Confident Firewall Policy Configuration Management using Description Logic

William M. Fitzgerald[1,2] Simon N. Foley[2] Mícheál Ó Foghlú[1]

[1] Telecommunications Software & Systems Group
Waterford Institute of Technology, Ireland
(wfitzgerald,mofoghlu)@tssg.org
[2] Department of Computer Science, University College Cork, Ireland.
s.foley@cs.ucc.ie

Abstract. The provisioning of a firewall is one of the first important steps toward securing access control to a network. However, the effectiveness of a firewall's access control may be limited or compromised by poor configuration and management of firewall policy decisions. Firewall configuration management involves, either the use of a command-line interface with a deep knowledge requirement of the firewall's complex low-level command syntax, or to a limited extent, the use of a graphical management console. Confidence in a firewall configuration is hampered by the complexity of properly comprehending a configuration that achieves the desired business-level security requirements. We outline an approach that enables network administrators to provision firewall configuration policies in a reliable, convenient, conflict-free and automated way. The approach uses Description Logic and the Semantic Web Rule Language to model and infer reliable firewall configuration policies for Linux Netfilter.

1 Introduction

Management of firewall policy configurations can be complex, error-prone, costly and inefficient for many large networked organizations [1]. Implementing a firewall configuration policy either involves writing low-level command syntax via a Command Line Interface (CLI) or the use of a graphical management console. Typical errors in a firewall configuration policy range from invalid syntax and inappropriate rule ordering, to errors in properly comprehending the configuration, given its scale and complexity [2]. The Graphical User Interface, (GUI), is the most commonly used method to configure a firewall in a timely manner, especially amongst inexperienced administrators. One of the major disadvantages of using this method is configuration granularity: as with most modern firewalls there are options that cannot be configured using a graphical interface [2, 1]. Manually managing firewall policy configurations becomes impractical and time consuming as the number of firewall decisions increases beyond the reasonable scope and scale of a human oriented management system [3].

We propose the use of Description Logic to define firewall configuration policies. Description Logic (*DL*) is a knowledge representation language that can be used to express terminological knowledge of an application domain in a structured and formally unambiguous manner [4]. In using *DL*, we capture the firewall knowledge domain and provide a general vocabulary for firewall configuration.

Reasoning plays a vital role in ensuring the quality of a firewall configuration and the Semantic Web Rule Language, *SWRL* [5], provides an ability to validate abstract firewall requirements. Description Logics are a decidable subset of first order logic which facilitates automated inference of implied information from a firewall configuration and the detection of inconsistencies.

2 Firewall Conflicts

Table 1 provides a fragment of a Linux Netfilter firewall access control configuration. In general, firewall decisions are activated in sequence starting at Decision 1. A *shadowed* [6] decision is one that is never activated due to previous decisions filtering the same kinds of packets but those decisions

Decision	Chain	Src IP	Src Port	Dst IP	Dst Port	State	Action	Conflict
1	Forward	*.*.*.*	any	192.168.1.2	80		Drop	
2	Forward	192.168.1.6	any	192.168.1.2	80		Accept	Shadowed by (1)
3	Output	192.168.1.1	any	10.37.2.*	21	Rel	Drop	
4	Output	192.168.1.1	any	10.37.2.*	22	Est	Drop	
5	Output	192.168.1.1	any	10.37.2.3	21,22	Est,Rel	Accept	Shadowed by (3,4)

Table 1. Firewall Decision Policy Example Extract

having different target actions. In Table 1, Decision 2 is shadowed by Decision 1, with the result that `http` traffic from host 192.168.1.6 which is intended to be acceptable, is not permitted. In [6] a number of classes of firewall conflicts are considered, including redundancy, correlation and generalization. For reasons of space we do not consider them in this paper.

3 Description Logic and Ontologies

An ontology is an explicit specification of a conceptualisation using an agreed vocabulary and provides a rich set of constructs to build a more meaningful level of knowledge. An important characteristic of the Semantic Web (as constrained by *DL*) is that the information contained in it is specified using a formal language in order to enable automated reasoning and the derivation of new knowledge from existing knowledge.

Description Logic (*DL*) is a family of logic-based formalisms that forms part of the W3C recommendation for the Semantic Web. *DL* uses classes (concepts) to represent sets of individuals (instances) and properties (roles) to represent binary relations applied to individuals. For example, the *DL* assertion

$$ServerNode \sqsubseteq Node \sqcap \exists hasTrustService.Service \sqcap \exists hasFirewall.Firewall$$

specifies that a server node (class) hosts trusted services (class) and has (property) a firewall (class) protecting them.

The Semantic Web Rule Language *SWRL* complements *DL* providing the ability to infer additional information in *DL* ontologies, but at the expense of decidability. *SWRL* rules are Horn-clause like rules written in terms of *DL* concepts, properties and individuals. A *SWRL* rule is composed of an antecedent part and a consequent part, both of which consist of positive conjunctions of atoms. For example, the requirement *servers hosting ssh based services protected by a firewall require that firewall to open port 22* is expressed in SWRL as:

$$ServerNode(?n) \wedge hasTrustService(?n, s) \wedge hasPort(?s, ssh) \wedge hasFirewall(?n, ?f) \\ \rightarrow hasPortOpen(?f, ssh)$$

4 Netfilter Configuration Ontology

We have developed a *DL*-based ontology for Netfilter. Figure 1 depicts a fragment of the class taxonomy for this model. The taxonomy provides the classes, subclasses and individuals that are inferred from the *DL* specification of the ontology. For example, firewall decision classes are defined as subclasses of the class $NamedFirewallDecision$ which defines the *necessary & sufficient* conditions for the composition of a firewall decision. A Netfilter firewall decision is composed of exactly one chain, one or more condition filters and a single permission target. This is expressed as the *DL* assertion:

$$NamedFirewallDecision \equiv NetfilterFirewall \sqcap \exists_{=1} hasChain.Chain \sqcap \\ \exists_{\geq 1} hasConditionMatch.ConditionFilter \sqcap \exists_{=1} hasTarget.Target$$

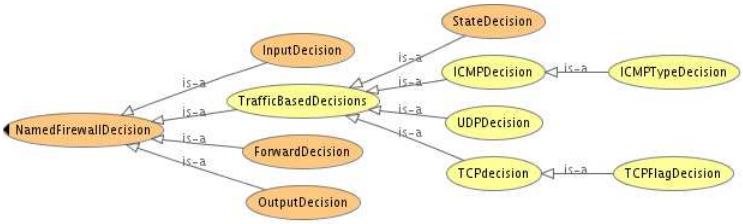

Fig. 1. *DL* Class Taxonomy Fragment

Netfilter's global firewall configuration policy is composed of a number of sub-governing local policies controlled by each of its in-built chains (*Chain* class) to which various protective packet condition filters (*ConditionFilter* class) and their respective verdict permissions (*Target* class) are applied. The following Netfilter CLI syntax represents a decision to accept inbound packets that are part of previously permitted sessions based on stateful filtering: iptables -A INPUT -m state - -state RELATED,ESTABLISHED -j ACCEPT. To capture this in our model, it is first necessary to define the membership constraints of class *InputDecision* (a more specialised *NamedFirewallDecision* class):

$$InputDecision \equiv NamedFirewallDecision \sqcap \in hasChain.inputChain$$

Following this, one can now instantiate an individual (for example, *id*) of *InputDecision*, a class that contains the set of firewall INPUT decisions in our model, by assigning specified properties from individual *id* to individuals of relevant classes:

$$< id >: InputDecision = < id, inputChain >: hasChain \sqcap < id, rel >: hasState \sqcap \\ < id, est >: hasState \sqcap < id, accept >: hasTarget$$

5 Firewall Configuration Conflict Detection

Recall the example of shadowed firewall decisions in Section 2. A *SWRL* rule that tests for a shadowed decision is expressed as

$$ForwardDecision(?x) \wedge ForwardDecision(?y) \wedge \\ abox : hasProperty(?ip1, hasIPMember, protoIP192.168.1.6) \wedge \\ abox : hasProperty(?ip2, hasIPMember, protoIP192.168.1.2) \wedge \\ hasSrcIP(?x, protoIP192.168.1.6) \wedge hasDstIP(?x, protoIP192.168.1.2) \wedge hasDstPort(?x, portHTTP) \wedge \\ hasSrcIP(?y, ?ip1) \wedge hasDstIP(?y, ?ip2) \wedge hasDstPort(?y, portHTTP) \wedge \\ hasTarget(?x, acceptTarget) \wedge hasTarget(?y, dropTarget) \wedge \\ hasRuleOrder(?x, ?a) \wedge hasRuleOrder(?y, ?b) \wedge \\ swrlb : greaterThan(?a, ?b) \wedge differentFrom(?x, ?y) \rightarrow \\ Shadowed(?x) \wedge isShadowedTo(?x, ?y)$$

This states that, on discovering a FORWARD chain super-set firewall decision ($?y$) previous to the firewall decision in question ($?x$) then that subset decision ($?x$) should be re-classified under the class *Shadowed* and to have that subset decision ($?x$) set its *isShadowedTo* property to that of super-set decision ($?y$). Thus, firewall Decision 2 is classified as shadowed by firewall Decision 1.

While automatic resolution of conflicts is not currently expressed in the Netfilter knowledge base, it can be added by expressing it in the ontology using *DL* and *SWRL*. This extensibility provides a 'plug and play' approach to managing policies and is one of the advantages of using semantic *DL* knowledge. *SWRL* also provides an SQL-like notation that can be used to structure information retrieved by the *DL* knowledge base. For example, the query

$$Shadowed(?x) \wedge isShadowedTo(?x, ?y) \rightarrow query : select(?x, ?y)$$

gives a list of tuples $x \mapsto y$, where x is a decision and y the decision that shadows x. When executed against the knowledge in Table 1 it returns tuples $2 \mapsto 1$, $5 \mapsto 3$ and $5 \mapsto 4$.

The *DL* constrained ontology approach provides more than just classification and conflict detection. While we have codified extended forms of the conflict-tests in [6], our approach provides a more general framework for exploring firewall and network configurations. For example, queries such as *what entities on subnet 192.168.1.* can access the ssh service on 192.168.1.1 (the firewall)?* or *list all open ports on 192.168.1.6 accessible to entities other than the 192.168.1.* subnet.*

6 Discussion and Conclusion

In this paper, we outline our approach to using an ontology to represent firewall configuration knowledge. This model effectively reflects the semantic knowledge that a firewall administrator should 'keep in their head' when writing and/or updating firewall decisions. While it may be relatively straightforward to fully comprehend a configuration containing a small number of decisions, it is evident from the apparent complexity of the proposed *DL* model, that this human comprehension does not scale to a large number of decisions. It can be difficult for an administrator to fully appreciate the impact of adding or changing firewall decisions. Thus the need for automated support. We have implemented our *DL* model using Protégé [7] with its ontology *DL* plug-ins, and this provides an administrator with configuration models that are consistent, and more straightforward to navigate and comprehend. Thus, the administrator can have more *confidence* in the policies that they configure.

A number of approaches have been proposed for the formal analysis of firewalls [8–12]. For example, model-checking techniques [8, 11] are used to test that a configuration of firewalls uphold a global routing policy that restricts certain data to certain sub-nets. In [10] constraint programming is used as an approach to finding suitable firewall rules from higher level policy constraints. The focus in these approaches is more on analyzing that firewall rules uphold particular correctness properties, or on synthesising firewall rules from specified correctness properties. While this notion of correctness does, in effect, provide semantics for firewall configuration under a limited number of a priori properties, it is not intended to provide a framework for general knowledge representation about firewalls. The *DL* approach, while not as expressive as the logics that underlie [8–12], is intended to allow the knowledge base to be extended and managed in general.

This paper focuses on the issues of firewall conflicts classified by [6] as a useful case study to demonstrate the appropriateness of the *DL* approach in resolving firewall policy conflicts. The provision of reasoning, in particular within the context of OWA, provides the DL approach with greater flexibility and extendability for incorporating new knowledge. Our model not only caters for conflict firewall decision anomalies, it also provides a means to provide proper explanation for misconfigured firewall rules.

Our *DL* model not only detects pair-wise conflicts between two firewall decisions (like the approach taken by [6]) but it can readily detect the conjunctions of partial conflicts in a set-wise fashion that may occur across multiple decisions in regard to a specified decision being analysed. For example, Decisions 3-5 of Table 1 captures the filtering of stateful outward ftp and ssh traffic of the firewall itself towards the 10.37.2.* subnet and 10.37.3.3 node respectively whereby connections are Established or Related. Our model can capture that Decision 5 is 'partially shadowed' by the conjunction of a number of individual proceeding decisions namely 3 & 4 (ports and state).

In general, *DL* provides a basis for *extendability*, *interoperability* and *complex composition* of other security domains of interest. For example, by developing new ontologies for application-level proxies, one can then model and reason about configurations that involve firewall decisions and proxy decisions. Conflict anomaly-style analysis could be carried out to determine how decisions of one security mechanism may interfere with the decisions of another.

References

1. Wool, A.: A Quantitative Study of Firewall Configuration Errors. COMPUTER, IEEE Computer Society Press, Vol. 37, No. 6, pp. 62-67 (2004)
2. Wack, J., Cutler, K., Pole, J.: Guidelines on Firewalls and Firewall Policy: Recommendations of the National Institute of Standards and Technology. NIST, Special Publication 800-41 (2002)
3. Golnabi, K., Min, R., Khan, L., Al-Shaer, E.: Analysis of Firewall Policy Rule Using Data Mining Techniques. In the 10th IEEE/IFIP Network Operations and Management Symposium, (NOMS) (2006)
4. Baader, F., Calvanese, D., McGuinness, D., Nardi, D., Patel-Schneider, P.: The Description Logic Handbook: Theory, Implementation and Applications. Cambridge University Press (2003)
5. Horrocks, I., Patel-Schneider, P.F., Boley, H., Tabet, S., Grosof, B., Dean, M.: SWRL: A Semantic Web Rule Language Combining OWL and RuleML. W3C Member Submission (2004)
6. Al-Shaer, E., Hamed, H., Boutaba, R., Hasan, M.: Conflict Classification and Analysis of Distributed Firewall Policies. In IEEE Journal on Selected Areas in Communications, Volume 1-1 (2005)
7. Gennari, J., Musen, M.A., Fergerson, R.W., Grosso, W.E., Crubezy, M., Eriksson, H., Noy, N.F., Tu., S.W.: The Evolution of Protege: An Environment for Knowledge-Based Systems Development. In: Proceedings of International Journal of Human-Computer Studies, Volume 58 , Issue 1. (2003)
8. Guttman, J.D.: Filtering Postures: Local Enforcement for Global Security Policies. IEEE Symposium on Security and Privacy, Oakland (1997)
9. Mayer, A., Wool, A., Zishind, E.: Fang: A Firewall Analysis Engine. 2000 IEEE Symposium on Security and Privacy, p. 0177 (2000)
10. Eronen, P., Zitting, J.: An Expert System for Analyzing Firewall Rules. (In: In Proceedings of the 6th Nordic Workshop on Secure IT Systems (NordSec 2001), pages 100-107)
11. Hazelhurst, S.: A Proposal for Dynamic Access Lists for TCP/IP Packet Filtering. South African Computer Journal, Vol. 33 (2004)
12. Marmorstein, R., Kearns, P.: A Tool for Automated iptables Firewall Analysis. (USENIX Annual Technical Conference, FREENIX Track)