Assurance for CyberSecurity with Assume-Guarantee Reasoning

1st Ali Alfageeh
Computer Engineering and Sciences
Florida Institute of Technology
Melbourne, USA
aalfageeh2017@my.fit.edu

2nd Siddhartha Bhattacharyya
Computer Engineering and Sciences
Florida Institute of Technology
Melbourne, USA
sbhattacharyya@fit.edu

3rd Samuel Perl
Software Engineering Institute
Carnegie Mellon University
Pittsburg,USA
sjperl@cert.org

4th Milankumar Patel
Computer Engineering and Sciences
Florida Institute of Technology
Melbourne, USA
patelm2017@my.fit.edu

***Abstract*—Design of cybersecurity solutions for a network is a challenging problem as it involves the integration of several complex components such as routers, servers, computers, smart devices, that include not only essential networking algorithms but also security algorithms. This is further complicated by the need to have robust security policies implemented to prevent violation of confidentiality as the networked devices interact. The design of such complex networked systems demand a more rigorous approach to the modeling and analysis at a higher level of abstraction, which can be inherited from the field of software engineering for safety critical systems. Presently, network or security engineers do not use a system/software engineering approach to design and build cybersecurity systems. Thus, we propose a software engineering approach to model and analyze cybersecurity, that along with identification of concrete requirements enables automated analysis to guarantee cybersecurity properties are satisfied within the network. In this research paper, we discuss an ontology guided *assume-guarantee* based formal approach to model and analyze a designed network.**

***Index Terms*—cybersecurity, assurance, design analysis, ontology, confidentiality**

I. Introduction

Research in cybersecurity emphasizes on designing security algorithms and protocols to preserve security properties such as, confidentiality, privacy, integrity, authentication, authorization without utilizing a systemic approach to model and analyze composition of a network at a higher level of abstraction. Research by Schieferdecker et.al. [22] points to this similar concern by focusing on a model based testing approach to generate test cases for automated testing. It does not discuss a formal method based approach as we investigate in our research which is still fairly new in the design and analysis of network for cybersecurity. Microsofts research report [14] pointed out that more than 50% of software security flaws stem from unsafe design, and will lead to higher repair costs in the later stages of product development lifecycle. In this paper our research is focused on developing a system/software engineering approach for designing security to check satisfaction of critical properties early in the design phase.

Significant research effort has been conducted that focused on perimeter protection based on the philosophy of trust on insiders and distrust on outsiders of a network in developing solutions for cybersecurity such as, firewalls. Our use case is based on the philosophy of zero trust to address insider threat analysis by utilizing formal method based approaches. This insider threat case study was selected as, recent research in cybersecurity emphasized the need to design policies to detect and prevent insider threats in cyber infrastructures. Spafford et al. [26] defines insider attackers as individuals who have been assigned privileges and the authority to execute their responsibilities, but they use that authority to damage the system or steal sensitive data. So, it is critical to design the system or software with zero trust philosophy.

The benefits of a formal method based software engineering approach is that it helps to capture the requirements in a concrete way early in the design phase. Once the design is completed with formal annotations it enables early automated reasoning to identify any conflicts in the requirements or errors in the design as has been proven with decades of research for safety critical systems [3], [21]. Additionally, introducing a formal language that contains terms from the domain allows for quicker and easier adaptation of the new methodology.

Our first challenge was in developing an ontology that represents the the cybersecurity domain with well-defined concepts to design a realistic model and facilitate computer-based analysis. In order to better understand the concepts of the domain which is insider threat analysis we narrowed down our scope to evaluate research conducted to identify insider attack dealing with confidentiality preservation only, while exchanging sensitive data within the organization network. There are many strategies in the literature to detect insider threats including modeling the abnormal behavior of the system, and detecting the vulnerabilities, the relationship among these vulnerabilities and the attack signature. The first strategy

concerned with the abnormal behavior of the software utilizes inference rules to compute the level of the threat [3]. The second strategy focused on modeling ontology for vulnerabilities to identify weaknesses and vulnerabilities in cybersecurity. The goal of another ontology was in detecting the missing security points in the security requirements framework [28]. Also, the Unified Cybersecurity Ontology (UCO) is utilized to detect and prevent an attack on the organizations' assets based on the attack signatures that have been stored in the knowledge base [25]. These strategies were focused on detecting the attacks. In this work, we aim to develop a formal approach to reason about confidentiality preservation violation which helps in identifying the gaps in security of the present design and any policies that need to be implemented to prevent attacks on the confidentiality preservation in a zero-trust environment.

Once the essential elements that need to be modeled for insider threat analysis were identified in our research effort, we modeled the elements of the network, the data being transmitted within the network of the organization, and the relationship among these elements. Additionally, we modeled the actions that are performed by the users. Our ontology has a generic representation of the network elements that are at a high level of abstraction. It does not need to represent the details of protocols or cryptographic algorithms as we do not focus on the specifics associated with the implementation of the content instead focusing on the possible interactions. In other words, when we modeled the content of the message, we did not model the content as a medical record for a patient, bank account information for a client, or an academic record for a student. We modeled the content by annotating it as an encrypted content or normal content. The reason for our abstracted approach for insider threat ontology was to make it more generic and not specific to a particular type of implementation. This higher level of modeling and analysis helps to identify the requirements, gaps and conflicts in strategies for a network much better than making it application specific.

The second challenge was in the creation of a Domain specific language (DSL) that captures all the essential elements from the domain to enable modeling the network architecture with annotations that will enable automated reasoning. This was achieved by mapping the concepts developed in the ontology to the reserved keywords in the DSL along with the integration of architectural representation and dependencies. The grammar for the DSL was implemented in Antlr.

Finally, the challenge was to enable formal reasoning once the model of the network is created using the DSL. To meet this objective the reasoner was developed which utilized architectural and relational associations among words in the programming language to perform automated analysis. This allowed automatically guaranteeing satisfaction of properties within the designed network. In section III-A we show how ontology can be utilized to capture the requirements of insider threat domain, explain our ontology and SWRL rules to enable reasoning for consistency check of the modeled concepts and how HermiT reasoner was utilized to run SWRL rules over our cybersecurity ontology. In addition, we developed a formal grammar associated with the architecture for verification which will be further explained in III-C. In III-D, we explain the *assume-guarantee* based automated reasoner for cyber systems.

II. Literature Review

A. Cybersecurity Ontology

Ontologies can model data as concepts and formally identify the associations among them. An ontology can be used to create well-defined concepts which will help the researchers when exchanging information about a particular domain. In addition, ontologies provide reusability for an agreed upon set of domain knowledge [17]. Several languages are used to develop ontologies. For example, Resource Description Framework (RDF), Ontology Inference Layer (OIL), Unified Modeling Language (UML), and Web Ontology Language (OWL) [13]. Protégé is a graphical environment that can be used to create an ontology. Protégé also supports the use of the OWL language [16]. Cybersecurity ontologies can be used to detect software vulnerabilities and their relationships.

Many studies have been conducted about cybersecurity ontologies. Undercoffer et al. [19] have been working on an Intrusion Detection System (IDS) ontology. The IDS ontology analyzed more than 4000 classes of computer attacks and how these attacks can be performed. The IDS ontology includes the following four categories: the system components that have been targeted by the attackers; the effect of the attacks on the system components; the means of the attack; and the attacker location. The IDS ontology was then extended by Finin et al. [25] to include information integration and to write general rules, and it became the Unified Cybersecurity Ontology (UCO) .

The UCO ontology added more features such as gathering numerous security information from different sources including Common Vulnerabilities and Exposures (CVE) [1] and the capability of reasoning and inferring new knowledge. Another ontology has been proposed to integrate weaknesses and vulnerabilities [9]. The goal of this ontology is detecting the missing security points in the security requirements frameworks. In addition, another security ontology called ontology-based model of network and computer attacks for security assessment has been built to categorize the attacks into proper classification to assess the security from the attack point of view [10].

We found that the modeled ontologies did not address the concerns related to the insider threat activities since they focused on the attacks that can be performed by the outsiders where they combined the vulnerabilities of systems, software, or networks from many resources to reason about these vulnerabilities and how outsiders can exploit them. Thus, it is essential to model insider threats activities as they are severe threats to an organization. Therefore, organizations must protect their assets from insider threats as much as outsider attacks.

However, the absence of a standardized approach to represent the terms insider threats and insider was the problem that

the researchers faced in the insider threats activities field [23]. A team from Carnegie Mellon University has developed one ontology called insider threat indicator ontology. Costa et al. [7] developed the insider threat indicator ontology to detect, create, share, and analyze the indication of insider threats. The insider threat indicator ontology connects the malicious activities of the insider, the natural language that is used to describe the activities of malicious employees in the internal network, and the data that is generated by machines which are used to capture the changes of the behavior of the insider activities .

None of these ontologies focused on modeling the interaction among insiders when exchanging sensitive messages to capture any potential violation of policies to attack confidentiality preservation and developing a system/software engineering approach to analyze and model the architecture of the network. We next discuss our approach to model and analyze insider threat interactions.

B. Confidentiality Preservation

The integrity of data and the confidentiality of messages being transmitted between two people is a big concern for them when they exchange messages. In order to overcome the concern about the integrity and the confidentiality of messages, there are approaches such as zero trust approach for the internal network, as well as cryptography based security technologies. For example Encryption Protocols including Transport Layer Security (TLS) protocol [8] and Hypertext Transfer Protocol Secure (HTTPS) [24].

Recent work on "Zero trust" addresses the principle of not trusting any user on the internal network. Their research effort focuses on the development of a domain specific language (DSL) to evaluate the users and devices on the network, but this effort is limited to the security property of accessibility. Google has also been working on a zero trust security framework where the major focus has been to shift access control to the individual nodes in the network [27]. Reverse proxies are deployed at each resource node which authenticates and then authorizes access to that nodes resources. This is implemented in a DSL named ACL which works with a centralized policy enforcement service. The language allows compilation of ACLs statically and helps reduce the logical gap between the policy and its implementation. The limitation in these methods is that the focus of the effort has been only on access control. Our approach broadens the focus with the formal language supporting the verification of security properties beyond just accessibility.

C. Node Reputation

Node reputation also plays a key role in insider threat analysis as according to the reputation or level of trust different policies can be implemented. The definition of Level of trust between two systems in the network means how much one system can trust another system to interact with it. Therefore, the reputation of that system can be used to measure the level of trust that can be assigned to one system by another system to work it. Al-hamadani [2] mentioned that Buchegger [5] gave reputation a definition which says that the reputation of one node is the opinion of another node about it. Michiardi [15] defines a measurement of the level of trust by the amount of knowledge that one community member gained in a particular domain. Thus, if a system chooses to work with other systems, it will build an excellent reputation for itself in the community. An uncooperative system that acts maliciously in the network will build a bad reputation for itself among other systems. This system will be secluded and will not be able to communicate with the other systems until it chooses to act safely with the other systems.

A systems final reputation rating depends on two kinds of gathered information. The first one called first-hand information where the information of one system is directly gathered from the observation of that system. The second called second-hand information where the information used for the final reputation rating is gathered from other systems [2].

III. Domain Specific Analysis with Formal Representation

Diaz defined a domain specific analysis process from software engineering point of view as the process of identifying, capturing, and organizing the information of a specific when developing a new system. One of the objectives of analysing a specific domain is to gather information about critical properties in that domain. Also, the reusability of the gathered information feature will allow a software engineers to decide about the approach they will use to develop another system in the same domain [20].

The notion of trusted and untrusted users enables network operators to control accessibility but fails to address how the actions of trusted users might violate security properties. Let us consider the network connection as shown in Fig 1. According to this example; systems A and B interact with each other and system B also interacts with system C. We first assume the network has a global policy that system A can send message to system B and system B can send message to system C. Also, system B only should handle messages sent by system A and not forward system A messages to system C. This is implemented in practice by the systems A and B sharing keys and encrypting the messages between them so that their communications are confidential and integrity. This approach can be violated in practice since system B also shares secure communication with system C. So, the message received from A could be transmitted to C by B. It is essential to identify such possible interactions in the network as this is a violation of the global security policy which might lead to system C receiving information from system A via system B. To analyze this, we use a formal representation in an ontology models and analyze security properties of a given network by analyzing the interactions among the systems in the network to identify if any of the network policies are violated. Previously, the Burrows Abadi Needham (BAN) [6] logic was formulated for the verification of information exchange protocols for communication. However, its semantics

do not provide necessary capabilities needed to perform the verification of cybersecurity properties [4].

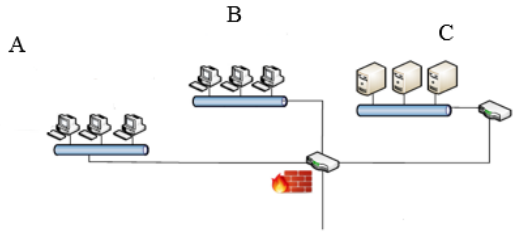

Fig. 1. An example of network interactions

A. Ontology for Cyber Requirements

Our ontology is built to model distributed systems as represented in a network at a high level of abstraction. So, the models developed in our framework captures the architectural representation such as, having connection to send and receive messages between distributed systems. We ignore the type of protocol to exchange messages to reason at a higher level of abstraction to understand the conflicts in requirements.
In our ontology a user can use a computer to encrypt, decrypt, send, receive, and forward messages. As shown in Fig. 2 it contains the top-level classes of our ontology enumerated as user, action, asset, message and key. The associations are shown as arrows going out from the user class to message class that represents the object properties. These object properties represent the performed actions by users on the messages. Users have the capability to encrypt and decrypt messages, and the knowledge of the key required for encrypting and decrypting the content of a message. A computer which is a subclass of asset class can send and receive messages. A user can use a computer. A computer can be connected to other computers. This ontology is developed in OWL which uses Description Logic [11] for reasoning to identify if the instantiated model satisfies some basic properties. For example, the computers have encryption and decryption methods within the instances of the systems created. Our ontology consists of multiple classes as the following:

- Action Class holds several actions which are performed by the user class on the message class. We modeled several actions such as encrypt and decrypt where the user can encrypt and decrypt messages. Also, send, receive and forward actions which represent the exchange of messages operation. Additionally, we also model a create action where the user can create a message.
- Asset class consists of computer and the data. In our case we only modeled the computers on the network and the data which can be normal data or sensitive data. These computers can be used by a user to perform several actions which are described in the action class. In addition, there are two types of data that could be included in the message. First, normal data which is the data that is not encrypted. An example of normal data is any non-sensitive information exchanged within the organization. There are many organizations that consider all internal exchanges to be private within the organization even though they are not encrypted but our assumption is any sensitive exchange must be encrypted. The second type of data is sensitive data which is always encrypted by our assumption and should not be forwarded to an unauthorized user. For example, an employee record containing private information such as personnel identification number, salary, or benefits are examples of sensitive data which should be encrypted all the time. According to the exchange policy, when the received message is normal which means it is unencrypted, the message can be forwarded to other users. If the message is sensitive which means it is encrypted, the message can be forwarded based on who is the owner of the message.
- Key class represents the keys that a user has that are used to encrypt and decrypt transmitted messages between users through the organization network. In this research we focused on only shared key policy (in the future this could be extended to support public key cryptography). The key will be shared by the owner of the message with the receiver of the encrypted message.
- User class defines a set of users who work for the organization and use the organization network. In our case, we modeled three different users who are manager, team leader, and team member. Those users can perform different kind of actions as indicated in the action class.

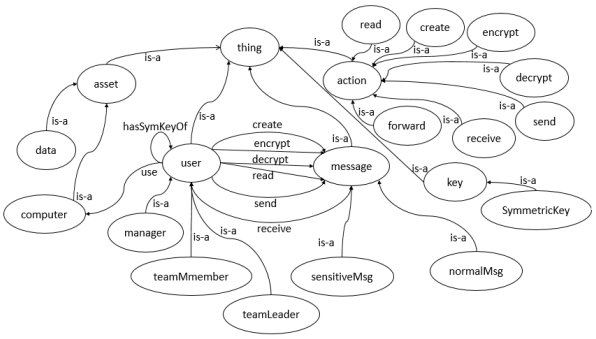

Fig. 2. Main classes and sub-classes of our ontology

B. SWRL Rules for Automated Reasoning

The SWRL language was been created by combining OWL sub languages. These sub languages are OWL, DL and OWL Lite [12]. The existing knowledge within our ontology can be used to infer new knowledge. The capability to automate reasoning over our ontology is achieved by writing SWRL rules. Reasoners can check rules about individuals based on the existing knowledge and infer new knowledge about classes, relations, individuals, etc [18].

After designing and developing our ontology, we created different insider communication scenarios written in Natural English language. We then developed SWRL rules to check

the consistency of the designed ontology and if the desired properties are satisfied by executing the SWRL rules on different individuals and object properties in our ontology. Our main focus was verifying the satisfaction of Reachability and Confidentiality properties.

C. Cybersecurity Architecture Design Language (CSADL)

CSADL is a system/software engineering guided formal language to model the network with its elements and the relationships between these elements.

CSADL is the language that we designed to enable a network engineer to model a network and automatically reason about it. CSADL allows modeling of different systems within the network such as computers and routers. Additionally, CSADL allows checking the security properties that are associated with the different systems in the network, and its elements. CSADL has language constructs that enables automated reasoning for performing *assume-guarantee* based reasoning to check preservation of confidentiality. This way, in a CSADL program network engineers will be able to identify the needed rules that should be modeled to prevent confidentiality related insider threats.

1) The Creation of CSADL Formal Language: The ontology that has been demonstrated in Fig. 2 guides the development of the grammar for the formal language CSADL. This is achieved by mapping the conceptual classes and object proprieties of the ontology that builds the vocabulary of CSADL. In developing the grammar, we developed one lexer rule and one parser rule. The lexer rules are the reserved words within CSADL and these reserved words are used to build the base of the parser rules. The parser rules are the syntax of our language that allows writing the statements following the different structures allowed in CSADL. Some of the main statements in CSADL are as shown below in Fig. 3.

```
assumption1: ASSUMPTION ASSET
             ASSOCIATION (ASSET|MESSAGE) ;

assumption2: ASSUMPTION MESSAGE
             SENSITIVITY ;

guarantee:   GUARANTEE ASSET
             ACTION  (ASSET|MESSAGE) ;
```

Fig. 3. CSADL main statements to develop the network architecture

1) assumption1 statement is developed to represent the association between two different assets or between assets and messages. An example of this statement is shown in Fig. 4.
2) assumption2 statement is developed to differentiate between message with sensitive content and normal content.
3) guarantee statement is used to represent the actions that can be performed by assets on other assets or messages as shown in Fig. 5.

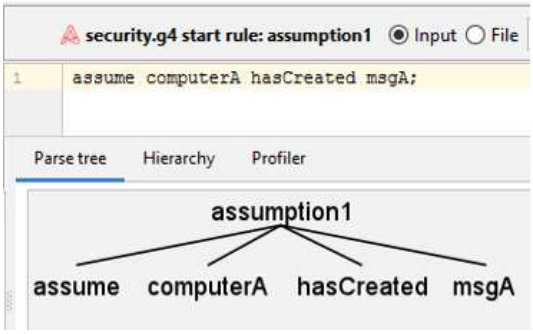

Fig. 4. Represent connection between two assets

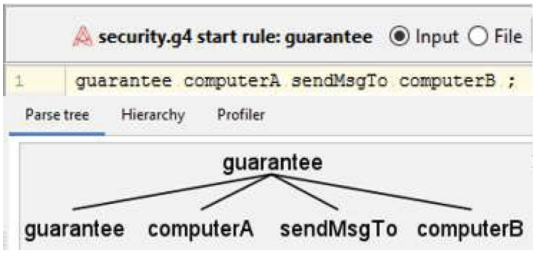

Fig. 5. Represent send message action between two computers

D. Assume-Guarantee Based Automated Reasoning for Confidentiality Preservation in the Network

1) System/Software Engineering approach for Assume-Guarantee based network design: In our methodology when network engineers start designing the network they need to first identify the systems in the network, then they would need to identify the associations among the systems and the actions that can be performed by these systems. The association and actions enable an *assume-guarantee* based modeling and analysis framework. For example, an association discussed by our ontology and CSADL language is *hasConnectionWith*. This association must guarantee two actions which are *send* and *receive*. In other words, the assumption in the CSADL program will have statements that indicate the existence of a connection between two systems. As a result, the network engineer should be able to guarantee in the CSADL program that these two systems can interact by sending and receiving messages. Presently, network engineers do not use such a system engineering approach to model the network and analyze the desired network along with the policies that they want to implement. This can be achieved in our method by designing the network with assumption and guarantee statements. Furthermore, any conflicts or dissatisfaction of properties can be captured early enough in the design process using our approach. According to our method, the network designer needs to associate assumptions about each component in the network. This method will help in inferring if the guarantees can be met or not. An automated reasoner is needed to perform the reasoning when several models are being evaluated. Thus, we design an automated *assume-guarantee* reasoner which will check the guarantees and check if this

guarantee is satisfied or not based on the assumptions.

Algorithm 1 Automated Assume-Guarantee Reasoning Algorithm

```
/initialization/
initialize variable for number of rows ;
initialize variable to the number of assume statements
initialize variable to the number of guarantee statements
initialize data store to store the code file content
initialize data store to store assumptions and guarantees

/store the program content/
while (code file not empty) do
   Store code file content in data store
    if Line Start with "assume" then
        Store line in assumption data store
    else if Line Start with "guarantee" then
        Store line in guarantee data store
    end if
end while

/Extract action from guarantee statement/
for each guarantee do
    get action

    /Retrieve associated association from action association library/
    for each assumption do
    Extract association from assumption statement
    if Association from library = Association from assumption then
        Extract assumption assets and guarantee assets

        /Check satisfaction of guarantee/
        if Assumption assets = guarantee assets then
            Print guarantee satisfied
        else
            Print assumption asset does not have guarantee asset in assumption
```

2) Automated Assume-Guarantee Reasoner Algorithm: We proposed an automated *assume-guarantee* reasoner algorithm (AAGRA), shown as Algorithm 1. Once the program is completer AAGRA reads the *assume-guarantee* code that has been written. Then, the assumptions and guarantees are stored in the respective data structures. AAGRA checks the action within the guarantee statement. Next, it retrieves the association that is associated with the action from the actions associations library which is demonstrated by the previous pseudo code and extract the association within the assumptions statement. In the next step, the program will check if the two associations are equal or not. If yes, the program will then check the assumption assets and the guarantee assets. If not, the program will check with the next assumption statement. After checking the assumption assets and the guarantee assets, if the assumption assets and guarantee assets match each other a message is printed that tells the guarantee is satisfied. Otherwise, another message is printed that tells assumption asset does not have the guarantee asset in assumption.
The *assumptions* are like the premises in a proposition logic and the *guarantees* are the conclusion we want to reach. So, these statements can be formulated as shown in equation 1.

$$\frac{assume \quad p_{ij}}{guarantee \quad q_{ij}} = \frac{premises}{conclusion} \tag{1}$$

where :

$p_{ij} \implies asset_i \ association_i \ asset_j$ **or** $asset_i \ association_i \ msg_j$
$q_{ij} \implies asset_i \ action_i \ asset_j$ **or** $asset_i \ action_i \ msg_j$

$$i \neq j \quad where \quad i \quad and \quad j \quad are \quad integers$$

$m_{ik} \implies msg_i \ association_k \ content$
$where \ k \ is \ an \ integer$
$action_i \implies equivalent \ relation\{association_i, association_k\}$

3) Experiment: Different scenarios were designed, developed, and tested as a prerequisite for confidentiality preservation scenario. The network architecture design was tested first to guarantee that any two computers could interact with each other. Next, the infrastructure or resources required to support cybersecurity was tested such as, for symmetric key cryptography the interacting computers should share a secret key to encrypt and decrypt messages. Then, we tested for confidentiality preservation, one such scenario is described next. This scenario shows how our algorithm can detect the action of forwarding message that contains sensitive content which is *msgA* in this scenario. *computerA* as shown in Figure 6 is the owner of *msgA* based on *hasCreated* association. Our reasoner guarantees the *forward* action because *computerA* is the owner of the message. On the other hand, when *computerA* is not the owner of *msgA*, our reasoner will check the sensitivity assumption statement and will make a decision based on that. If *msgA isNormal*, our reasoner will let *computerA* to *forward msgA*. However, if *msgA isSensitive*, our reasoner will deny the *forward* action that is performed by *computerA*.

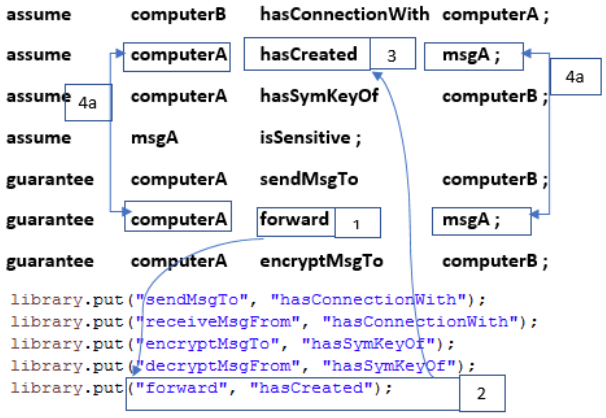

Fig. 6. Confidentiality Preservation Experiment

1) Develop the network architecture with assume statements as shown in Fig. 6
2) Once the design is complete check the actions that need to be satisfied in the guarantee statements, which is *forward* for the second guarantee.

3) Get the association relation with the corresponding action (*forward*) in the guarantee statement by searching the action/association library.
4) Check the association in the library (*hasCreated*) = the association in the assumption statement (*hasCreated*) is true.
    a) Check if assumption asset (computerA) = Guarantee asset (computerA) **and** assumption message (msgA) = Guarantee message (msgA)
    If Yes
    i) Print guarantee satisfied.
    else
    i) Check the sensitivity of msgA which is stated in the fifth assumption statement.

IV. Conclusion and Future work

We proposed a system/software engineering approach to design a network with critical properties into consideration. Our approach allows network engineers to reason about the security properties of the designed network before building it. We built a cybersecurity ontology which enabled us to formally represent different elements of the network domain for the chosen cybersecurity problem which is insider threats problem. We showed a formal way to model and analyze about confidentiality attacks that can occur due to users sharing messages that they are not allowed to share. Our formal language focused on designing the network with critical security properties into consideration based on *assume-guarantee* approach. Finally, we built an automated *assume-guarantee* based reasoner which reasons over the formal program based on the assumption and guarantee statements. Our approach checks violation of confidentiality since it could identify if an action in the guarantee statements can be guaranteed based on the assumption statements or not. Our approach can be extended to add new rules and specific network elements to the ontology such as, router and firewall by adding device specific behaviors at higher level of abstraction. In addition, new scenarios can be modeled to extend our approach beyond confidentiality preservation.

References

[1] CVE-2014-0160. Available from MITRE, CVE-ID CVE-2014-0160., December 3 2013.

[2] Ahmed Taha Hammo Al-hamadani. *Mitigating the Effects of Selfish Behavior in Mobile Ad Hoc Networks Using Reputation-aware Routing*. PhD thesis, Florida Institute of Technology, 2013.

[3] J. P. Bowen and v. Stavridou. Safety-critical systems formal methods and standards. *Software Engineering Journal*, 8, 1993.

[4] Colin Boyd and Wenbo Mao. On a limitation of ban logic. *Workshop on the Theory and Application of of Cryptographic Techniques*, pages 240–247, 1993.

[5] Sonja Buchegger and Jean-Yves Le Boudec. Self-policing mobile ad hoc networks by reputation systems. *IEEE Communications Magazine*, 43(7):101–107, 2005.

[6] Michael Burrows, Martin Abadi, and Roger Michael Needham. A logic of authentication. *Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences*, 426(1871):233–271, 1989.

[7] Daniel L Costa, Matthew L Collins, Samuel J Perl, Michael J Albrethsen, George J Silowash, and Derrick L Spooner. An ontology for insider threat indicators development and applications. Technical report, CMU, Pittsburg, PA, Software Engineering Institute, 2014.

[8] Tim Dierks and Eric Rescorla. The transport layer security (tls) protocol version 1.2. Technical report, 2008.

[9] Golnaz Elahi, Eric Yu, and Nicola Zannone. A modeling ontology for integrating vulnerabilities into security requirements conceptual foundations. In *International Conference on Conceptual Modeling*, pages 99–114. Springer, 2009.

[10] Jian-bo Gao, Bao-wen Zhang, Xiao-hua Chen, and Zheng Luo. Ontology-based model of network and computer attacks for security assessment. *Journal of Shanghai Jiaotong University (Science)*, 18(5):554–562, 2013.

[11] Ian Horrocks. Owl: A description logic based ontology language. In Peter van Beek, editor, *Principles and Practice of Constraint Programming - CP 2005*, pages 5–8, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[12] Ian Horrocks, Peter F Patel-Schneider, Harold Boley, Said Tabet, Benjamin Grosof, Mike Dean, et al. Swrl: A semantic web rule language combining owl and ruleml. *W3C Member submission*, 21(79), 2004.

[13] Ian Horrocks, Peter F Patel-Schneider, and Frank Van Harmelen. From shiq and rdf to owl: The making of a web ontology language. *Web semantics: science, services and agents on the World Wide Web*, 1(1):7–26, 2003.

[14] H. Michael and L David. Writing secure code. 2003.

[15] Pietro Michiardi and Refik Molva. Core: a collaborative reputation mechanism to enforce node cooperation in mobile ad hoc networks. In *Advanced communications and multimedia security*, pages 107–121. Springer, 2002.

[16] Mark A. Musen. The protégé project: a look back and a look forward. *AI Matters*, 1(4):4–12, 2015.

[17] Natalya F Noy, Deborah L McGuinness, et al. Ontology development 101: A guide to creating your first ontology, 2001.

[18] Martin Oconnor, Holger Knublauch, Samson Tu, and Mark Musen. Writing rules for the semantic web using swrl and jess. *Protégé With Rules WS, Madrid*, 2005.

[19] John Pinkston, Jeffrey Undercoffer, Anupam Joshi, and Timothy Finin. A target-centric ontology for intrusion detection. In *In proceeding of the IJCAI-03 Workshop on Ontologies and Distributed Systems. Acapulco, August 9 th*. Citeseer, 2004.

[20] Rubén Prieto-Díaz. Domain analysis: An introduction. *ACM SIGSOFT Software Engineering Notes*, 15(2):47–54, 1990.

[21] J. Yuang S. Smolka B. Meng C. Sticksel S. Bhattacharyya, S. Miller and C. Tinelli. Verification of quasi-synchronous systems with uppaal. In *Digital Avionics Systems Conference*, pages 738–743, 2014.

[22] Ina Schieferdecker, Juergen Grossmann, and Martin A Schneider. Model-based security testing. In *Proceedings 7th Workshop on Model-Based Testing*, 2012.

[23] George J Silowash, Dawn M Cappelli, Andrew P Moore, Randall F Trzeciak, Timothy Shimeall, and Lori Flynn. Common sense guide to mitigating insider threats. 2012.

[24] SK Rajinder Singh. An overview of world wide web protocol (hypertext transfer protocol and hypertext transfer protocol secure). *Int. J. Adv. Res. Comput. Sci. Softw. Eng.*, 3:251–259, 2013.

[25] Zareen Syed, Ankur Padia, Tim Finin, Lisa Mathews, and Anupam Joshi. Uco: A unified cybersecurity ontology. In *Workshops at the Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

[26] T Tuglular and EH Spafford. A framework for characterization of insider computer misuse. *Unpublished paper, Purdue University*, 1997.

[27] Rory Ward and Betsy Beyer. Beyondcorp: A new approach to enterprise security. 2014.

[28] Sujeong Woo, Jinho On, and Moonkun Lee. Behavior ontology: A framework to detect attack patterns for security. In *2013 27th International Conference on Advanced Information Networking and Applications Workshops*, pages 738–743. IEEE, 2013.