Management of Heterogeneous Security Access Control Configuration using an Ontology Engineering Approach

William M. Fitzgerald
Computer Science Department
University College Cork, Ireland
wfitzgerald@4c.ucc.ie

Simon N. Foley
Computer Science Department
University College Cork, Ireland
s.foley@cs.ucc.ie

ABSTRACT

Management of heterogeneous enterprise security mechanisms is complex and requires a security administrator to have deep knowledge of each security mechanism's configuration. Effective configuration may be hampered by poor understanding and/or management of the enterprise security policy which, in turn, may unnecessarily expose the enterprise to known threats. This paper argues that knowledge about detailed security configuration, enterprise-level security requirements including best practice recommendations and their relationships can be modelled, queried and reasoned over within an ontology-based framework. A threat-based approach is taken to structure this knowledge. The management of XMPP application-level and firewall-level access control configuration is investigated.

Categories and Subject Descriptors

K.6.5 [**Management of Computing and Information Systems**]: Security and Protection

General Terms

Security

Keywords

Configuration Management, Ontology, Access Control

1. INTRODUCTION

An enterprise security policy is a high-level policy document that defines a "*set of rules and practices that specify or regulate how a system or organization provides security services to protect sensitive and critical system resources*" [35]. An enterprise security policy provides high-level requirements and is typically not intended to prescribe low-level security controls. In practice, enterprise-level security requirements are implemented as a series of heterogeneous inter-dependent security mechanism configurations that span multiple subnets and system services. As a consequence, proper service operation is dependent on each control configuration. Application-level services are becoming more sophisticated and may provide their own security controls. For example, an XMPP-based Instant Messaging server may use a server white-list to limit connections to trusted domains. Securing this service goes beyond configuring the application-level mechanism: best practice [37] recommends that network access controls such as firewalls should also be used as part of security in depth strategy.

Access control configuration may be hampered by poor understanding and/or management of the enterprise security policy. For example, consider the following enterprise-level security requirements regarding an XMPP server: permit client-to-server (C2S) access to the XMPP server; permit clients to transfer files over XMPP on the internal network and permit server-to-server (S2S) access with trusted external XMPP servers. However, implementing access-control rules for XMPP clients and servers is not just about making ports 5222 (C2S) and 5269 (S2S) accessible on the XMPP server and/or intermediatory access controls such as firewalls. One may wish to only permit certain C2S communication (for example, by an IP address whitelist), require clients to use TLS encryption, accept only SASL [31] authentication from some S2S communication and dialback [24] authentication for others, prevent external file transfer and deal with XMPP traffic that is tunneled through proxies (for example, XMPP HTTP-bind). Furthermore, XMPP traffic does not necessarily have to communicate on the IANA [19] recommended XMPP ports. It may also be prudent to provide content sanitation at the application-layer. For example, controlling Spam over Instant Messaging (SPIM) by filtering known SPIM signatures within XMPP message stanzas.

Notwithstanding the challenge of providing an accurate enterprise security policy, one must also consider how each security mechanism may contribute to policy enforcement. For example, regardless of a firewall's capability to perform deep packet inspection, application-level content filtering may only be feasible using the XMPP server's access control when C2S and S2S communication is encrypted end-to-end; while connection-throttling is best implemented at the firewall. Thus, designing an access control configuration is complex and challenging, and is largely dependent on the expert knowledge of the security administrator drawing upon best practice and standards.

We argue that a framework is required in which one can uniformly represent and reason about the knowledge associated with a security configuration. We take an ontology-

SafeConfig'10, October 4, 2010, Chicago, Illinois, USA.

engineering approach to modelling this security configuration knowledge. Previous research [9, 10] focused on using ontologies to model firewall based access controls; in this paper we consider the modeling of and interplay between these firewall controls and the security controls provided by the application.

An ontology provides a conceptual model of a domain of interest [36]. It does so by providing a formal vocabulary describing various aspects of the domain of interest and provides a rich set of constructs to build a more meaningful level of knowledge. In the case of access control configuration management, an ontology provides the ability to make logical assertions and inferences with which to structure, share and infer new knowledge about the application-level access control and network access control domains.

A threat-based approach is proposed as a means of structuring the knowledge about the management of access control configuration. *Semantic Threat Graphs* [11], a variation of the traditional threat tree, are encoded within the ontology-based framework in order to relate knowledge about enterprise-level security requirements, best practice recommendations and access-control rules in terms of assets, threats, vulnerabilities and countermeasures. Threats are organised into a hierarchical structure such as a Microsoft STRIDE-based [17] hierarchy. Identifying threats in this way, for example Denial of Service attacks, facilitates the generation of appropriate access-control rules (countermeasures) such as connection-throttling and/or automatic blacklisting of offending servers (IP addresses). The semantic threat graph approach takes advantage of an ontology's ability to share and integrate knowledge within other ontologies. Thus, the XMPP and iptables ontologies are reused to describe detailed countermeasure configurations.

The paper is organised as follows. Section 2 outlines an ontology for XMPP security configuration. A threat model for XMPP server hosting is proposed in Section 3. Section 4 describes how enterprise-level security requirements can be captured in terms of semantic threat graphs. A knowledge-base of best practice countermeasures against known threats are encoded as semantic threat graphs in Section 5. For example, XEP-0205 [30] and NIST-800-123 [34] recommend multiple countermeasures that mitigate the threat of Denial of Service when hosting an XMPP server. Section 6 provides some examples that demonstrate synthesis and analysis of these access control configurations.

2. XMPP SECURITY ONTOLOGY

The Extensible Messaging and Presence Protocol (XMPP) is a protocol that is used for instant messaging, voice and video calls, presence, and publish-subscribe services [29]. There are three scenarios whereby access control needs to be considered, namely client-to-server (C2S) communication, client-to-client (C2C) server-less communication and server-to-server (S2S) communication. Server-to-server communication is known as a 'federation' that exists between different enterprise XMPP servers, thus facilitating inter-domain client message routing and search of client contacts [32].

An ontology for XMPP security configuration is developed. For reasons of space only a fragment of the ontology outlining the S2S access control is presented. Concepts represent sets of individuals (instances) and properties (roles) represent binary relations applied to individuals. Note that in presenting the XMPP model components, for reasons of space, complete specifications in particular, disjoint axioms, sub-properties, data type properties or closure axioms are omitted. The ontologies in this paper were developed using Description Logic [5] and SWRL [25], an overview of which is provided in the appendix.

Authentication and Encryption. Concept $S2SAuthRule$ represents the set of authentication rules (individuals) that define one or more authentication methods supported by an XMPP server. Encryption (*tlsEncrypted* property) is also supported over the Transport Layer Security protocol [8].

$$\begin{aligned} S2SAuthRule \sqsubseteq\ & XMPPAuthRule \sqcap \\ & \exists_{\geq 1} authenticateWith.Authenication \sqcap \\ & \exists_{=1} tlsEncrypted.Boolean \end{aligned}$$

Concept *Authentication* defines the vocabulary for federated S2S authentication.

$$\begin{aligned} Authentication \equiv\ & XMPPRuleComponent \sqcap \\ & \{\texttt{saslExternal}, \texttt{dialback}\} \end{aligned}$$

Server federation supports two kinds of authentication methods: Simple Authentication and Security Layer (SASL) EXTERNAL [31] and server dialback [24]. SASL EXTERNAL provides a basis for the initiating server to reuse the certificate presented during TLS negotiation for authentication. While, server dialback is the less secure authentication method, it is commonly used in practice [32].

For example, individual $\texttt{xmppr}_{\texttt{Dialback}}$ represents a requirement for (unencrypted) 'server dialback' for S2S federation.

$$\begin{aligned} & S2SAuthRule(\texttt{xmppr}_{\texttt{Dialback}}) \\ \leftarrow\ & authenticateWith(\texttt{xmppr}_{\texttt{Dialback}}, \texttt{dialback}) \sqcap \\ & tlsEncrypted(\texttt{xmppr}_{\texttt{Dialback}}, \texttt{false}) \end{aligned}$$

For the sake of clarity, meaningful identifiers are associated with individuals. For example, the individual $\texttt{xmppr}_{\texttt{Dialback}}$ identifies an XMPP rule that permits the use of 'server dialback' as its authentication method.

Access Control List. An access control list specifies the XMPP servers ($IPAddress$ or $DNSName$) which a server may $federateWith$ (property), where $Permission$ is an enumerated set of individuals `allow` and `deny`.

$$\begin{aligned} ACLRule \sqsubseteq & XMPPRule \sqcap \\ & \exists_{=1} federateWith.(IPAddress \sqcup DNSName) \sqcap \\ & \exists_{=1} hasPermission.Permission \end{aligned}$$

For example, individual $\texttt{xmppr}_{\texttt{AllowUCC}}$ an instance of concept $ACLRule$, states that the XMPP server will `allow` federated requests from an XMPP server with the domain name `www.ucc.ie` (individual identifier `ucc`).

$$\begin{aligned} & ACLRule(\texttt{xmppr}_{\texttt{AllowUCC}}) \\ \leftarrow\ & federateWith(\texttt{xmppr}_{\texttt{AllowUCC}}, \texttt{ucc}) \sqcap \\ & hasPermission.(\texttt{xmppr}_{\texttt{AllowUCC}}, \texttt{allow}) \end{aligned}$$

Denial of Service Mitigation. Concept $DoSRule$ defines the maximum number of simultaneous S2S connections that may be permitted with a remote XMPP server.

$$\begin{aligned} DoSRule \sqsubseteq\ & XMPPRule \sqcap \\ & \exists_{=1} hasMaxS2SConn.Integer \sqcap \\ & \exists_{=1} appliedTo.(IPAddress \sqcup DNSName) \end{aligned}$$

Indiviudal $\texttt{xmppr}_{\texttt{LimitConn}}$ limits the number of simultaneous connections that can be established with any XMPP server

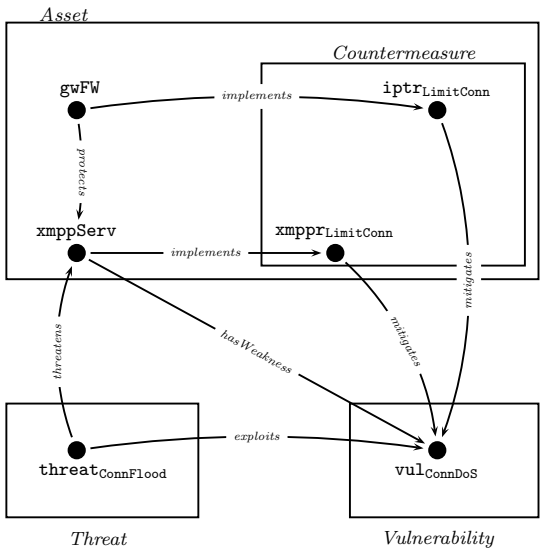

Figure 1: Semantic Threat Graphs Fragment.

(denoted by the wildcard IP address of `ip*.*.*.*`) to `1`.

$$DoSRule(\texttt{xmppr}_{\texttt{LimitConn}}) \leftarrow hasMaxS2SConn(\texttt{xmppr}_{\texttt{LimitConn}}, \texttt{1}) \sqcap appliedTo.(\texttt{xmppr}_{\texttt{LimitConn}}, \texttt{ip*.*.*.*})$$

Application-Layer Content Filtering. Concept $CFRule$ defines the set of rules that filters XMPP message stanza payloads. Such filtering may be used to mitigate against the threat of known SPIM, Malware and URL Phishing.

$$CFRule \sqsubseteq XMPPRule \sqcap \exists_{\geq 1} hasPattern.String \sqcap \exists_{=1} hasPermission.Permission$$

For example, $\texttt{xmppr}_{\texttt{BadWord}}$ is used to deny XMPP message stanzas that contain the word `"sex"`.

$$CFRule(\texttt{xmppr}_{\texttt{BadWord}}) \leftarrow hasPattern(\texttt{xmppr}_{\texttt{BadWord}}, \texttt{"sex"}) \sqcap hasPermission.(\texttt{xmppr}_{\texttt{BadWord}}, \texttt{deny})$$

3. THREAT MODEL FOR XMPP

A semantic threat graph [11], constructed in terms of an ontology, can be defined as a graph that represents the meaning of a threat domain. Enterprise assets are represented as individuals of the *Asset* concept. An asset may have one or more *hasWeakness*'s (property relationship) that relate to individuals categorised in the *Vulnerability* concept. Individuals of the *Vulnerability* concept are exploitable (*exploitedBy*) by a threat or set of threats (*Threat* concept). As a consequence, an asset that has a vulnerability is, therefore, also *threatenedBy* a corresponding *Threat*. A countermeasure *mitigates* particular vulnerabilities. Countermeasures are deemed to be kinds-of assets and thus are defined as a *subConceptOf Asset*. Figure 1 illustrates an example instantiation of a semantic threat graph that mitigates a Denial of Service attack against the XMPP server.

Asset. Concept *Asset* represents any enterprise entity that may be the subject of a threat. While assets can include people and physical infrastructure, this paper only considers computer-system based entities such as XMPP servers, firewalls and so forth.

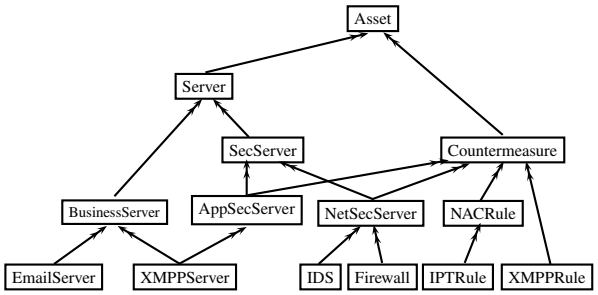

Figure 2: Fragment of Enterprise Asset Hierarchy.

Individuals of concept *Asset* may have zero or more vulnerabilities (∀ restriction) along property $hasWeakness$. As a result, those assets may be exposed to various individuals of the *Threat* concept. An asset may have the capability to implement a countermeasure to protect itself or other assets.

$$Asset \sqsubseteq \forall hasWeakness.Vulnerability \sqcap \forall_{\geq 0} isThreatenedBy.Threat \sqcap \forall_{\geq 0} implements.Countermeasure$$

Concept *Asset* is further specialised to have more specific kinds of asset concepts. For example, the set of business servers and security servers (individuals) an enterprise may have may be represented as $BusinessServer, SecServer \sqsubseteq Server$, where *Server* is a sub-concept of *Asset*. Figure 2 depicts a fragment of the *Asset* hierarchy. Note the double-headed arrow represents a subsumption relation.

An individual `xmppServ` of the *BusinessServer* concept (inferred as an individual of concept *Asset*) is vulnerable to a connection-based Denial of Service ($\texttt{vul}_{\texttt{ConnDoS}}$) weakness. As a consequence, the `xmppServ` *isThreatenedBy* a connection-flood attack represented as $\texttt{threat}_{\texttt{ConnFlood}}$ individual.

$$Asset(\texttt{xmppServ}) \leftarrow hasWeakeness(\texttt{xmppServ}, \texttt{vul}_{\texttt{ConnDoS}}) \sqcap isThreatenedBy(\texttt{xmppServ}, \texttt{threat}_{\texttt{ConnFlood}})$$

Concept *NetSecServer* represents the network access control systems that protect (*protects*) internal servers (including themselves). Individuals of concept *NetSecServer* implement one or more ($\exists_{\geq 1}$) individuals of the *NACRule* concept (sub-concept of *Countermeasure*). Note, the *protects* property is inherited from its parent concept *SecServer*.

$$NetSecServer \sqsubseteq SecServer \sqcap \exists_{\geq 1} protects.Server \sqcap \exists_{\geq 1} implements.NACRule$$

The following ontology fragment asserts that the gateway firewall (`gwFW`) protects the XMPP server from a Denial of Service attack by implementing a TCP connection-limit countermeasure (iptables rule $\texttt{iptr}_{\texttt{LimitConn}}$).

$$NetSecServer(\texttt{gwFW}) \leftarrow protects(\texttt{gwFW}, \texttt{xmppServ}) \sqcap implements(\texttt{gwFW}, \texttt{iptr}_{\texttt{LimitConn}})$$

Threat. A threat is a potential for violation of security [35]. An individual of the *Threat* concept is considered to exploit

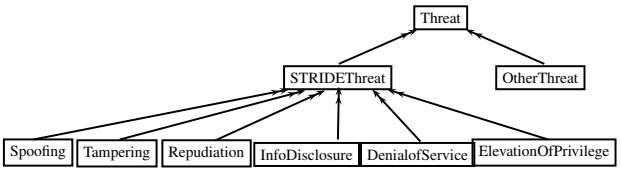

Figure 3: Fragment of Threat Hierarchy.

one or more vulnerabilities.

$$Threat \sqsubseteq \exists_{\geq 1} exploits.Vulnerability \sqcap \exists_{\geq 1} threatens.Asset$$

For example, individual $\texttt{threat}_{\texttt{ConnFlood}}$ exploits the $\texttt{vul}_{\texttt{ConnDoS}}$ vulnerability and threatens the `xmppServ`.

$$Threat(\texttt{threat}_{\texttt{ConnFlood}}) \leftarrow exploits(\texttt{threat}_{\texttt{ConnFlood}}, \texttt{vul}_{\texttt{ConnDoS}}) \sqcap threatens(\texttt{threat}_{\texttt{ConnFlood}}, \texttt{xmppServ})$$

The *Threat* concept defines a number of sub-concepts in accordance with best practice, such as the Microsoft STRIDE standard [17] whereby threats are categorised as: ***S**poofing identity*, ***T**ampering with data*, ***R**epudiation*, ***I**nformation disclosure*, ***D**enial of service* and ***E**levation of privilege*. Figure 3 depicts a fragment of the threat hierarchy.

Vulnerability. A vulnerability is an asset flaw or security weakness that has the potential to be exploited by a threat.

$$Vulnerability \sqsubseteq \exists_{\geq 1} isExploitedBy.Threat \sqcap \exists_{\geq 1} isWeaknessOf.Asset$$

The following ontology fragment states that the `xmppServ` is susceptible to a $\texttt{threat}_{\texttt{ConnFlood}}$ attack via the $\texttt{vul}_{\texttt{ConnDoS}}$ weakness. Note, $\texttt{vul}_{\texttt{ConnDoS}}$ represents a weakness in the XMPP stack whereby it is possible to surpass the maximum number of XMPP-based socket connections permitted by the unprotected XMPP protocol [30].

$$Vulnerability(\texttt{vul}_{\texttt{ConnDoS}}) \leftarrow isExploitedBy(\texttt{vul}_{\texttt{ConnDoS}}, \texttt{threat}_{\texttt{ConnFlood}}) \sqcap isWeaknessOf(\texttt{vul}_{\texttt{ConnDoS}}, \texttt{xmppServ})$$

Countermeasure. A countermeasure is an action or process that mitigates vulnerabilities and prevents and/or reduces threats. A countermeasure is an asset.

$$Countermeasure \sqsubseteq Asset$$

Concepts $XMPPRule$ and $NACRule$ are sub-concepts of concept $Countermeasure$. Concept $XMPPRule$ is representative of the XMPP access-control rules that mitigate one or more XMPP vulnerabilities.

$$XMPPRule \sqsubseteq Countermeasure \sqcap \exists_{\geq 1} mitigates.Vulnerability \sqcap \forall_{\geq 0} implementedBy.XMPPServer$$

The following ejabberd [1] XMPP rule:

```
{access, max_s2s_connections, [1, all]}
```

is represented as individual $\texttt{xmppr}_{\texttt{LimitConn}}$ presented in Section 2, where it is implemented by the `xmppServ` to mitigate the vulnerability $\texttt{vul}_{\texttt{ConnDoS}}$ on the XMPP server itself.

$$XMPPRule(\texttt{xmppr}_{\texttt{LimitConn}}) \leftarrow mitigates(\texttt{xmppr}_{\texttt{LimitConn}}, \texttt{vul}_{\texttt{ConnDoS}}) \sqcap implementedBy(\texttt{xmppr}_{\texttt{LimitConn}}, \texttt{xmppServ})$$

Note, while sample low-level ejabberd configurations are used within this paper, the ontology for XMPP access control is implementation neutral and is equally applicable to other XMPP implementations such as Openfire [21].

Concept $NACRule$ represents network access control rules. Concept $IPTRule$ is a sub-concept of concept $NACRule$ and represents the set of iptables firewall rules.

$$IPTRule \sqsubseteq Countermeasure \sqcap \exists_{\geq 1} mitigates.Vulnerability \sqcap \forall_{\geq 0} implementedBy.Firewall$$

Individual $\texttt{iptr}_{\texttt{LimitConn}}$, mitigates the XMPP server vulnerability $\texttt{vul}_{\texttt{ConnDoS}}$ and is implemented by the `gwFW`.

$$IPTRule(\texttt{iptr}_{\texttt{LimitConn}}) \leftarrow mitigates(\texttt{iptr}_{\texttt{LimitConn}}, \texttt{vul}_{\texttt{ConnDoS}}) \sqcap implementedBy(\texttt{iptr}_{\texttt{LimitConn}}, \texttt{gwFW})$$

The following iptables rule:

```
iptables -A FORWARD -p tcp -d xmppServIP --dport
5269 -m connlimit --connlimit-above 2 -j DROP
```

limits the number of connections to the XMPP server to 1 and, based on [9,10], is represented as individual $\texttt{iptr}_{\texttt{LimitConn}}$.

$$IPTRule(\texttt{iptr}_{\texttt{LimitConn}}) \leftarrow hasChain(\texttt{iptr}_{\texttt{LimitConn}}, \texttt{forward}) \sqcap hasProtocol(\texttt{iptr}_{\texttt{LimitConn}}, \texttt{tcp}) \sqcap hasDstIP(\texttt{iptr}_{\texttt{LimitConn}}, \texttt{xmppServIP}) \sqcap hasDstPort(\texttt{iptr}_{\texttt{LimitConn}}, \texttt{5269}) \sqcap hasModule(\texttt{iptr}_{\texttt{LimitConn}}, \texttt{connlimit}) \sqcap hasConnectionAbove(\texttt{iptr}_{\texttt{LimitConn}}, \texttt{2}) \sqcap hasTarget(\texttt{iptr}_{\texttt{LimitConn}}, \texttt{drop})$$

Assets, threats, vulnerabilities and countermeasures have additional properties and for the sake of clarity, a number of property relationships have been excluded from this discussion. For example, servers are assigned one or more IP addresses (depending on the number of network interfaces they may have). Access is provided based on the server's port, for example a XMPP server is accessible over port 5269. However, not all servers (or daemons) are accessible through a port number, for example, an iptables firewall server does not directly support network access. Each server has a particular role (*operatesAs* property), for example, the `gwFW` individual is identified as an iptables firewall via its string assignment `'iptables'`. Threats originate from one or more IP addresses. Countermeasures are executable on a particular type of security server. For example, individual $\texttt{iptr}_{\texttt{LimitConn}}$ $isExecutableOn$ the `gwFW` iptables firewall.

$$\begin{aligned} Asset &\sqsubseteq \exists_{\geq 1} hasIPAddress.IPAddress \sqcap \forall_{\geq 0} hasPort.Port \sqcap \exists_{=1} operatesAs.String \\ Threat &\sqsubseteq \exists_{\geq 1} hasThreatSource.IPAddress \\ Countermeasure &\sqsubseteq \exists_{=1} isExecutableOn.SecServer \end{aligned}$$

4. ENTERPRISE SECURITY POLICY

This section considers the relationship between the enterprise security policy and the lower-level security controls (iptables and XMPP) that implement the policy.

Example 1. Consider the enterprise-level security requirement that states the following: '*The enterprise XMPP server asset is permitted to federate with other XMPP servers*'. Table 1a depicts some of the threats, vulnerabilities and countermeasures associated with upholding this policy.

Unavailability of service ($\texttt{threat}_{\texttt{NoListeningS2SPort}}$)on the S2S port is considered to be a threat related to federation. Enabling the correct service port (IANA port `5269`) by implementing countermeasure $\texttt{xmppr}_{\texttt{EnableS2SPort}}$ will ensure that the XMPP server is capable of federating with other XMPP servers. Table 1a outlines the identified threats, vulnerabilities and associated countermeasures. Countermeasures are also required to mitigate threats originating from an overly-restrictive firewall: $\texttt{iptr}_{\texttt{AllowInS2SPort}}$ permits inbound access to the XMPP server over port `5269` and $\texttt{iptr}_{\texttt{AllowOutS2SPort}}$ permits the corresponding outbound traffic.

In order to uphold the enterprise-level security policy, each countermeasure must be implemented by its respective access control mechanism. Sample application-level and network access-control rules related to the ejabberd XMPP server and iptables firewall are outlined in Table 2.

Example 2 In this example, the enterprise-security requirement described in Example 1 is further restricted to prevent federation with untrusted XMPP servers. That is, '*Server-to-Server federation is not permitted except between trusted XMPP servers with an IP address of 143.239.75.235 and 193.1.193.140*'. This enterprise-level security requirement is structured according to the threats identified in Table 1b.

A threat of unintended access, $\texttt{threat}_{\texttt{UnintendedAccess}}$, regarding S2S federation is identified. Disabling the S2S service port ($\texttt{xmppr}_{\texttt{DisableS2SPort}}$) is not applicable in this example. Rather, the recommended countermeasures are to enable the S2S service port ($\texttt{xmppr}_{\texttt{EnableS2SPort}}$) and implement a default deny access control policy ($\texttt{xmppr}_{\texttt{DefaultDenyS2SAccess}}$) where exceptions must be defined. As a consequence of countermeasure $\texttt{xmppr}_{\texttt{DefaultDenyS2SAccess}}$, two additional vulnerabilities are introduced: $\texttt{Vul}_{\texttt{NoS2S143.239.75.235AllowXMPPRule}}$ and $\texttt{Vul}_{\texttt{NoS2S193.1.193.140AllowXMPPRule}}$. Thereby, giving rise to threats $\texttt{threat}_{\texttt{NoS2S143.239.75.235}}$ and $\texttt{threat}_{\texttt{NoS2S193.1.193.140}}$. In terms of the `xmppServ` configuration, these vulnerabilities are mitigated by the following countermeasures: $\texttt{xmppr}_{\texttt{Allow143.239.75.235}}$ and $\texttt{xmppr}_{\texttt{Allow193.1.193.140}}$, providing a white-list of trusted XMPP servers for the `xmppServ` to federate with.

Best practice [37] requires defense in-depth and therefore the firewall countermeasures outlined in Table 1a are further restricted to filter access to the S2S service port to the trusted XMPP servers only. That is, a deny by default policy ($\texttt{iptr}_{\texttt{DefaultDenyS2SAccess}}$) is implemented by the gateway firewall (`gwFw`) where permitted inbound and outbound IP address exceptions are also implemented (Table 1b).

Example 3. Consider the enterprise-level security requirement that restricts how individual trusted XMPP servers may authenticate: '*Server-to-Server federation is not permitted except between trusted XMPP servers with an IP address of 143.239.75.235 and 193.1.193.140, where the server with an IP address of 143.239.75.235 is permitted to authenticate using dialback and the server with an IP address of 193.1.193.140 is permitted to authenticate using SASL EXTERNAL*'.

The XMPP protocol does not currently support such fine-grained access control. However, this requirement may be emulated using firewalls capable of deep packet inspection such as iptables. Building upon the countermeasures already defined in Table 1b, two additional countermeasures outlined in Table 1c are required. For example, countermeasure $\texttt{iptr}_{\texttt{AllowDialbackAuthIn143.239.75.235}}$ ensures that packets originating from IP address 143.239.75.235 must have a packet with a known dialback signature in order to begin S2S federation communication with the enterprise XMPP server. All other packets originating from a different source, such as IP address 193.1.193.140, that try to federate using dialback authentication will be dropped by the iptables default deny policy $\texttt{iptr}_{\texttt{DefaultDenyS2SAccess}}$.

Note, provisioning an access control configuration for server federation must also consider additional threats that are not identified in the previous examples, such as Denial of Service or IP address spoofing. However, for reasons of space, additional threats are not discussed. For example, the firewall access-control rules outlined in Example 3 are intended to filter valid XMPP message stanzas to the XMPP server and do not consider the vulnerability of IP header forgery, where the threat of IP spoofing for IP addresses 143.239.75.235 and 193.1.193.140 may be realised. Such a threat coupled with the forging of layer-7 payloads, enable the required deep packet inspection rules implemented by the firewall to be used in an unintended manner. As a consequence, additional threats, vulnerabilities and countermeasures must also be considered. It may not always be possible to uphold an enterprise-level security requirement using the current set of access controls. Therefore, additional access controls such as an IPSec VPN [12] may be required. In some cases, the enterprise-level security requirement itself may require further refinement, for example the enforcement of strong authentication using SASL EXTERNAL only.

5. CATALOGUES OF BEST PRACTICE

Best practice recommendations, for example [30, 34], encoded as semantic threat graphs require for the most part customised 'tweaking' depending on the network in which they are applied. Therefore, modelling recommendations means that not all assets, threats, vulnerabilities, countermeasures and their relationships are known in advance. Ontologies are based on Open World Assumption [5], thereby making it an ideal knowledge framework with which to model both known and unknown facts.

For example, it may not be known in advance what the IP address range of a particular enterprise network is in which the catalogue is being applied to, or if a server is listening on a different port from the IANA [19] recommended default. Therefore, *template* semantic threat graph individuals are defined as place holders for unknown knowledge. For example, a template XMPP server individual, `xmppServ`, and associated template iptables individuals $\texttt{iptr}_{\texttt{AllowInS2SPort}}$ that is intended to permit S2S federation access, may have the following known facts:

$$
\begin{aligned}
&Asset(\texttt{xmppServ})\\
&\leftarrow\ hasIPAddress(\texttt{xmppServ}, \texttt{-}) \sqcap\\
&\quad\ hasPort(\texttt{xmppServ}, \texttt{5259}) \sqcap
\end{aligned}
$$

$$
\begin{aligned}
&IPT\,Rule(\texttt{iptr}_{\texttt{AllowInS2SPort}})\\
&\leftarrow\ hasChain(\texttt{iptr}_{\texttt{AllowInS2SPort}}, \texttt{forward}) \sqcap\\
&\quad\ hasDstIP(\texttt{iptr}_{\texttt{AllowInS2SPort}}, \texttt{-}) \sqcap\\
&\quad\ hasProtocol(\texttt{iptr}_{\texttt{AllowInS2SPort}}, \texttt{tcp}) \sqcap\\
&\quad\ hasDstPort(\texttt{iptr}_{\texttt{AllowInS2SPort}}, \texttt{5269}) \sqcap\\
&\quad\ hasAction(\texttt{iptr}_{\texttt{AllowInS2SPort}}, \texttt{accept})
\end{aligned}
$$

Threat	Vulnerability	Countermeasure
$\text{threat}_{\text{NoListeningS2SPort}}$	$\text{Vul}_{\text{S2SPortDisabled}}$	$\text{xmppr}_{\text{EnableS2SPort}}$
$\text{threat}_{\text{NoS2SInS2S}}$	$\text{Vul}_{\text{NoS2SInAllowIPTRule}}$	$\text{iptr}_{\text{AllowInS2SPort}}$
$\text{threat}_{\text{NoS2SOutS2S}}$	$\text{Vul}_{\text{NoS2SOutAllowIPTRule}}$	$\text{iptr}_{\text{AllowOutS2SPort}}$

(a) XMPP and iptables Server-to-Server Alignment.

Threat	Vulnerability	Countermeasure
$\text{threat}_{\text{UnintendedAccess}}$	$\text{Vul}_{\text{NoS2SDefaultDenyXMPPRule}}$	$\text{xmppr}_{\text{DefaultDenyS2SAccess}}$
$\text{threat}_{\text{UnintendedInAccess}}$	$\text{Vul}_{\text{NoS2SDefaultInDenyIPTRule}}$	$\text{iptr}_{\text{DefaultDenyS2SAccess}}$
$\text{threat}_{\text{NoListeningS2SPort}}$	$\text{Vul}_{\text{S2SPortDisabled}}$	$\text{xmppr}_{\text{EnableS2SPort}}$
$\text{threat}_{\text{NoS2S143.239.75.235}}$	$\text{Vul}_{\text{NoS2S143.239.75.235AllowXMPPRule}}$	$\text{xmppr}_{\text{Allow143.239.75.235}}$
$\text{threat}_{\text{NoS2S193.1.193.140}}$	$\text{Vul}_{\text{NoS2S193.1.193.140AllowXMPPRule}}$	$\text{xmppr}_{\text{Allow193.1.193.140}}$
$\text{threat}_{\text{NoS2SIn143.239.75.235}}$	$\text{Vul}_{\text{NoS2S143.239.75.235InAllowIPTRule}}$	$\text{iptr}_{\text{AllowInbound143.239.75.235}}$
$\text{threat}_{\text{NoS2SOut143.239.75.235}}$	$\text{Vul}_{\text{NoS2S143.239.75.235OutAllowIPTRule}}$	$\text{iptr}_{\text{AllowOutbound143.239.75.235}}$
$\text{threat}_{\text{NoS2SIn193.1.193.140}}$	$\text{Vul}_{\text{NoS2S193.1.193.140InAllowIPTRule}}$	$\text{iptr}_{\text{AllowInbound193.1.193.140}}$
$\text{threat}_{\text{NoS2SOut193.1.193.140}}$	$\text{Vul}_{\text{NoS2S193.1.193.140OutAllowIPTRule}}$	$\text{iptr}_{\text{AllowOutbound193.1.193.140}}$

(b) XMPP and iptables Access Control.

Threat	Vulnerability	Countermeasure
$\text{threat}_{\text{NoS2SInDialbackAuth143.239.75.235}}$	$\text{Vul}_{\text{NoS2S143.239.75.235InDialbackAuthenAllowIPTRule}}$	$\text{iptr}_{\text{AllowDialbackAuthIn143.239.75.235}}$
$\text{threat}_{\text{NoS2SInSASLExternalAuth193.1.193.140}}$	$\text{Vul}_{\text{NoS2S193.1.193.140InSASLExternalAuthAllowIPTRule}}$	$\text{iptr}_{\text{AllowSASLExternalAuthIn193.1.193.140}}$

(c) Fine-grained XMPP Access Control using iptables.

Table 1: Example Inter-Operation of XMPP and iptables Access Control Configuration.

Countermeasure	Low-Level Device Specific Access Control Rule
$\text{xmppr}_{\text{EnableS2SPort}}$	{5269, ejabbered_s2s_in}
$\text{xmppr}_{\text{DefaultDenyS2SAccess}}$	{s2s_default_policy, deny}
$\text{xmppr}_{\text{Allow143.239.75.235}}$	{s2s_host, "143.239.75.235"}
$\text{xmppr}_{\text{Allow193.1.193.140}}$	{s2s_host, "193.1.193.140"}
$\text{iptr}_{\text{DefaultDenyS2SAccess}}$	iptables -P FORWARD DROP
$\text{iptr}_{\text{AllowInS2SPort}}$	iptables -A FORWARD -i eth0 -d xmppServIP --dport 5269 -j ACCEPT
$\text{iptr}_{\text{AllowOutS2SPort}}$	iptables -A FORWARD -o eth0 -s xmppServIP --sport 5269 -j ACCEPT
$\text{iptr}_{\text{AllowIn143.239.75.235}}$	iptables -A FORWARD -i eth0 -s 143.239.75.235 -d xmppServIP --dport 5269 -j ACCEPT
$\text{iptr}_{\text{AllowOut143.239.75.235}}$	iptables -A FORWARD -o eth0 -s xmppServIP -d 143.239.75.235 --sport 5269 -j ACCEPT
$\text{iptr}_{\text{AllowIn193.1.193.140}}$	iptables -A FORWARD -i eth0 -s 193.1.193.140 -d xmppServIP --dport 5269 -j ACCEPT
$\text{iptr}_{\text{AllowOut193.1.193.140}}$	iptables -A FORWARD -o eth0 -s xmppServIP -d 193.1.193.140 --sport 5269 -j ACCEPT
$\text{iptr}_{\text{AllowDialbackAuthIn143.239.75.235}}$	iptables -A FORWARD -s 143.239.75.235 -d xmppServIP --dport 5269 -m string --string "<dialback xmlns="urn:xmpp:features:dialback">" -j ACCEPT
$\text{iptr}_{\text{AllowSASLExternalAuthIn193.1.193.140}}$	iptables -A FORWARD -s 193.1.193.140 -d xmppServIP --dport 5269 -m string --string "<mechanism>EXTERNAL</mechanism>" -j ACCEPT

Table 2: Corresponding Countermeasure Low-Level Access Control Rules

where "−" signifies that the range of a given property is an unknown individual. Deploying the XMPP server within the network, means that it will be assigned an IP address, with the result of the template XMPP server and iptables rule individuals being modified to reflect this new knowledge.

Automatic Synthesis of Access-Control Rules. As knowledge about assets, threats and vulnerabilities become known, it becomes possible to consider automatic synthesis of access-control rules. The following SWRL rule dynamically creates a set of iptables firewall rules (using built-in *swrlx:makeOWLIndividual*), that will protect XMPP servers from known SPIM threats. Knowledge about an XMPP server's IP address (variable $?xip$) and the source IP addresses in which the threat of SPIM ($SPIMThreat(?spim)$) has been identified is used to synthesise specific firewall rules (*?specific*) from a template iptables rule ($\texttt{iptr}_{temp}$).

$XMPPServer(?xmpp) \wedge SPIMThreat(?spim) \wedge$
$Vulnerability(?vul) \wedge TemplateIPTRule(\texttt{iptr}_{temp}) \wedge$
$hasWeakness(?xmpp, ?vul) \wedge exploits(?spim, ?vul) \wedge$
$mitigates(\texttt{iptr}_{temp}, ?vul) \wedge hasThreatSource(?spim, ?tip) \wedge$
$hasIPAddress(?xmpp, ?xip) \wedge hasPort(?xmpp, ?xp)$
swrlx:makeOWLIndividual(?specific,$\texttt{iptr}_{temp}$*, ?spim, ?xmpp)*
$\rightarrow IPTRule(?specific) \wedge$
$hasChain(?specific, \texttt{forward}) \wedge$
$hasSrcIPAddress(?specific, ?tip) \wedge$
$hasDstIPAddress(?specific, ?xip) \wedge$
$hasDstPort(?specific, ?xp) \wedge$
$hasAction(?specific, \texttt{drop})$

6. SYNTHESIS AND ANALYSIS

6.1 Synthesis of Access Control Configuration

Synthesis of an appropriate access control configuration relies on the existence of a knowledge-base of candidate XMPP and firewall access-control rules that are consistent with the enterprise-level security requirements. These could, for example, represent considered best practice for systems that protect XMPP-based applications.

The following is a generic SWRL rule that examines the threats ($?threat$) and vulnerabilities ($?vul$) that each enterprise server ($?serv$) has and searches for suitable countermeasures ($?rule$) that may be implemented by the appropriate security servers.

$Server(?serv) \wedge SecServer(?sec) \wedge Threat(?threat) \wedge$
$Vulnerability(?vul) \wedge Countermeasure(?rule) \wedge$
$hasWeakness(?serv, ?vul) \wedge threatens(?threat, ?serv) \wedge$
$exploits(?threat, ?vul) \wedge mitigates(?rule, ?vul) \wedge$
$protects(?sec, ?serv) \wedge operatesAs(?sec, ?secType) \wedge$
$isExecutableOn(?rule, ?ruleType) \wedge$
$swrlb : equal(?ruleType, ?secType)$
$\rightarrow implements(?sec, ?rule)$

6.2 Analysis of Access Control Configuration

The following SQWRL [26] query analyses an existing access control configuration regarding the current security servers ($?sec$) and their corresponding access-control rules ($?rule$) that mitigate the threat of a connection-flood Denial of Service attack ($\texttt{threat}_{\texttt{ConnFlood}}$) against the `xmppServ`.

$XMPPServer(\texttt{xmppServ}) \wedge SecServer(?sec) \wedge$
$Threat(\texttt{threat}_{\texttt{ConnFlood}}) \wedge Vulnerability(?vul) \wedge$
$Countermeasure(?rule) \wedge hasWeakness(\texttt{xmppServ}, ?vul) \wedge$
$threatens(\texttt{threat}_{\texttt{ConnFlood}}, \texttt{xmppServ}) \wedge$
$exploits(\texttt{threat}_{\texttt{ConnFlood}}, ?vul) \wedge$
$mitigates(?rule, ?vul) \wedge protects(?sec, \texttt{xmppServ})$
$\rightarrow sqwrl : select(?sec, ?rule)$

The query returns a single tuple $\texttt{xmppServ} \mapsto \texttt{xmppr}_{\texttt{LimitConn}}$. As part of defense in-depth, the `gwFW`, should also implement $\texttt{iptr}_{\texttt{LimitConn}}$. Thus, further synthesis is required.

7. DISCUSSION AND CONCLUSION

A number of existing techniques can be used to generate [7, 13, 14], query [15, 22, 23] and perform structural analysis [3, 4, 6] on network access control configurations. However, these homogeneous firewall-centric approaches tend not consider their interoperation with other and application-layer access controls. This paper extends previous results [9, 10] on ontology-based modeling of iptables and TCP-Wrapper firewalls by considering how these controls can be managed in conjunction with the application-level controls provided by XMPP. Section 6 demonstrates query-analysis and synthesis of the combined iptables and XMPP configuration ontology. We are currently extending the analysis techniques in [10] to provide structural analysis, such as shadowing, across the combined iptables and XMPP ontology.

A threat-based approach, based on [11], is proposed in this paper as a strategy for providing a uniform interpretation of the heterogeneous security controls. A knowledge-base of best practice XMPP and firewall countermeasures against known threats are encoded as semantic threat graphs. This knowledge-base is searchable where knowledge about XMPP operational requirements (for example the XMPP port), the enterprise-level security requirements (for example, deny access to untrusted IP addresses) and the threats (for example, SPIM) are used to search for suitable firewall countermeasures. Thus, the approach provides a basis for the automatic generation of access control configurations applicable to specific security mechanisms.

The ontology described in this paper has been populated with approximately fifty threat, vulnerability and countermeasures for XMPP and iptables, derived from existing best-practices and standards. These include a number of XEP recommendations [2] for XMPP configuration, for example, XEP-0205 (Denial of Service), XEP-0165 (JID Mimicking) and XEP-0159 (SPIM). These are combined with best-practice recommendations for safe firewall configuration, including [20, 33, 34], demonstrating that the ontology can effectively capture realistic configuration requirements.

Acknowledgments

This research has been supported by Science Foundation Ireland grant 08/SRC/11403.

8. REFERENCES

[1] ejabberd 2.1.4: Installation and Operation Guide. website `www.ejabberd.im`.

[2] XMPP extensions and proposals. website `xmpp.org/extensions`.

[3] M. Abedin, S. Nessa, L. Khan, and B. M. Thuraisingham. Detection and resolution of anomalies in firewall policyrules. *Data and Applications Security XX, 20th Annual IFIP WG 11.3 Working Conference on Data and Applications Security*, July 2006.

[4] E. S. Al-Shaer, H. H. Hamed, R. Boutaba, and M. Hasan. Conflict Classification and Analysis of Distributed Firewall Policies. *IEEE Journal on Selected Areas in Communications, Issue: 10, Volume: 23, Pages: 2069 - 2084*, October 2005.

[5] F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, and P. Patel-Schneider. *The Description Logic Handbook: Theory, Implementation and Applications*. Cambridge University Press, March 2003.

[6] F. Cuppens, N. Cuppens-Boulahia, and J. García-Alfaro. Detection and Removal of Firewall Misconfiguration. *IASTED International Conference on Communication, Network and Information Security (CNIS)*, November 2005.

[7] F. Cuppens, N. Cuppens-Boulahia, T. Sans, and A. Miège. A Formal Approach to Specify and Deploy a Network Security Policy. *2nd Workshop on Formal Aspects in Security and Trust (FAST)*, August 2004.

[8] T. Dierks and E. Rescorla. RFC5246: The Transport Layer Security (TLS) Protocol Version 1.2. *http://ietf.org*, August 2008.

[9] W. M. Fitzgerald and S. N. Foley. Aligning Semantic Web Applications with Network Access Controls. *International Journal on Computer Standards & Interfaces, Elsevier, Article in Press*, October 2009.

[10] W. M. Fitzgerald, S. N. Foley, and M. O. Foghlú. Network Access Control Configuration Management using Semantic Web Techniques. *Journal of Research and Practice in Information Technology, Volume 41 (2)*, May 2009.

[11] S. N. Foley and W. M. Fitzgerald. An Approach to Security Policy Configuration using Semantic Threat Graphs. *23rd Annual IFIP WG 11.3 Working Conference on Data and Applications Security (DBSec), Springer LNCS, Canada*, July 2009.

[12] S. Frankel, K. Kent, R. Lewkowski, A. Orebaugh, R. Ritchey, and S. Sharma. Guide to IPsec VPNs : Recommendations of the National Institute of Standards and Technology. *NIST Special Publication 800-77*, December 2005.

[13] M. G. Gouda and X.-Y. A. Liu. Firewall Design: Consistency, Completeness and Compactness. *24th IEEE International Conference on Distributed Computing Systems (ICDCS)*, March 2004.

[14] J. D. Guttman. Filtering Postures: Local Enforcement for Global Policies. *IEEE Symposium on Security and Privacy*, pages 120–129, May 1997.

[15] S. Hazelhurst. Algorithms for Analysing Firewall and Router Access Lists. *Technical Report TR-Wits-CS-1999-5, Witwatersrand University*, 1999.

[16] J. Hebeler, M. Fisher, R. Blace, and A. Perez-Lopez. *Semantic Web Programming*. Wiley, April 2009.

[17] S. Hernan, S. Lambert, T. Ostwald, and A. Shostack. Uncover Security Design Flaws Using The STRIDE Approach. *http://microsoft.com/*.

[18] I. Horrocks, P. F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, and M. Dean. SWRL: A Semantic Web Rule Language Combining OWL and RuleML. *W3C Member Submission*, May 2004.

[19] IANA. Port Numbers. *Internet Assigned Numbers Authority (IANA), http://www.iana.org/assignments/port-numbers*.

[20] IANA. RFC 3330: Special-Use IPv4 Addresses. *http://ietf.org*, September 2002.

[21] ignite realtime. Openfire: Real Time Collaboration (RTC). *http://www.igniterealtime.org/*.

[22] R. Marmorstein and P. Kearns. A Tool for Automated iptables Firewall Analysis. *Usenix Annual Technical Conference, Freenix Track, Pages: 71-81*, April 2005.

[23] A. Mayer, A. Wool, and E. Ziskind. Offline Firewall Analysis. *International Journal of Information Security*, 5(3):125–144, May 2006.

[24] J. Miller, P. Saint-Andre, and P. Hancke. XEP0220: Server Dialback. *http://xmpp.org*, March 2010.

[25] M. O'Connor, H. Knublauch, S. Tu, B. Grossof, M. Dean, W. Grosso, and M. Musen. Supporting Rule System Interoperability on the Semantic Web with SWRL. *4th International Semantic Web Conference (ISWC2005)*, 2005.

[26] M. J. O'Connor and A. K. Das. SQWRL: A Query Language for OWL. *6th International Workshop on OWL: Experiences and Directions (OWLED)*, 2009.

[27] J. Postel. RFC768: User Datagram Protocol. *http://ietf.org*, August 1980.

[28] J. Postel. RFC793: Transmission Control Protocol. *http://ietf.org*, September 1981.

[29] P. Saint-Andre. RFC3920: Extensible Messaging and Presence Protocol (XMPP): Core. *http://ietf.org*, October 2004.

[30] P. Saint-Andre. XEP-0205: Best Practice to Discourage Denial of Service Attacks. *http://xmpp.org*, January 2009.

[31] P. Saint-Andre and P. Millard. XEP0178: Best Practices for Use of SASL EXTERNAL with Certificates. *http://xmpp.org*, February 2010.

[32] P. Saint-Andre, K. Smith, and R. Troncon. *XMPP: The Definitive Guide, Building Real-Time Applications with Jabber Technologies*. O'Reilly, 2009.

[33] K. Scarfone and P. Hoffman. Guidelines on Firewalls and Firewall Policy: Recommendations of the National Institute of Standards and Technology. *NIST Special Publication 800-41, Revision 1*, September 2009.

[34] K. Scarfone, W. Jansen, and M. Tracy. Guide to General Server Security: Recommendations of the National Institute of Standards and Technology. *NIST Special Publication 800-123*, July 2008.

[35] R. Shirey. RFC 2828: Internet Security Glossary. *http://ietf.org*, May 2000.

[36] D. Taniar and J. W. Rahayu. *Web Semantics Ontology*. Idea Publishing, 2006.

[37] J. Wack, K. Cutler, and J. Pole. Guidelines on Firewalls and Firewall Policy: Recommendations of the National Institute of Standards and Technology. *NIST-800-41*, 2002.

APPENDIX

A. DL OVERVIEW

Description Logic (*DL*) is a formalism for representing knowledge (ontology) and belongs to a family of logic that represents a decidable portion of first-order logic [5]. A DL-based ontology is comprised of two components: terminological and assertional knowledge. *Terminological knowledge* contains intensional knowledge in the form of a terminology, that is, a vocabulary consisting of concepts and property relationships. Concepts represent sets of individuals and properties represent binary relations applied to individuals. Terminological knowledge is constructed through declarations that describe properties of concepts. *Assertional knowledge* contains extensional knowledge that is specific to individuals (instances of concepts) of the domain of interest.

Description Logic is characterised by a set of constructors (Table 3) and axioms (Table 4) that allow for the construction of complex concepts and property relationships (roles) from atomic concepts or properties.

Concepts are interpreted as sets of individuals with common properties. The following DL notation explicitly asserts HTTP Web port (`port80`) as instance of the *Port* concept:

$$Port(\texttt{port80})$$

The following naming convention is adopted; concepts and properties are written in *italic* font, where a concept begins with an uppercase letter and a property begins with a lowercase letter. Individuals are written in a lowercase `typewriter` font.

Concepts can be organised into a sub-concept hierarchy. For example, concept $TCPHeaderField$ represents the fields of a TCP header as defined by [28]. The following DL assertion:

$$Port \sqsubseteq TCPHeaderField$$

states that: concept *Port* is a specialisation ($\sqsubseteq$) of concept $TCPHeaderField$, membership of concept *Port* implies membership of $TCPHeaderField$, and the membership constraints of concept $TCPHeaderField$ are inherited by concept *Port*.

A *partial* concept is specified with *necessary conditions* ($\sqsubseteq$) which state that if an individual is a member of a particular concept, then it must satisfy the conditions that characterise that concept. However, it cannot be said that any (random) individual that satisfies these conditions must be a member of this concept. When a concept is defined with *necessary and sufficient conditions* ($\equiv$), like partial concepts, an individual must satisfy those conditions if it is a member of that concept. However, with the *sufficient condition* included, then any (random) individual that satisfies these conditions must be a member of this concept. Concepts that have at least one set of necessary and sufficient conditions are known as *complete* concepts.

A common TCP and UDP header field is the port field [27, 28]. The *Port* concept can be defined as having two overlapping parent concepts (intersection) as illustrated by the following DL fragment.

$$Port \quad \sqsubseteq \quad TCPHeaderField \sqcap UDPHeaderField$$

Enumerated concepts are those that exhaustively enumerate their individuals. In [28], the TCP protocol defines a set of six flags used to establish the three-way handshake and control ongoing data communication. The following enumerates the *Flag* concept:

$$Flag \sqsubseteq \{\texttt{syn}, \texttt{ack}, \texttt{fin}, \texttt{rst}, \texttt{psh}, \texttt{urg}\}$$

Properties are used to construct binary relationships between individuals and have a domain and range associated with them. For example, an individual of the *Port* concept, `port80`, has a relationship to `80` (an individual of the *Integer* concept) along the $hasPortValue$ property. Properties can also have additional characteristics; inverse, functional, symmetric and transitive. Domain and range constraints are axioms and are used during reasoning. The following defines an excerpt of TCP header and TCP header field property domain and ranges modelled within the ontology. Note, the $\nrightarrow$ symbol denotes a partial function. For example, a $TCPHeader$ individual may have a currently unknown *Port* individual.

$$\begin{aligned} hasPortValue &: Port \nrightarrow Integer \\ hasSrcPort &: TCPHeader \nrightarrow Port \\ hasDstPort &: TCPHeader \nrightarrow Port \end{aligned}$$

As with concepts, properties can be hierarchical, where sub-properties specialise their super-properties. For example, the property $hasSrcPort$ specialises the property $hasPort$.

Restrictions can be applied to properties and are used to constrain an individual's membership to a specific concept. A property restriction effectively describes an anonymous or unnamed concept that contains all the individuals that satisfy the restriction. When restrictions are used to describe concepts they specify anonymous super-concepts of the concept being described. Restrictions fall into three categories: *Quantifier, hasValue* and *Cardinality* restrictions.

An existential ($\exists$) restriction requires *at least one* relationship for a given property to an individual that is a member of a specific range concept. 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 range concept.

$$\begin{aligned} Port \quad \sqsubseteq \quad & TCPHeaderField \sqcap UDPHeaderField \sqcap \\ & \exists hasPortValue.Integer \sqcap \\ & \forall hasPortValue.Integer \end{aligned}$$

The following DL fragment asserts that individual `port22`, has a $hasPortValue$ property relationship to an individual of the Integer concept and does not have a $hasPortValue$ property relationship to an individual that is not a member of the Integer concept. Therefore the assertion representing individual `port22`, upholds the *Port* concept constraints for membership. Informally a DL individual assertion is to be interpreted to mean if the right hand-side of the assertion ($\leftarrow$) holds then the left hand-side must also hold.

$$Port(\texttt{port22}) \quad \leftarrow \quad hasPortValue.(\texttt{port22}, 22)$$

A hasValue restriction, denoted by $\ni$, describes a set of individuals that are members of an anonymous concept (domain) that are related to a specific individual along the range of a given property. For example, the hasValue restriction $hasSrcPort \ni \texttt{port22}$ describes the anonymous set of individuals that have at least one relationship along the $hasSrcPort$ property to the specified individual `port22`. Note, a hasValue restriction is semantically equivalent to an existential restriction along the same property (for example,

Constructor	DL Syntax	Example
Intersection	$C_1 \sqcap ... \sqcap C_n$	$TCPHeaderField \sqcap UDPHeaderField$
Union	$C_1 \sqcup ... \sqcup C_n$	$Flag \sqcup Port$
Complement	$\neg C$	$\neg Port$
Universal Quantifier	$\forall P.C$	$\forall hasSrcPort.Port$
Existential Quantifier	$\exists P.C$	$\exists hasSrcPort.Port$
Max Cardinality	$\leq_n P$	$\leq_6 hasFlag$
Min Cardinality	$\geq_n P$	$\geq_1 hasFlag$
Exact Cardinality	$=_n P$	$=_1 hasSrcPort$

Table 3: DL Constructors.

Axiom	DL Syntax	Example
Concept Inclusion	$C_1 \sqsubseteq C_2$	$Port \sqsubseteq TCPHeaderField$
Concept Equivalence	$C_1 \equiv C_2$	$Port \equiv TCPHeaderField \sqcap UDPHeaderField$
Property Inclusion	$P_1 \sqsubseteq P_2$	$hasSrcPort \sqsubseteq hasPort$
Concept Assertion	$C(a)$	$Flag(syn)$
Property Assertion	$R(a,b)$	$hasBooleanValue(syn, true)$
Disjoint Individual	$a \neq b$	$syn \neq ack \ where \ Flag(syn), Flag(ack)$

Table 4: DL Axioms.

$hasSrcPort$) as the hasValue restriction, which has a range that is an enumerated concept that contains the specific individual used in the hasValue restriction.

Cardinality restrictions (=,≤,≥) specify the exact number of relationships that an individual must participate in for a given property. For example, each port number is assigned a single integer value and is asserted by the following cardinality (functional) restriction: $\exists_{=1} hasPortValue$. The following is an extended DL fragment of the $Port$ concept:

$$\begin{aligned} Port \quad \sqsubseteq \quad & TCPHeaderField \sqcap UDPHeaderField \sqcap \\ & \exists_{=1} hasPortValue.Integer \sqcap \\ & \exists_{=1} hasStringValue.String \end{aligned}$$

Reasoning in DL provides inference of new knowledge from statements asserted within the ontology. This new knowledge is inferred by applying classification, consistency checking and concept satisfiability to the existing ontology [5]. Classification is applied to infer subsumption relationships between concepts from their asserted formal definitions. Ensuring that the ontology does not contain contradictory facts is performed by consistency checking. Concept satisfiability determines if it is possible for a concept to contain individuals. If a concept is unsatisfiable, then creating an individual of that concept causes the entire ontology to become inconsistent. The ability to reason over the ontology is important. A new concept or individual can easily be added to an existing ontology by simply defining its logical characteristics; the reasoner automatically inserts that concept or individual into its correct taxonomy position.

B. A SWRL OVERVIEW

The *Semantic Web Rule Language* (*SWRL*), complements Description Logic by providing the ability to infer additional information from an ontology, but at the expense of decidability. SWRL rules are Horn-clause like rules written in terms of DL concepts, properties and individuals [25]. Informally, a SWRL rule is interpreted to mean that whenever the antecedent holds (i.e., its true), then the conditions in the consequent must also hold. For example, a SWRL rule to express that all ports (variable $?p$) are to be inferred as both a TCP header field and a UDP header field can be expressed by the following SWRL assertion.

$$Port(?p) \rightarrow TCPHeaderField(?p) \wedge UDPHeaderField(?p)$$

Note, while SWRL can be used to express an individual's hierarchical concept membership, DL should be used to define the subsumption relationship of concepts. In practice, SWRL is used to express that what is not expressible in DL (a SWRL built-in example is shown below).

The SWRL language includes support for user-defined *built-ins* that are common to most programming and scripting languages [16, 25]. A built-in, is a predicate that takes one or more arguments and evaluates to true if the arguments satisfy the predicate. The following categories of built-ins are supported by SWRL: comparison, mathematical, list operators, strings, temporal, boolean, URI, TBox and ABox [18, 25]. The following SWRL rule:

$$\begin{aligned} & Port(?p) \wedge hasPortValue(?p, ?v) \wedge \\ & swrlb : lessThanOrEqual(?v, \texttt{1024}) \\ & \rightarrow PrivilegedPort(?p) \end{aligned}$$

states that any port ($?p$) that has a numerical value less than or equal to `1024` ($swrlb : lessThanOrEqual$) is to be classified as a member of the $PrivilegedPort$ concept.

Semantic Query-Enhanced Web Rule Language (SQWRL) reuses components of the SWRL language to perform DL-based ontology queries. The SWRL rule antecedent is used as a pattern specification, while the consequent is replaced with a retrieval specification. For example, the most common SQWRL consequent is the *sqwrl:select* operator, which takes one or more arguments (variables) that correspond to those already specified in the antecedent. Note, all valid SWRL rule antecedent built-ins are valid within SQWRL.

SQWRL queries can operate in conjunction with SWRL rules in an ontology and can be used to retrieve knowledge inferred by those rules. SQWRL queries do not modify the knowledge within the ontology. A SQWRL query that returns all the privileged ports in a given ontology can then be written as:

$$PrivilegedPort(?p) \rightarrow sqwrl : select(?p)$$