An Approach to Security Policy Configuration Using Semantic Threat Graphs

Simon N. Foley and William M. Fitzgerald

Cork Constraint Computation Centre,
Computer Science Department, University College Cork, Ireland
s.foley@cs.ucc.ie, info@williamfitzgerald.net

Abstract. Managing the configuration of heterogeneous enterprise security mechanisms is a wholly complex task. The effectiveness of a configuration may be constrained by poor understanding and/or management of the overall security policy requirements, which may, in turn, unnecessarily expose the enterprise to known threats. This paper proposes a threat management approach, whereby knowledge about the effectiveness of mitigating countermeasures is used to guide the autonomic configuration of security mechanisms. This knowledge is modeled in terms of *Semantic Threat Graphs*, a variation of the traditional Threat/Attack Tree, extended in order to relate semantic information about security configuration with threats, vulnerabilities and countermeasures. An ontology-based approach to representing and reasoning over this knowledge is taken. A case study on Network Access Controls demonstrates how threats can be analyzed and how automated configuration recommendations can be made based on catalogues of best-practice countermeasures.

1 Introduction

A significant challenge in the process of securing complex systems is attaining a degree of confidence that a security configuration adequately addresses the (security) threats. *Threat Trees* [1,2], *Attack Trees* [3] and similar tree-based threat-modeling methodologies [4,5] are used to help identify, represent and analyze about threats to an enterprise's assets. Their top-down approach provides a semi-formal and methodical way to determine viable threat vectors (who, why and how a system can be compromised). In practice these trees are used for threat elicitation and analysis: representing threats at a high-level of abstraction and they tend not to be used to capture low-level or concrete security configuration detail. For example, while a threat tree may identify a firewall countermeasure for a Denial of Service attack, it is not advantageous/intended to model, for example, the distinctions between SYN-proxy versus SYN-threshold configurations [6] for a firewall in a sub-net that is downstream from other similarly configured firewalls. In the latter case much of semantics of the threats and countermeasures must be modeled implicitly and outside of the tree structure. Threat trees are useful for analyzing threats in a local context that is decomposed from some root

E. Gudes, J. Vaidya (Eds.): Data and Applications Security 2009, LNCS 5645, pp. 33–48, 2009.

threat, however their advantages are diminished when one considers the threat in a global context (across multiple root threats).

In this paper we consider how a threat tree style approach can provide a basis for automatically testing whether a security configuration adequately mitigates the identified threats. In order to achieve this, we extend the threat tree model to include semantic knowledge about the security configuration and how it relates to assets, threats, vulnerabilities and countermeasures. Knowledge, such as security and network configuration, and relationships with vulnerabilities and threats, is represented using ontologies [7], providing a framework in which to extend threat trees to *Semantic Threat Graphs* (*STG*s).

Semantic threat graphs are used to model knowledge about threat mitigation by security configurations. We take the Open World Assumption [8], with the result that semantic threat graphs are easily extended to incorporate knowledge about the configuration of new threats and/or additional security mechanisms. For example, building a semantic threat graph using existing firewall and proxy ontologies [9,10] in order to describe specific syn-proxy and syn-threshold countermeasure configurations.

We can use this model to build a knowledge-base of best-practice defenses and systems security policies against known threats. For example, bogon firewall rules [11,12,13] are best-practice protection against spoofing-threats for internal servers and end-user workstations, while NIST recommend multiple countermeasures over an n-tier network hosting a Web-server [14]. This knowledge-base is searchable—a suitable countermeasure/policy can be found for a given threat—and provides the basis for autonomic security configuration.

This paper is outlined as follows. Section 2 provides an introduction to threat trees highlighting their limitation in the context of low-level configuration. Section 3 proposes semantic threat graphs as a more a natural approach to construct and analyse security policies. A formal specification of a semantic threat graph, grounded in the *NAC* domain, is modelled in Section 4. Section 5 provides a case study that describes the basis for automated analysis and synthesis of suitable catalogue configuration recommendations.

2 Threat Trees

A *threat* can be defined as *"a potential for violation of security, which exists when there is a circumstance, capability, action, or event that could breach security and cause harm"*[15]. Threat trees (similarly, *attack trees* [3]) provide a semi-formal way of structuring the various threats that an asset may encounter. Various extensions of the threat tree paradigm have been developed. For example, the inclusion of countermeasures within the tree provides a new kind of tree called a *Defense Tree* or *Protection Tree* [4,5]. For simplicity, and if no ambiguity arises, we refer to these approaches collectively as threat trees.

A threat tree is composed of a single root node that defines the primary threat to an asset ('Disrupt Web Server', Figure 1(a)). A threat may be decomposed into additional fine-grained sub-threats ('Denial of Service'), thereby forming

a tree hierarchy [1]. A *threat profile* can be described as the path from a leaf node to the root node which represents a specific set of states involved in either achieving the primary threat or countering it.

Everything is a Threat. Each node is either a threat or a countermeasure. In practice, threats are not viewed in isolation and additional concepts must be implicitly encoded within the nodes of the tree. For example in Figure 1(a), various enterprise *Assets* are referenced: a Web server and firewall are implicit by the 'Disrupt Web Server', 'Firewall-1' nodes respectively. Similarly the Web server's TCP/IP stack indicates an implicit *Vulnerability* in the 'Exploit 3-way Handshake' threat node. By viewing everything as a threat, implicit information may be overlooked.

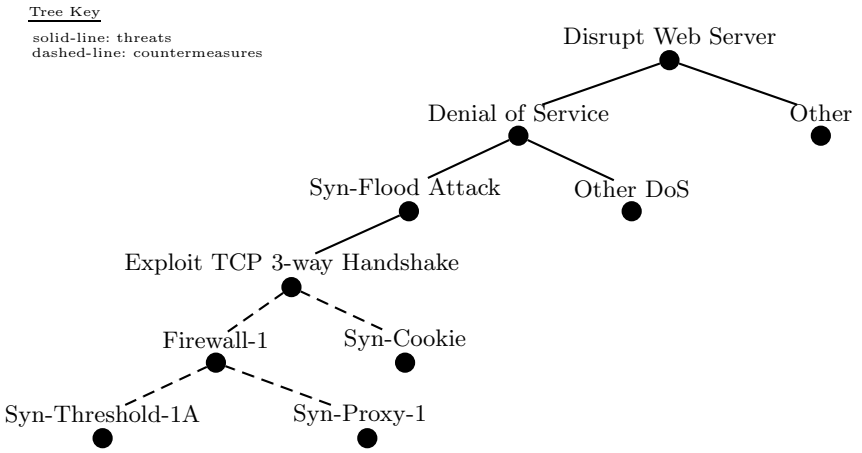
Implicit Threat Relations. A threat tree represents threat-decomposition and does not explicitly model other relationships between threats (or concepts related to threats). For example, in Figure 1(a), the 'Syn-Flood Attack' *exploits* (relationship) a TCP/IP 3-way handshake vulnerability (implied concept) and *threatens* (relationship) the Web server (implied concept).

Cascading Threats. Countermeasures themselves may have threats whereby the entity that protects another, is itself vulnerable. Cyclic dependencies between disparate trees cannot be explicitly modelled using threat tree constraints. From Figure 1(a), installing a firewall ('Firewall-1') with policy configuration 'Syn-Threshold-1A' and 'Syn-Proxy-1' will mitigate or reduce the threat of a 'Syn-Flood Attack' on the Web server. The 'Syn-Proxy-1' countermeasure in effect shifts the threat posed to the the Web server onto the firewall itself and thus the 'Syn-Flood Attack' has now indirectly migrated to 'Firewall-1' giving rise to the threat tree outlined in Figure 1(b). One of the ways that 'Firewall-1' can be protected is for 'Firewall-2' (implicitly defined asset) to filter traffic via its 'Syn-Threshold-2B' and 'Syn-Proxy-2' policy, thereby giving rise to the implicit dependency cycle.

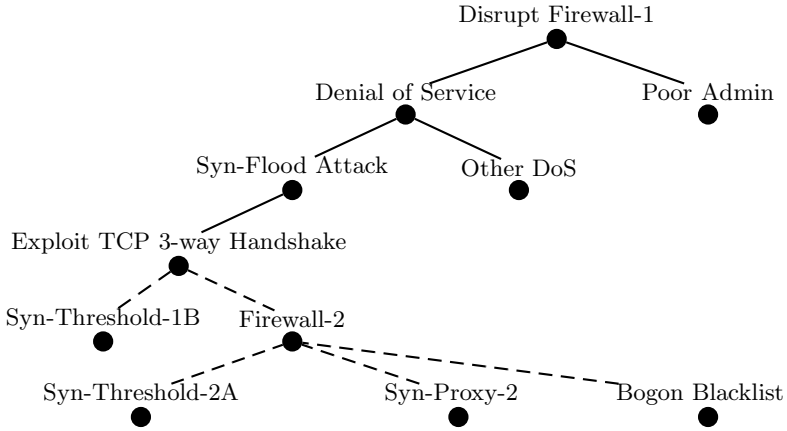
Unclear Threat Hierarchy. Threat trees are typically developed in isolation, that is they focus on a single threat target (decompose 'Disrupt Web Server' threat), and as a consequence it becomes difficult to inter-relate implicit information across multiple threat trees. Both 'Disrupt Web Server' and 'Disrupt Firewall' (Figure 1) form part of a forest of (implicitly related) threat trees. By the definition of a tree, a node should have one parent in the tree and given that the threat tree structure allows for one type of relationship, subsumption, a problem arises. The question is how should the hierarchy be constructed when assembling the overall tree forest? Either a new root node is created, 'Disrupt Servers' where both 'Disrupt Web Server' and 'Disrupt Firewall' are treated as disjoint siblings or the 'Disrupt Firewall' tree becomes a sub-node of the 'Firewall-1' node within the 'Disrupt Web Server' threat tree. The language provided by threat trees is not rich enough to state explicitly that the former sibling approach should be adopted with the inclusion of a dependency relationship that links the two trees together.

Tree Complexity. Modeling complex system threats can be challenging [5] due to tree-explosion and disparate trees (forest). While this may be addressed by software tools, much of the complexity must be managed implicitly and outside of the tree.

Semi-Formal. The semi-formal nature of a threat tree means that, in practice, any reasoning must be done outside of the tree structure [4]. For example, it would be useful to reason whether the concept of 'Denial of Service' has the same meaning in both trees of Figure 1.



(a) Partial Threat Tree: Web Server Syn-Flood DoS Attack



(b) Partial Threat Tree: Firewall Syn-Flood DoS Attack

Fig. 1. Threat Tree Forest

3 Semantic Threat Graphs

A semantic threat graph is an extended threat tree that addresses the issues discussed in Section 2. A semantic threat graph can be defined as a graph that represents the semantics (meaning) of a threat domain. Intuitively, a semantic threat graph makes explicit the information that is typically implicit in a threat tree.

Figure 2 provides a model of the components in a semantic threat graph and Figure 3 depicts an instantiation of this model for the threat tree example in the previous section. Enterprise IT assets are represented as *instances* of the *Asset* concept. An asset may have one or more *hasWeakness*'s (property relationship) that relate to individuals categorised in the *Vulnerability* concept (Figure 2). Instances 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, thus are defined as a *subConceptOf Asset*. Note, the *subConceptOf* has a double arrow head and defines a subsumption relation.

Explicit Concepts & Relationships. Domain experts explicitly specify the concepts (set of instances) of the threat domain and characterise their relationships to other instances and/or concepts. For example, a Network Access Controls expert specifies firewall configurations that adequately mitigates threats identified by a security manager.

Cascading Threats. With the graph-based approach cascading threats are identified explicitly within the threat graph model. Figure 3 demonstrates a threat that cannot be easily represented within a threat tree structure. Asserted relationships (for example *protects*) define that the *web server* is protected by *firewall-1* which, in turn, is protected by *firewall-2*, thus identifying the cascade threat. For simplicity, the scenario shown in Figure 3 models the *3-way handshake* as the

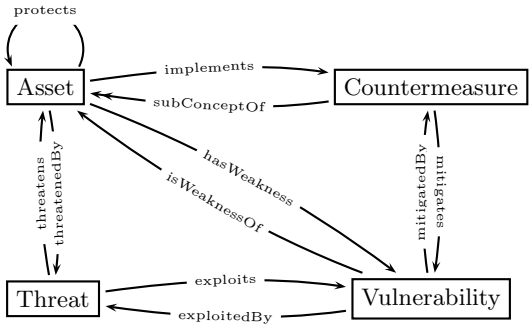


Fig. 2. Abstract Threat Graph Model

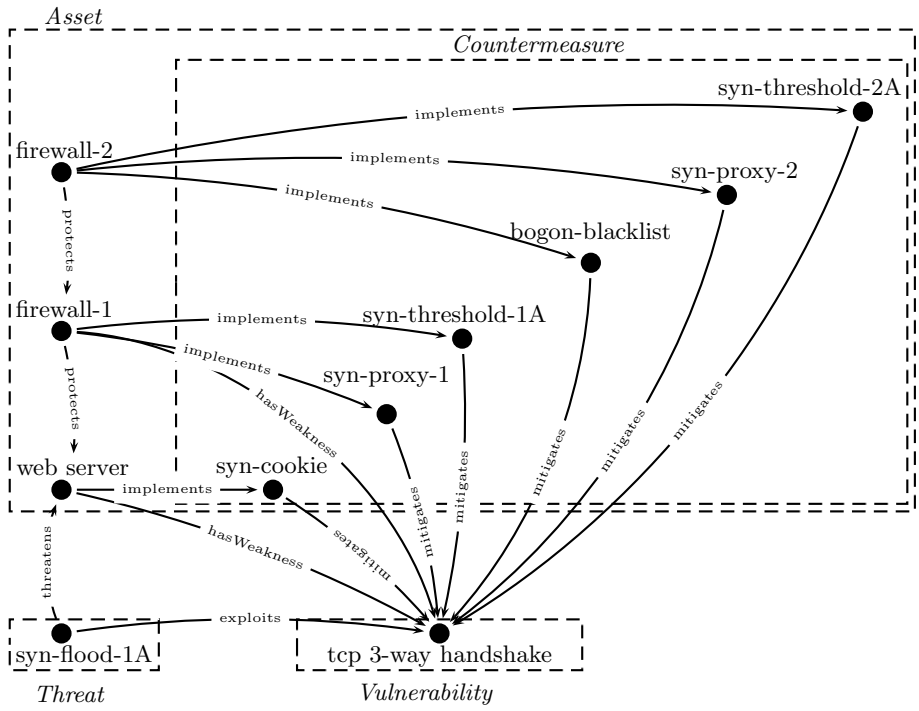


Fig. 3. Threat Graph: Web Server & Firewall Syn-Flood Countermeasure Dependency

same vulnerability for firewalls and systems. As a result of *firewall-1* mitigating the *syn-flood-1A* threat on the *web server* by way of a *syn-proxy-1*, it then adopts that threat while proxying the web servers TCP stream vulnerability.

Taxonomic Graph Hierarchies. Although threat trees provide hierarchies of the *Threat* concept, it lacks the capability to define a hierarchy for the implicit concepts within the tree. The threat graph model presented in Figure 2 can be further refined with sub-concepts that are more refined than their parent concepts. For example, the *Asset* concept can define a sub-concept *Server* to represent the set of servers (instances) an enterprise might have. This concept can in turn be further categorised as *Business Server* (containing Web, Email, Application servers and so forth) and *Protection Server* (for example, Firewalls, IDS's, VPN's, Anti-Spam). The *Threat* concept, as an additional example, can define a number of sub-concepts in accordance with best practice such as the Microsoft STRIDE standard (an acronym) whereby threats are categorised as follows: *S*poofing identity, *T*ampering with data, *R*epudiation, *I*nformation disclosure, *D*enial of service and *E*levation of privilege [16].

Managing Graph Complexity. Like threat trees, semantic threat graphs are prone to excessive graph complexity due to a large number of concept relationships.

The ability to compute a taxonomic hierarchy for a graph can assist navigation of complex graphs. Additionally, semantic reasoners and tools such as *Protégé* [17] help manage, navigate and ensure verifiable consistent graphs.

Formal Semantics. The concepts and relationships in semantic threat graphs are formally specified in terms of an ontology using Description Logic (DL) [8]. This is a decidable portion of first-order logic and is well-suited to the representation of and reasoning about domain knowledge.

4 An Ontology for Semantic Threat Graphs

An ontology provides a conceptual model of a domain of interest [7]. It does so by providing a vocabulary describing various aspects of the domain of interest and provides a rich set of constructs to build a more meaningful level of knowledge. Figure 2 depicts an abstract model for semantic threat graphs, whereby *classes* (concepts) represent sets of individuals (instances) and *properties* (roles) represent binary relations applied to individuals. Note that in presenting the model components, for reasons of space, we do not provide complete specifications in particular, definitions do not include disjoint axioms, sub-properties, data type properties or closure axioms.

Asset. Class $Asset$ represents any entity of interest within the enterprise that may be the subject of a threat. While assets can include people and physical infrastructure, in this paper we consider computer-system based entities such as Web servers, firewalls, databases, and so forth.

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

$$Asset \sqsubseteq \forall hasWeakness.Vulnerability \sqcap \forall threatenedBy.Threat \sqcap \forall implements.Countermeasure$$

For example, an instance `webServer` of the $Asset$ class that is vulnerable to a syn-flood based Denial of Service (DoS) attack can be defined by the following DL assertion (A-box), representing a fragment of knowledge in the ontology. Note atomic individuals are written in a `typewriter` font.

$$Asset(\texttt{webServer}) \leftarrow hasWeakeness(\texttt{webServer}, \texttt{tcpHandshake}) \sqcap threatenedBy(\texttt{webServer}, \texttt{synFlood})$$

Class $Asset$ can be further sub-classed, to more specific kinds of asset concepts, for example, $ProtectionServer \sqsubseteq Server$, whereby $Server$ is a sub-class of $Asset$. The class $ProtectionServer$ represents the NAC system individuals within the network (Cisco Pix, Linux Netfilter and so forth) and have a role in protecting

(*protects*) internal servers (including themselves). The *ProtectionServer* definition further restricts the *implements* property by requiring a protection server to implement one or more ($\exists_{\geq 1}$) countermeasures.

$$ProtectionServer \sqsubseteq Server \sqcap \exists_{\geq 1} protects.Server \sqcap \exists_{\geq 1} implements.Countermeasure$$

For example, the following fragment of knowledge in the ontology defines that the perimeter firewall (*gatewayNAC*) protects the Web server from a DoS by implementing a syn-threshold filtering countermeasure.

$$ProtectionServer(\texttt{gatewayNAC}) \leftarrow protects(\texttt{gatewayNAC}, \texttt{webServer}) \sqcap implements(\texttt{gatewayNAC}, \texttt{synThres})$$

Threat. A threat is a potential for violation of security [15]. An individual of the *Threat* class is considered to exploit one or more vulnerabilities ($\exists_{\geq 1}$ restriction):

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

For example, the DoS threat `synFlood` threatens a DMZ `webServer`.

$$Threat(\texttt{synFlood}) \leftarrow exploits(\texttt{synFlood}, \texttt{tcpHandshake}) \sqcap threatens(\texttt{synFlood}, \texttt{webServer})$$

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

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

For example, mis-configured *NAC* configurations have the potential to expose both internal servers and *NAC*'s alike to threats. The following fragment in the ontology states that the `webServer` is susceptible to a `synFlood` attack via the weakness `tcpHandshake`.

$$Vulnerability(\texttt{tcpHandshake}) \leftarrow isExploitedBy(\texttt{tcpHandshake}, \texttt{synFlood}) \sqcap isWeaknessOf(\texttt{tcpHandshake}, \texttt{webServer})$$

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

$$Countermeasure \sqsubseteq Asset \sqcap \exists_{\geq 1} mitigates.Vulnerability$$

Countermeasures can be further sub-classed into specific concepts, if desired. For example, an *AccessCtrlPolicy* is a countermeasure configured as one or more *NACPolicy* rules.

$$AccessCtrlPolicy \sqsubseteq Countermeasure \sqcap \exists_{\geq 1} configuredAs.NACPolicy$$

Countermeasure `synThres` mitigates the vulnerability `tcpHandshake` on the Web server (`webServer`); this countermeasure is configured as a collection of device-specific (Netfilter) *NACPolicy* rules, `synDoSLimit` and `synDoSDrop`.

$$\begin{aligned} AccessCtrlPolicy(\texttt{synThres}) \leftarrow\ & mitigates(\texttt{synThres}, \texttt{tcpHandshake}) \sqcap \\ & configuredAs(\texttt{synThres}, \texttt{synDoSLimit}) \sqcap \\ & configuredAs(\texttt{synThres}, \texttt{synDoSDrop}) \end{aligned}$$

An example of a low-level Netfilter syn-threshold rule-set that: a) limits the number of TCP connections to the web server to 1 per second after 4 connections have been observed and b) offending packets that exceed the limit are dropped, is expressed as follows:

"iptables -A FORWARD -d WebServerIP -p tcp –syn -m limit –limit 1/s –limit-burst 4 -j ACCEPT"
"iptables -A FORWARD -d WebServerIP -p tcp –syn -j DROP"

Based on previous research [9,10], the following are *DL* fragments that are representative of the above Netfilter rules:

$$\begin{aligned} NetfilterRule(\texttt{synDoSLimit}) \leftarrow\ & hasChain(\texttt{synDoSLimit}, \texttt{forward}) \sqcap \\ & hasDstIP(\texttt{synDoSLimit}, \texttt{webServerIP}) \sqcap \\ & hasProtocol(\texttt{synDoSLimit}, \texttt{tcp}) \sqcap \\ & hasTCPFlag(\texttt{synDoSLimit}, \texttt{syn}) \sqcap \\ & hasLimit(\texttt{synDoSLimit}, 1) \sqcap \\ & hasLimitBurst(\texttt{synDoSLimit}, 4) \sqcap \\ & hasTarget(\texttt{synDoSLimit}, \texttt{accept}) \end{aligned}$$

$$\begin{aligned} NetfilterRule(\texttt{synDoSDrop}) \leftarrow\ & hasChain(\texttt{synDoSDrop}, \texttt{forward}) \sqcap \\ & hasDstIP(\texttt{synDoSDrop}, \texttt{webServerIP}) \sqcap \\ & hasProtocol(\texttt{synDoSDrop}, \texttt{tcp}) \sqcap \\ & hasTCPFlag(\texttt{synDoSDrop}, \texttt{syn}) \sqcap \\ & hasTarget(\texttt{synDoSDrop}, \texttt{drop}) \end{aligned}$$

Threshold. This value is used to define the minimum degree of effectiveness of a countermeasure in mitigating the impact of a threat on an asset.

$$Countermeasure \sqsubseteq \exists_{=1} minEffect.Threshold$$

Simlarly, each threat has a threat level that defines the maximum impact on the asset.

$$Threat \sqsubseteq \exists_{=1} maxImpact.Threshold$$

For example, *Threshold* can be defined as enumerated class:

$$Threshold \sqsubseteq \{\texttt{high,medium,low,nil}\}$$

and a fragment in the ontology is

$$\begin{aligned} Threat(\texttt{synFlood}) \leftarrow\ & exploits(\texttt{synFlood}, \texttt{tcpHandshake}) \sqcap \\ & threatens(\texttt{synFlood}, \texttt{webServer}) \sqcap \\ & maxImpact(\texttt{synFlood}, \texttt{high}) \end{aligned}$$

The threshold level is used to characterize the extent to which a countermeasure mitigates a threat: an asset is considered secure if the effectiveness (threshold) of the countermeasures are greater than the impact (threshold) of the related threats. For example, if a `synCookie` was considered to have `medium` effectiveness at mitigating a `synFlood` then the asset remains under threat, albeit less threat than having no countermeasure. While CVSS [18], CVE [19], DREAD [20] and OSVDB [21] for example may provide suitable threshold metrics, the elicitation of threshold weightings is not the focus of this paper.

5 Case Study

5.1 Network System Configuration

A simplified 3-tier e-commerce *NAC* architecture is illustrated in Figure 4. The network at tier-1, also known as a Demilitarized Zone (*DMZ*), hosts the Web server that is accessible from the Internet. The gateway *NAC*, a firewall, implements a configuration that permits inbound packets from the Internet to the Web server on ports HTTP and HTTPS by way of an *AccessCtrlPolicy* $\texttt{cntr}_{web}$ countermeasure and drops all other irrelevant packets ($\texttt{cntr}_{denyOtherPkt}$). The following is a fragment of knowledge that defines the gateway firewall:

$$\begin{aligned} ProtectionServer(\texttt{gatewayNAC}) \leftarrow\ & protects(\texttt{gatewayNAC}, \texttt{webServer}) \sqcap \\ & implements(\texttt{gatewayNAC}, \texttt{cntr}_{web}) \sqcap \\ & implements(\texttt{gatewayNAC}, \texttt{cntr}_{denyOtherPkt}) \end{aligned}$$

Within tier-2, the application server communicates with the Web server over an SSL tunnel. The application firewall, (`appNAC`), implements countermeasure $\texttt{cntr}_{sslTunnelApp}$ to permit the correct SSL access.

$$\begin{aligned} ProtectionServer(\texttt{appNAC}) \leftarrow\ & protects(\texttt{appNAC}, \texttt{appServ}) \sqcap \\ & implements(\texttt{appNAC}, \texttt{cntr}_{sslTunnelApp}) \sqcap \\ & implements(\texttt{appNAC}, \texttt{cntr}_{denyOtherPkt}) \end{aligned}$$

Hosted in tier-3 is the database server. The backend data *NAC* is configured to permit SSH traffic, ($\texttt{cntr}_{sshTunnelDB}$), from the application server destined for the database server only.

$$\begin{aligned} ProtectionServer(\texttt{dataNAC}) \leftarrow\ & protects(\texttt{dataNAC}, \texttt{dbServer}) \sqcap \\ & implements(\texttt{dataNAC}, \texttt{cntr}_{sshTunnelDB}) \sqcap \\ & implements(\texttt{dataNAC}, \texttt{cntr}_{denyOtherPkt}) \end{aligned}$$

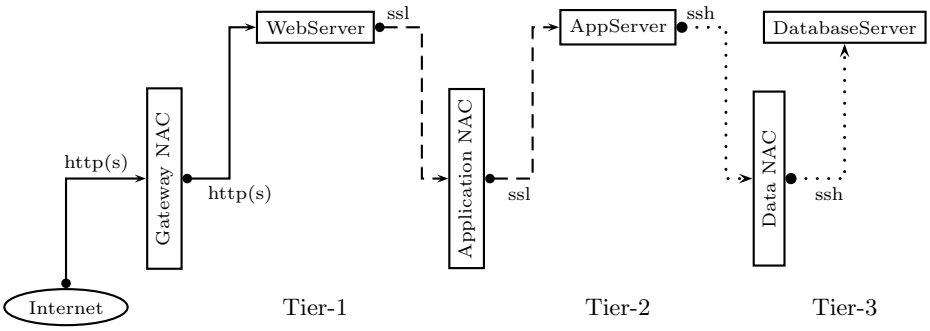


Fig. 4. Abstract 3-Tier Enterprise E-Commerce *NAC* Architecture

5.2 Threshold Satisfiability Analysis

The ideal *NAC* configuration is one that permits only valid traffic, and, no more and no less. An example of threats to the database, asserted within the ontology, is identified in Table 1. The focus here is on the threat of permitting a set of clients (IP Addresses) direct or indirect access to the database. For example, whether the database be accessible directly by the Web server IP address, in which case this threat impact is considered `high` as identified by $\texttt{threat}_{web}$. Note that $\texttt{threat}_{tier1}$ subsumes $\texttt{threat}_{web}$ since the Web server's IP address is contained in the network IP range of tier-1.

Table 1. Example Threats of Unintended IP Address Access to the Database

Asset	Threat	Unintended IP Access	$\text{Threat}_{maxImpact}$
dbServer	threat_{web}	webServIP	high
dbServer	threat_{tier1}	tier1Subnet	high
dbServer	threat_{tier2}	tier2Subnet	medium
dbServer	threat_{app}	appServerIP	nil

The following *DL* fragment represents the multiple *NAC* systems protecting the database server as part of a defense-in-depth strategy:

$$\begin{aligned} \mathit{DatabaseServer}(\texttt{dbServer}) \leftarrow\ & \mathit{isProtectedBy}(\texttt{dbServer}, \texttt{gatewayNAC}) \\ & \mathit{isProtectedBy}(\texttt{dbServer}, \texttt{appNAC}) \sqcap \\ & \mathit{isProtectedBy}(\texttt{dbServer}, \texttt{dataNAC}) \end{aligned}$$

A sample collection of *NAC* policy configurations (defense-in-depth) currently protecting the database is provided in Table 2. We consider a number of inadequate countermeasures that expose the database to unintended access. The table outlines the name of the *NAC* system, its implemented countermeasure

Table 2. Example NAC Policy Countermeasures and their associated Effectiveness

NAC	NAC_{cntr}	$cntr_{minEffect}$	NAC_{rule}	Src IP	Dst IP	DstPort
appNAC	$cntr_{webTrafficGen}$	low	$nacr_{app1}$	webServIP	tier2Subnt	any
dataNAC	$cntr_{sshTunnelDB}$	high	$nacr_{bac1}$	appServIP	dbServIP	22
dataNAC	$cntr_{sshTunnelDBGen}$	medium	$nacr_{bac2}$	tier2Subnet	dbServIP	22
dataNAC	$cntr_{sshTier1Gen}$	low	$nacr_{bac3}$	tier1Subnet	dbServIP	22

(NAC_{cntr}), the mitigation effectiveness ($cntr_{minEffect}$) and the corresponding low-level NAC_{rule} components (such as source IP address).

The `appNAC` firewall implements a generic access countermeasure that enables much more than the intended SSL access to the application server. In effect, $\texttt{cntr}_{webTrafficGen}$ permits the Web server access to all systems and services in tier-2 and is rated as having a `low` effectiveness in mitigating unnecessary database access. A compromised Web server can now be used as a launch pad for attacks on systems in tier-2, inclusive of the `dataNAC` Ethernet interface within that network tier. A more restrictive countermeasure ($\texttt{cntr}_{sslTunnelApp}$) is required.

The correctly implemented countermeasure ($\texttt{cntr}_{sshTunnelDB}$) within the `dataNAC` policy configuration helps in mitigating the misconfiguration of firewalls upstream. Thus defence-in-depth is borne out as a result. However, in practice, managing *NAC* policy configurations is complex [9] and there may be a requirement for multiple application servers within tier-2 to communicate with the database server. Rather than defining specific countermeasures for each application server, an administrator may, in the knowledge of being protected by a firewall upstream, implement a generic countermeasure such as $\texttt{cntr}_{sshTunnelDBGen}$, that provides blanket SSH access to the database server from all systems in tier-2. Other non-bastion hardened servers (such as intranet LDAP server) in tier-2 could then be used as launch pad when attacking the database server in tier-3, for example a SSH brute force attack. On individual basis both $\texttt{cntr}_{webTrafficGen}$ and $\texttt{cntr}_{sshTunnelDBGen}$ may appear as minor oversights in their respective tiers, however it is their conjunction that provides indirect and unacceptable access from the Web server to the database.

The `dataNAC` firewall inadequately mitigates the threat $\texttt{threat}_{tier1}$ (outlined in Table 1), as countermeasure $\texttt{cntr}_{sshTier1Gen}$ directly permits SSH access from all tier-1 systems to the database. Perhaps remote database administration is a requirement through an SSH proxy server in the DMZ. Unintended access from the tier-1 through the `dataNAC` may be considered to have less of a threat impact if one can guarantee that firewalls upstream apply more restrictive port-forwarding controls. However, and in keeping with the defense-in-depth approach, countermeasure $\texttt{cntr}_{sshTier1Gen}$ should be refined to have a more restrictive access policy.

Analysis of security policy configuration is performed using the Semantic Web Rule Language *SWRL* [22]. *SWRL* complements *DL* providing the ability to infer additional information from *DL* constrained ontologies. *SWRL* rules are

Table 3. Inadequate Countermeasure Analysis Report

Threat	$\text{Threat}_{maxImpact}$	NAC	$\text{NonCompliantNAC}_{cntr}$	$\text{cntr}_{minEffect}$	NAC_{rule}
threat_{web}	high	appNAC	$\text{cntr}_{webTrafficGen}$	low	nacr_{app1}
threat_{tier1}	high	dataNAC	$\text{cntr}_{sshTier1Gen}$	low	nacr_{bac3}

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.

The following is an excerpt of a *SWRL* query ($sqwrl : select$) that analyses the overall *NAC* configuration for breaches in threshold satisfiability due to inadequate countermeasures. It reports if countermeasures ($?c$) implemented by a *NAC* ($?nac$) to protect vulnerable assets ($?a$) are not effective ($swrlb : lessThan$ comparison) in mitigating the threat impacts ($maxImpact$).

$$threatenedBy(?a, ?t) \wedge hasWeakness(?a, ?v) \wedge exploits(?t, ?v) \wedge mitigates(?c, ?v) \wedge$$
$$isProtectedBy(?a, ?nac) \wedge implements(?nac, ?c) \wedge maxImpact(?t, ?imp) \wedge$$
$$minEffect(?c, ?eff) \wedge \ldots \wedge swrlb : lessThan(?eff_{value}, ?imp_{value})$$
$$\rightarrow sqwrl : select(?t, ?imp, ?nac, ?c, ?eff, ?nr)$$

Table 3 depicts the results of this query on the knowledge-base regarding inadequate threat mitigation of unintended client access.

5.3 Configuration Recommendation Synthesis

Synthesis provides configuration recommendations from a catalogue of best-practice countermeasures. For example, it is considered best practice that *NAC*'s implement anti-spoofing bogon rules as described by [11,12,13] to protect its internal servers and end-user work stations. As a consequence, a *NAC* policy should prohibit incoming packets claiming to originate from the internal network. Similarly both [6,23] are examples of best practice with regard to mitigating DoS attacks. Countermeasures `synDoSLimit` and `synDoSDrop` introduced in Section 4 are examples of DoS catalogue countermeasures.

The following *SWRL* rule excerpt, states if an asset is threatened by a spoofing attack (`spoof` individual) as a result of a particular weakness in the TCP/IP stack, `ipHeaderForgery` (individual) then that asset's protecting *NAC* should implement a set of best practice catalogue countermeasures that adequately ($swrlb : greaterThanOrEqual$) reduce the threat.

$$threatenedBy(?a, \texttt{spoof}) \wedge hasWeakness(?a, \texttt{ipHeaderForgery}) \wedge$$
$$exploits(\texttt{spoof}, \texttt{ipHeaderForgery}) \wedge mitigates(?c, \texttt{ipHeaderForgery}) \wedge$$
$$isProtectedBy(?a, ?nac) \wedge maxImpact(\texttt{spoof}, ?imp) \wedge minEffect(?c, ?eff) \wedge$$
$$\ldots \wedge swrlb : greaterThanOrEqual(?eff_{value}, ?imp_{value}) \rightarrow implements(?nac, ?c)$$

6 Tool Support

The *semantic threat graph* (ontology) and case study described in this paper where implemented in OWL-DL, a language subset of OWL which is a W3C standard that includes *DL* reasoning semantics [24]. *Protégé* is a plug-and-play knowledge acquisition framework that provides a graphical ontology editor [17]. Protégé interfaces with a *DL* based reasoner called *Pellet* providing model classification and consistency [25]. In conjunction to *DL* reasoning support, the *SWRL* Protégé plug-in (SWRLTab), allows for the creation of horn-like logic rules that interfaces with an expert system called *Jess* [26,22]. In practice, a domain expert using such tools can avoid or at least limit having to become an expert in *DL* and/or *OWL* notation, as these semantic editors and underlying reasoning tools hide much of the underlying complexity.

7 Related Research

A number of existing research approaches extend the threat tree model in particular [4,27,5,28]. Additional boolean node operators: *NAND, XOR and NOR*, and the incorporation of *defense nodes* as countermeasures is described by [28]. Edge et al [5] define a Protection Tree and Bistarelli et al [4] define a Defense Tree as countermeasure-centric extensions to the threat tree approach. The research carried out by [27] describes an *Enhanced Attack Tree (EAT)* that supports temporal dependencies and sequential threat events that must occur for an attack to be successful.

While these approaches are aimed at resolving particular inadequacies within the vanilla threat tree model, they still operate at rather high-levels of abstraction and are limited with regard to viable threat-only-vectors that contain implicit information such as assets, vulnerabilities and so forth. Our approach differs by extending the threat tree model to include semantic knowledge about fine-grained security configuration and, how it relates to assets, threats, vulnerabilities and countermeasures. Thus, the graph based approach makes explicit the information that is typically implicit in a threat tree. The semantic threat graph model is implemented in a knowledge representation system (Protégé) that provides a language for defining an ontology and an ability to conduct inferences accross the graph.

While the semantic threat graph model proposed in this paper is applicable to any threat and countermeasure domain, the case study focused on NAC countermeasures, building on existing ontologies [9,10]. An ontology for Netfilter firewall is described in [10] and is used to (binary) test the consistency of firewall rules with respect to a Semantic Web application policy. [9] considers the inter-operation of multiple Netfilter firewall and TCPWrapper proxies with respect to business and network service requirements.

8 Conclusion

This paper outlined a threat management approach using an ontology to construct, reason about and manage security policy configurations in the context of semantic threat graphs. Threat tree models used to represent threats at a high-level of abstraction, their singular threat vector focus and their practical suitably in a localised context (individual trees) do not explicitly capture all the entities involved in the threat management process. The *STG* extends the threat tree model to include semantic knowledge about low-level security configurations.

The model was used to build a knowledge-base of best-practice countermeasures (e.g. PCI-DSS & NIST) and system security policies against known threats. The ontology was populated with around one hundred threat, vulnerability and countermeasure combinations providing a relatively small catalogue for testing purposes. A case study on *NAC* demonstrated how security configurations can be analysed using knowledge about the countermeasures effectiveness in mitigating threats and how automated security mechanism configuration recommendations can be made based on catalogues of best-practice countermeasures.

Future research shall investigate how larger catalogues might be pre-populated from existing vulnerability databases such as OSVDB using knowledge engineering techniques such as case-based reasoning. An investigation of scalability regarding our approach will also need to be conducted. Emphasis on dynamic elicitation of threshold weightings also needs to be considered.

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

References

1. Mauw, S., Oostdijk, M.: Foundations of Attack Trees. In: Won, D.H., Kim, S. (eds.) ICISC 2005. LNCS, vol. 3935, pp. 186–198. Springer, Heidelberg (2006)
2. Stamatelatos, M., Vesely, W., Dugan, J., Fragola, J., Minarick, J., Railsback, J.: Fault Tree Handbook with Aerospace Applications. NASA Office of Safety and Mission Assurance NASA Headquarters, Washington, DC 20546, Version 1.1 (August 2002)
3. Schneier, B.: Secrets and Lies Digital Security in Networked World. Wiley Publishing, Chichester (2004)
4. Bistarelli, S., Fioravanti, F., Peretti, P.: Defense trees for economic evaluation of security investments. In: 1st International Conference on Availability, Reliability and Security (ARES), Vienna (April 2006)
5. Edge, K., Raines, R., Grimaila, M., Baldwin, R., Bennington, R., Reuter, C.: The Use of Attack and Protection Trees to Analyze Security for an Online Banking System. In: Proceedings of the 40th Annual Hawaii International Conference on System Sciences (HICSS 2007) (2007)
6. Eddy, W.: RFC 4987: TCP SYN Flooding Attacks and Common Mitigations (August 2007), `http://ietf.org`
7. Taniar, D., Rahayu, J.W.: Web Semantics Ontology. Idea Publishing (2006)

8. Baader, F., Calvanese, D., McGuinness, D., Nardi, D., Patel-Schneider, P.: The Description Logic Handbook: Theory, Implementation and Applications. Cambridge University Press, Cambridge (2003)
9. Fitzgerald, W.M., Foley, S.N., Foghlú, M.O.: Network Access Control Interoperation using Semantic Web Techniques. In: 6th International Workshop on Security In Information Systems (WOSIS), Barcelona, Spain (June 2008)
10. Foley, S.N., Fitzgerald, W.M.: Semantic Web and Firewall Alignment. In: First International Workshop on Secure Semantic Web (SSW 2008), Cancun, Mexico. IEEE CS Press, Los Alamitos (2008)
11. IANA: RFC 3330: Special-Use IPv4 Addresses (September 2002), `http://ietf.org`
12. Rekhter, Y., Moskowitz, B., Karrenberg, D., de Groot, G., Lear, E.: RFC1918: Address Allocation for Private Internets (Febuary 1996), `http://ietf.org`
13. Wack, J., Cutler, K., Pole, J.: Guidelines on Firewalls and Firewall Policy: Recommendations of the National Institute of Standards and Technology. NIST-800-41 (2002)
14. Tracy, M., Jansen, W., Scarfone, K., Winograd, T.: Guidelines on Securing Public Web Servers: Recommendations of the National Institute of Standards and Technology. NIST Special Publication 800-44, Version 2 (September 2007)
15. Shirey, R.: RFC 2828: Internet Security Glossary (May 2000), `http://ietf.org`
16. Hernan, S., Lambert, S., Ostwald, T., Shostack, A.: Uncover Security Design Flaws Using The STRIDE Approach (2009), `http://microsoft.com/`
17. Gennari, J., Musen, M.A., Fergerson, R.W., Grosso, W.E., Crubezy, M., Eriksson, H., Noy, N.F., Tu., S.W.: The Evolution of Protege: An Environment for Knowledge-Based Systems Development. Journal of Human-Computer Studies 58(1) (2003)
18. FIRST: Common Vulnerability Scoring System (2009), `http://first.org/cvss/`
19. International, C.: Common Vulnerabilities and Exposures (2009 `http://cve.mitre.org/`
20. Meier, J., Mackma, A., Dunner, M., Vasireddy, S., Escamilla, R., Murukan, A.: Improving Web Application Security: Threats and Countermeasures. Microsoft Press (2003)
21. OSVDB: Open Source Vulnerability Database (2009), `http://osvdb.org/`
22. O'Connor, M.J., Knublauch, H., Tu, S.W., Grossof, B., Dean, M., Grosso, W.E., Musen, M.A.: Supporting Rule System Interoperability on the Semantic Web with SWRL. In: Gil, Y., Motta, E., Benjamins, V.R., Musen, M.A. (eds.) ISWC 2005. LNCS, vol. 3729, pp. 974–986. Springer, Heidelberg (2005)
23. Ferguson, P.: RFC 2827: Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source Address Spoofing (May 2000), `http://ietf.org`
24. Smith, M.K., Welty, C., McGuinness, D.L.: OWL Web Ontology Language Guide. W3C Recommendation, Technical Report (2004)
25. Parsia, B., Sirin, E.: Pellet: An OWL DL Reasoner. In: McIlraith, S.A., Plexousakis, D., van Harmelen, F. (eds.) ISWC 2004. LNCS, vol. 3298. Springer, Heidelberg (2004)
26. Friedman-Hil, E.J.: Jess the Rule Engine for the Java Platform. Version 7.0p1 (2006)
27. Camtepe, S.A., ulent Yener, B.: Modeling and Detection of Complex Attacks. In: 3rd International Conference on Security and Privacy in Communications Networks, Secure Comm, Nice, France (September 2007)
28. Opel, A.: Design and Implementation of a Support Tool for Attack Trees. Internship Thesis, Otto-von-Guericke University Magdeburg (March 2005)