In proceedings of IEEE International Conference on Big Data Security on Cloud, May 2019

A Policy based Framework for Privacy-Respecting Deep Packet Inspection of High Velocity Network Traffic

Arya Renjan*, Sandeep Nair Narayanan*, Karuna Pande Joshi†
*Department of Computer Science and Electrical Engineering, University of Maryland, Baltimore County
†Department of Information Systems, University of Maryland, Baltimore County
Email: {arenjan1, sand7, karuna.joshi}@umbc.edu

***Abstract*—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. Until recently, passive proxying using certain aspects of TLS 1.2 were used to perform decryption and further DPI analysis. With the introduction of TLS 1.3 standard that only supports protocols with Perfect Forward Secrecy (PFS), many such techniques will become ineffective. Several security solutions will be forced to adopt active proxying that will become a big-data problem considering the velocity and veracity of network traffic involved. 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 attributes of network connections against user access privileges using policies described with semantic technologies. In this paper, we 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 technique makes selective identification of connections based on policies, both processing and memory load at the gateway will be reduced significantly.**

***Index Terms*—Attribute-based Access Control (ABAC), Deep Packet Inspection, TLS 1.3, Perfect Forward Secrecy, Semantic Technologies, Privacy**

I. Introduction

With the unprecedented growth of the Internet, security and privacy have become critical concerns to its users. 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]. One side-effect of the adoption of encryption is its usage by malicious hackers to conceal malware from signature-based detection strategies. Gartner [3] 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 [4], POODLE [5], 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 [6] 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 [7]. 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

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

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 Cisco ETA (Encrypted Traffic Analysis)[3] that tries to identify malware without decryption. Anderson et al. [8] present a study on using TLS meta-characteristics for malware detection. Yet another competing technique is from ExtraHop Networks [6] where the session keys are retrieved from the client for decryption. In this paper, 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.

The rest of this paper is organized as follows: Section II describes a brief literature review. Section III presents our system's architecture, and Section IV discusses the implementation. Section V demonstrates the usefulness of our technique with the help of use-case scenarios followed by the conclusions and future work in Section VI.

II. 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 [9], Jin et al. investigate formal connections between these three classical models and ABAC models. ABAC techniques are used in several fields like cloud computing [10], web services [11], Internet-of-Things [12], [13], and grid computing [14].

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. [15] 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. [16] 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 [17], 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 describe our 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. [18] patented a policy enabled Deep Packet Inspection framework for telecommunications network. In contrast, our system extracts several direct and extended attributes and uses ABAC using semantic technologies with a goal of specifically supporting DPI in perfect forward secrecy implementations like the new TLS 1.3 standard.

III. System Architecture

In this section, we discuss the architecture of our policy-based framework. The main goal of our approach 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. For this purpose, we envision a multi-agent system as presented in Fig. 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.

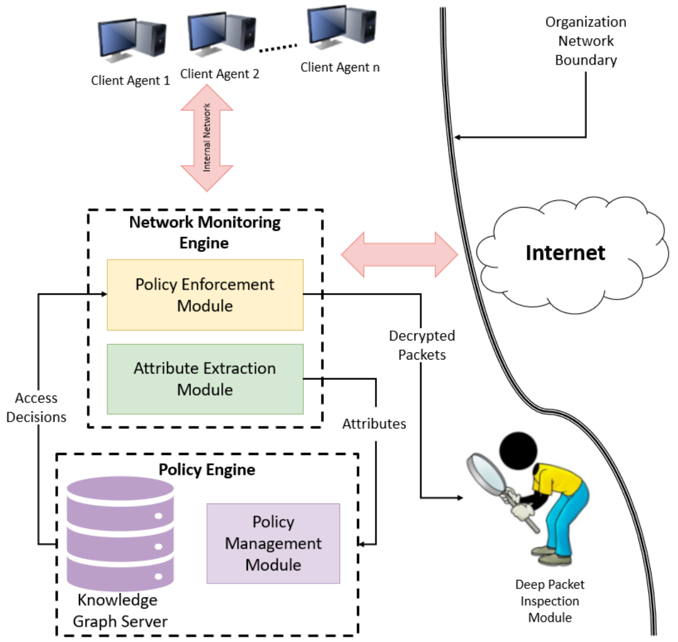

Fig. 1: System Architecture

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

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 III-A2. The decisions are then sent to the policy enforcement module of the network monitoring engine for further processing and enforcement. The policy enforcement module interacts with the client-user agents and supports efficient 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 subsections.

A. 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 module is policy enforcement module which enforces the access decisions generated by the policy engine.

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.

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:

- *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*.
- *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.
- *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.
- *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, the policy enforcement module interacts with the client user-agent and requests permission. If the user responds with 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*.

B. 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

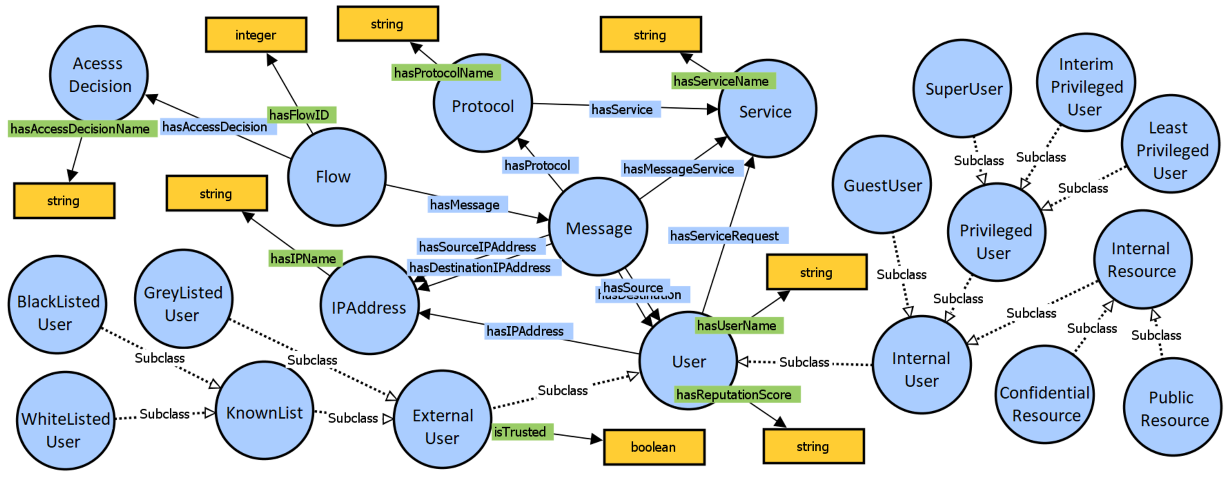

Fig. 2: Relevant extract from our knowledge graph

module as required. The client 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.

C. 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 III-A2. 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 subsections.

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. Fig. 2 presents a relevant part of the knowledge graph designed for this purpose.

Of the many classes in the knowledge graph, *User* class plays an important role while developing policies. The *User* 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 *InternalUser* (users who are part of the organizational network) and *ExternalUser* (users who are external to the network). The *ExternalUser* class is further classified into sub-classes *BlackListedUser*, *WhiteListedUser*, *GreyListedUser*, 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[4] (Semantic Web Rule Language) or using other policy specification frameworks like Rein [19]. 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 section V. However, different policies may result in conflicting 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]https://www.w3.org/Submission/SWRL/

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.

IV. Implementation

This section describes the tools and API's used in the development of our system. In our implementation, the network monitoring engine is developed using *mitmproxy*[5] 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 III-A1, our system interfaces this module to a reputation scoring engine *DAbR* [20]. *DAbR* will extract several attributes about the external IP addresses and generates a numeric reputation score. All these information are 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 system uses the RDF triple store, *Apache Fuseki*[6], 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*[7] 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.

V. Use-case Scenario

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 the 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 *SuperUser* class, developers as instances of *InterimPrivilegedUser* class, and contract employees as instances of *LeastPrivilegedUser* class. We used the list of known IP blacklists from hpHosts[8], and some whitelists from IP addresses corresponding to top domains from OpenDNS[9] as the instances of *BlackListedUser*, and *WhiteListedUser* classes respectively. Information about different flows and messages are inserted into the knowledge graph during runtime.

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

```
hasMessage (?flow, ?message) ^
    hasSourceIPAddress (?message, ?srcIP) ^
    hasIPAddress (?srcUser, ?srcIP) ^
    GuestUser (?srcUser) ^ hasServiceRequest
    (?message, ?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, SNI/IP address pointing to a well-known file-server, etc. In our 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*. Policy enforcement module will then contact the client agents, extract specific session key from the internal user, and send the traffic for further inspection.

```
hasMessage(?flow, ?message) ^
    hasSourceIPAddress(?message, ?srcIP) ^
    hasIPAddress(?srcUser, ?srcIP) ^
    hasDestinationIPAddress(?message,
    ?destIP) ^ hasIPAddress(?destUser,
    ?destIP) ^ SuperUser(?srcUser) ^
    GreyListedUser(?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 *SuperUser* class even though the external

[5]https://mitmproxy.org/
[6]https://jena.apache.org/documentation/fuseki2/
[7]https://rdflib.github.io/sparqlwrapper/
[8]https://www.hosts-file.net/
[9]https://github.com/opendns/public-domain-lists

resource is not completely trustworthy. As per the above policy, if a *SuperUser* is trying to access an external user with low reputation score, the policy engine will decide to perform *OptionalInspection*. The assumption is that *SuperUser* 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 *SuperUser*. 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 and their privileges.

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

The above policy specifies another scenario where an *InterimPrivilegedUser* 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.

VI. Conclusion and Future Work

We have developed an ABAC (Attribute-Based Access Control) policy framework that supports existing Deep Packet Inspection tools in ‘Perfect Forward Secrecy’ implementation like TLS 1.3. It respects user’s privacy requirements and organizational policies and also gives the user the ability to accept or decline the access decision based on his privileges. 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. Security intelligence is a highly dynamic domain, and new intelligence is 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.

Acknowledgment

We would like to thank Prof. Anupam Joshi, the Oros Family Professor and Chair of CSEE Department, Dr. Tim Finin, Willard and Lillian Hackerman Chair in Engineering and other members of the Ebiquity Research Lab at University of Maryland Baltimore County, for their constant guidance and support in this research.

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] “Encrypted malware: a threat facilitated by the gdpr?” https://www.pandasecurity.com/mediacenter/malware/encrypted-malware-facilitated-gdpr/, accessed: 2019-03-27.
[4] “Sloth: Tls 1.2 vulnerability (cve-2015-7575),” https://access.redhat.com/articles/2112261, accessed: 2019-03-28.
[5] “Poodle ssl vulnerability now attacking tls security protocol,” https://thehackernews.com/2014/12/SSL-Poodle-TSL-attack.html, accessed: 2019-03-28.
[6] “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.
[7] “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.
[8] 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.
[9] 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.
[10] 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.
[11] 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.
[12] 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
[13] 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.
[14] 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
[15] S. Berger, A. Vensmer, and S. Kiesel, “An abac-based policy framework for dynamic firewalling,” in *ICSNC 2012*, 2012.
[16] 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.
[17] C. Basile, A. Lioy, S. Scozzi, and M. Vallini, “Ontology-based security policy translation,” *Journal of Information Assurance and Security*, vol. 5, 01 2010.
[18] 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.
[19] 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.
[20] 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.