Lightbulb: A Toolkit for Analysis of Security Policy Interactions

Derrick Kong, David Mandelberg, Andrei Lapets, Ronald Watro, Daniel Smith, and Matthew Runkle
Raytheon BBN Technologies
Cambridge MA USA
email: {dkong, dmandelberg, alapets, rwatro, dsmith, mrunkle}@bbn.com

***Abstract*— Lightbulb is a toolkit for analysis of the combined impact of a set of diverse security policies. It is designed to securely access and collect the security policy configuration data from the hosts, routers, and firewalls that comprise a network enclave. Lightbulb loads the collected security configuration data into a modeling tool and allows system administrators to run queries against the model with the intent to verify desired security properties of the composite system. If a policy query fails, the user is given a specific instance of the policy violation that can be investigated and resolved. The overall toolkit provides an extensible framework for rigorous verification of security policies of network devices.**

Keywords-cyber security; security policy; network security policy; access control; logic programming; formal verification.

I. INTRODUCTION

Security configuration management has been a problematic issue ever since security devices have existed. In modern, heterogeneous networks, misconfigurations are not just occasional nuisances, but common problems that can lead to serious security breaches. Current work in the field has shown that rigorous verification of security policies is possible, but published research [1][2] has generally been limited to particular aspects of either policy or security configurations. The next logical step is to apply these principles across a broader spectrum of policies and security appliances and to compose multiple policies into a coherent system specification.

The primary challenge for building a coherent system-wide tool for managing security polices is that there can be dozens to hundreds of heterogeneous configuration files residing on devices in a typical enterprise network that will have an impact on some aspect of security. Without an easy way to collect, organize and provide end-to-end analysis, administrators must look at configurations in isolation or in small groups to verify that desired policies are being enforced.

This paper presents Lightbulb, an integrated toolkit of components that support rigorous automated security verification of a variety of network devices and clients. These components have been designed to fit within a general framework; individual components handle tasks, such as ingesting security policy specifications or query inputs (expressed in a Domain-Specific Language (DSL)), controlling and extracting the security-relevant configuration files from components located in a managed network and converting them to an intermediate form and, finally, performing rigorous verification of security policies against the configuration data. Models are built using Prolog within the Ciao logic programming environment and employ features such as constraint programming.

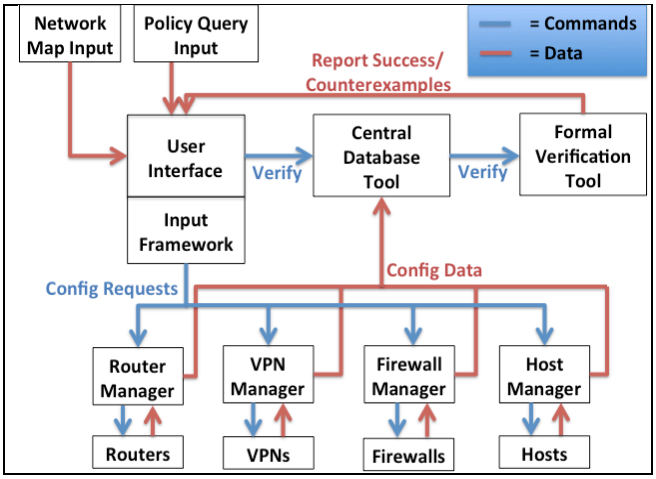

Figure 1. Toolkit architecture showing detailed data flows and interfaces with external inputs and devices to be managed.

The paper is organized as follows. Section 2 describes the general approach of the Lightbulb system. Section 3 provides an overview of the system user interface. Section 4 documents the currently supported network devices. Section 5 covers sample use cases and Section 6 covers currently supported queries. Related research efforts are described in Section 7 and the paper concludes with a summary and recommendations in Section 8.

II. APPROACH

A toolkit architecture diagram for the toolkit is shown in Fig. 1. Starting at the top left, a user provides inputs such as a network map and a security policy verification question via the User Interface. The Input Framework reads the map and uses it to identify the relevant devices in the network. It passes the results down to the Capability Manager Tools, each of which interfaces with one or more specific devices. An individual Capability Manager possesses the specific data needed to locate and interpret the security relevant configuration files from particular classes of devices; for example, the Router Manager is programmed with information on routers, including make, model and other relevant differences. (Support for basic Cisco devices covers a large portion of the install base; see Section VI for notes about extensibility.) A network device with more than one capability (for example, a firewall which is also a router) will report to more than one Capability Manager.

 ISBN: 978-1-61208-398-8

This architecture is easily extensible in that new Capability Managers can be added to the overall system and new device configurations can be programmed into existing Managers. Also, Capability Managers can easily be combined or split up as the Managers can be run on separate processors or the same processor. Our current architecture provides examples for the most common devices that should be present in a typical enterprise network. Extension and modification of these examples is a relatively straightforward process, allowing a local administrator to customize models for their devices, although a radically different architecture or device would require some significant work in order to translate its configuration and capability into our internal representation.

Once the configuration data have been obtained, the Capability Managers pass them up to the Central Database Tool, which stores them. The Database Tool also is responsible for processing the configuration files, including filtering and conversion to our Prolog representation for input to the Verification Tool.

Finally, the Formal Verification Tool takes in all the processed input that has been loaded into the Central Database, combines it with the input data from the User Interface, and performs the policy query check. The results of the check (either success or a listing of counterexamples) are sent back to the User Interface for display.

The formal verification module uses Ciao, a particular implementation of Prolog (with some extensions). Ciao is a modern tool for logic programming that supports strong modularization, which was often lacking in earlier logic programming systems. Ciao also includes support for constraint logic programming, which is very helpful when dealing with network packet data, and also tabling, memoization, and higher-order functions.

Logic programming is a very natural approach for modeling security policies, as both make use of the "negation as failure" concept [3]. In SELinux policy, for example, the active access vector rules are just the ALLOW rules, which permit certain actions. The lack of an ALLOW rule for an action means that that action is blocked. In our Prolog models for SELinux policies, the ALLOW rules become first order facts, and the Prolog engine can implicitly interpret the absence of a fact in a model as the assertion of the fact's negation, which matches SELinux semantics. There are indeed NEVERALLOW rules in SELinux, but they are passive specification rules, to be checked against the ALLOW rules after policy compilation. This approach to modeling SELinux policy using Prolog was first popularized by Scott Stoller and his student at SUNY Stony Brook [4].

If desired, a user can swap out the default formal verification module in the Lightbulb framework and replace it with a completely different engine or processor. This might be done if a particular security policy query requires a different logic or handling process. For example, the Accumulo query on prohibited label combinations employs a Python engine to handle the data processing as opposed to the Ciao engine.

III. User Interface Overview

The Lightbulb User Interface (UI) is designed to provide a single, convenient input and control interface for users, while still allowing developers an easy way of modifying existing functionality or adding additional modules to the toolkit. The UI provides the input and output mechanisms for the network map and policy queries; it works in close concert with a backend that handles operations for retrieving and managing configurations from networked devices.

The UI is based on open-web technologies. It makes use of three main frameworks:

- Flask, a python-based web server that handles processing
- Bootstrap, a frontend framework that provides templating and styling, and
- SigmaJS, a JavaScript graphing library used to draw network maps.

The UI provides a human interface to the Input Framework and negotiates communication with the Formal Verification Tool.

When Lightbulb is started, the user is presented with the configuration page, starting with the Network Map (Fig. 2). If this is the first run, the user is given the option of starting with a blank pane or to input an existing UML map; otherwise the last edited version of the Network Map is presented. At this stage, the user is free to alter the map to reflect the current network architecture. Lightbulb currently only read in maps files that use a UML deployment diagram representation.

The second configuration step allows the user to specify device credentials, including SSH-enabled, telnet-enabled, and Accumulo (a highly scalable database) node devices. Devices with common passwords can be supported as well as devices needing additional authentication (such as enable mode on Cisco devices).

In the final step of the configuration wizard, the Capability Managers are instructed to fetch device security policies. While the system contacts each device, a visual overlay is provided to indicate the status of the fetch process. Each device security configuration is retrieved and converted to an internal representation (expressed in Prolog) in parallel. The user interface periodically polls the server for updates, changing the status as fetches complete (or time out).

A. Security Policy Query Interface

The query page is an interface to the Prolog modeling engine. In order to make each type of query easy to formulate, the page provides a preset structure and list of options

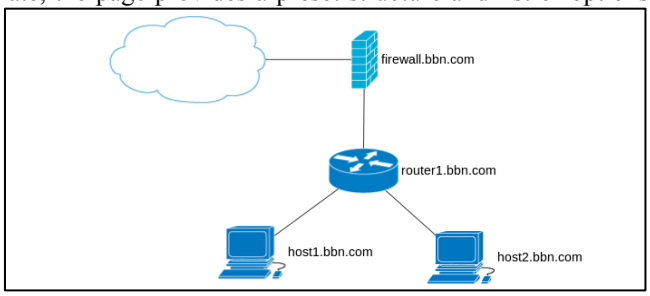

Figure 2. Network Configuration pane detail showing an example network.

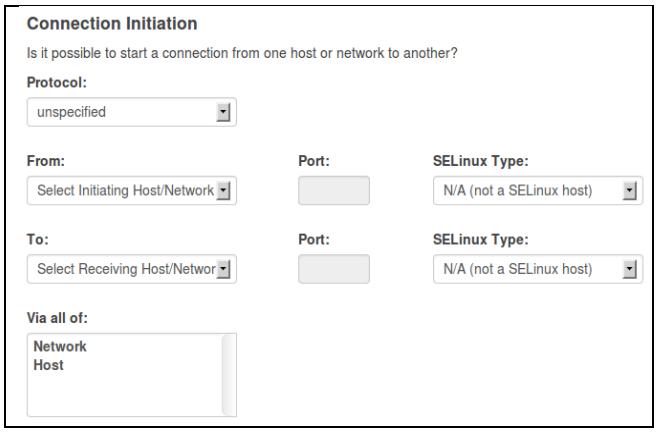

Figure 3. Query interface example. Shown is the selector for connection initiation queries. Values in the pulldown menus are automatically taken from the network map that is input by the user.

for each type of query. Submitting a query is as simple as selecting the appropriate options for each field and pressing submit. A small delay will occur as the system converts the information to the appropriate Prolog commands and submits them for processing. A simple "yes" or "no" response will return in typical Prolog fashion, along with counterexample data (in limited cases) if the answer is "no." An example of the query interface for reachability and connection queries is shown in Fig. 3 and a complete list of supported queries is given in Section 6.

IV. Currently Supported Devices

We currently support policy analysis for multiple types of devices: Iptables, Cisco routers/firewalls, SELinux, UNIX discretionary access control, and Accumulo hosts. Each of these analyses is discussed below.

A. Iptables

Iptables is an application program that allows a system administrator to configure the tables provided by the Linux kernel firewall and thus the firewall policy. We have completed a Prolog model that implements the iptables policy. Because iptables is so general, we are able to use our model as a basis for covering other packet polices, such as the Cisco IOS access list rules (discussed in the next section).

In constructing an iptables policy, the largest granularity item is called a table. Tables consist of one or more chains, where chains can be built-in or user-defined. Chains may contain multiple rules, where rules determine an action to take on packets.

There are four built-in tables: filter, NAT, raw, and mangle. Each built-in table contains a few built-in chains. Each rule in a chain contains a goal and a target. If the rule goal is matched, then processing continues onto the rules specified in the target. If the goal is not matched, then processes moves to the next rule. The default value in iptables is to accept if no rule applies, but this default can be changed by the policy. Lightbulb currently only considers the filter table for its policy analyses.

A rule in iptables may take a number of actions on a packet, including:

- Accept the packet
- Drop the packet
- Queue the packet for user interaction
- Return the packet to the calling chain.

An example of a very basic set of rules that Lightlbulb can parse and verify is shown below:

```
iptables -A INPUT  -j ACCEPT -p all -s 192.168.1.0/24 -i eth1
iptables -A OUTPUT -j ACCEPT -p all -d 192.168.1.0/24 -o eth1
```

These rules allow traffic to/from a specific subnet to pass through the firewall.

Our Prolog implementation of iptables policy uses nested lists to represent the chains and rules inside each table. We use recursive list traversal in order to ensure that the chains and rules are examined in order and that the first rule matching a particular connection (i.e., a hypothetical packet with particular source and destination addresses and ports) is selected for application.

B. Cisco Routers/Firewalls

The Cisco IOS support for router and firewall access control lists has evolved over the years and currently includes a wide array of options. In IOS, access control lists are numbered or named, and a numbered list can be applied to either the in-bound or out-bound traffic on an interface. Each access control list may contain explicit permit or deny rules, and the order of the rules is important, as the first match of a packet to a rule determines the status of the packet. There is also an implicit deny at the end of each access control list, so that if a packet matches no rule, it is rejected. Due to the presence of both explicit and implicit denial, we explicitly process Cisco access control lists to find the first match to a packet, rather than storing them as Prolog facts and allowing failure to define denial.

The general syntax for a CISCO extended rule is:

[permit/deny] protocol source destination parameters

The protocol can be one of IP, TCP, ICMP, and UDP. The exact syntax of a rule varies for each protocol. In general, the source and destination can be ranges of IP addresses, expressed using an IP address and a host mask; in this manner, "host 192.168.30.5" means the same as "192.168.30.5 0.0.0.0". Ports can be specified either numerically or by using names for the well-known ports.

For the current system, we have chosen to model the Cisco extended rule set, an example of which is verifiable by Lightbulb is shown here:

```
access-list 101 permit tcp any host 192.168.35.1 range 20 21
access-list 101 permit tcp host 192.168.30.5 host 192.168.35.1
   eq telnet
access-list 102 permit tcp host 192.168.35.1 any
access-list 102 permit tcp host 192.168.35.1 eq 20 any gt 1023
access-list 102 deny udp any

interface Ethernet0
access-group 101 out
access-group 102 in
```

 ISBN: 978-1-61208-398-8

C. SELinux

As described in Section 2 above, we start with the Prolog approach found in previous work [4] to build a model of the security policy for SELinux hosts. We include allow rules, type transitions, conditional rules and SELinux policy booleans in the model. In addition, we link the SELinux policy to the network activity by explicit tracking of both the assignment of port numbers to types and then enabled allow access rules for sockets associated with these port numbers. Under SELinux, the use of a network port number in TCP or UDP traffic communications is limited by policy and our tool analyzes these limitations when it performs checks on connections.

D. Unix Discretional Access Control

We include a Prolog model for the well-known discretionary access control in conventional Unix, based on settings for the user that owns the object, the group that the object is in, and all others. This was implemented without use of the Prolog cut operation, which is commonly used to control backtracking, but in this case would limit the usability of the definition. Instead of cuts, the group and other rules contain explicit hypotheses that express the proper order handling of user/group/world permissions.

Note that the user interface does not currently support preformed queries for discretionary access control, but it is accessible via direct Prolog input.

E. Accumulo

Apache Accumulo is a scalable data store based on Google's BigTable model. One of its enhancements beyond BigTable is the implementation of cell-level access controls which allow data cells at different security levels to be stored in the same table.

Accumulo clusters may be composed of a variety of complex network configurations and storage topologies; Lightbulb enables seamless reasoning over Accumulo cluster security by modeling policies at both the network and database configuration levels. While Accumulo may not be as common as other data storage technologies like MySQL, it uses access control permissions similar to those found in other databases and combines them with new types of security controls. As a result, the models and queries developed for Lightbulb's Accumulo support can be ported to support other database systems.

Like many common databases, an Accumulo cluster does not have a single well-defined policy file for the entire system. Instead, its security properties are defined by four separate configurations: the configuration files, user authentication, database access control permissions, and data cell visibilities.

Of these configuration types, the configuration files are of comparatively minor importance. The configuration files primarily specify the network configuration of the Accumulo cluster and the user authentication information required to provide management oversight of the system. They can be used to derive the network topology of the Accumulo cluster given direct access to an Accumulo master server, but they do not contain information on Accumulo user accounts, access controls, or visibilities. The other configuration types must be obtained by interacting directly with the Accumulo database.

The Accumulo configuration structure described above presents a challenge to the Lightbulb data ingest model. To interface with Accumulo, a new access module was developed to extract configuration files and interact directly with the Accumulo database to extract the security properties. Because each cell in an Accumulo database has its own visibility, it is impractical to recover all of the relevant security data. Lightbulb captures only the information required to construct a Prolog model of the basic system and access control configuration and does not attempt to recover the visibility of individual cells. As a result, some security queries are executed using the Prolog system model while queries requiring access to data cell visibility are executed using a Python query engine and run against the live Accumulo cluster.

V. USE CASES

In this section, we present a set of typical use cases in which a network administrator might desire to verify security policies. These use cases were used to derive the set of supported queries that are listed in the next section.

A. Typical Enterprise Network

The first use case is a typical enterprise enclave with two independent connections to the Internet, a DMZ zone, and an internal backbone that spans multiple subnets. A set of standard security policies applicable to many typical corporate and university networks is relevant here, such as the provision of particular public services in the DMZ to the outside world, but limited or no exposure of services and hosts inside the inner firewalls, except in particular cases to the DMZ (such as an internal database accessed by a DMZ web service). Verification of firewall, routing, and service access rules would be relevant in this case.

B. Multiple Enclave Enterprise

A variant of the first use case is one in which an enterprise consists of multiple independent enclaves separated across the Internet. In addition to the questions in the previous section, relevant policy questions include enclave-to-enclave communication configurations, such as verification that VPN traffic between enclaves is encrypted and is being correctly routed through the designated endpoints.

C. Combined Network and SELinux

The next use case is shown in Fig. 4. The policy question illustrated in the figure is whether it is possible for data to flow from Host A to Host Z. The key concept here is the need to combine security policies; in this case, the access control policies on firewalls 1 and 2 must be combined with the security policies on the host systems, which are assumed to be SELinux. The blow-up of Host C illustrates the issue that data may transition in type as it flows across an SELinux host.

We note that enforcement of SELinux types upon data transmitted across a network is generally not enabled. To

 ISBN: 978-1-61208-398-8

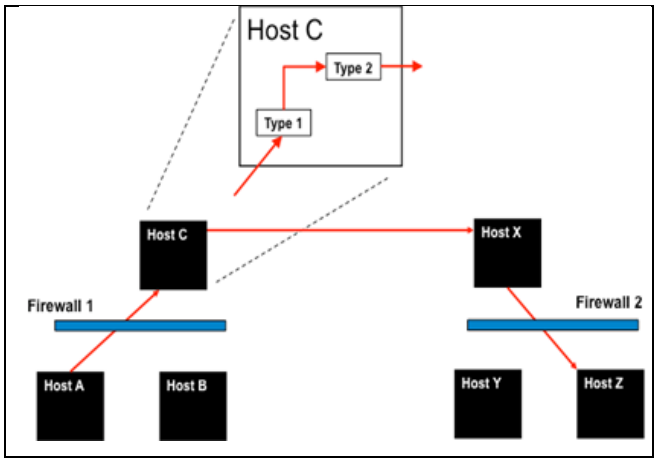

Figure 4. Basic combination of host (SELinux) and network policies.

do so would require all network hosts to be SELinux enabled or for mechanisms such as IPsec tunneling (with bindings established between Security Associations and types) between all relevant hosts to be enabled. While the latter is a more interesting case in terms of security policy, it is much less common in practice. Thus, we do not address this latter case in this version of the toolkit. Also, we assume that the Tresys tools are available to answer security policy questions that are completely self-contained on a single host.

Relevant queries then include basic routing questions as well as whether information flow can occur from one domain on an initial host to another domain on a destination host. For purposes of this work, we assume that identically labeled types on different hosts are equivalent; this is not enforced on general networks, but is often assumed to be so in practice. Type transition rules, as well as roles, are also relevant to policy queries, but require more advanced handling; Lightbulb currently does not support analyses using such transitions, but could do so with some additional development support.

D. Accumulo

In addition to the network connectivity and routing related queries described previously, a network administrator must be able to assess the security properties of the Accumulo database itself in addition to those of the hosts it runs on. Network related queries are still relevant for an Accumulo cluster, but new queries are also required to validate its user account, table access control, and cell visibility properties.

An administrator may need to determine whether Accumulo users have certain permissions (read, write, etc.) on tables in the database or hold other system-wide administrative rights, a type of query equally applicable to other databases. In addition to this general database permissions query, the data cell visibilities particular to Accumulo demand two unique queries.

The first addresses the need to determine if an Accumulo user account has the proper authorizations to view a cell with a particular visibility. The result allows an administrator to confirm that a user account's access to system data is not overly restricted by unintentional consequences of the applied security configuration.

The second type of query allows an administrator to determine if restricted cell data is leaking between users with exclusive authorizations due to cell-level misconfigurations. Accumulo's data cell visibilities allow the storage of data at different security levels within the same table; if cells are inserted with misconfigured combinations of security labels (e.g., the visibility is restricted to users holding SECRET or PUBLIC authorizations), restricted data can become visible to unintended users. This invalid visibility query allows an administrator to quickly identify every data cell violation in the system given a policy that describes the permitted relationships of security labels.

VI. Currently Supported Queries and Extensibility

The set of preformatted policy queries supported within Lightbulb is shown in Table 1. These policies were chosen as exemplars of each class of query; future work will extend these to related domains such as confidentiality, authorizations, etc. In addition to these, the Lightbulb interface allows an expert user to formulate an arbitrary query in Prolog that is passed directly to the Formal Verification Tool; no checks or constraints are applied to this query, so only expert users should attempt to use this option.

A user can create additional preformatted policy queries by formulating a native Prolog query and following the existing templates presented in the user interface to provide arguments. This process requires Javascript programming knowledge and understanding of the Prolog modules, which will require in-depth knowledge in those areas.

Lightbulb currently supports SELinux hosts, basic Cisco firewall/routers, hosts running IPtables, and Accumulo clusters. Lightbulb also provides a generic temple (in Python) that a user can adapt to read the configuration files from other similar device types. However, a device using a com-

TABLE 1: Currently Supported Preformatted Queries

Query Type	Query	Parameters
Reachability	Does a path exist between these hosts or networks?	From (host/network), To (host/network)
Connection	Is it possible to start a connection from one host or network to another?	Protocol, From (host/network), From (port number), To (host/network), To (port number), Via (hosts/networks)
SELinux	Is it possible for data of a one type on a SELinux host to transition to data of another type on another SELinux host?	Originating host, Originating type, Destination Host, Destination Type
Accumulo Permissions	Do users have a specified permission on particular tables?	Host, Users, Tables, Permission (read/write)
Accumulo Visibility	Are invalid visibilities present in the database?	Host, Tables, System Labels, Allowed Label Coexistence
VPN	Is it possible to start a connection from one host to another, going through VPNs?	Protocol, From (host/network), From (port number), Originating VPN, Receiving VPN, To (host/network/port)
Database (MySQL)	Is a user connecting from a host authorized with a particular permission on a particular table?	User, Host, Permission, Table

 ISBN: 978-1-61208-398-8

pletely new paradigm of storing and reading configuration data may require a new access method not currently supported by the existing work.

VII. Related Work

Early concepts related to the current Lightbulb toolkit where developed at BBN under the Cyber Command System (CCS) project in the DARPA Information Assurance program [5].

The applicability of logic programming for security policy modeling has been noted and exploited in numerous papers [4][6][7]. The early research of this type used the original Prolog language [8] but the later work, including this paper, employ more general logic programming techniques such as tabling and data constraints. Bounded model checking is another approach to the policy analysis problem, using tools such as Alloy [9] and Margrave [10]. The model checking approaches can excel at analyzing changes in security policies.

Previous work has addressed information flow polices in networks and in SELinux [1][2]. The current paper builds off that work by creating a single model that includes the policy requirements from all the components in an enclave.

Accumulo is a relatively recent system and as such, most existing research for it has been in developing architectures and analyzing performance; a few analyses of security have been performed [11], but not in conjunction with other systems.

Finally, there is much current research on Software Defined Networks (SDNs), where specific custom-built network hardware such as routers or firewalls are being replaced with generalized all-purpose network appliance with the power to dramatically redesign a network just by accepted new configuration data [12][13]. Creating assurance in the configuration settings of an SDN is a vitally important challenge that could be supported by future extensions of the Lightbulb toolkit.

VIII. Summary and Conclusions

This paper has presented the Lightbulb, an integrated set of toolkit components for networked system security analysis. Lightbulb allows a network administrator to collect and analyze the various access control and security policies that exist inside a network or collection of networks. Lightbulb includes a user interface tool for network system definition and also automated support for extracting policy data from various configuration files. A query tool is provided with templates for common policy statements. A user can submit a security assertion as a query and receive a verification that the assertion holds for the policy, or a counterexample to the assertion that the user can examine to refine the policy.

We plan to continue to develop the Lightbulb toolkit. Our highest priorities are to expand the set of supported queries and to construct models for new devices from the realm of Software Defined Networking.

References

[1] J. D. Guttman, and A. L. Herzog, “Rigorous automated network security management,” International Journal for Information Security, vol. 3, no. 3, pp. 29-48, 2004.

[2] J. D. Guttman, A. L. Herzog, J. D. Ramsdell, and C. W. Skorupka, “Verifiyng information-flow goals in Security-Enhanced Linux,” Journal of Computer Security, vol. 13, no. 1, pp. 115-134, 2005.

[3] K. L. Clark, “Negation as failure,” in Logic and Data Bases, H. Gallaire and J. Minker, Eds., Springer-Verlag, 1978, pp. 293–322, doi:10.1007/978-1-4684-3384-5_11.

[4] B. Sarna-Starosta, and S. D. Stoller, “Policy analysis for Security-Enhanced Linux,” In WITS’04: Workshop on Issues in the Theory of Security, 2004. Available at http://www.cs.sunysb.edu/~stoller/WITS2004.html [retrieved: February 2015]

[5] D. F. Vukelich, D. Levin, and J. Lowry, “Architecture for cyber command and control: experiences and future directions,” DARPA information survivability conference and exposition, vol. 1, DARPA Information Survivability Conference and Exposition (DISCEX II'01), Volume I, 2001, pp. 155-164.

[6] B. Hicks, S. Rueda, L. St.Clair, T. Jaeger, and P. McDaniel, “A logical specification and analysis for SELinux MLS policy,” Proceedings of the 12th ACM Symposium on Access Control Models and Technologies, June 2007, pp. 91-100.

[7] L. A. Wahsheh, D. Conte de Leon, and J. Alves-Foss, “Formal verification and visualization of security policies,” Journal of Computers, vol. 3, no. 6, June 2008, pp. 22-31.

[8] W. F. Clocksin, and C. S. Mellish, Programming in PROLOG, Springer-Verlag, 1981.

[9] D. Jackson, Software Abstractions, revised edition, Logic, Language, and Analysis, MIT Press, 2011.

[10] K. Fisler, S. Krishnamurthi, L. A. Meyerovich, and M. C. Tschantz, “Verification and change-impact analysis of access-control policies,” Proceeding ICSE '05 Proceedings of the 27th International Conference on Software Engineering, ACM New York, NY, 2005, pp. 196-205.

[11] M. Allen, “Past and Future Threats: Encryption and security in Accumulo.” Presentation at Accumulo Summit, June 2014.

[12] G. Lauer, R. Irwin, C. Kappler, and I. Nishioka, “Distributed resource control using shadowed subgraphs,” ACM Conference on Networking Experiments and Technologies (CoN-EXT), 2013, pp. 43-48, ISBN: 978-1-4503-2101-3.

[13] T. Nelson, A. D. Ferguson, M. J. G. Scheer, and S. Krishnamurthi, “Tierless programming and reasoning for software-defined networks,” NSDI'14 Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation, April 2014, pp. 519-531, ISBN: 978-1-931971-09-6

 ISBN: 978-1-61208-398-8