    Input:
        $ReputationModel$, $liveIPList$
    Output:
        $DAbRScoreList$

1: liveIPVectorArray = getAttributes(liveIPList)
2: for i in liveIPVectorArray do
3:     DAbRVector = generateEuclideanVector[i]
4:     DAbRScore = getScore(DAbRVector) using Eq: 5.2
5:     append DAbRScore to DAbRScoreList
6: return DAbRScoreList
```

5.2 Reputation Score Interpretation

The DAbR score of an IP address, defined in Eq: 5.2, will always be a value in the range 0 - 10. DAbR scores closer to 0 implies very low reputation and those closer to 10 implies very good reputation. Mathematically, the score is inversely proportional to the euclidean distance of IP vector from origin in the new vector space. The vector components are the normalized frequencies of occurrences of their respective attributes in the bad IP attribute list. Hence higher values indicate higher similarity toward IP addresses in bad lists. If we map all bad IP addresses (from the bad list used for model generation) into the vector space, they will not be closer to the origin. Similarly, if the euclidean distance of the IP vector is very small (closer to origin), we will get a higher reputation score.

5.2.1 Example Scenario

Consider an example scenario with 1000 bad IP addresses and 3 attributes - Country, ASN, and, ISP. Let's assume that 500 IP's have Country as $CountryX$ and 250 of these 500 IP's have ISP as $ISPX$. Now when we generate the model, $CountryX$ will get a very high NormalizedFrequency value of 0.5 ($500/1000 = 0.5$) and $ISPX$ will get a NormalizedFrequency value of 0.25 ($250/1000 = 0.25$). Figure 5.2 depicts a representative figure of the new 3-dimensional state space generated by the Model Generation phase. If we map all the IP's with country $CountryX$ and ISP $ISPX$ from the training set, they will be positioned farther from the origin as shown. All those points will get very low reputation scores by default using the Eq: 5.2. When a new IP address is available for finding reputation, its attributes are fetched from the sources. Let's assume the new IP address is from country $CountryX$ and ISP $ISPX$. Now when this IP address is mapped to the euclidean space, it will be positioned farther from the origin and it will get a lower reputation. On the other hand, if the IP has attributes which are not at all present in any of the bad IP attributes, it will be placed at the origin, because our threat intelligence has not found any malicious activities corresponding to them. DAbR will give a very high reputation score for such IP addresses.

Chapter 6

USE-CASE SCENARIOS AND EVALUATION

Our framework helps to enable efficient deep packet inspection using various organizational policies. To demonstrate its capabilities, we created a virtual corporate network with several employees and user roles. We populated our knowledge graph using Policy Management module of the Policy Engine. First, we added information about the employees in our virtual corporation. In our setup, all the managerial employees are mapped as instances of *SuperUserEntity* class, developers as instances of *InterimPrivilegedEntity* class, and contract employees as instances of *LeastPrivilegedEntity* class. We used the list of known IP blacklists from hpHosts[1] and some whitelists from IP addresses corresponding to top domains from OpenDNS[2] as the instances of *BlackListedEntity*, and *WhiteListedEntity* classes respectively. The information about the different flows and messages are inserted into knowledge graph during runtime.

6.1 Access Control Policies

We defined several access control policies to our use-case scenario using SWRL rules. Some of them are presented below:

---

[1]https://www.hosts-file.net/
[2]https://github.com/opendns/public-domain-lists

```
Flow (?flow) ^ hasSourceIPAddress (?flow, ?srcIP) ^ hasIPAddress
   (?srcUser, ?srcIP) ^ GuestEntity (?srcUser) ^
   hasServiceRequest (?flow, ?service) ^ hasServiceName
   (?service, ``FileTransfer'') ^ AccessDecision (?decision) ^
   hasAccessDecisionName (?decision, ``MandatoryInspection'') ->
   hasAccessDecision (?flow, ?decision)
```

The policy defined above using SWRL suggests that *Guest users should be monitored if they are making any "FileTransfer" requests*. A variety of attributes may decide if a message is requesting a "FileTransfer" service, such as, the protocol being FTP/SFTP, connections using well-defined port number 21, or SNI/IP address pointing to a well-known file-server, etc. In our proposed architecture, the administrators can specify simple policies as presented above and the reasoner will automatically infer whether the connection is requesting a "FileTransfer" using the knowledge graph. Hence, if a guest user tries to access, say, a known file-server, it first infers the connection as a "FileTransfer" request. Since the connection initiated from a guest user, the policy engine will generate the access decision as *MandatoryInspection*. The policy enforcement module will then contact the user client agents, extract the specific session key from internal user, and send the traffic for further inspection.

```
Flow (?flow) ^ hasSourceIPAddress(?flow, ?srcIP) ^
   hasIPAddress(?srcUser, ?srcIP) ^
   hasDestinationIPAddress(?flow, ?destIP) ^
   hasIPAddress(?destUser, ?destIP) ^ SuperUserEntity (?srcUser)
   ^ GreyListedEntity(?destUser) ^ hasReputationScore(?destUser,
   ``low'') ^ AccessDecision(?decision) ^
   hasAccessDecisionName(?decision, "OptionalInspection") ->
   hasAccessDecision(?flow, ?decision)
```

The policy presented above gives additional privileges to users who belong to the *SuperUserEntity* class even though the external resource is not completely trustworthy. As per the above policy, if a *SuperUserEntity* is trying to access an external user with low reputation score, the policy engine will decide to perform *OptionalInspection*. The assumption is that the *SuperUserEntity* users can determine if the connection is spurious or not, for instance, the security team who wants to download malware samples for analysis. In our implementation, if such a request comes from the security team, the above policy will generate an *OptionalInspection* because the requesting user is inferred as a *SuperUserEntity*. In this scenario, the policy enforcement module will request the user client agent for the session keys and depending on the response from the user, the connection will proceed further. For other users, however, the decision may be different depending on other attributes.

```
Flow(?flow) ^ InterimPrivilegedEntity(?srcUser) ^
   ConfidentialResource(?destUser) ^ hasSourceIPAddress(?flow,
   ?srcIP) ^ hasIPAddress(?srcUser, ?srcIP) ^
   hasDestinationIPAddress(?flow, ?destIP) ^
   hasIPAddress(?destUser, ?destIP) ^ AccessDecision(?decision)
   ^ hasAccessDecisionName(?decision, "MandatoryInspection") ->
   hasAccessDecision(?flow, ?decision)
```

The above policy specifies another scenario where an *InterimPrivilegedEntity* is trying to access a *ConfidentialResource*. According to this policy, the access decision should be *MandatoryInspection*. A software developer trying to access user credentials stored in a secure server is one such use-case. This scenario mandates extra inspections because of the sensitive contents in the resource.The policy engine now generates *MandatoryInspection* that will trigger the policy enforcement to retrieve the session keys and further inspection.

6.2 Evaluation of DAbR

DAbR associates a reputation score for unknown IP addresses using a model generated from known bad IP address lists. In this section, we evaluate the performance of DAbR using real datasets. We created a dataset of about 87k IP addresses to evaluate DAbR. Sections 6.2.1 and 6.2.2 discuss the new dataset, its characteristics, and DAbR's evaluation using this dataset.

6.2.1 Data Collection

As discussed in Section III, the proposed technique is performed in two phases; a model generation phase and a reputation scoring phase. Bad IP addresses and their attributes are required for model generation. We also require another combined set of bad and good IP addresses for evaluating DAbR.

For bad IP addresses, we used known sources like *Talos IP blacklist feeds*[3]. To improve the total number of IP addresses, we also downloaded around 100k blacklisted domains from services like hpHosts[4] (hpHosts is a hosts file for Windows that allows protection against access to spammer, scammer, pornographic, spoofed and malicious websites.) and converted them to IP addresses. Unlike bad IP addresses available directly from intelligent sources, the good IP addresses are harder to collect. We downloaded a list of top domains from services like OpenDNS[5] and top one million ranked websites as ranked by Alexa[6]. We also performed a preliminary analysis to verify if these domains are men-

[3]`http://talosintel.com/feeds/ip-filter.blf`
[4]`https://www.hosts-file.net/`
[5]`https://github.com/opendns/public-domain-lists`
[6]`http://stuffgate.com/stuff/website`

tioned among the blacklisted domains collected from hpHosts. These cleaned domains are then converted to IP addresses to get good IP addresses. In total, we gathered around 100k blacklisted IP addresses and 20k good IP addresses. We collected less number of non-black listed IP addresses because they are not being used for model generation and are only used for evaluation.

The next step is the collection of attributes corresponding to each IP address. We used services like Maxmind GeoIP[7] (Maxmind is an IP Intelligence tool that provides various network and geographical features of user requested IP addresses) databases for collecting various attributes. The collected attributes include Autonomous System Number (ASN), Internet Service Provider (ISP), IP registration country, usertype (residential/commercial user), IP location country, subdivision, and cityname. We faced many challenges during data collection and cleaning. First, attributes were not directly available for many bad IP addresses. Some of these IP addresses were not in service also. Second, the intelligence sources provided limited information about many IP addresses owing to the fact that some providers do not use standard formats for publishing their information. After the attribute collection and data cleaning, the final dataset consisted of 70635 bad IP addresses, 17101 good IP addresses, and attributes associated with each of these collected IP addresses.

6.2.2 Results

We evaluate DAbR using the newly generated dataset of bad and good IP addresses. For this, we convert the reputation scoring task to a classification task using a threshold. The classification task is to identify a given IP address as a bad IP address or a good IP address. First, a DAbR score is calculated using Eq: 5.2 for each input IP address after generating a model described in section 5.1.1. If the score falls below a specific threshold,

[7] `https://www.maxmind.com/en/home`

$ReputationScore_{thr}$, it is detected as bad. Otherwise, it is detected as good.

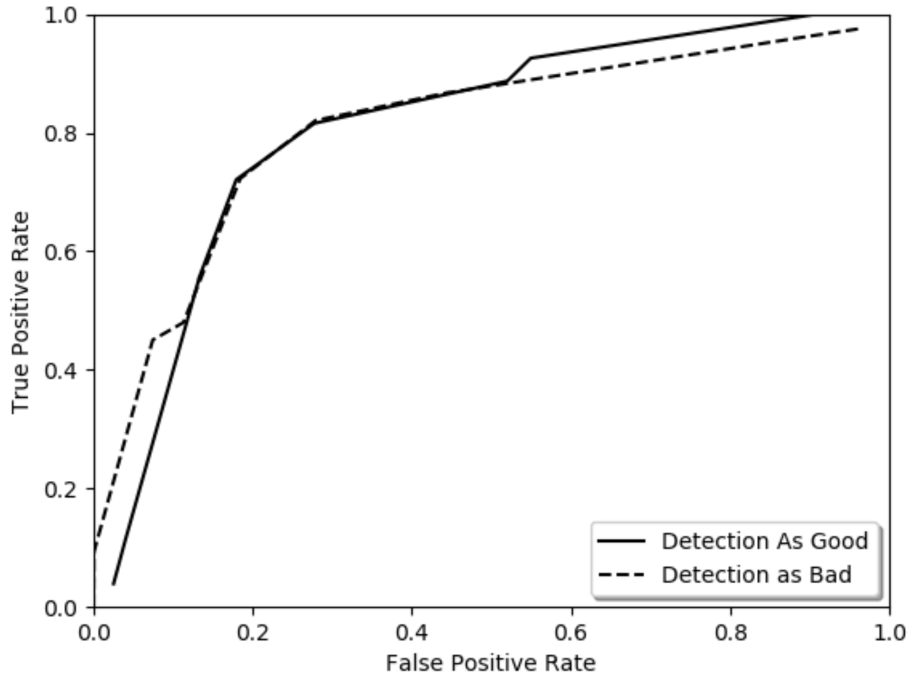

FIG. 6.1: ROC Curve

We used a modified 4 fold cross validation for evaluation. In each fold of a traditional 4 cross validation, 3 out of 4 parts of the input data is used for model generation and the remaining part is used for testing. However, DAbR uses only bad IP address's attributes for model generation. Hence in our evaluations, for each of the 4 folds in the 4 cross validation, we use 3 out 4 parts of the input bad IP dataset for model generation. For reputation scoring, we used a union of the remaining one part of the bad IP dataset and the complete set of good IP's. The Receiver Operating Characteristic (ROC) curve, Figure 6.1, is plotted using the average of true positive rates and false positive rates across the 4 folds in cross validation. In this figure, we plotted the performance of DAbR by varying the $ReputationScore_{thr}$ from 0 to 10. Two ROC curves are depicted in Figure 6.1. The solid line represents the ROC curve, when we consider True Positive as "Good IP detected as

$ReputationScore_{thr}$	Precision	Recall	Accuracy	F-1 Score
0.0	0.491	1	0.491	0.659
2.0	0.516	0.999	0.538	0.680
4.0	0.740	0.816	0.768	0.776
6.0	0.802	0.554	0.713	0.655
8.0	0.733	0.113	0.543	0.196

Table 6.1: Evaluation metrics for thresholds 0.0 - 8.0

$ReputationScore_{thr}$	Precision	Recall	Accuracy	F-1 Score
4.4	0.740	0.815	0.768	0.776
4.5	0.741	0.815	0.768	0.776
4.6	0.749	0.810	0.773	0.779
4.7	0.752	0.809	0.775	0.779
4.8	0.754	0.807	0.776	0.780
4.9	0.756	0.793	0.773	0.774
5.0	0.795	0.721	0.771	0.756

Table 6.2: Evaluation metrics for thresholds 4.4 - 5.0

Good" and False Positive as "Bad IP detected as Good". The dotted lines represent the ROC curve when True Positive is "Bad IP detected as Bad" and False Positive is "Good IP detected as Bad". The ROC curve shows that the classification task performs well with a True Positive rate of around 80% along with a False Positive rate of below 20%. Other classic performance metrics for a classification task like F-1 score, accuracy, precision, and recall are also reported. Table 6.1 reports these metrics for $ReputationScore_{thr}$ from 0.0 to 8.0. Classification metrics for more granular values of $ReputationScore_{thr}$ from 4.4 to 5.0 is reported in Table 6.2, where we achieve maximum F-1 scores. As seen from the tables 6.1 and 6.2, the maximum value of F-1 score = 78.034% is seen at a threshold of 4.8. At this threshold for reputation scoring, the average true positive rate is about 76.5% and the average false positive rate is about 22%. Also, Precision = 75.45%, Recall= 80.79% and accuracy of detection is 77.6% at threshold of 4.8. The confusion matrix at this threshold is

		Detection outcome		
		Good IP	Bad IP	Total
Actual Value	Good IP	55268	13136	68404
	Bad IP	17979	52656	70635
	Total	73247	65792	

Table 6.3: Confusion Matrix

presented in Table 6.3. In this confusion matrix, the total number of IP's is the cumulative sum across all the 4 folds in our cross validation.

In a large enterprise with a very large number of connections, a 20% false positive rate is high. However, it should be noted that DAbR is not a classification technique. The main task of DAbR is to quickly associate a numeric value corresponding to every incoming IP address, otherwise considered normal. The numeric value indicates how similar or dissimilar the IP address is to the existing blacklisted IP addresses. As DAbR is designed to work in conjunction with a policy enforcement module, instead of detecting it as either Good or Bad, it delegates the response action to the enforcement module. In existing systems, 100% of incoming packets need to be sent for further investigation. With the introduction of TLS 1.3 and wider adoption of encryption, techniques like deep packet inspection will be highly costly and privacy-invasive because it could involve actual decryption. The DAbR score enables to reduce this cost by avoiding such inspections for a large share of packets according to a predefined policy. This would be a very substantial performance improvement and allows penalizing only suspicious connections rather than everyone connected.

Chapter 7

CONCLUSION

In this thesis, we proposed an ABAC (Attribute-Based Access Control) policy framework that supports existing Deep Packet Inspection tools in ‘Perfect Forward Secrecy’ implementation like TLS 1.3. Our framework also respects user’s privacy requirements and organizational policies. In our framework, various observed and derived attributes of the network connections are evaluated against user access privileges defined using semantic technologies. We implemented a prototype system for our framework and demonstrated the efficacy of our technique with the help of meaningful use-case scenarios. We developed DAbR, a lightweight reputation scoring system for IP addresses which associate a numeric reputation score for each IP address in network traffic. DAbR may also be integrated with a policy enforcement module that filter malicious network traffic in a more efficient and limited privacy-invasive way. The DAbR score is representative of the IP address’s trustworthiness based on the existence of its attributes in known malicious attributes. To evaluate DAbR, we aggregated a dataset of 87k IP addresses and their attributes. We used a threshold on DAbR score to classify good/bad IP addresses and observed a classification F-1 score of 78% with selected attributes. The results demonstrate DAbR score’s usefulness in separating network traffic into different classes. IP’s with bad reputation scores should be sent for detailed evaluation while traffic corresponding to good reputation can pass through without penalization.

7.1 Future Work

Security intelligence is a highly dynamic domain, and new intelligence is being pushed by a variety of structured and semi-structured sources daily. In the future, we plan to ingest knowledge from such sources to automatically generate policies and adapt to the modified landscape. An easy extension to our system is to allow policies which can use decisions from the DPI techniques it supports. An example is when the DPI technique identifying a malicious connection and blocking the connection once this decision is made. Yet another direction we can do with minimal efforts is to integrate information from other known knowledge-graphs similar to CKG[1] and and UCO [35] (Unified CyberSecurity Ontology). Since, our policy system is driven by W3 standard semantic technologies such integration requires minimal efforts from the software coding perspective.

[1]`http://eb4.cs.umbc.edu:9090/`

Appendix A

CODE

A.1 Network Monitoring Engine

```
from query_sparql import *
from gateway_keyRequest import *
from sparql import *
import time
import datetime
import os
import json
from mitmproxy import http

monitoring_conns = {}

def inspection(DestIP, monitoring_conns, curr_key, SourceIP, acc):
  filter_cmd = "tcpdump -r test.pcap -w outTemp_"+ DestIP +
     ".pcap host " + DestIP
  os.system(filter_cmd)
  time.sleep(3)
```

```
create_json_cmd = "tshark -r outTemp_" + DestIP + ".pcap -T
   json > outTemp_" + DestIP + ".json"
os.system(create_json_cmd)
time.sleep(3)
with open ("outTemp_" + DestIP + ".json" ) as json_file:
  data = json.load(json_file)

  for each_packet in data:
    try:
      random_str =
         each_packet["_source"]["layers"]["ssl"]["ssl.record]\\
         ["ssl.handshake"]["ssl.handshake.random"]
      packet_dest =
         each_packet["_source"]['layers']['ip']['ip.dst']
      if packet_dest == DestIP:
        crandom = random_str.replace(":", "")
        if not (crandom in
           monitoring_conns[curr_key]['ClientRandom']):
          outfil = "output_folder/" + curr_key + ".ssl"
          if not monitoring_conns[curr_key]['IsFirst']:
            create_connection(SourceIP, DestIP, 'MAND',
               crandom, outfil, monitoring_conns, curr_key)
          else:
            create_connection(SourceIP, DestIP, acc, crandom,
               outfil, monitoring_conns, curr_key) #acc =
               'MAND' for mandatoryinspection ; OPTL for
               optional
```

```
              monitoring_conns[curr_key]['IsFirst'] = False
            monitoring_conns[curr_key]['ClientRandom'].append(crandom)
      except:
        continue

def request(flow: http.HTTPFlow):
  cr = flow.client_conn.connection.client_random()
  crandom = str(cr.hex())
  ts= time.time()
  ts_str =
     datetime.datetime.fromtimestamp(ts).strftime('%Y-%m-%d_%H:%M:%S')
  sni = flow.server_conn.sni
  DestIP = flow.server_conn.ip_address[0]
  ipv6src = flow.client_conn.ip_address[0]
  ipv6src_s = ipv6src.split(":")
  SourceIP = ipv6src_s[len(ipv6src_s) - 1]
     #flow.server_conn.source_address[0]
  Flowid = "Flowgit" + ts_str
  Message = "Message" + ts_str
  protocol = "HTTP"
  inp = {"Flow":Flowid, "Message":Message, "SourceIP":SourceIP,
     "DestIP":DestIP, "Protocol":protocol}
  print (inp)
  curr_key = SourceIP + "_"+ DestIP + "_" + protocol
  outfil = "output_folder/" + curr_key + ".ssl"
  curr_access_decision = ''
```

```
# Getting Access Decision
if True:
  all_mon_conns = monitoring_conns.keys()
  if curr_key in all_mon_conns:
    # If the key is found in the monitoring set
    curr_acccess_decision =
        monitoring_conns[curr_key]['AccessDecision']

  else:
    # Starting to see if we need to monitor new connection
    query = GenerateQuery(1, inp)
    InsertIndividualVariable(query)
    curr_access_decision = Query(Flowid)
    if curr_access_decision:
      print ("Current Access Decision is " + curr_access_decision)
    else:
      print ("Current Access Decision is None")
    monitoring_conns[curr_key] = {}
    #curr_access_decision = 'AllowConnection'
    monitoring_conns[curr_key]['AccessDecision'] =
        curr_access_decision
    monitoring_conns[curr_key]['ClientRandom'] = []
    monitoring_conns[curr_key]['IsFirst'] = True
    monitoring_conns[curr_key]['OPTLResult'] = False
    monitoring_conns[curr_key]['OPTLResultSet'] = False

# Depending on access decision we will decide to monitor or not
```

```
if curr_access_decision == "MandatoryInspection":
  print ("Got Mandatory Inspection")
  if not (crandom in
     monitoring_conns[curr_key]['ClientRandom']):
    create_connection(SourceIP, DestIP, 'MAND', crandom,
       outfil, monitoring_conns, curr_key)
    monitoring_conns[curr_key]['ClientRandom'].append(crandom)
  else:
    print ("Session Key already Exist")

if curr_access_decision == "BlockConnection":
  flow.response = http.HTTPResponse.make(418, b"The current
     action you have done is against the organizational
     policies. Please review it!. ",)

if curr_access_decision == "OptionalInspection":
  print ("Got Optional Inspection")
  if monitoring_conns[curr_key]['OPTLResultSet']:
    if monitoring_conns[curr_key]['OPTLResult']:
      if not (crandom in
         monitoring_conns[curr_key]['ClientRandom']):
        create_connection(SourceIP, DestIP, 'MAND', crandom,
           outfil, monitoring_conns, curr_key)
        monitoring_conns[curr_key]['ClientRandom'].append(crandom)
        # If it is a new random and the user already said monitor
      else:
        print ("Optional but key exist")
    else:
      print ("Optional but user said no")
  else:
    print ("Optional but need to ask user")
    create_connection(SourceIP, DestIP, 'OPTL', crandom,
        outfil, monitoring_conns, curr_key)
    if monitoring_conns[curr_key]['OPTLResult']:
      print ("abc.py: User asked to monitor")
    else:
      print ("abc.py: User asked NOT to monitor")

if curr_access_decision == "AllowConnection":
  print ("Got AllowConnection")
```

A.2 Client User

```
import csv
import socket
import sys
from ast import literal_eval as make_tuple

IP_address = "10.0.2.15"
byte_count = 1024

def send_key_optional(conn, random, destIP):
    print ("\n\nYou have made a connection to "+ destIP )
    inp = input("Proceed with inspecting the connection? (Y/N): ")
```

```
   if inp.upper() == ’Y’:
      send_key(conn, random)
   elif inp.upper() == ’N’:
      conn.sendall(bytes("\00", ’ascii’))

def send_key_mandatory(conn, random):
   send_key(conn, random)

def send_key(conn, random):
   filename = ’sslkeylogfile.log’
   f = open(filename,’r’)
   for line in f:
      if random in line:
         conn.send(line.encode(’ascii’))
         print(’Sent ’,repr(line))
   f.close()
   print (’Done sending’)

def recv_connection(sock):
   # Wait for a connection
   print ( ’Waiting for a connection’)
   conn, client_address = sock.accept()
   print ( ’Got connection from ’ , client_address)
   # Receive the data in small chunks and retransmit it
   data = conn.recv(byte_count)
   print(’Data received’, repr(data))
   data1 = data.decode(’ascii’).split(",")
   destIP = data1[1]
   access_decision = data1[2]
   random = data1[3]
   if access_decision == 'MAND':
      send_key_mandatory(conn, random)
   elif access_decision == 'OPTL':
      send_key_optional(conn, random, destIP)
   conn.close()


def create_connection():
   # Create a TCP/IP socket
   sock = socket.socket()#socket.AF_INET, socket.SOCK_STREAM)
   # Bind the socket to the port
   server_address = (IP_address, 10000)
   print ( 'starting up on %s port %s' %server_address)
   sock.bind(server_address)
   # Listen for incoming connections
   sock.listen(5)

   while True:
      recv_connection(sock)


create_connection()
```

A.3 DAbR

```
import sys
```

```
#from sets import Set
import csv
import pickle
no_of_attr = 7


def create_vector(categories, data, ip, no_of_attr):
  ret_data = []
  for i in range(no_of_attr):
    try:
      ret_data.append(categories[i][data[ip][i]])
    except:
      ret_data.append(0.0)
  return ret_data


def conver_categorical_to_score(data, item_no):
  keys = data.keys()
  item = set()
  for i in keys:
    item.add(data[i][item_no])
  ret = {}
  for i in item:
    ret[i] = 0
  for i in keys:
    ret[data[i][item_no]] += 1
  ret_score = {}
  for i in item:
    ret_score[i] = float(ret[i])/float(len(data))
  return ret_score



def CreateModel (data, no_of_attr, isReturn):
  normFreq = []
  for i in range(no_of_attr):
    normFreq.append(conver_categorical_to_score(data, i))
  ips = data.keys()
  vec_dict = {}
  for i in ips:
    vec_dict[i] = create_vector(normFreq, data, i, no_of_attr)
    #returns dictionary of vectors for each training IP address
  largest=0.0
  for ip, attr in vec_dict.items():
    sum_sq = 0
    for i in range(no_of_attr):
      sum_sq = sum_sq + attr[i] * attr[i]
    dist = sum_sq ** (1/2.0)
    if dist > largest:
      largest = dist
  model = {}
  model["normFreq"] = normFreq
  model["largest"] = largest
  model["vectors"] = vec_dict
  if isReturn:
    return model
  with open("DAbR_data/DAbR_model.pkl", 'wb') as f:
    pickle.dump(model, f)


def get_DAbRScore(realVector, no_of_attr, largest):
  sq_sum = 0
  for i in range(no_of_attr):
    sq_sum = sq_sum + (realVector[i] * realVector[i])
  DAbRScore = 10 - ( (sq_sum ** (1/2.0)) * 10 / largest)
  return DAbRScore


def readModel():
  with open("DAbR_data/DAbR_model.pkl", 'rb') as f:
    model = pickle.load(f)
  return model


def getReputationScore(IP, real_IP_data, model, no_of_attr):
  normFreq = model["normFreq"]
  largest = model["largest"]
  realVector = create_vector(normFreq, real_IP_data, IP,
     no_of_attr)
  DAbRScore = get_DAbRScore(realVector, no_of_attr, largest)
  return DAbRScore


def get_realIP_data(IP, train_data, real_data):
  train_IP = train_data.keys()
  real_IP = real_data.keys()
  IP_dict = {}
  if IP in train_IP:
    IP_dict[IP] = train_data[IP]
  elif IP in real_IP:
    IP_dict[IP] = real_data[IP]
  else:
    attr_list = []
    for i in range(no_of_attr):
      attr_list.append("null")
    IP_dict[IP] = attr_list

  return IP_dict

def create_data_list(filename):
  con = []
  fil = open(filename, 'r', encoding = "ISO-8859-1")
  for i in fil:
    con.append(i.strip('\n').strip('\r'))
  dat1=[]
  for l in csv.reader(con, quotechar='"', delimiter=',',
     quoting=csv.QUOTE_ALL, skipinitialspace=True):
    dat1.append(l)
  dat =dat1[1:]
  return dat

def create_data_dictionary(X_train):
  data ={}
  for i in X_train:
    app = []
    for j in range(1, len(i)):
      app.append(i[j])
    data[i[0]] = app
    no_of_attr = len(app)
  return data, no_of_attr

isnewModel = False

def DAbR_pipeline(IP):
  black_data, no_of_attr =
     create_data_dictionary(create_data_list(sys.argv[1]))
  if isnewModel:
    CreateModel(black_data, no_of_attr, False)

  white_data, no_of_attr =
     create_data_dictionary(create_data_list(sys.argv[2]))
  model = readModel()
  real_IP_data = get_realIP_data(IP, black_data, white_data)
  DAbRScore = getReputationScore(IP, real_IP_data, model,
     no_of_attr)
  print (DAbRScore)

if __name__== "__main__":
  main()
```

REFERENCES

[1] "Google transparency report: Https encryption on the web," https://transparencyreport.google.com/https/overview?hl=en, accessed: 2019-03-27.

[2] "Alexa top 1 million analysis - august 2018," https://scotthelme.co.uk/alexa-top-1-million-analysis-august-2018/, accessed: 2019-03-27.

[3] "Most cyber attacks now use encryption: Are you prepared for the good, the bad and the ugly?" https://www.venafi.com/blog/most-cyber-attacks-now-use-encryption-are-you-prepared-for-good-bad-and-ugly, accessed: 2018-06-22.

[4] "February 2018 zscaler ssl threat report," https://www.zscaler.com/blogs/research/february-2018-zscaler-ssl-threat-report, accessed: 2018-06-20.

[5] "Encrypted malware: a threat facilitated by the gdpr?" https://www.pandasecurity.com/mediacenter/malware/encrypted-malware-facilitated-gdpr/, accessed: 2019-03-27.

[6] "Sloth: Tls 1.2 vulnerability (cve-2015-7575)," https://access.redhat.com/articles/2112261, accessed: 2019-03-28.

[7] "Poodle ssl vulnerability now attacking tls security protocol," https://thehackernews.com/2014/12/SSL-Poodle-TSL-attack.html, accessed: 2019-03-28.

[8] "What is perfect forward secrecy?...and what does it mean for you?" https://www.extrahop.com/company/blog/2017/what-is-perfect-forward-secrecy/, accessed: 2019-03-27.

[9] “The impact on network security through encrypted protocols tls 1.3,” https://blogs.cisco.com/security/the-impact-on-network-security-through-encrypted-protocols-tls-1-3, accessed: 2019-03-27.

[10] B. Anderson, S. Paul, and D. McGrew, “Deciphering malwares use of tls (without decryption),” *Journal of Computer Virology and Hacking Techniques*, pp. 1–17, 2016.

[11] X. Jin, R. Krishnan, and R. Sandhu, “A unified attribute-based access control model covering dac, mac and rbac,” in *Data and Applications Security and Privacy XXVI*, N. Cuppens-Boulahia, F. Cuppens, and J. Garcia-Alfaro, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 41–55.

[12] M. Ed-Daibouni, A. Lebbat, S. Tallal, and H. Medromi, “A formal specification approach of privacy-aware attribute based access control (pa-abac) model for cloud computing,” in *2016 Third International Conference on Systems of Collaboration (SysCo)*, Nov 2016, pp. 1–5.

[13] E. Yuan and J. Tong, “Attributed based access control (abac) for web services,” in *IEEE International Conference on Web Services (ICWS’05)*, July 2005, p. 569.

[14] M. Gupta, J. Benson, F. Patwa, and R. Sandhu, “Dynamic groups and attribute-based access control for next-generation smart cars,” in *Proceedings of the Ninth ACM Conference on Data and Application Security and Privacy*, ser. CODASPY ’19. New York, NY, USA: ACM, 2019, pp. 61–72. [Online]. Available: http://doi.acm.org/10.1145/3292006.3300048

[15] P. K. Das, S. Narayanan, N. K. Sharma, A. Joshi, K. Joshi, and T. Finin, “Context-sensitive policy based security in internet of things,” in *2016 IEEE International Conference on Smart Computing (SMARTCOMP)*. IEEE, 2016, pp. 1–6.

[16] B. Lang, I. Foster, F. Siebenlist, R. Ananthakrishnan, and T. Freeman, "A flexible attribute based access control method for grid computing," *Journal of Grid Computing*, vol. 7, no. 2, p. 169, Nov 2008. [Online]. Available: https://doi.org/10.1007/s10723-008-9112-1

[17] S. Berger, A. Vensmer, and S. Kiesel, "An abac-based policy framework for dynamic firewalling," in *ICSNC 2012*, 2012.

[18] M. Burmester, E. Magkos, and V. Chrissikopoulos, "T-abac: An attribute-based access control model for real-time availability in highly dynamic systems," in *2013 IEEE Symposium on Computers and Communications (ISCC)*, July 2013, pp. 000 143–000 148.

[19] C. Basile, A. Lioy, S. Scozzi, and M. Vallini, "Ontology-based security policy translation," *Journal of Information Assurance and Security*, vol. 5, 01 2010.

[20] Q. J. Hu, M. Klenzak, and P. Smith, "Policy-enabled dynamic deep packet inspection for telecommunications networks," Dec. 11 2012, uS Patent 8,331,229.

[21] L. Page, S. Brin, R. Motwani, and T. Winograd, "The pagerank citation ranking: Bringing order to the web." Stanford InfoLab, Tech. Rep., 1999.

[22] S. Katkar, K. Pande, P. Patil, and Q. Ahmed, "Implementation of trust reputation system (trs) for e-commerce applications using data mining," *Imperial Journal of Interdisciplinary Research*, vol. 3, no. 4, 2017.

[23] M. El Marrakchi, H. Bensaid, and M. Bellafkih, "Scoring reputation in online social networks," in *10th International Conference on Intelligent Systems: Theories and Applications (SITA)*. IEEE, 2015.

[24] Y. Wang and J. Vassileva, "Trust and reputation model in peer-to-peer networks," in *Proceedings Third International Conference on Peer-to-Peer Computing (P2P2003)*, Sept 2003, pp. 150–157.

[25] R. Zhou and K. Hwang, "Gossip-based reputation aggregation for unstructured peer-to-peer networks," in *2007 IEEE International Parallel and Distributed Processing Symposium*. IEEE, 2007, p. 95.

[26] L. Mui, M. Mohtashemi, C. Ang, P. Szolovits, and A. Halberstadt, "Ratings in distributed systems: A bayesian approach," in *Proceedings of the Workshop on Information Technologies and Systems (WITS)*, 2001.

[27] T. K. Kim and H. S. Seo, "A trust model using fuzzy logic in wireless sensor network," *World academy of science, engineering and technology*, vol. 42, no. 6, pp. 63–66, 2008.

[28] H. Esquivel, A. Akella, and T. Mori, "On the effectiveness of ip reputation for spam filtering," in *Communication Systems and Networks (COMSNETS), 2010 Second International Conference on*. IEEE, 2010.

[29] R. Hegli, H. Lonas, and C. K. Harris, "System and method for developing a risk profile for an internet service," uS Patent 8,438,386.

[30] D. Chiba, K. Tobe, T. Mori, and S. Goto, "Detecting malicious websites by learning ip address features," in *Applications and the Internet (SAINT), IEEE/IPSJ 12th International Symposium on*, 2012.

[31] M. Antonakakis, R. Perdisci, D. Dagon, W. Lee, and N. Feamster, "Building a dynamic reputation system for dns." in *USENIX security symposium*, 2010, pp. 273–290.

[32] L. Kagal and T. Berners-Lee, “Rein: Where policies meet rules in the semantic web,” *Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA*, vol. 2139, 2005.

[33] A. Renjan, K. P. Joshi, S. N. Narayanan, and A. Joshi, “Dabr: Dynamic attribute-based reputation scoring for malicious ip address detection,” in *2018 IEEE International Conference on Intelligence and Security Informatics (ISI)*, Nov 2018, pp. 64–69.

[34] H. Elkhannoubi and M. Belaissaoui, “Assess developing countries’ cybersecurity capabilities through a social influence strategy,” in *2016 7th International Conference on Sciences of Electronics, Technologies of Information and Telecommunications (SETIT)*, Dec 2016, pp. 19–23.

[35] Z. Syed, A. Padia, T. Finin, L. Mathews, and A. Joshi, “Uco: A unified cybersecurity ontology,” in *Workshops at the Thirtieth AAAI Conference on Artificial Intelligence*, 2016.