Network Access Control Configuration Management using Semantic Web Techniques

William M. Fitzgerald and Simon N. Foley

Cork Constraint Computation Centre, Department of Computer Science, University College Cork, Ireland
wfitzgerald@tssg.org
s.foley@cs.ucc.ie

William M. Fitzgerald and Micheál Ó Foghlú

Telecommunications Software & Systems Group, Waterford Institute of Technology, Ireland
mofoghlu@tssg.org

Network Access Control requirements are typically implemented in practice as a series of heterogeneous security-mechanism-centric policies that span system services and application domains. For example, a Network Access Control policy might be configured in terms of firewall, proxy, intrusion prevention and user-access policies. While defined separately, these security policies may interoperate in the sense that the access requirements of one may conflict and/or be redundant with respect to the access requirements of another. Thus, managing a large number of distinct policies becomes a major challenge in terms of deploying and maintaining a meaningful and consistent configuration. It is argued that employing techniques of the Semantic Web—an architecture that supports the formal representation, reasoning and sharing of heterogeneous domain knowledge—provides a natural approach to solving this challenge. A risk-based approach to configuring interoperable Network Access Control policies is described. Each Network Access Control mechanism has an ontology that is used to represent its configuration. This knowledge is unified with higher-level business (risk) rules, providing a single (extensible) ontology that supports reasoning across the different Network Access Control policy configurations.

Keywords: Security Configuration, Network Access Control, Ontology, Risk, Semantic Web
ACM Classification: C.2.0., C.2.3., C.2.5., I.2.4.

1. INTRODUCTION

While application-level services may provide their own access controls, it is standard practice to deploy additional security mechanisms, such as firewalls and proxies in order to provide a 'layered approach' to security. In practice, security requirements are implemented as a series of independent security-mechanism-centric policies that span multiple system services and application domains. As a consequence, proper application operation is dependent on each security policy configuration. An overly-restrictive policy configuration may prevent normal interoperation of services with the resulting failure of the application. An overly-permissive policy configuration, while permitting normal operation of the application, may render the layers of security ineffective; a service hosted on a subnet is at greater risk of compromise if the firewall is configured with an open-access policy.

Manuscript received: 23 April 2008
Communicating Editor: Eduardo Fernandez-Medina Paton

In this paper we are interested in the configuration of Network Access Control's (*NAC*'s). Configuration of a single *NAC* policy, notwithstanding multiple interoperable *NAC*'s, can be complex, for example; a firewall policy may run to many thousands of rules and is typically maintained on an ad-hoc basis (Al-Shaer, Hamed, Boutaba and Hasan, 2005; Gheorghe, 2006). New access control rules are often added to a NAC policy configuration with little regard to how they interoperate with existing rules and likely resulting in an overly-restrictive and/or overly-permissive configuration. Similarly, changes to the policy of one *NAC* mechanism (for example, a firewall) may indirectly impact the intent of a policy of another *NAC* mechanism (for example, a proxy). The ideal *NAC* configuration provides for consistent interoperability of NAC policies that support valid service traffic, and, preferably, no more and no less.

The vision of the Semantic Web is the ability to express World Wide Web information in a natural and formal language that can be interpreted by intelligent agents, thus permitting them on behalf of the human user to locate, share and integrate information in an automated way. It provides a framework for dynamic, distributed and extensible structured knowledge (ontology) founded on formal logic (Alesso and Smith, 2006; Smith, Welty and McGuinness, 2004). An ontology is an explicit specification of a conceptualization using an agreed vocabulary and provides a rich set of constructs to build a more meaningful level of knowledge. Ontologies and their associated reasoners are the building blocks of the Semantic Web initiative.

We argue that the Semantic Web framework provides a natural approach to constructing, reasoning about and managing security policies: a security policy can be regarded as an explicit specification of terminological knowledge regarding security mechanism configuration, that is, an ontology. Separate ontologies can be developed for different security policies that are naturally composable under the open-world assumption, providing a unified view of the configuration of enterprise-wide security policy. It provides for separation of concerns, whereby security concerns (security policy) and business concerns (business policy) can be separately developed, with reasoning and deployment over their composition. It also means that information about new mechanisms, policy configurations and vulnerabilities can be incorporated as new facts within the existing ontology knowledge-base.

This paper describes a risk-driven model for heterogeneous *NAC* configuration interoperability.

A series of ontologies are developed using *Description Logic* that describe network and host-based access-control knowledge at the systems-layer and at the application-layer. Ontologies are described for the Netfilter/iptables firewall (Gheorghe, 2006), TCP-Wrapper (Venema, 1992) and for application-level access requirements that reflects the high-level business goals.

The contribution of this paper is a *NAC* policy model ontology, over which both, intra- and inter-operation of policies can be reasoned. At the intra-operation level, one can reason about conflicts within a policy configuration: for example, firewall consistency checking can be performed over the Netfilter ontology. One can also reason across different policies (inter-operation): by combining the *NAC* ontology with the application-level risk access ontology, one can reason about service reachability and access permissiveness. The resulting model also demonstrates the practical effectiveness of using Semantic Web techniques in constructing, reasoning about and managing the configuration of security policies.

This paper is a revised and extended version of Fitzgerald, Foley and Foghlú (2008) and is organised as follows. Section 2 introduces Description Logic (DL) (Baader, Calvanese, McGuinness, Nardi and Patel-Schneider, 2003). Section 3 outlines the security policy architecture. Individual ontologies for Netfilter firewall, TCP-Wrapper proxy and application-level access requirements are described in Section 4. Section 5 describes a typical small to medium enterprise environment that is used to represent configuration of a network of systems, services, proxies, and so forth. This enterprise environment provides the basis for configuration reasoning (synthesis and analysis), which is considered in Section 6.

2. DESCRIPTION LOGIC AND ONTOLOGIES

Description Logic (*DL*) is a family of logic-based formalisms that are well-suited for the representation of and reasoning about terminological (*T-box*) and assertional (*A-box*) knowledge based on the Open World Assumption (OWA) (Baader *et al*, 2003; Haarslev and Moller, 2003). DL represents a decidable portion of first-order logic and forms part of the W3C recommendation for the Semantic Web (Smith *et al*, 2004).

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:

$$\begin{aligned}&Server \subseteq Node \cap \\ &\qquad \exists\ hasHosted.BusinessService \cap \exists\ hasProtection.ProtectionService\end{aligned}$$

specifies that a server node (class *Server*) hosts business services (class *BusinessService*) and has (property) a protection service (class) protecting them. Note that properties are conventionally prefixed by "*has*" or "*is*"; for instance, *hasProtection*, is the property over the individuals of the class *Server* (domain) that is protected by a protection service *ProtectionService* (range). The inverse of *hasProtection* is the property *isProtectionOf*.

The Semantic Web Rule Language, (*SWRL*), complements *DL* by 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 (body) part and a consequent (head) part, both of which consist of positive conjunctions of atoms (O'Connor, Knublauch, Tu, Grosof, Dean, Grosso and Musen, 2005). For example, the requirement: *servers hosting* ssh *based business services protected by a firewall require that firewall to open port 22* is expressed in *SWRL* as:

$$\begin{aligned}&Server(?n) \wedge hasHosted(?n, ?s) \wedge hasPort(?s, \texttt{ssh}) \wedge hasFirewall(?n, ?f) \\ &\rightarrow hasPortOpen(?f, \texttt{ssh})\end{aligned}$$

where ssh is an atom/constant and *?s*, *?n* and *?f* represent unbound variables in the rule.

3. SECURITY POLICY MODEL

It is considered best practice to secure business critical applications/services using a layered approach, in particular, by employing low-level infrastructure access control (firewalls, intrusion prevention and so forth) in addition to any security provided at the service-level. Figure 1 provides an abstract policy architecture whereby the elements represent *DL* classes that are defined in later sections and arcs represent relationships between the classes. The architecture provides for three levels of access control: Netfilter, TCP-Wrapper and a business-risk policy. The Netfilter and TCP-Wrapper policies are explicitly enforced by their underlying mechanisms. Section 6 considers how the overall configuration is reasoned over to determine whether the business-risk policy is upheld.

A network is modeled as a collection of systems *Nodes* that may host *BusinessServices*. A service *RiskPolicy* defines the risk of allowing particular clients (*ClientRange*) having access to the service. For example, a particular range of IP addresses may be considered high-risk to a mail server due to a large volume of spam messages, while IP addresses with a service agreement are considered low-risk. Each node is deployed with zero or more *ProtectionService*(s) that provide

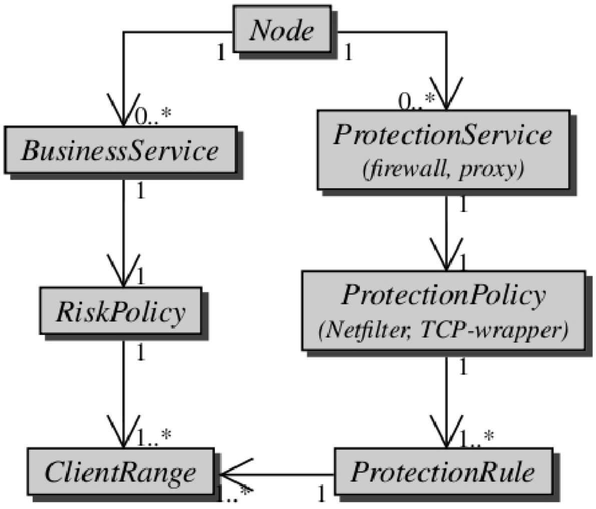

Figure 1: Ontology Policy Architecture

access control mechanisms such as *Firewall*(s) and *Proxy*(s). These protection services are configured in terms of a set of *ProtectionRule*(s) that define its *ProtectionPolicy*. A protection rule, amongst other components, is defined in terms of a range of IP addresses (*ClientRange*) that may be permitted access.

4. SECURITY POLICY ONTOLOGY

In this section *DL* ontologies are outlined for the model components in Figure 1.

4.1 System Ontology

Class *Node* represents the set of system nodes. Nodes may host business and protection services and are connected to each other (*hasConnectionTo*) to form a network.

$$\begin{aligned} Node \subseteq\ & \exists_{\geq 1}\, hasConnectionTo.Node\ \cap \\ & \forall_{\geq 0}\, hasHosted.BusinessService\ \cap \\ & \forall_{\geq 0}\, hasProtection.ProtectionService \end{aligned}$$

Note that both disjoint and covering axioms are used extensively in class definitions throughout our formal model. However, for reasons of space, these axioms are not included. The reader should assume that all sibling classes defined in the paper are disjoint unless stated otherwise. The Appendix provides some background information on *DL*.

A *Node* instance, for example; lanNode that hosts a business service openVPN that is connected to a gateway node, (gwNode) and is also protected by various access control service mechanisms namely a gateway firewall (gwfw), a local firewall (lfw) and a local proxy (lproxy) is illustrated in the following expression. In this paper atomic individuals are written in a San Serif font.

$$\begin{aligned} Node(\mathsf{lanNode}) \leftarrow\ & hasConnectionTo.(\mathsf{lanNode}, \mathsf{gwNode})\ \cap \\ & hasHosted.(\mathsf{lanNode}, \mathsf{openVPN})\ \cap \\ & hasProtection.(\mathsf{lanNode}, (\mathsf{gwfw}, \mathsf{lfw}, \mathsf{lproxy})) \end{aligned}$$

The *BusinessService* class represents the set of business services deployed to nodes within the network.

$$BusinessService \subseteq \text{Service} \cap \exists_{\geq 1}\ hasDeployedTo.Node$$

The class of services, *Service* is a business service or a protection service.

$$ProtectionService, BusinessService \subseteq Service$$

Access control based services (for example, TCP-Wrapper, Netfilter) are subclasses of *ProtectionService*, where membership of these classes requires protection of at least one node.

$$ProxyService, FirewallService \subseteq ProtectionService \cap \exists_{\geq 1}\ isProtectionOf.Node$$

4.2 Risk Policy Ontology

4.2.1 Security Confidence

Every node has a security confidence; (high, medium and low), that indicates the degree to which the services that it hosts are protected.

$$Node \subseteq \exists_{=1}\ hasAssurance.SecurityConfidence$$

Thus a *Node* individual must also have a *hasAssurance* relationship to an individual of *SecurityConfidence*. This value could be based on an assessment of security of the node itself (for example, Unix or Security Enhanced Linux), the subnet it resides in, and/or the protection services that it hosts. Incorporating the property *hasAssurance*.(lanNode, high) into the same individual lanNode described earlier indicates that the high security assurance reflects that lanNode has multiple and combined protection by various access control mechanisms namely a gateway firewall (gwfw), a local firewall (lfw) and a local proxy (lproxy).

Each business service has a minimum security confidence requirement for any node on which it is hosted and is defined in the business service definition as:

$$BusinessService \subseteq \exists_{=1}\ hasAssurReq.SecurityConfidence$$

4.2.2 Risk Metric

Each business service has associated risks (*hasRisk*) of permitting clients (IP address) access to the service. Ideally the sum of the associated risks should not be greater than the maximum acceptable risk threshold (*hasRiskThreshold*).

$$BusinessService \subseteq \exists_{\geq 1}\ hasRisk.RiskPolicy \cap \exists_{=1}\ hasRiskThreshold.Float$$

While a service will have clients with which it has service agreements (risk value of zero), there can be scenarios where it is expedient to tolerate potential service access from other client/sources *hasRisk.RiskPolicy*, (notwithstanding the service's internal access controls). For example, it might be considered low to moderate-risk for a mail service to accept packets from addresses that are believed to be spam sources, while accepting packets from addresses that have been blacklisted as botnet sources is considered high risk. The service risk threshold reflects an business/security trade-off decision. Section 6.3.1 considers how a configuration is tested against this measure.

$$\begin{aligned} RiskPolicy \subseteq\ & \exists_{\geq 1}\ hasRiskIPStartRange.Integer \cap \\ & \exists_{\geq 1}\ hasRiskIPEndRange.Integer \cap \\ & \exists_{=1}\ hasRiskV\ alue.Float \end{aligned}$$

4.3 Netfilter Ontology

Netfilter (Gheorghe, 2006) is a framework that enables packet filtering, network address translation (NAT) and packet mangling. As a firewall, it is both a stateful and stateless packet filter that is characterised by a sequence of firewall rules against which all packets traversing the firewall are filtered. Each firewall rule takes the form of a series of conditions representing packet attributes that must be met in order for that rule to be applicable, with a consequent action for the matching packet (accept, drop, log and so forth).

Netfilter requires the specification of a *table* (`filter`, `NAT` or `mangle`), a *chain*, the accompanying rule *condition* details and an associated *target* outcome. A table is a set of chains and it defines the global context, while chains define the local context within a table. Our research focuses on the firewalling aspects of Netfilter and hence our current model only incorporates the `filter` table attributes. A chain (`INPUT`, `OUTPUT`, `FORWARD`) is a set of firewall rules and those rules in a chain are applied to the context defined both by the chain itself and the particular table. By default there is no need to specify the `filter` table (using `-t` option) when defining firewall rules. A Netfilter rule has the following components.

```
[Table][Chain Type][Filter Conditions][Target Decision]
```

For example, a rule that states that HTTP traffic along the `FORWARD` chain outward bound from the trusted internal interface `eth0` is permitted is defined by the following syntax:

```
iptables -t filter -A FORWARD -o eth0 -p tcp --dport 80 -j ACCEPT
```

4.3.1 Firewall Rule Composition

A Netfilter firewall rule is composed of exactly one chain, one or more condition filters and a single permission target. Various kinds of firewall rules can be defined as subclasses of the *NamedFirewallRule* class in our model. The following definition is constructed in terms of the model vocabulary. Note for reasons of space, we do not provide the semantic explanations of individual model components, however such explanations are described in Foley and Fitzgerald (2008a) and Foley and Fitzgerald (2008b):

$$\begin{aligned} \mathit{NamedFirewallRule} \equiv \mathit{NetfilterFirewall} \cap \; & \exists_{=1}\, \mathit{hasChain}.\mathit{Chain} \cap \\ & \exists_{\geq 1}\, \mathit{hasCondition}.\mathit{ConditionFilter} \cap \\ & \exists_{=1}\, \mathit{hasTarget}.\mathit{Target} \end{aligned}$$

To instantiate `INPUT` rules, for example, it is first necessary to define the membership constraints of class *InputRule*. This class is a more specialised *NamedFirewallRule* class whereby `INPUT` rules must have a relationship to a specific individual of the *Chain* class called inputChain:

$$\begin{aligned} \mathit{InputRule} \equiv \mathit{NamedFirewallRule} \cap \; & \\ & \in \mathit{hasChain}.\mathsf{inputChain} \end{aligned}$$

4.3.2 Firewall Configuration

The previous section describes an ontology blueprint for firewall configuration. A firewall rule instance, defined in terms of specific ports, protocols, etc., is represented as an instance of the ontology defined in terms of class individuals. For example, the firewall rule

```
iptables -A INPUT -s 79.8.2.9 -d 192.168.1.3 -p tcp --dport 22 -j ACCEPT
```

is represented by an individual fwInRule in the ontology whereby *InputRule*(fwInRule) holds; intuitively, we can think of fwInRule as an individual of class InputRule. This individual is defined within the ontology as:

$$\begin{aligned} \mathit{InputRule}(\text{fwInRule}) \leftarrow\ & \mathit{hasChain}(\text{fwInRule, inputChain}) \cap \\ & \mathit{hasSrcIP}(\text{fwInRule, ip79.8.2.9}) \cap \\ & \mathit{hasDstIP}(\text{fwInRule, ip192.168.1.3}) \cap \\ & \mathit{hasProtocol}(\text{fwInRule, tcp}) \cap \\ & \mathit{hasDstPort}(\text{fwInRule, ssh}) \cap \\ & \mathit{hasTarget}(\text{fwInRule, acceptTarget}) \end{aligned}$$

Note that the low-level facts of a firewall configuration are presented as individuals rather than classes on the basis that they are atomic and will not be further decomposed. Using instances (rather than subclasses) allows subsequent reasoning of collections of firewall rules using *SWRL* (Horrocks and Patel-Schneider, 2004).

4.4 TCP-Wrapper Ontology

The Linux/Unix-based TCP-Wrapper service is a host-based transport layer proxy that provisions network access control to local daemons spawned by the Internet services daemon (*inetd*). Under typical circumstances, Linux environments use a super server (*inetd*) to invoke TCP/IP based network services, for example the *SSH* service. Instead of invoking the `sshD` daemon directly the *inetd* daemon will invoke the TCP-Wrapper daemon. The TCP-Wrapper proxy will permit or deny access to the requested service daemons it protects based on the requesting client (for e.g. an IP address) as ascertained from the *inetd* network connection. If a rule has concluded with a permit action then the TCP-Wrapper proxy shall invoke the appropriate service daemon.

A TCP-Wrapper configuration – a set of access control rules to protect host-based services – is specified in terms of access control files *hosts.allow* and *hosts.deny*. Recent versions of TCP-Wrapper allow the access-controls to be specified in a single file. Like firewall rules, the ordering of rules is vital, thus once a rule has been matched no further rules are processed. A TCP-Wrapper rule has the following components.

```
[Daemon List][Client List][Options][Target Action]
```

For example, a rule to deny access to an ftp server from the requesters of a particular subnet whereby attempts by IP addresses of that subnet to access the protected service are logged and time-stamped (using `spawn`) can be written as follows.

```
ftpD:192.168.1. :spawn /bin/echo '/bin/date' %h >> /var/log/ftp.log:deny
```

4.4.1 Proxy Rule Composition

In this section, we provide an overview of TCP-wrapper rule constraints. A complete model for TCP-wrapper rules is described in Foley and Fitzgerald (2008a). A TCP-Wrapper rule at its simplest level can be described as having the following components; one or more service daemons, one or more requesting clients, zero or more options and exactly one permission target action and is represented by the assertion:

$$\begin{aligned} \mathit{NamedProxyRule} \equiv\ & \mathit{TCPWrapperDomain} \cap \\ & \exists_{\geq 1}\ \mathit{hasDaemon.Daemon} \cap \\ & \exists_{\geq 1}\ \mathit{hasClient.Client} \cap \\ & \forall_{\geq 0}\ \mathit{hasOption.Option} \cap \\ & \exists_{=1}\ \mathit{hasAction.Action} \end{aligned}$$

4.4.2 Proxy Configuration

A TCP-Wrapper rule prule, (an individual of *NamedProxyRule*) that states a trusted client IP address is permitted access to the protected ssh server (sshD daemon) is represented by the following:

$$\begin{aligned} \mathit{NamedProxyRule}(\text{prule}) \leftarrow\ & \mathit{hasDaemon}.(\text{prule, sshD}) \cap \\ & \mathit{hasClient}(\text{prule, ip79.8.2.9}) \cap \\ & \mathit{hasAction}(\text{prule, allow}) \end{aligned}$$

The corresponding TCP-Wrapper syntax detailing the access control of the previous knowledge base individual prule is written as:

```
sshD: 79.8.2.9 : allow
```

5. SYSTEM CONFIGURATION MODEL

Intuitively, a configuration is a collection of nodes, services, proxies, et cetera, and their relationships; it is represented as collections of instances from the ontologies. A small to medium enterprise (SME) network environment typically deploys a gateway firewall as the initial step in provisioning access control and a succession of locally hosted access control mechanisms may follow. Figure 2 illustrates an example of a network access control architecture whereby internal nodes hosting business applications are protected by low-level infrastructure namely a gateway firewall and may also be further protected by a local firewall and/or local proxy deployed on those same nodes.

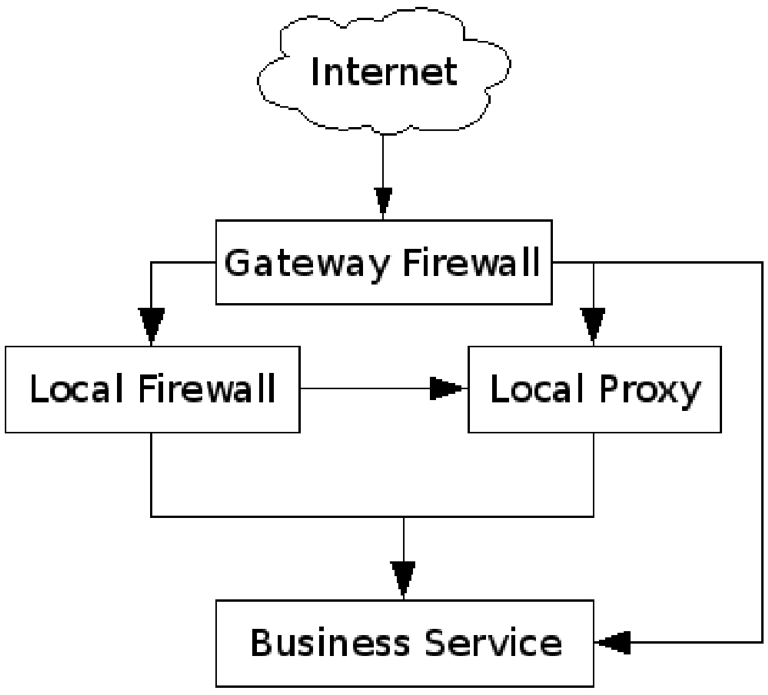

Figure 2: Example SME Network Access Control Architecture

6. SYNTHESIS AND ANALYSIS

Secure service deployment and configuration management of *NAC*s has, to date, been complex, error-prone, costly and inefficient for many large networked organizations (Foley, Fitzgerald, Bistarelli, O'Sullivan and Foghlú, 2006; Wool, 2004). Our approach is to represent both the knowledge about the NAC configuration, business access policies and the business services in terms of a uniform ontology. In doing this, it becomes possible to reason about configurations; in particular, synthesis and analysis of inter-operable conflict-free policy configurations. This section explores a number of scenarios of how our approach could be used in practice.

6.1 Synthesizing Service Deployment

When deploying services to suitable nodes protected by varying degrees of *NAC* protection, the synthesis process searches for a system configuration that meets the assurance requirement (*hasAssurReq.SecurityConfidence*) that the service requires of the node on which it is hosted.

SWRL rules and queries can be defined to infer and discover facts about the current system configurations within the network. Taking the scenario of the SME enterprise system configuration presented in Section 5, a node's level of security confidence is based on an assessment of the protection services protecting the node itself. For example, a node protected by a 3-tier access control defense in depth strategy (gateway firewall, local firewall and local proxy) would have a high security confidence. In terms of secure service deployment in correlation with risk policy requirements (*RiskPolicy*), we seek hosting nodes that possess a 'security confidence' that at a minimum meets their 'assurance requirements' (high, medium and low).

Assuming that nodes *?n* and business services *?b* have been assigned security confidence levels (by instantiating the *hasAssurance* and *hasAssurReq* properties), we can reason over the knowledge base to find suitable nodes on which to host services, that is, a hosting nodes confidence is sufficient for the services it may host. Note, *SWRL* variables are prefixed with a "*?* " and are place holders for individuals of the ontology.

$$\begin{aligned}
&Node(?n) \wedge BusinessService(?b) \wedge hasAssurance(?n, ?a) \wedge hasAssurReq(?b, ?br) \wedge \\
&hasSecureConfidenceValue(?a, ?nrate) \wedge hasSecureConfidenceValue(?br, ?brate) \wedge \\
&swrlb{:}lessThanOrEqual(?brate, ?arate) \\
&\rightarrow hasHostingSuitabilityFor(?n, ?b)
\end{aligned}$$

6.2 Synthesizing NAC Policy Configurations

Reasoning accross the entire knowledge base provides an ability to synthesize a suitable *NAC* policy to protect each business level service. In this section, we consider the synthesis of a TCP-Wrapper proxy policy whereby proxy synthesis is done by searching an existing knowledge base of candidate proxy rules. This knowledge base corresponds to a catalog of best-practice rules and can be regarded as consolidating expert knowledge of known solutions to common problems in a reusable package. This catalog-based approach has a disadvantage of not discovering 'new' or previously unknown forms of *NAC* access rules. However, it has the advantage of modeling the behaviour of the *NAC* rules in their entirety, rather than taking a high-level of abstraction, such as Eronen and Zitting (2001), Guttman (1997), Hazelhurst (2004) and Marmorstein and Kearns (2005), in order to support a tractable search.

Table 1 outlines an extract of a *TCP-Wrapper* catalog of best practice rules that we have built based on our model. For the purposes of this paper, sample configurations were tested based on a TCP-Wrapper catalog with twenty rule sets. For simplicity the rules are given in their original syntactic form. For example, the TCP-wrapper rules related to Email are modeled within the ontology as follows.

$$
\begin{aligned}
\mathit{NamedProxyRule}(\text{sendmail}) \leftarrow\ & \mathit{hasDaemon}.(\text{sendmail, smtpD}) \cap \\
& \mathit{hasClient}(\text{sendmail, domain.local}) \cap \\
& \mathit{hasAction}(\text{sendmail, allow})
\end{aligned}
$$

$$
\begin{aligned}
\mathit{NamedProxyRule}(\text{sendmail2}) \leftarrow\ & \mathit{hasDaemon}.(\text{sendmail2, smtpD}) \cap \\
& \mathit{hasClient}(\text{sendmail2, emailSpamClientList}) \cap \\
& \mathit{hasAction}(\text{sendmail2, deny})
\end{aligned}
$$

$$
\begin{aligned}
\text{NamedProxyRule}(\text{imapreadmail}) \leftarrow\ & \mathit{hasDaemon}.(\text{imapreadmail, imapD}) \cap \\
& \mathit{hasClient}(\text{imapreadmail, domain.local}) \cap \\
& \mathit{hasAction}(\text{imapreadmail, allow})
\end{aligned}
$$

$$
\begin{aligned}
\mathit{NamedProxyRule}(\text{popreadmail}) \leftarrow\ & \mathit{hasDaemon}.(\text{popreadmail, pop3D}) \cap \\
& \mathit{hasClient}(\text{popreadmail, domain.local}) \cap \\
& \mathit{hasAction}(\text{popreadmail, allow})
\end{aligned}
$$

With respect to network access controls, there are two approaches that can be adopted against both known and unknown threats; the ‘*negative security model*’ – permit all traffic by default, thereafter explicitly denying traffic known to be threatening and the ‘*positive security model*’ (also known as a ‘whitelist security model’) – deny everything by default, thereafter explicitly permitting legitimate traffic. The later approach is commonly adopted as best practice, as it is highly effective at preventing known and unknown attacks (Wack, Cutler and Pole, 2002). In this paper, the catalog of best practice (Table 1) takes the ‘positive security model’ philosophy, as is evident from the first best practice rule and for example, from the client whitelist (proceeding three Email rules) whom ‘can’ access the mail server.

6.2.1 Risk-driven Policy Configuration Synthesis

The following *SWRL* rule searches the knowledge base for suitable candidate TCP-Wrapper rules (*?proxyR*), based on the pre-defined catalog, that adequately protects a business service (*?b*) according to that business service’s risk policy constraints;

$$
\begin{aligned}
& \mathit{BusinessService(?b)} \wedge \mathit{RiskPolicy(?r)} \wedge \mathit{ProxyService(?tcpd)} \wedge \\
& \mathit{hasRiskThreshold(?b, ?max)} \wedge \mathit{hasRisk(?b, ?r)} \wedge \mathit{hasPortNumber(?b, ?bpnum)} \wedge \\
& \mathit{hasRiskIPStartRange(?r, ?iprs)} \wedge \mathit{hasRiskIPEndRange(?r, ?ipre)} \wedge \\
& \mathit{hasRiskValue(?r, ?rval)} \wedge \mathit{hasProxyRule(?tcpd, ?proxyR)} \wedge \\
& \mathit{hasDaemon(?proxyR, ?dmn)} \wedge \mathit{hasClient(?proxyR, ?client)} \wedge \\
& \mathit{hasSrcIPStartRange(?client, ?sip)} \wedge \mathit{hasSrcIPEndRange(?client, ?eip)} \wedge \\
& \mathit{ipAddress(?sip, ?x)} \wedge \mathit{ipAddress(?eip, ?y)} \wedge \mathit{hasPortNumber(?dmn, ?dpnum)} \wedge \\
& \mathit{swrlb{:}greaterThanOrEqual(?iprs, ?x)} \wedge \mathit{swrlb{:}lessThanOrEqual(?ipre, ?y)} \wedge \\
& \mathit{swrlb{:}equal(?bpnum, ?dpnum)} \wedge \mathit{swrlb{:}lessThanOrEqual(?rval, ?max)} \\
& \rightarrow \mathit{hasProtectiveProxyRule(?b, ?proxyR)}
\end{aligned}
$$

Best Practice Goal	Explanation
Sample TCP-Wrapper rule-set configuration	
“Deny all requests by any client to all services”	Instantiate a ‘positive security model’ as this defines what is permitted, and rejects everything else. It is security best practice to deny all client connections by default and allow only intended access. Note this rule needs to be inserted as the last rule of the *hosts.allow* configuration.
`all: all: deny`	
“Permit access to email server by all corporate network clients”	Allow access to email (SMTP, IMAP & POP3) services by all hosts in the local domain. The example provided uses DNS name service and can be substituted by the IP address range.

`smtpD: .domainName.local : allow`
`imapD: .domainName.local : allow`
`pop3D: .domainName.local : allow`	

“Deny external blacklisted IP addresses accessing the mail server”	Deny all known SPAM source IP addresses from accessing the SMTP service. There are public spam repositories (for example DNSBL, RHSBL) where these lists can be retrieved. This particular rule will parse the blacklisted clients from a locally stored email spam list.
`smtpD: /blackistDirectory/emailSpamClients.txt: deny`	
“Log and Deny access to the FTP server except from trusted local domain”	Only permitted trusted internal LAN access to FTP resources and deny all others. On denying access, log the time and the offending clients indentity. This information can then be further processed and used to auto generate new proxy rules and/or used by external access control mechanisms such as firewalls and intrusion prevention systems.
`ftpD : all EXCEPT .domainName.local : spawn /bin/echo '/bin/date' %h >> /var/log/ftp.log : deny`	
“Alarm and Deny access to the Samba server except from trusted local domain”	Only permitted trusted internal LAN can use the SMB resources and deny all others. On denying access to offending clients, Email the administrator the offending client (%h) attempts on the SMB service (%d). Note, the rule assumes safe finger is installed in place of the more vulnerable finger daemon.

`smbD : .domainName.local: allow`
`smbD : all spawn /usr/bin/safe finger -l @%h | /usr/bin/mail -s finger %d -%h root : deny`	

Table 1: Catalog Extract of TCP-Wrapper NAC Rules

For example, if the business service (*?b*) in question was an Email daemon service, then one would expect the various email related rules from the TCP-Wrapper catalog defined in Table 1 to be assigned to the email service along the *hasProtectiveProxyRule* property, provided those TCP-Wrapper rules provision client access within the Email service's risk tolerance range.

6.3 Analysing NAC Policy Configurations

SWRL also provides the ability to analyze a policy configuration in order to discover potential conflicts. These may be conflicts between the *NAC* rules and the risk-driven access requirements of the business service, between the *NAC* rules themselves (intra-policy conflicts) and between interoperating *NAC* policy configurations (inter-policy conflicts).

6.3.1 Risk-driven Policy Configuration Analysis

The ideal *NAC* configuration is one that is aligned with the business service, that is, it permits only valid service traffic, and, no more and no less. The following is an example of a risk-based conflict alignment query, that tests whether the aggregate risk from clients that are permitted (by the protection services) to reach a service exceeds the risk threshold specified by the service. Note that it uses '*sqwrl: select*', which provides a SQL-like query.

$$\begin{aligned}
&BusinessService(?b) \wedge RiskPolicy(?r) \wedge\ hasRiskValue(?r,\ ?rval) \wedge \\
&hasRiskIPStartRange(?r,\ ?iprs) \wedge hasRiskIPEndRange(?r,\ ?ipre) \wedge \\
&hasRiskThreshold(?b,\ ?max) \wedge\ hasRisk(?b,\ ?r) \wedge\ hasProtectiveRule(?b,\ ?proxyR) \wedge \\
&hasClient(?proxyR,\ ?client) \wedge\ hasSrcIPStartRange(?client,\ ?sip) \wedge \\
&hasSrcIPEndRange(?client,\ ?eip) \wedge\ ipAddress(?sip,\ ?x) \wedge\ ipAddress(?eip,\ ?y) \wedge \\
&swrlb : greaterThanOrEqual(?iprs,\ ?x) \wedge swrlb : lessThanOrEqual(?ipre,\ ?y) \\
&\rightarrow\ sqwrl : select(?max) \wedge sqwrl : sum(?rval)
\end{aligned}$$

This compares the ranges of the clients that are made visible by the firewall and proxy decisions (*hasProtectiveRule(?b, ?proxyR)*) with the associated risks of each client range. Should the sum (sqwrl:sum) of individual risks be greater than the maximum acceptable risk threshold then that service is not protected sufficiently by its infrastructure *NAC* policy configuration.

6.3.2 Intra-Policy Conflict Analysis

Al-Shaer *et al* (2005) classify firewall conflicts into four categories: redundancy, shadowing, correlation and generalisation. Due to page constraints, we focus our attention to firewall intra conflicts detected using *SWRL* rules applied across the *DL* constrained knowledge base. Again for reasons of space, we do not consider all four conflict categories in this paper, however they are modeled within our ontology. Table 2 provides a fragment of a Linux Netfilter firewall access control configuration.

Shadowing. In general, firewall rules are activated in sequence starting at Rule 1. A shadowed rule is one that is 'never activated due to previous rules filtering the same kinds of packets but those rules having different target actions'. Table 2, illustrates that Rule 2 is shadowed by Rule 1. Since Rule 2 is never activated, intended http traffic from a specific host is not permitted. *SWRL* rules detect conflicts within a firewall configuration. For example, the following 'partial' *SWRL* rule provides a list of tuples $x \rightarrow y$, where x is a rule and y is the rule that shadows x. When executed against the knowledge in Table 2 it returns tuples $2 \rightarrow 1$.

Rule	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	Rel,Est	Accept	Shadowed by (3,4)

Table 2: Firewall Configuration Example Extract

$$\begin{array}{l}
\mathit{NamedFirewallRule}(?\mathit{fwrule1}) \land \mathit{NamedFirewallRule}(?\mathit{fwrule2}) \land \\
\mathit{decisionOrder}(?\mathit{fwrule1}, ?\mathit{order1}) \land \mathit{decisionOrder}(?\mathit{fwrule2}, ?\mathit{order2}) \land \\
\mathit{hasSrcIPStartRange}(?\mathit{fwrule1}, ?\mathit{ip1start}) \land \mathit{hasSrcIPEndRange}(?\mathit{fwrule1}, ?\mathit{ip1end}) \land \\
\mathit{hasSrcIPStartRange}(?\mathit{fwrule2}, ?\mathit{ip2start}) \land \mathit{hasSrcIPEndRange}(?\mathit{fwrule2}, ?\mathit{ip2end}) \land \\
\mathit{ipAddress}(?\mathit{ip1start}, ?\mathit{ip1s}) \land \mathit{ipAddress}(?\mathit{ip1end}, ?\mathit{ip1e}) \land \\
\mathit{ipAddress}(?\mathit{ip2start}, ?\mathit{ip2s}) \land \mathit{ipAddress}(?\mathit{ip2end}, ?\mathit{ip2e}) \land \\
\mathit{hasDstPortStartRange}(?\mathit{fwrule1}, ?\mathit{dst1ps}) \land \mathit{hasDstPortEndRange}(?\mathit{fwrule1}, ?\mathit{dst1pe}) \land \\
\mathit{hasDstPortStartRange}(?\mathit{fwrule2}, ?\mathit{dst2ps}) \land \mathit{hasDstPortEndRange}(?\mathit{fwrule2}, ?\mathit{dst2pe}) \land \\
\mathit{portNumber}(?\mathit{dst1ps}, ?\mathit{dst1ns}) \land \mathit{portNumber}(?\mathit{dst1pe}, ?\mathit{dst1ne}) \land \\
\mathit{portNumber}(?\mathit{dst2ps}, ?\mathit{dst2ns}) \land \mathit{portNumber}(?\mathit{dst2pe}, ?\mathit{dst2ne}) \land \\
\mathit{hasTarget}(?\mathit{fwrule1}, ?\mathit{tar1}) \land \mathit{hasTarget}(?\mathit{fwrule2}, ?\mathit{tar2}) \land \\
\mathit{differentFrom}(?\mathit{fwrule1}, ?\mathit{fwrule2}) \land \\
\mathit{swrlb} : \mathit{greaterThanOrEqual}(?\mathit{order2}, ?\mathit{order1}) \land \mathit{swrlb} : \mathit{lessThanOrEqual}(?\mathit{ip1s}, ?\mathit{ip2s}) \land \\
\mathit{swrlb} : \mathit{greaterThanOrEqual}(?\mathit{ip1e}, ?\mathit{ip2e}) \land \mathit{swrlb} : \mathit{lessThanOrEqual}(?\mathit{dst1ns}, ?\mathit{dst2ns}) \land \\
\mathit{swrlb} : \mathit{greaterThanOrEqual}(?\mathit{dst1ne}, ?\mathit{dst2ne}) \land \ldots.. \\
\rightarrow \mathit{isShadowedBy}(?\mathit{fwrule2}, ?\mathit{fwrule1})
\end{array}$$

Reasoning across the ontology not only detects pair-wise conflicts between two firewall rules (like the approach taken by Al-Shaer *et al*, (2005)) but it can readily detect the conjunctions of partial conflicts in a set-wise fashion that may occur across multiple rules in regard to a specified rule being analysed. For example, Rules 3-5 of Table 2 captures the filtering of stateful outward ftp and ssh traffic of the firewall itself towards the 10.37.2.* subnet and 10.37.2.3 node respectively whereby connections are Related or Established. Our model can capture that Rule 5 is 'partially shadowed' by the conjunction of a number of individual proceeding rules namely 3 & 4 (ports and state). Note, the same techniques of intra-policy conflict detection are also applicable to TCP-Wrapper proxy configurations.

6.3.3 Inter-Policy Conflict Analysis

Inter-policy conflicts within a configuration can occur between inter-operating *NAC*'s (that is, between inter-operating firewalls or firewalls inter-operating with proxies). *NAC* inter-operation can be analysed using the previous conflict categorisation methods. While individual policy configurations may be conflict free their inter-operation may not be. For example, a Netfilter firewall rule may unintentionally 'shadow' an intended TCP-Wrapper rule.

Netfilter Configuration Extract								
Rule	Chain	Src IP	Src Port	Dst IP	Dst Port	State	Action	Conflict
F1	Forward	*.*.*.*	any	192.168.1.3	22		Drop	Shadows P1
F2	Forward	192.168.1.*	any	192.168.1.3	80		Accept	Redundant to P2

TCP-Wrapper Configuration Extract				
Rule	Daemon	Client	Action	Conflict
P1	sshD	192.168.1.*	allow	Shadowed by F1
P2	httpD	192.168.1.*	allow	Redundant to F2

Table 3: Example of Inter-conflicts between Firewall and Proxy.

Redundancy. Redundancy occurs when 'two or more *NAC* rules can filter the same packets and those rules have the same target actions over those packets. In terms of intra-policy conflict analysis, redundancy adds to the complexity of the search space and can open a protected service to unintended access. For example, unknowingly defining multiple redundant *NAC* rules to permit access to a service and later modifying a single rule to provide a stricter set of access control constraints on that service can still leave that service, now thought to be protected, a security risk based on the remaining redundant rules. This is quite the contrary when performing inter-policy conflict analysis, as redundancy is what is required to avoid 'shadowing' between two or more *NAC* configurations. Redundancy is what provisions for a 'defence in depth' strategy.

Table 3 illustrates a simple example of undesired shadowing and desired redundancy. An example of desired redundancy where both firewall and proxy permit access to a HTTP service is illustrated by rules F2 and P2. The TCP-Wrapper rule, P1, provides the intended internal access to the SSH service from the corporates local subnet (192.168.1.0/24). However, the gateway firewall protecting the subnet, in which the SSH service also resides, is blocking the intended recipients due to rule F1 filtering and dropping 'all' source IP address's. While we currently do not provide conflict resolution there are a number of approaches one can adopt through synthesis to resolve the shadowed conflict and introduce redundancy. For example rule F1 can be modified to replace the source IP range with 192.168.1.* or introduce a new firewall rule F0, to specifically permit intended internal access to the SSH service.

The following *SWRL* rule analyses both the forward relating firewall rules (*?fwrule*) and proxy rules (*?proxyR*) assigned to protect business services for shadowing conflicts. The result of this *SWRL* rule when executed against the knowledge in Table 3 returns the tuple `F1 → P1` via the *hasInterShadowConflictWith* property being inferred in the ontology.

ForwardRule(?fwrule) ∧ *NamedProxyRule(?proxyR)* ∧ *hasClient(?proxyR, ?client)* ∧
hasSrcIPStartRange(?fwrule, ?fipstart) ∧ *hasSrcIPEndRange(?fwrule, ?fipend)* ∧
hasSrcIPStartRange(?client, ?pipstart) ∧ *hasSrcIPEndRange(?client, ?pipend)* ∧
ipAddress(?fipstart, ?fips) ∧ *ipAddress(?fipend, ?fipe)* ∧
ipAddress(?pipstart, ?pips) ∧ *ipAddress(?pipend, ?pipe)* ∧
hasDstPortStartRange(?fwrule, ?dstps) ∧ *hasDstPortEndRange(?fwrule, ?dstpe)* ∧
portNumber(?dstps, ?dstns) ∧ *portNumber(?dstpe, ?dstne)* ∧
hasDaemon(?proxyR, ?dmn) ∧ *portNumber(?dmn, ?dpnum)* ∧

$$\begin{aligned}&hasTarget(?fwrule, \text{dropTarget}) \wedge hasAction(?proxyR, \text{allow}) \wedge\\ &swrlb : lessThanOrEqual(?fips, ?pips) \wedge swrlb : greaterThanOrEqual(?fipe, ?pipe) \wedge\\ &swrlb : lessThanOrEqual(?dstns, ?dpnum) \wedge swrlb : greaterThanOrEqual(?dstne, ?dpnum) \wedge\\ &\rightarrow hasInterShadowConflictWith(?fwrule, ?proxyR)\end{aligned}$$

7. RELATED RESEARCH

An ontology-based model that can be used to (binary) test the safety of an individual firewall policy with respect to a Semantic Web application policy is described in Foley and Fitzgerald (2008b) and Foley and Fitzgerald (2008a). A TCP-wrappers ontology is used as a model a *NAC* policy and *SWRL* queries are used to test whether an *individual NAC* policy supports the access requirements of the Semantic Web application. The results in Foley and Fitzgerald (2008a) and Foley and Fitzgerald (2008b) do not consider *NAC* intra-operation nor inter-operation.

This paper builds on the results of Foley and Fitzgerald (2008b) and Foley and Fitzgerald (2008a) by considering how interoperation of multiple *NAC* policies (involving multiple firewalls and proxies) are affected by more general network service requirements. With this extended model, it is possible to analyze configurations for intra- and inter-policy conflicts, and to also synthesize suitable configurations based on partial configuration knowledge. Furthermore, the risk-metric provides a quantitative approach to aligning a *NAC* policy to service requirements. This paper described a catalog-driven approach when synthesising suitable best practice *NAC* rules. Additional research also illustrates a catalog of best practice rules for Netfilter. We are currently building more extensive catalogs of best practice, for instance, a catalog of *NAC* (Netfilter and TCP-Wrapper) rules reflecting PCI-DSS (DSS, 2006) best practice.

Previous research applying ontologies to the security domain focuses primarily on security taxonomies and tended to go no further than providing abstract models such as Herzog, Shahmehri and Duma (2007) and Kim, Luo and Kang (2005). The model presented in this paper supports reasoning at a lower-level of granularity. Low-level facts of a *NAC* configuration, for example specific protocols and ports from the actual *NAC* syntax, are represented semantically as individuals rather than classes on the basis that they are atomic and will not be further decomposed. Using instances (rather than subclasses) allows subsequent reasoning of collections of *NAC* configuration policies that utilises *SWRL* to extend the expressive capabilities of *DL*.

A number of approaches have been proposed for the formal analysis of firewall policies (Eronen and Zitting, 2001; Guttman, 1997; Hazelhurst, 2004; Marmorstein and Kearns, 2005; Mayer, Wool and Zishind, 2000). For example, model-checking techniques (Guttman, 1997; Hazelhurst, 2004) are used to test that firewall configurations uphold a global routing policy that restricts certain data to certain sub-nets. In Eronen and Zitting (2001) 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*/*SWRL* approach, while not as expressive as the logics that underlie (Eronen and Zitting, 2001; Guttman, 1997; Hazelhurst, 2004; Marmorstein and Kearns, 2005; Mayer *et al*, 2000), is intended to support a general purpose knowledge base about low-level configuration.

Al-Shaer’s research provides a filtering decision tree approach to discover and generate an effective and anomaly free firewall policy configuration (Al-Shaer *et al*, 2005; Golnabi, Min, Khan and Al-Shaer, 2006). We argue that the practicalities of the Semantic Web approach outlined in this paper greatly extends the research of Al-Shaer *et al* (2005). This was illustrated in Section 6.3.2, whereby our approach (unlike Al-Shaer *et al*, 2005) can identify partial conflicts in a set-wise fashion that may occur across multiple rules in regard to a specified rule being analysed. The Semantic Web approach adopted in our model can interpret not just firewall configurations but also those of *NAC*'s in general as demonstrated in Section 6.3.3, whereby both firewall and proxy policies could be analysed for inter-policy conflicts.

8. DISCUSSION AND CONCLUSION

This paper outlined an approach to using a *DL* constrained ontology to construct, reason about and manage *NAC* (Netfilter and TCP-Wrapper) policies in the context of risk-driven service policy requirements.

An ontology by definition is an explicit specification of a conceptualization using an agreed vocabulary. And as such, the *NAC* ontologies outlined in this paper reflect the semantic knowledge a network administrator should 'keep in their head' when writing rules to provide service access control. The provisioning of reasoning, both within the context of *DL*'s OpenWorld Assumption (*OWA*) characteristics (Baader *et al*, 2003) and *SWRL* facts, provides our ontology with the flexibility and extendability of incorporating new unknown *NAC* related knowledge. For example, the ontology not only provides risk-driven *NAC* alignment to business services (synthesis) but it also facilitates subsequent reasoning (analysis) of *NAC* inter- and intra-policy conflict detection. A catalog driven approach is adopted when synthesising candidate *NAC* rules to protect business services in a prescribed way.

Future research will investigate the development a prototype autonomic architecture that is based on this ontology and reasoning framework. Existing Semantic Web techniques can be used to provision an agent-based approach to deploying and maintaining the configuration of security policies in a automated/autonomic manner. Semantic *NAC* agents are responsible for managing the configuration of access controls (firewalls, proxies, IDS's and so forth). These agents negotiate *NAC* settings that are constrained by the current knowledge base, which is in turn controlled by the *NAC* agents and other application agents, managing for example, the business policies. The knowledge base is controlled by adding or deleting facts based on new knowledge and inferences by the agents. For example, a business agent informs a *NAC* agent of a new service offering whereby the *NAC* agent must reconfigure (new facts) to enable appropriate access.

ACKNOWLEDGEMENTS

This research has been funded by SFI Autonomic Management of Communications Networks and Services PI Cluster Award: 04/IN3/I404C.

REFERENCES

AL-SHAER, E., HAMED, H., BOUTABA, R. and Hasan, M. (2005): Conflict classification and analysis of distributed firewall policies, *IEEE Journal on Selected Areas in Communications*, 1-1.

ALESSO, H.P. and SMITH, C.F. (2006): *Thinking on the Web: Berners-Lee, Gdel and Turing*, Wiley-Interscience.

BAADER, F., CALVANESE, D., McGUINNESS, D.L., NARDI, D. and PATEL-SCHNEIDER, P. (2003): *The Description Logic Handbook: Theory, Implementation and Applications*, Cambridge University Press.

DSS, P. (2006): Payment card industry (PCI) data security standard, *PCI Security Standards Council*, version 1.1.

ERONEN, P. and ZITTING, J. (2001): An expert system for analyzing firewall rules, *6th Nordic Workshop on Secure IT Systems*, 100-107, Copenhagen, Denmark.

FITZGERALD, W.M., FOLEY, S.N. and FOGHLÚ, M.O. (2008): Network access control interoperation using semantic web techniques, *6th International Workshop on Security In Information Systems (WOSIS 2008)*, Barcelona, Spain, June 12-13.

FOLEY, S.N. and FITZGERALD, W.M. (2008a): Aligning semanticweb applications with network access controls, *Special Issue of Computer Standards and Interfaces*, Under Review.

FOLEY, S.N. and FITZGERALD, W.M. (2008b): Semantic web and firewall alignment, *1st International Workshop on Secure Semantic Web (SSW'08)*, Cancun, Mexico.

FOLEY, S.N., FITZGERALD, W.M., BISTARELLI, S., O'SULLIVAN, B. and FOGHLÚ, M.O. (2006): Principles of secure network configuration: Towards a formal basis for self-configuration, *6th IEEE International Workshop on IP Operations and Management, (IPOM06)*.

GHEORGHE, L. (2006): *Designing and Implementing Linux Firewalls with QoS using netfilter, iproute2, NAT and l7-filter*, PACKT Publishing.

GOLNABI, K., MIN, R., KHAN, L. and AL-SHAER, E. (2006): Analysis of firewall policy Rule using data mining techniques, *10th IEEE/IFIP Network Operations and Management Symposium, (NOMS)*.

GUTTMAN, J.D. (1997): Filtering postures: Local enforcement for global security policies, *IEEE Symposium on Security and Privacy*, Oakland.

HAARSLEV, V. and MOLLER, R. (2003): Description logic systems with concrete domains: Applications for the semanticWeb', *10th International Workshop on Knowledge Representation meets Databases, (KRDB)*, Hamburg, Germany.

HAZELHURST, S. (2004): A proposal for dynamic access lists for TCP/IP packet filtering', *South African Computer Journal*, 3.

HERZOG, A., SHAHMEHRI, N. and DUMA, C. (2007): An ontology of information security, *International Journal of Information Security and Privacy*, 1:4.

HORROCKS, I. and PATEL-SCHNEIDER, P.F. (2004): A proposal for an OWL rules language, *13th International conference on World Wide Web* .

KIM, A., LUO, J. and KANG, M. (2005): Security ontology for annotating resources, *4th International Conference on Ontologies, Databases, and Applications of Semantics, (ODBASE)*, Agia Napa, Cyprus.

MARMORSTEIN, R. and KEARNS, P. (2005): A tool for automated iptables firewall analysis, *USENIX Annual Technical Conference*, FREENIX Track, Anaheim, CA, USA .

MAYER, A., WOOL, A. and ZISHIND, E. (2000): Fang: A firewall analysis engine, I*EEE Symposium on Security and Privacy*, 017.

O'CONNOR, M., KNUBLAUCH, H., TU, S., GROSOF, B., DEAN, M., GROSSO, W. and MUSEN, M. (2005): Supporting rule system interoperability on the semanticWeb with SWRL, *4th International Semantic Web Conference (ISWC2005)*, Galway, Ireland.

SMITH, M.K.,Welty, C. and McGUINNESS, D.L. (2004): OWLWeb ontology language guide, *W3C Recommendation, Technical Report*.

VENEMA, W. (1992): TCP Wrapper: Network monitoring, access control, and booby traps, *3rd UNIX Security Symposium (Baltimore, September'92)*.

WACK, J., CUTLER, K. and POLE, J. (2002): Guidelines on firewalls and firewall policy: Recommendations of the National Institute of Standards and Technology, *NIST-800-41*.

WOOL, A. (2004): A quantitative study of firewall configuration errors, *COMPUTER, IEEE Computer Society Press*, 37(6):62-6.

APPENDIX: DESCRIPTION LOGIC

DL belongs to a family of logic that represents a decidable portion of first-order logic. The logic is characterised by a set of constructors (Table 4) that allows the construction of complex concepts and roles from atomic concepts or roles. Classes (concepts) represent sets of individuals and properties (roles) represent binary relations applied to individuals. Tables 4 and 5 illustrate parts of *DL* that are used in this paper. The reader is referred to Baader *et al* (2003) for further information.

Constructor	DL Syntax	Example
Intersection Of	$C_1 \cap \ldots \cap C_n$	$Protocol \cap Port$
Union Of	$C_1 \cup \ldots \cup C_n$	$PrivilegedPort \cup UnprivilegedPort$
Complement Of	$\neg C$	$\neg PrivilegedPort$
Universal Quantifier	$\forall P.C$	$\forall hasPort.PrivilegedPort$
Existential Quantifier	$\exists P.C$	$\exists hasPort.PrivilegedPort$
Max Cardinality	$\leq_n$ P	$\leq_{16}$ hasPort
Min Cardinality	$\geq_n$ P	$\geq_1$ hasPort
Exact Cardinality	$=_n$ P	$=_1$ hasPort

Table 4: DL Constructors

Axiom	DL Syntax	Example
SubClass Of	$C_1 \subseteq C_2$	$TCPParam \subseteq TCPFlag \cap Port$
Equivalent Class	$C_1 \equiv C_2$	$Port \equiv PrivilegedPort \cup UnPrivilegedPort$
Disjoint With	$C_1 \subseteq \neg C_2$	$PrivilegedPort \subseteq \neg UnPrivilegedPort$
SubProperty Of	$P_1 \subseteq P_2$	$hasSrcPort \subseteq hasPort$

Table 5: DL Axioms

Classes & Properties: Classes are interpreted as sets of individuals and can be organised into a superclass-subclass hierarchy. For example, *Protocol* is a class that represents the set of all individual protocols and its subclasses include *TCP* and *UDP* classes. Subsumption represents the superclass subclass hierarchy, for example, $TCP \subseteq Protocol$ indicates that TCP is a subclass of *Protocol*.

Properties are used to construct binary relationships between classes. They are used when making statements about classes. For example, the following defines the concept of "ports are either privileged or unprivileged but not both":

$$Port \equiv PrivilegedPort \cup UnPrivilegedPort$$
$$PrivilegedPort \subseteq \neg UnPrivilegedPort$$

Like classes, subproperties specialise their superproperties. For example, the property *hasSrcPort* specialises the property *hasPort*. This states that if two classes are related by the *hasSrcPort* property then an attributed source port is a more specific relationship than the general case of having a port relationship. *DL* has the following properties: functional, inverse functional, transitive and symmetric.

Property Restrictions: Object property restrictions are used to create constraints on individuals that belong to a particular class. Restrictions fall into three categories: *Quantifier*, *Cardinality* and *hasValue* restrictions. An existential ($\exists$) restriction requires *at least one* relationship for a given property to an individual that is a member of a specific class. A universal ($\forall$) restriction mandates that the *only* relationships for the given property that can exist must be to individuals that are members of the specified class. A property restriction effectively describes an anonymous or unnamed class that contains all the individuals that satisfy the restriction. When restrictions are used to describe classes they specify anonymous superclasses of the class being described.

Partial & Complete Classes: A *partial* class definition is specified with *necessary conditions* and is of the form $Class \subseteq SuperClass \cap PropertyConditions$. This states that if an individual is a member of the defined class it must satisfy the conditions that it characterises. However, it cannot be said that any (random) individual that satisfies these conditions must be a member of this class. When a class is defined with *necessary and sufficient conditions* ($\equiv$), like partial classes, then if an individual is a member of the class then it must then satisfy those conditions. However, with the *sufficient condition* included, then any (random) individual that satisfies these conditions must be a member of this class. Classes that have at least one set of *necessary and sufficient conditions* are known as *complete* classes.

Open World Assumption: The Closed World Assumption (*CWA*) and the Open World Assumption (*OWA*) represent two different approaches of how to evaluate implicit knowledge in a knowledge base (Baader *et al*, 2003). The CWA approach makes the presumption that what is not currently known to be true in a knowledge base is false, hence the interpretation of negation as failure.

However, OWA assumes that its knowledge of the world is incomplete. If something cannot be proved to be true in the known knowledge base, then it does not automatically become false. A simple example of OWA would be to assume that we know that a firewall rule has been applied to a range of privileged ports and from this information using the OWA approach one can not conclude that a firewall rule also has or has not some unprivileged ports assigned. Hence, if a class is to be confined to certain constraints, it must be explicitly stated that other unwanted constraints do not exist. In *DL*, OWA characteristics can be contained by stating exactly the components of a class using both *disjointness* and *covering axioms*.

BIOGRAPHICAL NOTES

William Fitzgerald has recieved a BSc honours degree (2000) and MSc by research (2002) from National University of Maynooth. He is currently pursuing a PhD in systems security at University College Cork. William is also a researcher at Telecommunications Software & Systems Group (TSSG), Waterford Instutite of Technology. At TSSG he has collaborated on a number of European FP6 and FP7 IST projects, for example: SecurIST and Daidalos. He previously held a short-term research appointment at Ericsson, Ireland (2003). A detailed biography is available at: http://www.williamfitzgerald.org

William Fitzgerald

Simon Foley is a statutory lecturer in computer science at University College Cork where he teaches and directs research on computer security. He is on the editorial board of the Journal of Computer Security and has served as Program chair of the IEEE Computer Security Foundations Workshop and the ACM/ACSAC New Security Paradigms Workshop. His resesearch interests include security modeling, distributed authorization and enterprise risk management.

Simon Foley

Micheal O Foghlu is research director of and one of the co-founders of Telecommunications Software & Systems Group, an ICT research centre in Waterford Institute of Technology. The TSSG has grown from around five people in 1996 to 160 in September 2008, spinning out companies that employ a further 60 people, and has won nearly 50 Million Euro in funding from Irish and European funding agencies in this period. With an academic background in AI techniques through undergraduate and postgraduate work in the UK (in Keele, Cambridge and Central Lancashire). His current research focus is on network management of IP networks and services. He is a member of the Advisory Committee of the W3C, the Advisory Council of the Internet Society and is Chair of the Irish IPv6 Task Force.

Micheal O Foghlu