CHAPTER 55

Detection of Conflicts in Security Policies

Cataldo Basile[1], Matteo Maria Casalino[2], Simone Mutti[3], and Stefano Paraboschi[3] [1]Politecnico di Torino, Torino, Italy [2]SAP Research Sophia-Antipolis, Mougins, France [3]Universita degli studi di Bergamo, Bergamo, Italy

Abstract

Tools are needed to support the analysis of the security policies, and a crucial element that signals problems in the policies is represented by the presence of conflicts (contradictions or ambiguities in the policy specification, which may lead to anomalies in the application of the policy). Several types of conflict can be identified. Each type has been the subject of significant investigation, and several approaches and techniques have been examined for their detection and management. Rather than present exhaustive coverage, the chapter seeks to identify common approaches to identifying security conflicts, considering three relevant scenarios: access control policies, policy execution, and network protection. The chapter focuses on the detection of conflicts. Limited attention is given to ways to manage a detected conflict. The basic assumption of the chapter is that the security administrator is notified of each detected conflict and that he will have the responsibility of choosing the correct approach to manage the conflict. In large policies, the number of notifications can be large, and the need arises to have tools that automatically manage conflicts by introducing corrections to the policy that follow a specific optimization criterion. We give only limited attention to this aspect.

Keywords

Abstract policies; Conflicts; Contradictory and redundant; Executable policies; Policies; Policy enforcement mechanisms; Privacy; Security policies; Security requirements

1. Introduction

The evolution of information systems is continuously increasing the capabilities and range of offered services, leading to infrastructures that see the participation of a larger number of users, a greater level of integration among separate systems, and a correspondingly larger impact of possible misbehaviors. Security solutions are available to protect the correct delivery of services, but these solutions have to adapt to the increasing complexity of system architectures. In this scenario, the management of security becomes a critical task. The goal is not only part of natural business practices; it is also required to show compliance with respect to the many regulations promulgated by governments.

In modern information systems, a particular area of security requirement is access control management, with security policies that describe how resources and services should be protected. These policies offer a classification of the actions on the system that distinguishes them into authorized and forbidden, depending on a variety of parameters. Given the critical role of security and their large size and complexity, concerns arise about the policy's correctness. It is no longer possible to rely on the security designer to guarantee that the policy correctly represents how the system should protect access to resources.

Examples will be used to support the explanation and try to make the description self-contained. We want to provide an understanding of what we perceive as the main applications of these techniques.

The chapter is organized as follows. Section 2 introduces the concept of conflict in a security policy; then the resolution of conflicts is discussed and the relationship with separation of duty constraints is illustrated. Section 3 presents conflicts that arise in executing a security policy; the example of security policies for Java EE is used as an example. Section 4 offers an extensive analysis of conflict detection for network policies; significant attention is dedicated to this area, because it represents the domain in which there is large experience in the use of conflict detection. Section 5 illustrates how Semantic Web technology can support the detection of conflicts in generic policies. Section 6 presents a few concluding remarks.

2. Conflicts in Security Policies

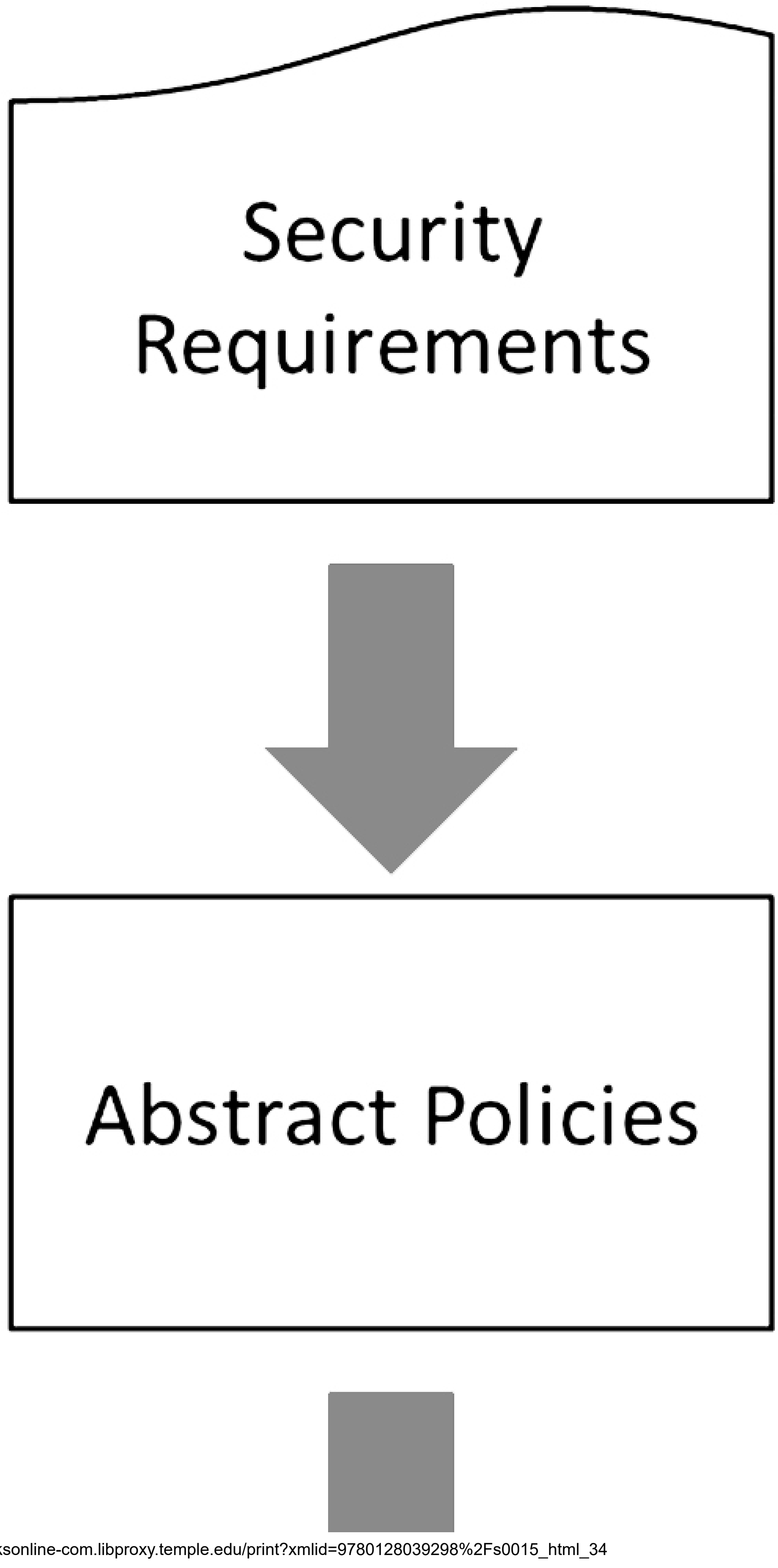
A typical top-down representation of the protection of an information system might consist of the five layers shown in Fig. 55.1.

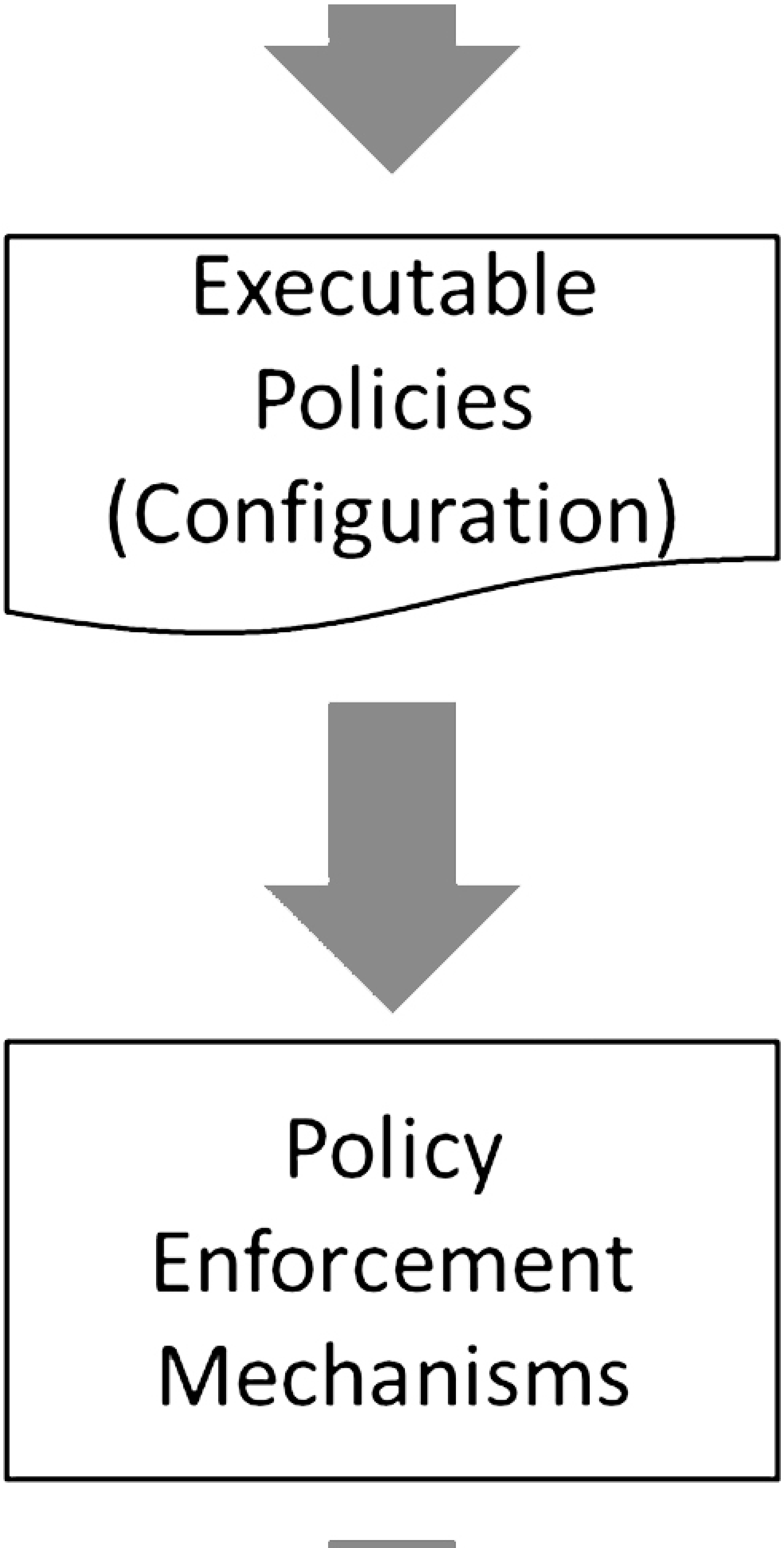
Security Requirements

Security requirements are a high-level, declarative representation of the rules according to which access control must be regulated. Security requirements largely ignore details of the system used to deliver the service, but focus on business concepts. This layer uses terminology and levels of detail typical of managers that are commonly expressed using natural language. For this reason, formal consistency verification cannot be applied automatically to security requirements, and so human intervention will be required to complete the task.

Policies

Policies represent how business requirements are mapped to the systems used for service provisioning. Policies can be defined at different levels, and the use of higher-level specification requires an approach to be adopted, possibly associated with a software tool, that supports the generation of lower-level representations.


Security Requirements
Abstract Policies


Executable Policies (Configuration)
Policy Enforcement Mechanisms

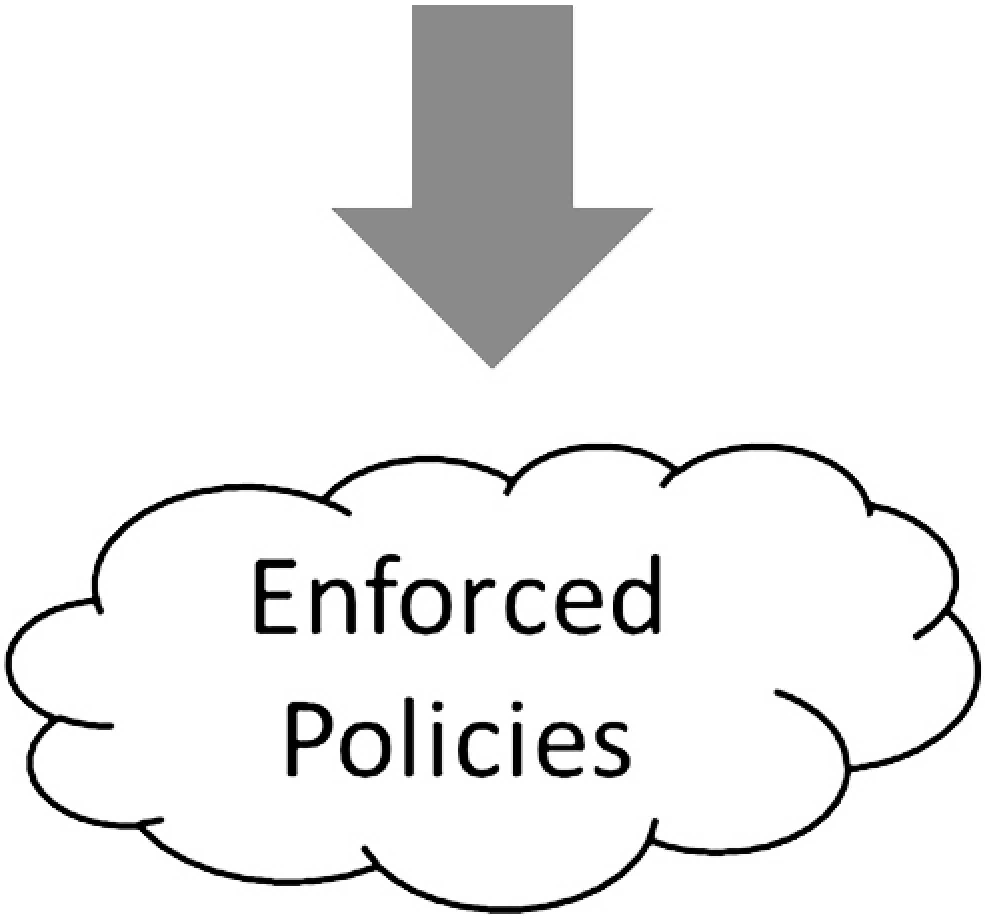


FIGURE 55.1 Top-down representation of the protection of an information system.

Abstract Policies

Abstract policies provide a formal representation of access control and its behavior. A policy may state, for instance, that an internal database storing credit card information must not be accessible from the Internet. It is declarative because it does not detail the actual mechanism used to enforce this policy. First, the policy is intended to define the desired behavior of the services. Because the policy will apply to abstract service definitions (services that are not yet instantiated), the specification cannot use full topological details.

Executable Policies

Executable policies describe the access control policy in a way that can immediately be processed by an access control component. Executable policies can be considered the security configuration of a system and are expressed in the specific language that a system recognizes. For instance, a policy for a relational Data Base Management System (DBMS) will typically be expressed by a sequence of Structured Query Language (SQL) statements.

Policy Enforcement Mechanisms

Policy enforcement mechanisms correspond to the low-level functions that implement the executable policies. It is convenient in the design and analysis of the system to separate the consideration of the policies (abstract and executable) from the mechanisms responsible for enforcing them, because each has its own weaknesses and threats.

Research has proposed multiple approaches for policy specification. Proposals have often been characterized by direct integration with the languages and models of the modern Web scenario. These models include industry standards such as eXtensible Access Control Markup Language (XACML) [1], which is interesting because it can be characterized as a mostly abstract policy language but it is also associated with tools that are able to process it directly, which makes it an executable policy. There are other abstract policy languages that a computer can directly process, such as rule-based policy notation using an if-then-else format, or proposals based on the representation of policies using Deontic logic for obligation and permissibility rules. Academic efforts produced solutions ranging from theoretical languages such as the one proposed by Jajodia et al. [2] to executable policy languages such as Ponder [3]. In the Semantic Web area proposals have emerged such as Rei [4] and KAoS [5]. Policy languages based on Semantic Web technologies allow policies to be described over heterogeneous domain data and promote a common understanding among participants who might not use the same information model.

A crucial advantage of using a formal policy representation, particularly at the abstract level, is the possibility of the early identification of anomalies. Security policies in real systems often exhibit contradictions (inconsistencies in the policy that can lead to an incorrect realization of the security requirements) and redundancies (elements of the policy that are dominated by other elements, increasing the cost of security management without providing benefits to the users or applications). The availability of a high-level and complete representation of the security policies supports the construction of services for the analysis of the policies able to identify these anomalies and possibly suggest corrections. A classical taxonomy of conflicts is shown in Fig. 55.2.

As depicted in Fig. 55.2, conflicts can be divided into two categories: (1) *intrapolicy* conflicts that may exist within a single policy and (2) *interpolicy* conflicts that may exist between at least two policies. For each category we have the following subcategories: (1) contradictory, (2) redundant, and (3) irrelevant.

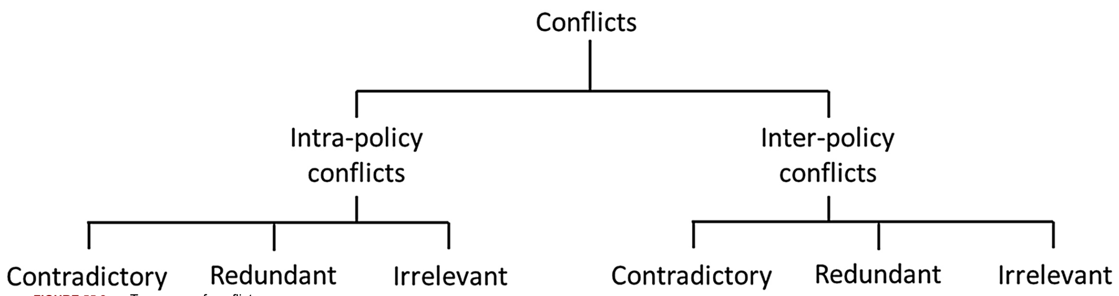


FIGURE 55.2 Taxonomy of conflicts.

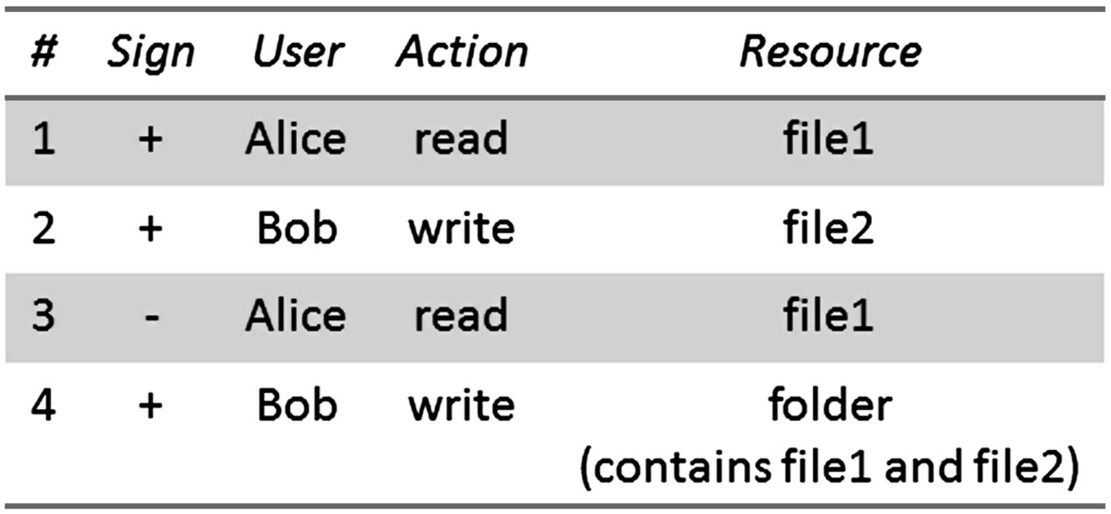
Contradictory

Contradictory conflicts arise when principals are authorized to do an action *a* on a resource *r* by *a* positive authorization, and are forbidden to do the same action *a* on the resource *r* by a negative authorization. In this case, the two authorizations are said to be incompatible. For example, in Fig. 55.3, Authorization 1 is in conflict with Authorization 3. In fact, Authorization 1 states that Alice can read file1, and Authorization 3 states that Alice cannot read file1.

Contradictory authorizations make the policy inconsistent. The security administrator has to be alerted to correct this error by editing or removing the conflict. In network policies, a classification is introduced that further refines this type of conflict.

Redundant

Redundant conflicts arise when an authorization is dominated by other authorizations and does not contribute to the policy (its removal would not modify the behavior of the system). Given two authorizations, a_1 and a_2, with the same action and sign, let us call p_i (respectively, r_i) the principals (respectively, resources) associated, directly or indirectly, with a_i. If $p_2 \subseteq p_1$ and $r_2 \subseteq r_1$ and a_2 is not involved in any conflict with other authorizations, then a_2 is redundant with respect to a_1 and can be safely removed from the policy without modifying the behavior of the system. For example, in Fig. 55.3, Authorization 2 is redundant with respect to Authorization 4 because Authorization 4 dominates (it is expressed on the folder) Authorization 2 (it is expressed on file1, but it is contained in the folder).



#	Sign	User	Action	Resource
1	+	Alice	read	file1
2	+	Bob	write	file2
3	-	Alice	read	file1
4	+	Bob	write	folder (contains file1 and file2)

FIGURE 55.3 Examples of contradictory and redundant conflicts.

Irrelevant

Irrelevant conflicts occur when the conflict can never manifest itself in a system. This may happen when specification of the elements of the authorizations cannot lead to activation of all of the authorizations involved in the conflict. Recognizing that a conflict is irrelevant may be hard, depending on the expressive power of the language used to represent authorizations. Examples are presented in Section 4 in a discussion of conflicts in network policies.

This classification is a starting point for evaluating the conflicts. In the next section we present some examples of conflicts and discuss introducing techniques to resolve conflicts to be able to solve contradictions in the policy.

Conflict Resolution

Security policies in real systems often exhibit conflicts and redundancies. The availability of a high-level and complete representation of the security policies supports the construction of services for the analysis of the policies able to identify these anomalies and possibly suggest corrections. Contradictions in the policy are also called *modality conflicts*. They arise when principals are authorized to do an action *a* on a resource *r* by a positive authorization, and are forbidden to do the same action *a* on resource *r* by a negative authorization.

In this case, the two authorizations are said to be *incompatible*. An example was presented earlier with Authorizations 1 and 3 in Fig. 55.3.

In the literature and in systems, several criteria have been proposed and implemented to manage this kind of conflict at the time policy execution [6] with the aim of removing ambiguity in the policy and solving the conflict. In that regard, consider the case of a generic network firewall device in which packet filtering rules are evaluated in a given order, and as such, any conflict among them (more than one rule matching to the same packet) is deterministically solved by evaluating the result dictated by the first matching rule. The rules that handle the composition of authorizations to solve the conflicts do not necessarily have to be fixed. More sophisticated languages are in fact equipped with specific constructs to instruct the policy evaluator to apply one of several possible composition strategies. The XACML language [1], for instance, defines so-called combining algorithms to compose the results of different access control rules. Examples of the available options are the "*deny-overrides*" algorithm, in which rules prescribing access denial take precedence. This means that in case of conflict, the negative authorization always wins, so a forbidden action will never be permitted. In the example presented in Fig. 55.3, this means that Authorization 3, which is negative for subject Alice, has priority over Authorization 1, so Alice is not allowed to read file1. Another strategy that sees extensive adoption in operating systems is the "*first-applicable*" one, in which rules are evaluated in order, such as in the case of a network firewall mentioned earlier. In a similar fashion, the Apache Web server access control configuration language permits specification of the order of priority of rule evaluation. For example, the "Order allow, deny" directive determines the priority of permissions over denials.

Other important criteria are those based on identification of a dominance relationship among rules. This is represented by the criterion "*most specific wins*," which states that when one authorization dominates the other, the more specific wins. In most cases this represents an adequate and flexible solution. A critical problem of this approach is that specificity may not always be defined for conflicting authorizations, for a variety of reasons.

A first case is represented by the authorizations supporting a hierarchy for any element of the (<*subject, action, resource*>) triple, with the possibility of being contained in more than one ancestor. For example, for a given action and resource, Authorization *A3* has a positive sign and is applied to Group *G1*, and Authorization *A4* has a negative sign and is applied to Group *G2*, with *G1* and *G2* not contained one into the other and with User *u* belonging to both groups. In this situation, the "*most specific wins*" does not solve the conflict. A second case occurs when containment hierarchies are possible on more than one element. For example, Authorization *A5* has a positive sign and applies to User *u* when accessing elements in Resource Group *RG*; Authorization *A6* has a negative sign and applies to User group *UG* when accessing Element *r*; if *u* is a member of *UG* and *r* is included in *RG*, the "*most specific wins*" criterion is not able to manage the conflict. Other solutions have been proposed that rely on the explicit specification of a priority for each authorization. If a partial order is specified using the same priority for sets of authorizations, the possibility of unresolved conflicts remains. If priorities build a total order on authorizations, conflicts would be solved, but it appears difficult to assign priorities efficiently that are consistent with the application semantics.

An option that can solve all of the conflicts is to combine multiple resolution criteria, applying each one only after the previous ones were not able to solve the conflict. For instance, the "*most specific wins*" can be applied first, and the "*deny overrides*" can be used to solve the remaining conflicts. In most cases this solution is preferable to identification of some fixed ordering of the authorization that is not consistent with the semantics of the policy. Another option that has a significant potential, particularly when dealing with abstract policies that will be mapped to executable policies, is to use the "*most specific wins*" criterion as a first step and let the conflict detection solutions notify the security administrator of the remaining conflicts, to modify the policy or introduce an ad hoc solution. Support for this approach can be found by using Semantic Web tools, as discussed in Section 5.

Separation of Duty

The conflicts presented until now derive from the presence of positive and negative authorizations in the same policy that can be applied to the same access request. A different kind of conflict derives from the definition in the security policy of constraints that the authorizations have to satisfy. An important class of constraints is *separation of duty* (*SoD*). These constraints follow the common best practice for which sensitive combinations of permissions should not be held by the same individual, to avoid violating business rules. The purpose of this constraint is to discourage fraud by spreading the responsibility and authority for an action or task, thereby raising the risk involved in committing a fraudulent act, by requiring the involvement of more than one individual. The idea of SoD existed long before the information age and is extensively used in some areas such as the banking industry and the military. *Role-based access control* can be adapted to express this kind of constraint because the role hierarchy allows easy mapping of real-world business rules to the access control model.

A well-known example is the process of creating and approving purchase orders. If a single person creates and approves purchase orders, it is easy and tempting for him to create and approve a phony order and pocket the money. If different people must create and approve orders, committing fraud requires a conspiracy of at least two people, which significantly lowers the risk.

The two main categories of SoD are (1) static SoD and (2) dynamic SoD (Fig. 55.4). The former category (also known as strong exclusion) is the simplest way to implement SoD. Given two roles, $role_1$ and $role_2$, static SoD between these two roles means that a User *u* must not exist who can activate both $role_1$ and $role_2$. For instance, if *Order Creator* and *Order Approver* are strongly exclusive roles, no one who may assume the Order Creator role would be allowed to assume the Order Approver role; on the other hand, no one who may assume the *Order Approver* role would be allowed to assume the *Order Creator* role.

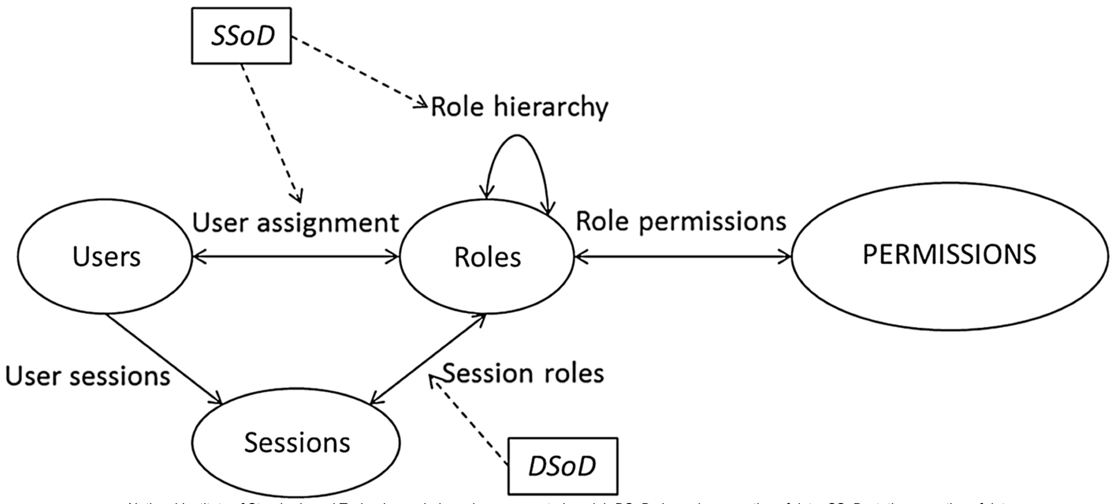


FIGURE 55.4 National Institute of Standards and Technology role-based access control model. *DSoD*, dynamic separation of duty; *SSoD*, static separation of duty.

The latter category (also known as weak exclusion) states that "A principal may be a member of any two exclusive roles, but he must not activate both at the same time." This definition implies that the system will keep precise track of each task. Before doing any task, the system will check that the SoD is not violated. Dynamic SoD allows users to perform roles that would be strongly exclusive in static systems.

Violations of the SoD constraints policy are another kind of conflict. Support for this conflict can be adequately provided by implementing conflict detection services, which are able to notify the security designer of inconsistencies in the policy. Resolution of such a conflict will typically require revising the policy, restricting the user's ability to enact conflicting roles, or modifying the specification of the constraint. The efficient identification of these violations can use ad hoc solutions. An interesting option is represented by the use of Semantic Web tools, as discussed in Section 5.

3. Conflicts in Executable Security Policies

So far, we have focused on how conflicts have been studied in the context of abstract security policies, in which the specific details of implementing policy enforcement mechanisms are not part of the model. Therefore they are not assumed to introduce any possible issue into the policy evaluation.

In this section we instead consider the case of concrete policy evaluation frameworks, in which an *evaluation algorithm* determines the effect of a given policy according to:

- An executable representation of the policy
- Context-dependent information

The executable policy can be seen as the configuration of the security enforcement mechanism, and it can be referred to as its *security configuration*. A security configuration is typically expressed according to a respective *configuration language*. The semantics of this language is ultimately given by the evaluation algorithm that computes the result of a configuration at operations time. As a consequence, configuration authors need to have a thorough understanding of the semantics evaluation, such that they can configure the behavior of the enforcing mechanism exactly according to the policy they want the system to implement.

Security configuration languages and corresponding evaluation semantics typically incorporate mechanisms to cope with conflicts that may arise at the evaluation stage. This can be achieved, on the one hand, by constraining the expressiveness of the configuration language so that some inconsistencies are syntactically ruled out; for instance, contradictions in the policy (see Section 2) cannot occur in the case of an access control configuration language that allows only specifying collections of positive (respectively, negative) authorization rules. On the other hand, solutions to resolve the conflicting situations can be included in the evaluation semantics. For example, the rules that determine the semantics of the composition of different constructs of the language can be designed to handle conflicts by applying a predetermined strategy, as discussed earlier.

Although policy conflicts are sorted out in the evaluation semantics of security configuration languages, errors still can be introduced by inexperienced configuration authors. As a matter of fact, the gap of abstraction that lies between a security configuration and the corresponding enforced abstract policy is similar to the difference between a program's source code and the behavior realized by an interpreter while executing the program (Fig. 55.5). As such, policies that enforce unintended security properties can stem from misconfigured security enforcement devices: for instance bugs in the program's source code producing incorrect runtime behavior.

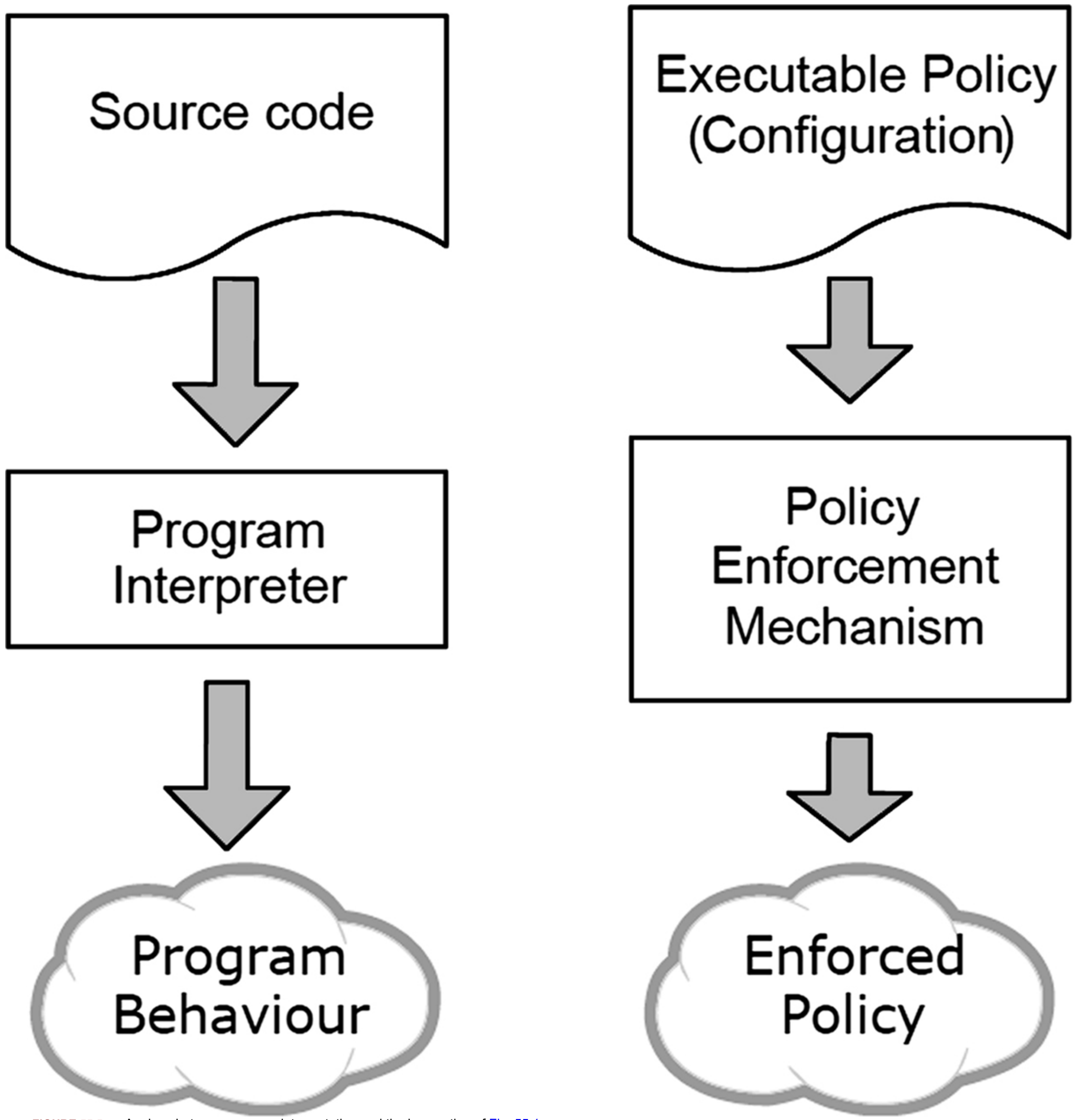


FIGURE 55.5 Analogy between program interpretation and the low portion of Fig. 55.1.

To solve this issue, researchers have been studying the characteristics of security policy languages and their semantics to:

- Identify in particular counterintuitive corner-cases or anomalous situations that likely stem from misconfigurations

- Propose models to detect and possibly solve such misconfigurations automatically

In the following we discuss these issues in the scenario of Java Enterprise Edition (EE). The treatment will also use some concrete examples of the security configuration language.

Java Enterprise Edition Access Control

The Java Enterprise Edition (Java EE) platform consists of a set of application program interfaces (APIs) and a runtime environment that allows development and execution of distributed Web-based applications. The basic execution model of a Java EE Web application is depicted in Fig. 55.6. Hypertext Transfer Protocol (HTTP) requests coming over the network are processed by the Java EE application server and abstracted to *HttpServlet Request* Java objects, which constitute the input of the Web application. Web applications are composed of *Web Components*, dealing with the client's requests and computing responses, and *JavaBeans Components*, which can be optionally involved to encapsulate the business logic of large-scale Web applications.

The interface between the Web Components and the application server, providing their execution environment, is standardized in the Java EE Servlet Specification [7]. This document establishes a contract between application server implementations on one side and Web applications on the other, prescribing, among others, a number of mechanisms to deal with security in Java EE Web applications.

Such mechanisms belong to two categories: programmatic security and declarative security. Programmatic security describes functionalities that developers can use through an API to implement security within their application's code. Declarative security refers instead to the enforcement of security properties (such as HTTP-based access control) achieved not through dedicated source code in the application, but rather through the declarative specification of security configurations. In the latter case, the enforcement of security at runtime is completely transparent to the Web application's developer. When the Web application is deployed within the application server, it comes together with a configuration file, the so-called deployment descriptor, in which security and several other aspects of the Web application's runtime environment are configured. Analogous mechanisms are likewise available for JavaBeans Components, as described in a dedicated specification [8].

Because we are interested in discussing examples of security configuration languages, in this section we focus on declarative security. We first provide an overview of its evaluation semantics and then examine different approaches to the analysis of Java EE security configurations.

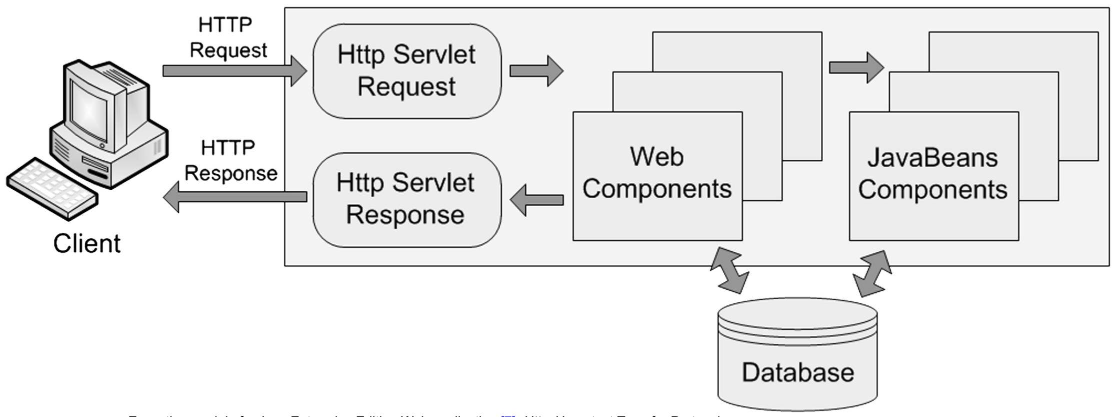


FIGURE 55.6 Execution model of a Java Enterprise Edition Web application [7]. *Http*, Hypertext Transfer Protocol.

The deployment descriptor of Web Components is an eXtensible Markup Language (XML) document that conforms to a grammar (XML schema) defined as part of the Servlet Specification. The security-related fragment of this grammar is the subtree rooted at the *security constraint* XML tag.

Every security constraint associates a set of resources [i.e., Uniform Resource Locators (URLs) of the Web application] with the required security properties. Two categories of security properties can be configured: *authorization constraints*, which are access control on URLs, and *user data constraints*, which stand for confidentiality or integrity requirements on data exchanged between the client and the Web application. Access control is configured by associating URL patterns and a (possibly empty) set of HTTP methods to at most one authorization constraint: that is, the set of roles allowed to access the mentioned resources. Wild cards are allowed in the definition of both URL patterns and granted roles: in particular (1) the special role name '∗' is shorthand for all the roles defined inside the deployment descriptor, and (2) entire URL hierarchies can be specified with URL patterns ending with the '/∗' wildcard.

Similarly, requirements for data confidentiality or integrity are specified by associating URL patterns and HTTP methods with one or more transport guarantee XML nodes, containing either the CONFIDENTIAL or INTEGRAL keywords.

According to the informal semantics from Rubinger et al. [9], to have access granted, a user must be a member of *at least one of the roles* named in the security constraint (or implied by '∗') that matches her or his HTTP request. An empty authorization constraint means that *nobody* can access the resources, whereas access is granted to *any* (possibly unauthenticated) user in case the authorization constraint is omitted. Unauthenticated access is also allowed by default to any unconstrained resources. An intuitively insignificant syntactic difference, such as omitting the authorization constraint instead of specifying an empty one, corresponds to a major gap in semantics: *allow all* or *deny all* behaviors, respectively, are obtained.

In case the same URL pattern and HTTP method occur in different security constraints, they have to be conceptually composed, because they apply to overlapping sets of resources. Concerning access control, if two nonempty authorization constraints are composed, the result is the *union* of the two sets of allowed roles. If one of the two allows unauthenticated access, the composition does so as well. In contrast, if one of the sets of roles is empty, their composition is empty; that is, the *intersection* of the two sets is performed in this case. Constraints on more specific URL patterns (*/a/b*) always override more general ones (*/a/∗*). The composition of user data constraints, instead, is always the union of the single requirements.

The following snippet is an example of two overlapping security constraints. As a result of their composition, no access is granted to the URL hierarchies '∗' and '*/acme/wholesale/∗*' via the *DELETE* and *PUT HTTP* methods. Access to '*/acme/wholesale/∗*' via *GET* is restricted to users with the role *SALESCLERK*. HTTP requests with any method other than the aforementioned are instead granted to anyone:

```
<security-contraint>
<web-resource-collection>
<url-pattern>/*</url-pattern>
<url-pattern>/acme/wholesale/*</url-pattern>
<http-method>DELETE</http-method>
<http-method>PUT</http-method>
</web-resource-collection>
<auth-constraint/>
</security-contraint>
<security-contraint>
<web-resource-collection>
<url-pattern>/acme/wholesale/*</url-pattern>
<http-method>GET</http-method>
<http-method>PUT</http-method>
</web-resource-collection>
<auth-constraint>SALESCLERK</auth-constraint>
</security-contraint>
```

It is suggested [10] that the evaluation semantics of security constraints for Java EE Web Components is partly counterintuitive, specifically in its fragments concerning composition, which is where the rules to deal with conflicts are encoded, as argued earlier. The peculiar handling of unconstrained HTTP methods and the fact that more specific URL patterns should override less specific ones, for instance, may lead to unexpected behaviors, as illustrated in the following example.

Let us consider again a couple of security constraints introduced previously. According to the earlier interpretation, the *HTTP DELETE* requests to the URL*/acme* are denied, because the first constraint applies. In contrast, requests to the same URL but through any unconstrained method, such as *GET*, are allowed to anyone.

We now assume that a system administrator, wanting to deny *GET* requests to the URL*/acme*, added the following constraint:

```
<security-contraint>
<web-resource-collection>
<url-pattern>/acme</url-pattern>
<http-method>GET</http-method>
</web-resource-collection>
<auth-constraint/>
</security-contraint>
```

Because this new constraint is the most specific for the URL*/acme*, and it does not specify any behavior for methods other than *GET*, it introduces a side effect by allowing requests that were previously denied. For example, the *DELETE* requests to*/acme* become *allowed to anyone* after introducing this constraint. This behavior is particularly counterintuitive because the new constraint does not include a reference to the *DELETE* method, which is nevertheless affected. Also, although apparently it specifies *access denial* (empty authorization constraint), it implicitly carries *access permission* semantics for every unconstrained method.

Al-Shaer and Hamed [10] argue for the need for a formal characterization of the semantics of security constraints, which can be used as a reference to check the correctness of the behavior of both application server implementations and Web application configurations. Hence, they propose a set-theoretic model that captures the expressiveness of Java EE Web Components' authorization constraints, in which resources form a set ordered according to the URL tree hierarchy and sets of roles, ordered by inclusion, form a lattice of permissions.

4. Conflicts in Network Security Policies

The identification of conflicts in security policies has been investigated especially in the scenario of the configuration of computer networks. This area of security sees significant industrial interest; it is currently one of the most critical components in the protection of an information system from external threats and relies on a protection model that is well understood and adequate to realizing a number of ad hoc solutions. Thus, we consider it interesting to analyze solutions that have been devised to detect policy conflicts in this scenario. The analysis will give a more precise understanding of problems that can be faced when managing conflicts in a real system. The results of work in this area provide important guidelines that can drive the design of this functionality in the different scenarios in which security policies are defined. First, we will consider the configuration of firewalls. Then, we will analyze how the configuration of channel protection solutions can identify other kinds of conflicts in the policy.

Filtering Intrapolicy Conflicts

Firewalls are devices used to separate parts of networks parts that have different security levels; in fact, they are able to enforce an authorization policy that selects the traffic to be allowed according to a security policy expressed as a set rules, often named the access control list (ACL). The rules are composed by a *condition* clause, formed by a series of predicates over some packet header fields, and an action clause, determining the action to be enforced, typically allowing or denying the traffic.

When a new packet arrives at one of the firewall network interfaces, the values from its headers are used to evaluate the condition clause predicates [10,11]. A packet matches a rule if all of the predicates of the rule are true. If a packet matches only one rule, the action enforced is taken of its action clause. However, in an ACL a packet can match more than one rule; therefore rules are prioritized and the action from the matching rule at the highest priority is enforced. This approach is often named the "*first applicable*" resolution strategy, based on ordering rules by priority and starting from the one with the highest priority; the action enforced is the one from the first matching rule. However, hardware-based approaches use ad hoc algorithms and fast memories that speed up the matching process considerably. In practice, the ACL is not scanned linearly, because the action is selected by fast look-up algorithms [12]. It also may happen that a packet does not match any of the ACL rules. In that case, a default action is enforced; typically, the traffic is denied and the packet is dropped.

Firewalls are categorized according to their capabilities or the layer at which they work (that is, the headers they can consider). The simplest firewall capability is the *packet filter*, working at the network and transport International Organization of Standardization/Open System Interconnection (ISO/OSI) layer, which makes decisions based on five fields: the Internet Protocol (IP) address and ports of the source and destination, and the IP protocol type. Packet filters do not maintain state information [distinguishing packets that belong to an established Transmission Control Protocol (TCP) connection], and they are also referred to as *stateless firewalls*.

A firewall that performs the stateful packet inspection is named a *stateful firewall*, and it usually maintains information about the TCP state, but also about other stateless protocols (Internet Control Message Protocol echo-request echo-reply sequences) or stateful application-layer protocols [understanding the opening of File Transfer Protocol (FTP) data ports in active or passive mode]. At the highest level of the ISO/OSI stack, there are the *application firewalls*. Because application protocols are heterogeneous, application firewalls are usually tailored to one or more specific protocols to perform a more focused analysis. The most widespread one is the Web Application Firewall, which observes HTTP properties and fields, including Multipurpose Internet Mail Extension objects, and, if integrated with the Web service it protects, can also circumvent common attacks and vulnerability exploits. In addition, application firewalls are able to check the "RFC compliance" that verifies whether the protocol traffic is consistent with the standards or with a set of nonharmful implementations. Therefore, the condition clause of stateful and application firewalls also contains predicates over state information and application protocol fields.

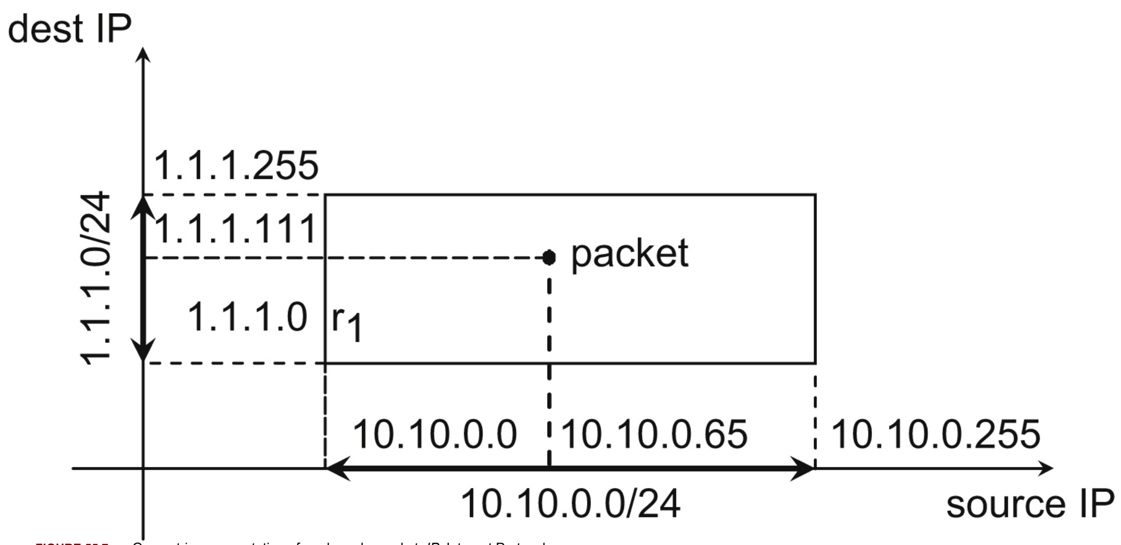


FIGURE 55.7 Geometric representation of a rule and a packet. *IP*, Internet Protocol.

Condition clauses are not just modeled as logical predicates; in fact, many works represent them as using geometrical models that are proven equivalent. According to the geometric view, every packet is a point in a decision space composed by many dimensions, one for each field for which it is possible to state a condition. A rule thus becomes a hyperrectangle. For instance, the decision space of packet filters is often named five-tuple space. A packet matches a rule if it is in the rule hyperrectangular area. For instance, a simple bidimensional case is represented in Fig. 55.7.

Because firewalls are a major security shield against attacks and intrusions, their correct configuration has always worried administrators. However, most firewalls are poorly configured, as Wool highlighted in a study in 2004 whose trend was confirmed [13,14]. Historically, three approaches are used to verify the correctness of intrafirewall policies: manual testing, query-based approaches, and the use of conflict and anomaly analysis tools. Companies have been using complex distributed systems with many firewalls and redundant controls; therefore, all of these approaches have been designed or extended to interfirewall policies analysis.

In interfirewall policy analysis, verification of the correctness of the action enforced by a firewall is extended to a more general case, evaluating actual reachability by analyzing the actions enforced by all of the firewalls encountered in a communication path.

Manual Testing

Manual testing is the first and simplest case. It can be performed by actually trying a set of connections to verify whether they succeed and comparing them with the authorization policy, or using software able to probe hosts, servers, and other devices for open ports and available features: that is, the vulnerability scanner. Many scanners are available for this purpose, mostly as open-source software such as Nmap [15], Amap [16], and Nessus [17]. They are sophisticated and can be used to detect more complex cases (to recognize operating system and software fingerprints, or to distinguish filtered ports from closed ones) or to identify known vulnerabilities. This approach is time-consuming and requires an effort that is beyond the administrator's possibilities, especially in large networks. In addition, it requires actual deployment of the policy and physical access to the network, which may also be flooded by probe packets that may interfere with normal network functioning. Although scanners' output is detailed, they need a further step to be compared with the firewall policy and to know if it has been correctly implemented.

5. Query-Based Conflict Detection

The first attempts to overcome the limitations of the manual approach consisted of representing a firewall policy using an abstract format to perform queries and figure out actual firewall behavior by evaluating the action it would enforce instead of trying the connections. Firewall queries can be considered questions concerning firewall behavior [16]. Examples of questions of interest to administrators are: “Which clients can access the server s1?” and “Which server is reachable from the Internet?” This query-based approach easily extends to the analysis of firewalls in distributed systems. In fact, firewall questions can be easily extended to more general reachability problems.

Querying a firewall requires an abstract representation of the policy it implements and an abstract representation of the issued question. One major theoretical problem is the query aggregation. In fact, the number of cases to be considered, for instance, to answer previous questions, is too large: For a five-tuple IPv4 packet filter there are 2^{104} different packets, potentially corresponding to cases to consider.[1] It is critical to use IP address ranges instead of single addresses and port intervals, merging adjacent intervals. The aggregation of the results is also complex and computationally expensive, because the union of rectangles is not always a rectangle.

The first tool produced was Fang [18], a simulation-based engine that performs simple query aggregation. Its successor, Firewall Analyzer (formerly known as Lumeta) [19,20], also provided standard queries and import functionalities to automate the analysis and facilitate the job of the administrators. Another early work from Hazelhurst [21] concentrated on a simple query-based analysis.

Liu proposed Structured Firewall Query Language (SFQL) and an associated intrafirewall query engine [22], which he extended to an analysis of corporate networks composed of packet filters with network address and port translation capabilities (Network Address Translation and Network Address and Port Translation) [11]. SFQL is a SQL-like language that permits specifying queries in a compact and familiar syntax. Assuming that D is the field name for the destination address, S for source address, N for the destination port, and P for the protocol type, the following query answers the following question: “Which are the IP (source) addresses of computers that can reach the Web service s_1 available at 10.0.0.1:80/TCP?” Please see the following lines of code:

```
Select S
from firewall
where {S∈all} ∧
{D∈10.0.0.1} ∧
{N∈{80}} ∧
{P∈{TCP}} ∧
{decision=accept}
```

The main limitation of the firewall querying approach is that the questions at issue are selected by administrators who have to identify the meaningful queries, write them correctly, aggregating if needed the results of more queries (similarly to the SQL union clause), and analyze the results, which may be large. However, they “often do not know what to query” [18]. In the literature, only the Firewall Analyzer addressed this problem, proposing a set of standard queries.

Conflict Detection by Anomaly Classification

A different method to identify whether the firewall policy is correct consists of performing an exhaustive analysis of the ACL, to detect all the situations that may be evidence of a misconfiguration. Although the terms *conflict* and *anomaly* are often used synonymously, in this field they have different meanings: a *conflict* is an occurrence that may stop the correct working (having two matching rules in a router that allows for only one matching rule at the time); an *anomaly* is a particular relation between one or more ACL rules that administrators have to consider, because it may be the evidence of specification mistakes, but that is perfectly allowed by the examined control.

Sloman [6,23,24] initially introduced the concept of conflicting policy, but the methods he presented are not directly applicable to firewall policies and in general to all of the low-level configurations. Many seminal articles present solutions for an analysis of packet filtering. First works concentrated on efficient representations of the ACL, because conflicting rules decrease performance or were not allowed in devices owing to the limited computation capabilities.

The approaches are mainly equivalent, even if they use different rule representations. Hazelhurst presented solutions based on binary decision diagrams (BDDs) [21], Hari [25] proposed the use of tries, Baboescu [26], the use of bit vectors, and Srinivasan [27], the Tuple Space Search classification algorithm.

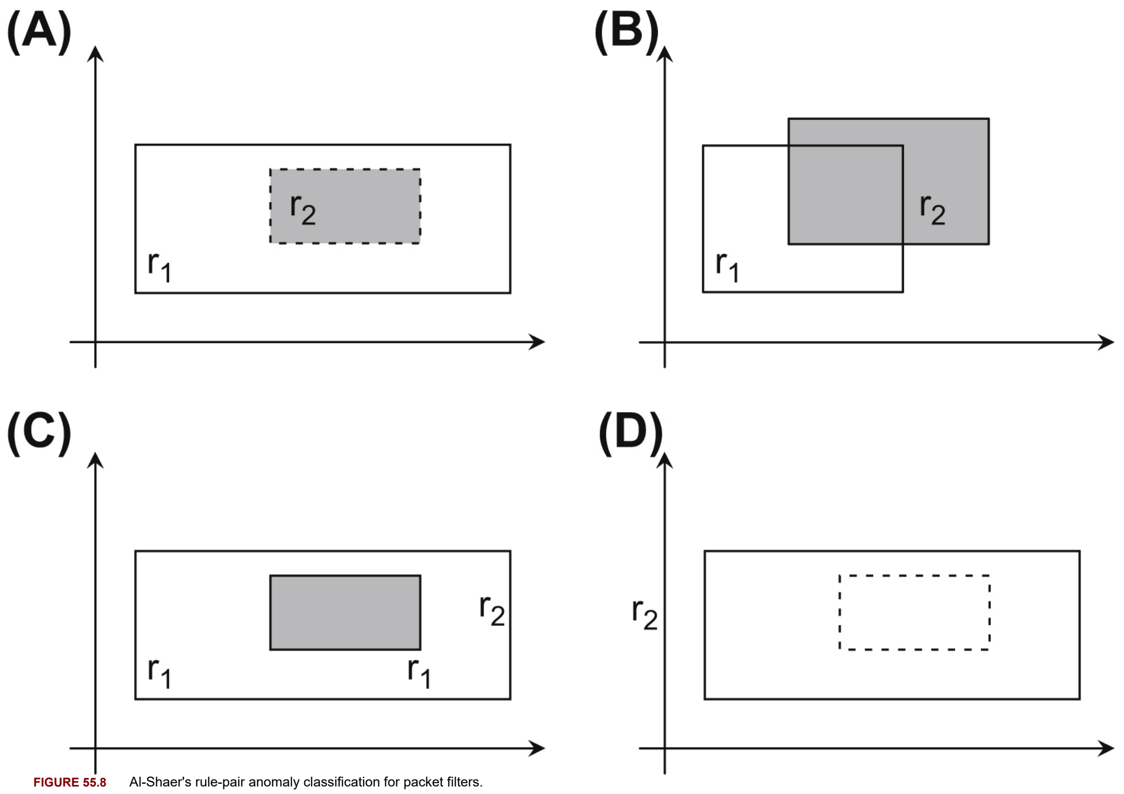


FIGURE 55.8 Al-Shaer's rule-pair anomaly classification for packet filters.

The first formalization of the anomaly concept was proposed by Al-Shaer, who focused on the intrapolicy analysis of packet filters [10]. He introduced the concept of anomaly, defined as "the existence of two or more filtering rules that may match the same packet, or the existence of a rule that can never match any packet," and identified five rule-pair anomaly types: *shadowing*, *correlation*, *generalization*, and *irrelevance*, and presented an algorithm to discover and manage anomalies in ordered rule lists. Given two rules, r_1 and r_2, where r_1 is the highest-priority rule, the rule-pair anomalies are:

1. *Shadowing anomaly*: r_2 is shadowed when r_1 matches all of the packets that r_2 matches, so that r_2 will never be activated (Fig. 55.9B);
2. *Correlation anomaly*: r_1 and r_2 are correlated if (1) they enforce different actions; (2) there exists some packet matching both r_1 and r_2; and (3) there exists some packet matching r_1 but not r_2, and vice versa (Fig. 55.8B);
3. *Generalization anomaly*[2]: r_2 is a generalization of r_1 if (1) they enforce different actions; and (2) all the packets matching r_1 also match r_2, but not the contrary (Fig. 55.8C);
4. *Redundancy anomaly*: r_2 is redundant if r_1 matches the same packets and enforces the same action as r_2, so the removal of r_2 will not change the policy behavior (Fig. 55.9A);
5. *Irrelevance anomaly*: A rule is irrelevant if does not match any packet that could pass through the firewall. It does not concern relations between rules, but rather between a rule and the enforcing device.

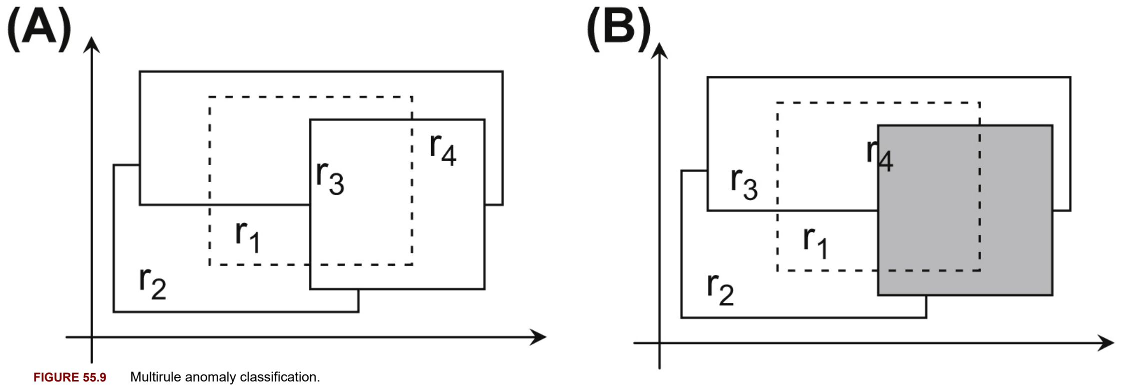


FIGURE 55.9 Multirule anomaly classification.

For instance, with reference to Table 55.1, r_1 is a generalization of r_2, and r_2 shadows r_3, makes r_4 redundant, and is correlated to r_5. Finally, r_6 is irrelevant if the traffic from IP address 5.5.5.5 cannot reach the firewall interfaces.

Al-Shaer's classification is limited because it detects only anomalies in rule pairs; anomalies that arise when more rules are considered are not discussed. Basile [28,29] generalized Al-Shaer's classification to multirule anomalies: that is, anomalies that involve more than two rules. The firewall policies are categorized as *conflicting* when at least two rules contradict each other (that includes the correlation, generalization, and shadowing anomalies), and suboptimal when the removal of one or more rules does not affect the behavior of the firewall (that includes shadowing and redundancy). Suboptimality is caused by *hidden rules* (rules that are never activated regardless of the number of rules that hide them). Hidden rules are further classified as *general redundant* if all of the rules hiding them enforce the same action (as presented in Fig. 55.9A) and *general shadowed*, if at least one enforces a different action (as presented in Fig. 55.9B).

A More In-Depth View of Packet Filter Conflict Analysis

Al-Shaer's classification is the starting point for several works that tried to improve the identification of the anomalies using different techniques. Bouhoula [30] used rule field logical relations permitting the analysis of different firewall rule formats, not only the five-tuples. Thanasegaran [31] used bit vectors that support the detection of rule-pair anomalies more efficiently and the definition of new fields, but they fail to express conditions effectively on ordered fields (ranges of port numbers). Ferraresi [32] presented a slightly alternative conflict classification and a proven correct algorithm that produces a conflict-free rule list. Other works propose rule set optimization by redundancy removal. Gouda [33] provided algorithms to verify the consistency, completeness, and compactness of packet filters, and Liu [34] introduced techniques to detect redundancy based on Firewall Decision Diagrams. Alfaro [35] proposed a set of algorithms to remove anomalies between packet filters and network intrusion detection systems in distributed systems, implemented in the Mirage tool [36]. Hu [37] also introduced an ontology-based anomaly management framework that delegates set operations to BDDs. A completely different approach was presented by Bandara [38], who used argumentation logic and achieved excellent performance. A complementary approach is represented by Liu's Firewall Compressor [39], which minimizes the ACL size by manipulating the specified rules to obtain an equivalent ACL with a minimal number of rules. Redundant and shadowed rules disappear but correlated rules are not examined. The compressed ACL serves only deployment purposes because it is no longer manageable by the administrators.

Stateful Firewall Analysis

Anomaly analysis of stateful firewalls is a less explored field. It is difficult to share the optimism of Buttyàn [40], who stated that “stateful is not harder than stateless.” Their scenario is oversimplified because they added one single field to the five-tuple decision space to describe all possible states. The stateful case is harder for at least two reasons: There are new anomalies that do not appear in the stateless case, and it is computationally more complex because many fields need to be considered. Gouda and Liu [41] presented a model of stateful firewalls that maps the stateful filtering functionalities to the packet filter case to use available detection algorithms. For this reason, they model stateful firewalls using two components: the stateful section and the stateless section. The stateful section inspects the transport headers and maintains a *state table* that associates each of the connections observed with a set of Boolean variables: for example, the established state for TCP connections. A set of (hard-coded) *stateful rules* regulates how the state table is updated according to previous states and received packets. The stateless section is simply a packet filter that includes predicates over the Boolean variables in the state table; the anomalies found are Al-Shaer's. Cuppens [42] extended Al-Shaer’s classification, adding stateful conflicts, such as situations connected to the specific protocol state machines: for example, rules that deny TCP setup and termination for allowed connections, or rules that block allowed, related FTP connections.

Table 55.1

Sample Filtering Policy With Anomalies

	Priority	Source IP	Source Port	Destination IP	Destination Port	Protocol	Action
r_1	1	10.0.0.64/28	Any	1.1.1.64/28	80	TCP	DENY
r_2	2	10.0.0.0/24	Any	1.1.1.0/24	80	TCP	ALLOW
r_3	3	10.0.0.2	Any	1.1.1.1	80	TCP	DENY
r_4	4	10.0.0.250	Any	1.1.1.1	80	TCP	ALLOW
r_5	5	10.0.0.16/24	Any	1.1.1.16/24	80	TCP	DENY
r_6	6	5.5.5.5	Any	6.6.6.6	Any	Any	ALLOW

IP, Internet Protocol; *TCP*, Transmission Control Protocol.

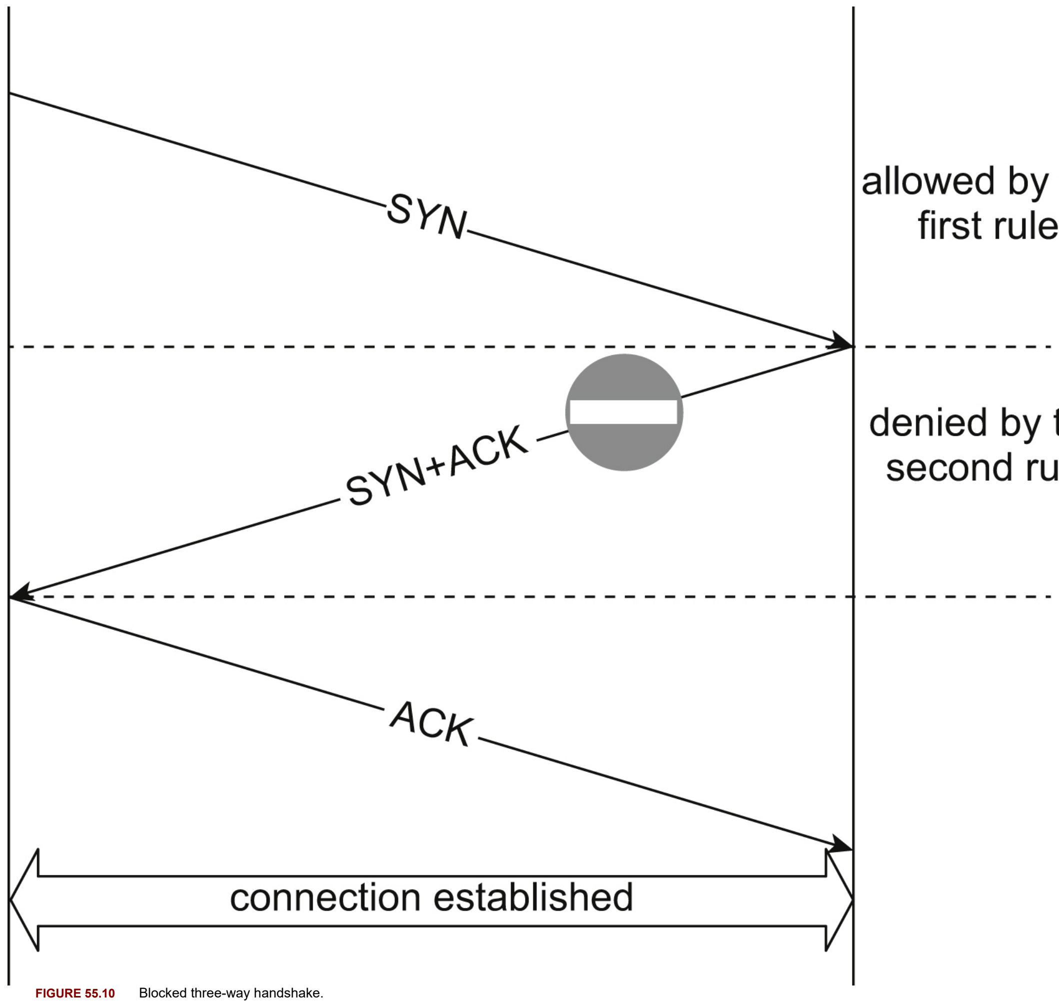


FIGURE 55.10 Blocked three-way handshake.

Finally, there is currently no extensive work to detect anomalies in application firewalls, if we exclude the effort to validate the factory-provided regular expressions used to avoid attacks in Web application firewalls. An example representative of the complexity of the analysis in this case is described by the following rules (the first rule is used to avoid denial of service attacks):

1. Deny packets with the SYN and ACK set to true from the external nodes;

2. Allow TCP connections from the internal node having IP 1.1.1.1 to the external server 2.2.2.2 (in the Internet).

These rules are apparently disjointed because one poses a condition with different values of IP addresses and TCP flags. However, according to the TCP specification, the three-way handshake cannot be terminated; the result is thus that the connection from 1.1.1.1 to 2.2.2.2 is forbidden (Fig. 55.10).

Interfirewall Analysis

Al-Shaer [43] also provided the first classification of anomalies in distributed systems. He considered the case of two serially connected stateless firewalls, named, respectively, upstream and downstream firewalls. Assuming *fwu* is the upstream firewall and *fwd* is the downstream firewall, four anomaly types are identified:

- *Shadowing anomaly*: occurs if *fwu* blocks traffic accepted by *fwd* (Fig. 55.11A);
- *Spuriousness anomaly*: occurs if *fwu* permits traffic denied by *fwd* (Fig. 55.11B);
- *Redundancy anomaly*: occurs if *fwu* denies traffic already blocked by *fwu* (Fig. 55.11C);
- *Correlation anomaly*: occurs when a rule *ru* in *fwu* and a rule *rd* in *fwd* are correlated (Fig. 55.11D).

Another work that addresses conflict analysis in distributed systems is presented in Gouda and Lin [41]. This work defines for every pair of nodes in the network two separate end-to-end policies: the allowed packets, named accept property, and the denied packets, named discard property. Based on an abstract representation of the firewall ACL, the Firewall Decision Diagram [31], they calculate the effects on the communication between two nodes n_1 and n_2, by superposing the actions taken by all of the cascading firewalls encountered in the path between n_1 and n_2. Then, they compare the results with the accept and discard properties; a conflict arises when an accept or discard property is not satisfied. Finally, a tool able to detect Al-Shaer's anomalies in distributed systems is FIREMAN, which also checks whether a distributed policy complies with an end-to-end policy [42].

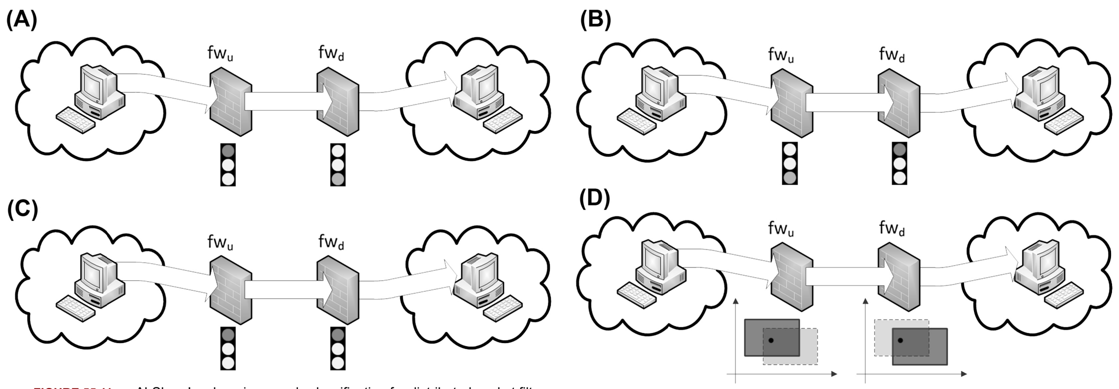


FIGURE 55.11 Al-Shaer's rule-pair anomaly classification for distributed packet filters.

Channel Protection Conflicts

The configuration of secure channels is also an error-prone activity; thus administrators need assistance and conflict detection mechanisms. A few differences can be highlighted with respect to the filtering case: The number of rules is usually orders of magnitude less than in the firewall case. However, they enforce more complex actions, they may enforce more than one action in case of multiple matches, and they have more complex dependencies on the actual distributed system topology.

Different technologies are available to protect channels, and these technologies work at different levels of the ISO/OSI stack. The most well-known solutions are Internet Protocol Security (IPsec), which works at the network layer, and the Transport Layer Security (TLS) protocol, which works up to the transport layer. IPsec allows the creation of secure communication channels between two end points. IPsec can enforce authentication and integrity of IP payload and header using the Authentication Header (AH) protocol, and confidentiality, authenticity, and integrity of the IP payload using the Encapsulating Security Payload (ESP). These end points can be the communicating peers (client and server or two peers) or two gateways used to allow two subnets or two offices to be securely connected over a public/insecure network (the Internet) and to establish a virtual private network (VPN). Also, TLS is used as the base protocol to create VPNs; when this technology is employed, the terms *OpenVPN* or *clientless VPN* are used. These techniques have many similarities from the configuration point of view; therefore they share the same anomalous situations. However, only IPsec VPNs have received attention from researchers, nevertheless, the results for IPsec VPNs are easily extendable to the OpenVPN scenario.

Internet Protocol Security Intrapolicy Conflict Detection

The IPsec configuration rules (the security policy) are stored in the local Security Policy Database (SPDB). These rules select the traffic to be protected by means of (condition clause) predicates on the source and destination IP addresses, IP protocol type fields, and traffic direction (in–out). Three types of anomalies can be found in this scenario: (intrapolicy) local anomalies (intrapolicy) topology-dependent local anomalies, and interpolicy anomalies.

Local anomalies are analogous to the packet filter scenario (see checklist: "An Agenda for Action for Developing Security Policies for Packet Filtering") and they can be identified using the same techniques (only a simple adaptation is needed). It is not surprising that Al-Shaer [10] proposed the application of its classification for packet filters for intra-IPsec policy analysis [44], extending an early work from Fu et al. [45].

An Agenda for Action for Developing Security Policies for Packet Filtering

IPsec can perform host-based packet filtering to provide limited firewall capabilities for end systems. You can configure IPsec to permit or block specific types of unicast IP traffic based on source and destination address combinations and specific protocols and specific ports. For example, nearly all of the systems illustrated in the following checklist can benefit from packet filtering to restrict communication to specific addresses and ports. You can strengthen security by using IPsec packet filtering to control exactly the type of communication that is allowed between systems (check all tasks completed):

_____1. The internal network domain administrator can assign an Active Directory-based IPsec policy (a collection of security settings that determines IPsec behavior) to block all traffic from the perimeter network (also known as a demilitarized zone or screened subnet).

_____2. The perimeter network domain administrator can assign an Active Directory-based IPsec policy to block all traffic to the internal network.

_____3. The administrator of the computer running Microsoft SQL Server on the internal network can create an exception in the Active Directory-based IPsec policy to permit SQL protocol traffic to the Web application server on the perimeter network.

_____4. The administrator of the Web application server on the perimeter network can create an exception in the Active Directory-based policy to permit SQL traffic to the computer running an SQL server on the internal network.

_____5. The administrator of the Web application server on the perimeter network can also block all traffic from the Internet, except requests to TCP port 80 for the HTTP and TCP port 443 for HTTP Secure Protocol (HTTP over Secure Sockets Layer/TLS Protocol), which are used by Web services. This provides additional security for traffic allowed from the Internet in case the firewall was misconfigured or compromised by an attacker.

_____6. The domain administrator can block all traffic to the management computer but allow traffic to the perimeter network.

The anomaly types are the same: Shadowed, redundant, correlated, and generalized rule pairs can be found in an IPsec SPDB. However, the effort required for the analysis is greater. The main difference is that the actions that can be enforced using IPsec are more complex, because confidentiality, authenticity, and integrity (of the IP payload only using ESP, or IP payload and header using AH) can be selected. Moreover, cryptographic algorithms need to be evaluated and compared. For instance, is it better to protect a channel using "ESP with hash message authentication code (HMAC)-SHA1 and Advanced Encryption Standard (AES) 256" or a channel using "ESP with reserve component 2128 encapsulated in AH with HMAC-MD5"? The answer is not easy and depends on the requirements specified at the business level.

Together with the previous anomalies, other types of anomaly appear from the analysis of a local SPDB, with the intrapolicy channel overlapping and multitransform anomalies. These anomalies depend on the possibility of applying in the same SPDB more than one transformation; choosing the correct order, modes, and algorithms becomes crucial.

Overlapping occurs when an SPDB contains more than one rule with the same source *s* and destination *d* that uses different tunneling devices, g_1 and g_2 (Fig. 55.12). For instance, if the SPDB contains the following rules:

- (short tunnel) tunnel to g_1 with protection p_1
- (long tunnel) tunnel to g_2 with protection p_2

and is applied in this order, the following communications are performed:

1. $s \rightarrow g_2$ protected with p_1 and p_2
2. $g_2 \rightarrow g_1$ protected with p_1 (p_2 is removed at g_2)
3. $g_1 \rightarrow d$ with no protection

Therefore, the rule order matters. In fact, if the rules are applied in the opposite order, the communications are, as expected:

1. $s \rightarrow g_1$ protected with p_1 and p_2
2. $g_1 \rightarrow g_2$ protected with p_2 (p_1 is removed at g_1)
3. $g_2 \rightarrow d$ with no protection

This anomaly can also occur with a transport transform (instead of the long tunnel) followed by a tunnel (short tunnel). Applying more than one transformation increases the risk of reducing the protection level. In fact, it is not always true that the combination of more transformations results in a stronger protection. Moreover, because the application of each transformation requires computational resources, using more than one transformation must be justified from the security point of view. The multitransform anomaly occurs when a weaker protection is applied after a stronger one. For instance, applying ESP after AH reduces the overall security because ESP transport does not provide IP header protection. On the other hand, applying AH after ESP is often justified to preserve the header integrity. In addition, a multitransform anomaly may also occur when the increase of cost–benefit in terms of security is not justified, as when one is applying ESP with AES 256 after having applied ESP with AES 128. Resolving these anomalies is delicate. Every case needs to be considered individually, because the "protection strength" needs to be measured and compared with the performance loss, but an official measure does not exist and every organization may have its own evaluation criteria.

Internet Protocol Security Interpolicy Conflict Detection

Together with the explicit deny action, communications can also be blocked in case of misconfigurations or if the peer authentication fails, if the *security association* defining the algorithms and keys to use to protect the channel is not available and nonnegotiable, or if there is more than one security association when a unique security association is expected. Therefore, the types of anomaly are analogous to those presented in distributed systems. In fact, IPsec communications can be shadowed (block traffic already blocked by the upstream device) and spurious (allow traffic already blocked by the upstream device).

In addition, it is possible to highlight *interpolicy channel overlapping*, presenting the same mechanism as the intrapolicy case but involving more than two elements. To protect communication between Source *s* and Destination *d*, there are three gateways, g_1, g_2, and g_3, which are encountered in this order by packets from *s* to *d* (Fig. 55.13). The following policy is enforced:

- *s* creates a secure channel (transport mode) to g_2 with protection p_1.
- g_1 creates a tunnel to g_3 with protection p_2.

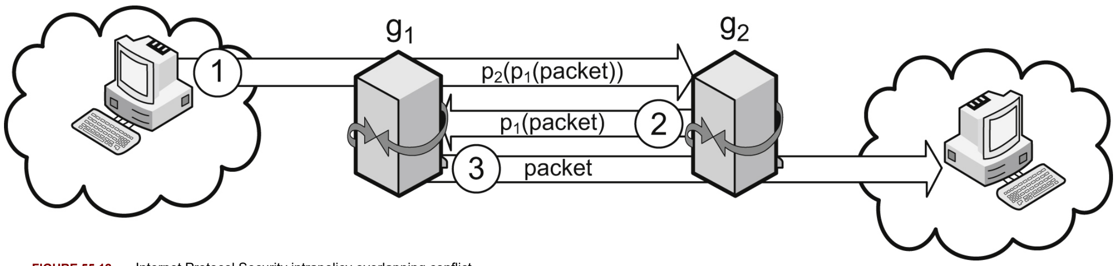


FIGURE 55.12 Internet Protocol Security intrapolicy overlapping conflict.

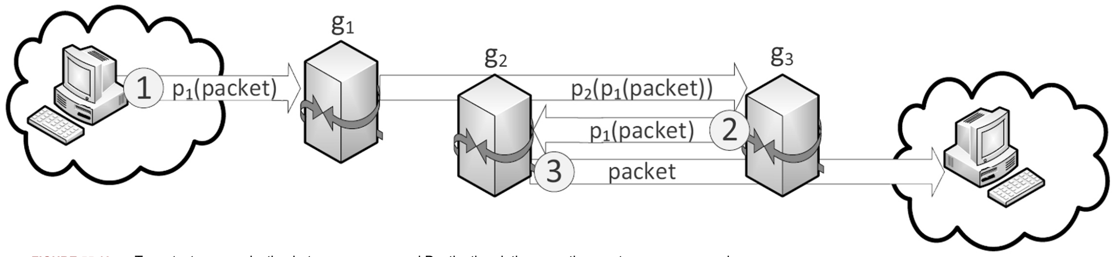


FIGURE 55.13 To protect communication between sources and Destination d, there are three gateways: g_1, g_2, and g_3.

Therefore, the resulting communications are:

1. $s \rightarrow g_1$ protected with p_1
2. $g_1 \rightarrow g_3$ protected with p_1 and p_2
3. $g_3 \rightarrow g_2$ protected with p_1 (p_2 is removed at g_3)
4. $g_2 \rightarrow d$ with no protection

IPsec devices are managed by different people who often work in different units; therefore interpolicy conflicts are relatively frequent. Sun et al. [46] proposed a new architecture that stores all of the IPsec policy centrally and offers access via a manager that also enforces an access control policy. In addition, the proposed work aims to manipulate the SPDB automatically to avoid or recover some of the anomalies presented before.

6. Semantic Web Technology for Conflict Detection

The term *Semantic Web* refers to both a vision and a set of technologies. The vision is articulated, in particular by the World Wide Web Consortium (W3C), as an extension to the current idea of the Web in which knowledge and data could be published in a form easy for computers to understand and reason with. Doing so would support more sophisticated software systems that share knowledge, information, and data on the Web just as people do by publishing text and multimedia. Under the stewardship of the W3C, a set of languages, protocols, and technologies has been developed to realize this vision partially, to enable exploration and experimentation, and to support the evolution of the concepts and technology. The current set of W3C standards is based on Resource Description Framework (RDF) [47], a language that provides a basic capability of specifying graphs with a simple interpretation as a *semantic network* and serializing them in XML and several other popular Web systems (e.g., JavaScript Object Notation). Because it is a graph-based representation, RDF data are often reduced to a set of triples in which each represents an edge in the graph or, alternatively, a binary predicate. The Web Ontology Language (OWL) [48] is a family of knowledge representation languages based on Description Logic (DL) [49] with a representation in RDF. OWL supports the specification and use of ontologies that consist of terms representing individuals, classes of individuals, properties, and axioms that assert constraints over them.

The use of OWL to describe and verify the properties of policies offers several important advantages that are particularly critical in distributed environments possibly involving coordination across multiple organizations. First, most policy languages define constraints over classes of targets, objects, actions, and other kinds of information (location). A substantial part of the development of a policy is often devoted to the precise specification of these classes. This is especially important if the policy is shared among multiple organizations that must adhere to or enforce the policy, even though they have their own native schemas or data models for the domain in question. The second advantage is that OWL's grounding in logic facilitates the translation of policies expressed in OWL to other formalisms, either for further analysis or for execution.

Semantic Web technology offers an extensive collection of tools that can be used to model and represent policy conflicts. Several approaches can be adopted, with different profiles in terms of abstractness and efficiency. There are three main approaches to discover conflicts: standard reasoners, ad hoc reasoning methods, and rule-based inferencing. In the next subsections we characterize these three alternatives.

Use of Standard Reasoners

The standard reasoner is one of the core elements of an ontology-based system. Starting from the information contained in the ontology described in OWL, it is able to perform several tasks (it is able to check the consistency and validity of the ontology, classify its information, answer queries, and generate inferences) using a variety of techniques derived from the work of the artificial intelligence community.

In particular, standard DL reasoning performed with regard to a formal ontology can check complex consistency constraints in the model. Such constraints are different from the usual ones from database and Unified Modeling Language (UML)-like systems. In OWL-DL, which is the portion of OWL restricted to the expressivity of DL, we can express:

- Constraints on properties, domains, and ranges
- Definitions of concepts (classes) in terms of relationships with other elements
- Boolean operations on classes

One of the main differences between DL-based schema definitions and UML or Entity relationship definitions concerns the constraints on property domains and ranges. Properties can be defined in a general way, and their behavior in terms of range type can be precisely described while refining the ontology concepts. This promotes the definition and reuse of high-level properties without losing the ability to force precise typing. In this way, the resolution of some conflicts can be explicitly expressed in the policy without the need to rely on an external conflict resolution option or implicit priorities. For instance, authorizations associated with a subject administrator can be denoted as having higher priority, dominating in possible conflicts with other authorizations.

Ad Hoc Reasoning Methods

Standard *DL* reasoners can answer complex questions and verify structural and nonstructural constraints. Furthermore, DL-based language expressiveness often exceeds classical solutions (such as UML for design and SQL for data storage models). This supports the description and verification of more complex structural constraints. For example, if we consider the approach used in Finin et al. [50], in which the roles are represented as individuals of the class Role, the *roleHierarchy*: *Role* → *Role* property[3] is used to connect each role to its direct subroles and the *canHaveRole+*: *Identity* → *Role* property is used to represent the roles that each identity (user) can activate, directly or indirectly, thanks to the presence of positive role authorizations. Thus, *roleHierarchy* (r_1, r_2) means that role r_1 is a superrole of r_2. Its transitive closure *canBe+*: *Role* → *Role* can be used to identify all the direct or indirect subroles. The subrole (as well as its inverse superrole) relationship is not a containment and does not define a taxonomy on identities (a superrole of Role R is intended to be more privileged than R and is available to a more restricted set of identities).

With these tools it is possible, for instance, to offer an immediate management of SoD constraints. The user role assignment relation is represented using role authorizations, which specialize authorizations with the specification of the role that the principal is allowed or forbidden to assume. An SoD constraint between Role r_1 and r_2 then can be expressed using a negative role authorization r_{auth} that forbids Role r_1 from enacting Role r_2. SoD constraints are enforced both at the role hierarchy level (in this way we directly prevent a Role r_1 from being declared superrole of another Role r_2, such that r_1 and r_2 are in an SoD constraint) and at the user hierarchy level (to prevent two Roles r_1 and r_2 from being assigned to a user, directly or indirectly, that are involved in an SoD constraint).

To show a more concrete example, we assume that class *RoleAuthorization* ⊆ Authorization represents the role authorizations, and properties *grantedTo*: *RoleAuthorization* → *Principal* and *enabledRole*: *RoleAuthorization* → *Role* are used to represent, respectively, the role enabled by the role authorization and the principal to which the role is assigned. To keep track of all SoD conflicts on roles, we can define a class *SoDOnRole* ⊆ *Role*. SoD constraints on the role hierarchy can be expressed adding to the ontology the following set of axioms:

$$\forall auth \in RoleAuthorization: sign(auth, -),$$

$$grantedTo(auth, r_1), enabledRole(auth, r_2)$$

$$SoDOnRole \equiv \exists canBe+.\{r_1\} \cap \exists canBe+.\{r_2\}$$

The interpretation of these axioms is that for each negative role authorization, there is an instance in class SoDOnRole only if there exists a single role that belongs to r_1 and to r_2. We can thus enforce the SoD at the role hierarchy level simply by adding the axiom *SoDOnRole* ⊆ ⊥ to the ontology, which declares as consistent the ontology only if the class is empty.

In a way similar to what we have done for the identification of SoD conflicts at the role hierarchy level, we can define a class *SoDOnUser* ⊆ *Identity* that keeps track of the conflicts on the user hierarchy. We then express SoD constraints using the following axioms:

$$\forall auth \in RoleAuthorization: sign(auth, -),$$

$$grantedTo(auth, r_1), enableRole(auth, r_2)$$

$$SoDOUser \equiv \exists canHaveRole$$

$$+.\{r_1\} \cap \exists canHaveRole + .\{r_2\}$$

and to enforce the SoD constraints we simply have to add to the ontology the axiom $SoDOnUser \subseteq \bot$.

This approach can easily be extended to handle other kinds of SoD constraints, such as *Permission-based SoD* (which requires that no user be allowed to do both Actions a_1 and a_2) or *Object-based SoD* (which requires that no user can access both Resources res_1 and res_2). However, DL systems, as well as *Semantic Web* tools in general, are designed and implemented with a focus on knowledge management services, such as knowledge integration, schema matching, and instance retrieval. Such a specialization raises some limitations on the use of pure DL reasoning in real scenarios, in which reasoning must be carried out on a well-defined and complete description of a closed system.

Closed World Assumption

Closed World Assumption (CWA) reasoning is a generally accepted requirement in model-driven systems. Conversely, DL reasoners usually work under the Open World Assumption. This means that the facts asserted in the model (about the layout topology or the authorization policies) are not assumed to be complete. Obviously, this can become a problem if model characteristics are described in terms of the existence of some properties or some relationships between model elements.

Reasoning on Complex Property Paths

Reasoning on complex property paths (commutatively of nontrivial graphs), creates uncertainty of the formal logics the language is based on. Checking the closure of complex paths is beyond the expressive power of classical database systems, but unfortunately it is sometimes necessary to check structural constraints. This is the case, for example, for the consistency loop in Fig. 55.14, which states that:

an authorization of executing an action must be assigned to a resource (Database) that runs on a system (DBMS) compatible with the action type (Select, Create, Delete).

Unique Name Assumption

Unique Name Assumption is a commonly accepted assumption in most model-driven tools. It consists of assuming that different names will always denote different elements in the model. This is usually not true in DL reasoners because of the essential nature of knowledge integration problems. In fact, in the Semantic Web scenario, different authors may describe the same entities (both shared conceptualizations and physical objects), assigning a new name, generally in the form of a Uniform Resource Identifier, defined independently from other users.

These properties must be considered carefully when applying DL and Semantic Web tools to the detection of policy conflicts. The obstacles introduced can be solved as long as attention is paid to them. Misbehaviors of the system can be observed otherwise.

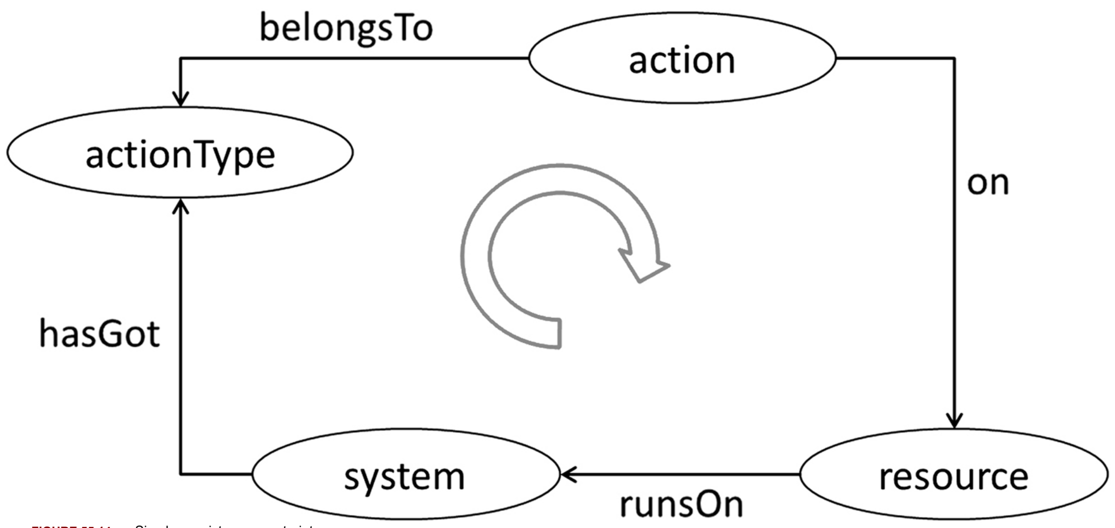


FIGURE 55.14 Simple consistency constraint.

Rule-Based Inferencing

Rule inference reasoning is widely used in knowledge management systems. Some combinations of theorem-proving systems (such as DL ones) and rule inference systems have been proposed to address some limitations of decidable theorem-proving systems.

Semantic Web Rule Language (*SWRL*) is the W3C standard proposal for integrating rule-based inferencing into systems that represent knowledge as a set of RDF triples and introducing some limitations to the use of the rules, to preserve joint system decidability. In a real scenario, the main advantage of combining rules and classical theorem-proving systems is the support for complex property chains. In fact, even if some *OWL* profiles introduce the support for the chaining of properties (aka roles in DL terminology), this may not be sufficient to express some complex topological properties. For example, the simple consistency loop shown in Fig. 55.14 is not enforceable at the schema level using only DL axioms. To perform a consistency check on Fig. 55.14, a simple SWRL rule like this one is needed:

$on(?a_1, ?r_1), belongsTo(?a_1, ?act_1), runsOn(?r_1, ?s_1), hasGot(?s_1, ?act_2), differentFrom(?act_1, ?act_2) \rightarrow Error(?a_1, Error)$

This SWRL rule verifies whether there is an Action a_1 belonging to Action type act_1 that is applied to Resource r_1, which runs on Service s_1, which has an Action type act_2; if act_1 and act_2 are verified to be incompatible (using the *differentFrom* predicate), an instance of class Error is created, recording a_1. Essentially, using this rule we can verify that all the *system* instances on which the action is granted are compatible with the *actionType*. Violations are recorded into Error.

As a technical note, we observe that as a general outcome of the adoption of the *Open World Assumption*, even if we could enforce the existence of the loops, we would not be able to require that such loops be explicitly stated into the assertional part of the semantic model (*A-box*). At the opposite end, owing to decidability issues (DL safe rules), the rule-based component of the language operates in a kind of CWA limited to the nodes. This means that a forward chaining rule can be triggered by any property derived by the reasoning, but involving only nodes that are explicitly named in the A-box. Then, we can operate only on nodes and properties explicitly stated in the semantic model. Furthermore, rules can freely combine as antecedent triple patterns to capture complex topological structures, and this solves the lack of complex property chains of *DLs*. This means that we can check for loops, or for the absence of loops, by adding custom rules to the ontology. However, SWRL safe rules can consume only positive knowledge, so they can be used directly to detect errors that consist of the existence of some structure in the ontology: that is, the existence of a loop.

Semantic Web technology offers an interesting potential for detecting conflicts in a variety of settings. Integration with a rich environment of tools, open source and commercial, together with the increasing familiarity that users are acquiring with them, make this option particularly interesting for the realization of sophisticated conflict-detection solutions. These approaches support the flexible definition and identification of conflicts, going beyond the classifications introduced in this chapter and adapting the model to the specific requirements of every application scenario.

7. Summary

The detection and management of conflicts in security policies is an important topic for both the research and industrial communities. The chapter was not exhaustive in its treatment of the topic, although it is extensive. The goal was to focus on the detection of conflicts, considering abstract and executable policies, and illustrating in greater detail the detection of policy conflicts in computer networks, which is the area that sees the greater industrial support. Support in industrial products can be expected to appear in the near future for security policies in other scenarios and at a variety of abstraction levels. The discussion of Semantic Web technology has shown how this family of tools can be applied to this task, offering a strategy that can be particularly interesting for deployment in real systems.

We expect that conflict detection techniques will become common components of tools for the design and configuration of security. The Policy and Security Configuration Management project described in Chapter 26 aims to realize a policy-based security management; it represents an interesting example of such a system.

Finally, let us move on to the real interactive part of this chapter: review questions/exercises, hands-on projects, case projects, and the optional team case project. The answers and/or solutions by chapter can be found in the Online Instructor's Solutions Manual.

Exercise

Problem

What is meant by the phrase "where technically feasible"?

Hands-on Projects

Project

What is meant by the phrase "reasonable business judgment"?

Case Projects

Problem

What is meant by *data*, *documents*, *documentation*, *logs,* and *records*? What are the differences between these terms?

Optional Team Case Project

Problem

What are some sample security policy test procedures?

Acknowledgments

This work was partially funded by the European Community in the scope of the research project PoSecCo (Project No. 257129), under the Information and Communication Technologies theme of the Seventh Framework Program for R&D (FP7). The work by Stefano Paraboschi was partially supported by the PRIN 2008 project PEPPER (2008SY2PH4) and the PRIN 2010-11 Project GenData-2020.

References

[1] Anderson A. eXtensible Access Control Markup Language (XACML), Identity. 2006. http://www.oasis-open.org/committees/xacml/.

[2] Jajodia S, Samarati P, Subrahmanian V.S. A logical language for expressing authorizations. In: *Proceedings of the 1997 IEEE Symposium on Security and Privacy (SP '97)*. Washington, DC, USA: IEEE Computer Society; 1997.

[3] Damianou N, Dulay N, Lupu E, Sloman M. The ponder policy specification language. In: *POLICY '01 Proceedings of the International Workshop on Policies for Distributed Systems and Networks*. London, UK: Springer-Verlag; 2001.

[4] Kagal L, Finin T, Joshi A. A policy language for a pervasive computing environment POLICY '03. In: *Proceedings of the 4th IEEE International Workshop on Policies for Distributed Systems and Networks*. Washington, DC, USA: IEEE Computer Society; 2003.

[5] Tonti G, Bradshaw J.M, Jeffers R, Montanari R, Suri N, Uszok A. Semantic web languages for policy representation and reasoning: a comparison of KAoS, Rei, and Ponder. In: Fensel D, Sycara K.P, Mylopoulos J, eds. *International Semantic Web Conference*. Springer; 2003:419–437.

[6] Lupu E, Sloman M. Conflicts in policy-based distributed systems management. *IEEE Trans. Software Eng*. 1999;25(6):852–869.

[7] Coward N, Yoshida Y. Java servlet specification version 2.4. *Tech. Rep*. 2003.

[8] Casalino M, Thion R, Hacid M.S, Fischer-Hbner S, Katsikas S, Quirchmayr G. Access control configuration for J2EE web applications: a formal perspective. In: Fischer-Hbner S, Katsikas S, Quirchmayr G, eds. *Trust, Privacy and Security in Digital Business*. *Lecture Notes in Computer Science*. vol. 7449. Berlin Heidelberg: Springer; 2012:30–35.

[9] Rubinger A, Burke B, Monson-Haefel R. Enterprise JavaBeans 3.1. In: *Java Series*. O'Reilly Media, Incorporated; 2010.

[10] Al-Shaer E, Hamed H. Modeling and management of firewall policies. *IEEE Trans. Network Serv. Manage*. 2004;1(1):2–10.

[11] Khakpour A.R, Liu A.X. Quantifying and querying network reachability. In: *Proc. of the 2010 IEEE 30th Int. Conf. on Distributed Computing Systems, Washington, DC, USA*. 2010:817–826.

[12] Taylor D. Survey and taxonomy of packet classification techniques. *ACM Comput. Surv*. 2005;37(3):238–275.

[13] Wool A. A quantitative study of firewall configuration errors. *Computer*. 2004;37:62–67.

[14] Wool A. Trends in firewall configuration errors: measuring the holes in swiss cheese. *IEEE Internet Comput*. 2010;14:58–65.

[15] Insecure.Com LLC: Network mapper. http://nmap.org/.

[16] The Hachers Choice, Amap. http://thc.org/thc-amap/.

[17] T.N. Security, Nessus vulnerability scanner. http://www.tenable.com/products/nessus/nessus-product-overview.

[18] Mayer A, Wool A, Ziskind E. Fang: a firewall analysis engine. In: *Proc. of the 2000 IEEE Symposium on Security and Privacy, Washington, DC, USA*. 2000:177–187.

[19] Wool A. Architecting the lumeta firewall analyzer. In: *Proc. of the 10th Conference on USENIX Security Symposium*. vol. 10. 2001:7 Berkeley, CA, USA.

[20] Mayer A, Wool A, Ziskind E. Offline firewall analysis. *Int. J. Inf. Secur*. 2006;5(3):125–144.

[21] Hazelhurst S, Attar A, Sinnappan R. Algorithms for improving the dependability of firewall and filter rule lists. In: *Proc. of the 2000 Int. Conf. on Dependable Systems and Networks, Washington, DC, USA*. 2000:576–585.

[22] Liu A.X, Gouda M.G. Firewall policy queries. *IEEE Trans. Parallel Distrib. Syst*. 2009;20:766–777.

[23] Moffett J.D, Sloman M.S. Policy conflict analysis in distributed system management. *J. Organ. Comput*. 1993;4(1):1–22.

[24] Sloman M. Policy driven management for distributed systems. *J. Network Syst. Manage*. 1994;2(4):333–360.

[25] Adiseshu H, Suri S, Parulkar G.M. Detecting and resolving packet filter conflicts. *INFOCOM*. 2000:1203–1212.

[26] Baboescu F, Varghese G. Fast and scalable conflict detection for packet classifiers. *Comput. Networks*. 2003;42(6):717–735.

[27] Srinivasan V, Suri S, Varghese G. Packet classification using tuple space search. In: *Proc. of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication, New York, NY, USA*. 1999:135–146.

[28] Basile C, Cappadonia A, Lioy A. Geometric interpretation of policy specification. In: *IEEE Policy 2008, New York, NY*. 2008:78–81.

[29] Basile C, Cappadonia A, Lioy A. Network-level access control policy analysis and transformation. *IEEE/ACM Trans. Networking*. 2012. .

[30] Benelbahri M, Bouhoula A. Tuple based approach for anomalies detection within firewall filtering rules. In: *ISCC 2007, Aveiro, Portugal*. 2007:63–70.

[31] Thanasegaran S, Yin Y, Tateiwa Y, Katayama Y, Takahashi N. A topological approach to detect conflicts in firewall policies. In: *IPDPS 2009, Rome, Italy*. 2009:1–7.

[32] Ferraresi S, Pesic S, Trazza L, Baiocchi A. Automatic conflict analysis and resolution of traffic filtering policy for firewall and security gateway. In: *ICC '07, Glasgow, Scotland*. 2007:1304–1310.

[33] Gouda M.G, Liu X.Y.A. Firewall design: consistency, completeness, and compactness. In: *Proc. of the 24th Int. Conf. on Distributed Computing Systems (ICDCS '04), Washington, DC, USA*. 2004:320–327.

[34] Liu A.X, Gouda M.G. Complete redundancy detection in firewalls. In: *Proc. of the 19th Annual IFIP WG 11.3 Working Conference on Data and Applications Security*. 2005:193–206.

[35] Alfaro J.G, Boulahia-Cuppens N, Cuppens F. Complete analysis of configuration rules to guarantee reliable network security policies. *Int. J. Inf. Secur*. 2008;7(2):103–122.

[36] Garcia-Alfaro J, Cuppens F, Cuppens-Boulahia N, Preda S. MIRAGE: a management tool for the analysis and deployment of network security policies. In: *Proc. of the 5th Int. Workshop on Data Privacy Management*. 2011:203–215.

[37] Hu H, Ahn G.J, Kulkarni K. Ontology-based policy anomaly management for autonomic computing. In: Georgakopoulos D, Joshi J.B.D, eds. *CollaborateCom*. 2011:487–494.

[38] Bandara A.K, Kakas A.C, Lupu E.C, Russo A. Using argumentation logic for firewall configuration management. *Integr. Network Manage*. 2009:180–187.

[39] Liu A.X, Torng E, Meiners C.R. Firewall compressor: an algorithm for minimizing firewall policies. *INFOCOM*. 2008:176–180.

[40] Buttyan L, Pék G, Thong T.V. Consistency verification of stateful firewalls is not harder than the stateless case. *Infocommun. J.* 2009;54(2–3):1–8.

[41] Gouda M.G, Liu A.X. A model of stateful firewalls and its properties. In: *Proc. of the IEEE Int. Conf. on Dependable Systems and Networks (DSN-05), Yokohama, Japan*. 2005.

[42] Cuppens F, Cuppens-Boulahia N, Garca-Alfaro J, Moataz T, Rimasson X. Handling stateful firewall anomalies. In: *Information Security and Privacy Research—27th IFIP TC 11 Information Security and Privacy Conference, SEC 2012, Heraklion, Crete, Greece, June 4–6, 2012, Proceedings*. vol. 376. 2012:174–186.

[43] Al-Shaer E, Hamed H, Boutaba R, Hasan M. Conflict classification and analysis of distributed firewall policies. *IEEE JSAC*. 2005;23(10):2069–2084.

[44] Hamed H, Al-Shaer E, Marrero W. Modeling and verification of IPsec and VPN security policies ICNP '05. In: *Proceedings of the 13th IEEE International Conference on Network Protocols*. Washington, DC, USA: IEEE Computer Society; 2005.

[45] Fu Z, Wu S.F, Huang H, Loh K, Gong F, Baldine I, et al. IPsec/VPN security policy: correctness, conflict detection, and resolution. In: *POLICY*. 2001:39–56.

[46] Sun H.M, Chang S.Y, Chen Y.H, He B.Z, Chen C.K. The design and implementation of IPsec conflict avoiding and recovering system. In: *TENCON 2007 – 2007 IEEE Region 10 Conference*. 2007:1–4.

[47] Lassila O, Swick R.R. *Resource Description Framework (RDF) Model and Syntax Specification*. 1999.

[48] Bechhofer S, van Harmelen F, Hendler J, Horrocks I, McGuinness D.L, Patel-Schneider P.F, et al. OWL Web Ontology Language Reference Technical report, W3C. 2004. http://www.w3.org/TR/owl-ref/.

[49] Baader F, Calvanese D, McGuinness D.L, Nardi D, Patel-Schneider P.F. Description logic handbook. In: Baader F, Calvanese D, McGuinness D.L, Nardi D, Patel-Schneider P.F, eds. *Description Logic Handbook*. Cambridge University Press; 2003.

[50] Finin T, Joshi A, Kagal L, Niu J, Sandhu R, Winsborough W, et al. ROWLBAC: representing role based access control in OWL. In: *Proc. of SACMAT*. ACM; 2008.

[1] 32 bits for source and destination IPv4 addresses, 16 bits for ports, and 8 for the protocol type.

[2] In Basile et al. [28] this is named an exception.

[3] The notation "*R: A* → *B*" has to be interpreted according to DL conventions. It states that *A* and *B* are, respectively, the domain and the range of Property *R*, with no further constraints about the functionality or completeness of *R*.