# D4.1
# Policy specification

| | |
|---|---|
| **Project number** | 611458 |
| **Project acronym** | SECURED |
| **Project title** | SECURity at the network EDge |
| **Project duration** | 36 months (1/10/2013–30/9/2016) |
| **Programme** | FP7 (Collaborative Project) |

| | |
|---|---|
| **Deliverable type** | **R** - Report |
| **Deliverable number** | D4.1 |
| **Version (date)** | v1.0 (10/4/2015) |
| **Work package(s)** | WP4 |
| **Due date** | 31/3/2015 – M18 |

| | |
|---|---|
| **Responsible organisation** | POLITO |
| **Editor** | Marco Vallini |
| **Dissemination level** | **PU** - Public |

| | |
|---|---|
| **Abstract** | This document contains the specification of the two mechanisms foreseen by SECURED for defining the user security policies, namely the High-level Security Policy Language (HSPL) and the Medium-level Security Policy Language (MSPL).<br>HSPL is suitable for expressing the protection requirements of typical non-technical users, while MSPL is a lower-level abstraction useful for expressing specific configurations of security controls in a generic format (as such it is more appealing for technical users). |
| **Keywords** | HSPL, MSPL |


**Editor**
Marco Vallini (POLITO)

**Reviewers**
Diego R. Lopez (TID)
Antonio Agustin Pastor (TID)
Antonio Lioy (POLITO) – quality control

**Contributors**
Cataldo Basile (POLITO)
Christian Pitscheider (POLITO)
Fulvio Valenza (POLITO)

**Acknowledgement**
This work was partially supported by the European Commission (EC) through the FP7-ICT programme under project SECURED (grant agreement no. 611458).

# Change Log

| Version | Date | Note | Author |
|---|---|---|---|
| v0.1 | 29.09.2014 | Base version | M.Vallini |
| v0.2 | 13.03.2015 | Final draft ready for internal review | M.Vallini |
| v0.3 | 30.03.2015 | Corrections after internal review, ready for quality control | M.Vallini |
| v1.0 | 10.04.2015 | Quality control | A.Lioy |

## Executive Summary

This deliverable presents the two abstractions defined within the SECURED project to specify the security policy: the High-level Security Policy Language (HSPL) and the Medium-level Security Policy Language (MSPL).

HSPL is the policy language suitable for expressing the general protection requirements of typical non-technical end-users, such as "do not permit access to illegal content" or "block access to peer-to-peer networks". HSPL is an authorization language that follows this form:

```
[ sbj ] act obj [ (field_type,value) ...  (field_type,value) ]
```

Where: `sbj` is the entity who is authorized to perform the authorization action `act` on the resource `obj`. A set of optional `(field_type,value)` pairs can be used to further constrain the resource or the scope of the operation. Another goal of HSPL is to hide configuration details (e.g. encryption algorithm, key length) during the definition of a security policy.

Based on the project use cases and a set of WP4-defined scenarios, SECURED identified a predefined set of subjects, actions, and objects able to cover the user needs (and are reported in this deliverable).

Moreover, to ease the specification process, HSPL policies may be also presented as sets of constrained natural language sentences that can be directly selected by the users (e.g. by using check-boxes) or customized with writing assistant tools. The natural language sentences are not an additional abstraction layer, they are obtained from HSPL as view by means of specific controllers that perform the translation (i.e. following the Model-View-Controller design pattern).

MSPL is the policy abstraction used for expressing specific configurations in a device-independent format. MSPL is an abstract language with statements related to the typical actions performed by various security controls (e.g. matching patterns against packet headers, keeping track of connection status, identifying the MIME type of a payload), but expressed in a generic (PSA-independent) syntax. MSPL is used to describe the configuration settings of a class of security controls.

MSPL is defined by a meta-model that specifies the main concepts (like policies, rules, conditions, and actions), and it is organized by *capabilities*. In this context, capabilities are defined as basic features that can be configured to enforce a security policy (e.g. channel protection, filtering, anti-virus, parental control). This deliverable presents the initial classification of the security capabilities as a UML model that includes the Capability class and several subclasses including Authorization, Authentication, Data Protection, and General Security Requirements capabilities.

Based on capabilities, we group actual configuration languages with similar concepts (e.g. attributes, actions, or condition types) into families that are captured by specific MSPL sub-models built by analysing several languages (e.g. netfilter/iptables) offering the same capability. For instance, MSPL permits the specification of configurations for a general packet filter, or to configure the options of a general anti-virus. This deliverable presents in details the Filtering and Data Protection sub-models. Overall, writing policies in MSPL demands the same security awareness and level of expertise as for specifying the configurations directly in the PSAs. The advantage, however, is that MSPL avoids experts the burden to master several semantically equivalent security controls and syntaxes.

# Contents


SECURED

# 1 Introduction

The design of SECURED meta-models started from the consideration of the current state of the technology. Nowadays the common approach to protect personal devices from Internet threats relies on the installation of appropriate security controls. As widely discussed in previous documents, this approach poses several issues which are addressed by SECURED offloading security tasks to one or more Personal Security Application (PSA) executed on a Network Edge Device (NED).

Each PSA implements some security control(s), configured according to the user security requirements. However the configuration of security applications is complex and not well understood by the majority of end-users (i.e. it typically requires specific skills). To simplify this task, we propose the definition of a user-oriented security policy language. This language offers two levels of abstraction:

- HSPL (High-level Security Policy Language) is suitable for expressing the general protection requirements of typical non-technical end-users, such as “do not permit access to illegal content” or “block access to peer-to-peer networks”;
- MSPL (Medium-level Security Policy Language) expresses specific configurations by technically-savvy users in a device-independent format, such as “deny *.sex”, “deny src 192.168”, or “inspect image/* for malware”.

To generate the configuration settings for a security control, MSPL is translated to low-level settings. These settings are expressed with the format required by a specific control (e.g. netfilter/iptables) selected for the actual implementation. The translators from MSPL to low-level settings are out of the scope of this document and will be discussed in another deliverable.

This approach has several advantages: it simplifies the configuration of the security controls, permits the use of different controls with the same abstract configuration and, in general, makes possible an easy migration of security configurations from one NED to another. More details on these advantages are discussed in the next sections.

The aim of this document is to provide specifications of the security policies (expressed by using HSPL) and configuration meta-model (represented by using MSPL). The applicability of the findings described here are useful for NED and PSA providers, and in general with respect to the applicability SECURED.

This document is organized in three parts: Section 2 introduces the HSPL, defining the structure of the language and proposing security policy examples, Section 3 presents the security capabilities required to classify security control features and Section 4 defines the meta-model for configuring security controls.

# 2 High-level Security Policy Language

The aim of SECURED is to offload security from end-nodes (e.g. smartphone, smart TV) to personal security applications (PSAs). Each PSA contains one or more security controls configured to enforce the security policy. A security control has configuration settings and features that require high technical skills to manage. This complexity get worse when more security controls must be configured to perform a particular activity (e.g. netfilter/iptables and Squid-cache configured together to perform parental control). In addition, the manual configuration of these settings, although performed by an expert user, is an error prone and time consuming task. This suggests the creation of a user-oriented language for expressing concepts related to end-point protection while hiding technical details of the actual implementation.

## 2.1 High-level policy languages

This section summarizes the analysis of high-level policy languages that was conducted as a preliminary step toward the design of the HSPL. The goal of high-level languages is to hide configuration details (e.g. blacklisted URLs) during the definition of a policy. On the contrary, user-oriented languages define which operations are permitted or denied to a user (e.g. Alice is authorized to access Internet) rather than configure a resource (e.g. configure firewall to allow traffic from Alice's PC to Internet).

Several languages have been developed and proposed during the last ten years. The most significant ones are XACML (eXtensible Access Control Markup Language) [1] and PCIM (Policy Core Information Model) [2]. XACML is widely adopted for specifying access control policies; PCIM provides a useful link with system description and was already used in the previous EC-funded projects POSITIF [3] and PoSecCo [4] to represent firewall and channel protection configurations. In addition to XACML and PCIM, we analysed also access control policy languages (Ponder [5, 6], SecPAL [7], Usage Control Languages [8]) and a format to express configurations and policy enforcement points (CIM-SPL [9]). Ontology-based languages have been considered as well (KAoS [10], rei [11], rein [12, 13]), together with languages that support representation of policy-related information (SAML [14], SCAP [15]).

Unfortunately most of these languages are related to specific context (e.g. access control) or in other cases are complex to be learned and managed by end-users. Therefore, none of the above languages can be considered user-oriented.

## 2.2 Requirements for HSPL

The definition of HSPL must satisfy the following requirements:

- simplicity – the definition of a security policy must be intuitive, the user must be assisted by using predefined statements (e.g. "block Internet traffic") and auto completion techniques;
- flexibility – to define every types of security policy, also supporting specific conditions (e.g. time constrains, content types, traffic types), for every security application;
- extensibility – must support future extensions, e.g. by introducing new policy types and specific conditions without changing the structure of HSPL.

## 2.3 Structure of HSPL

The idea of the HSPL is to define a policy by using high-level security requirements (e.g. "do not download any malware", "do not connect to illegal sites"). HSPL is composed of statements with the following structure:

```
[ sbj ] action obj [ (field_type,value) ...  (field_type,value) ]
```

Where:

- `sbj` is the user who needs to access or perform some operation on an object (e.g. employee, family member) and may be omitted if the policy is applied to the user that defines the HSPL;
- `action` is the operation performed on the object (e.g. protect, permit access, enable);

- `obj` is the entity (i.e. a resource such as e-mail scanning, Internet traffic, P2P traffic) target of the action (e.g. authorize access);
- `(field_type,value)` is an optional condition that add specific constraints to the action (e.g. time, content type, traffic type). The value part is a string with specific format depending on the field type.

The full specification of HSPL elements is provided in the next sections.

### 2.3.1 Subject

Considering the scope of SECURED and the use cases considered in the project, a set of suitable subjects (i.e. users) are proposed. In particular, this document addresses two scenarios: corporate and private. Considering the corporate scenario, it is useful to distinguish among employees with different privileges. For example, the subject can be used by the network administrator to define policies for user groups, e.g. marketing employees. Similarly, in the private scenario, several users with different privileges can be identified. For example, a father may wish to define different policies for his child and for guests. Therefore, the available subjects are:

- *specific user*, e.g. “Alice”;
- *current user*, that is omitted during the policy authoring;
- *user group*, e.g. “children”, “marketing employees”.

Finally, the special group “Users” identifies every user of the managed domain.

### 2.3.2 Action

Considering the objective of HSPL, i.e. expressing concepts related to end-point protection, a set of predefined actions is proposed. The following set of actions is identified:

- *is/are not authorized to access* – to explicitly deny a generic type of traffic (e.g. Internet traffic), a specific type of traffic (e.g. VoIP, P2P, DNS traffic) or the traffic related to a particular resource (e.g. FTP service);
- *is/are authorized to access* – to permit a generic type of traffic (e.g. Internet traffic), a specific type of traffic (e.g. VoIP, P2P, DNS traffic) or the traffic related to a particular resource (e.g. FTP service);
- *enable(s)* – to activate a particular type of control (e.g. antivirus, e-mail scanning, parental control) or feature (e.g. DDoS protection, logging);
- *remove(s)* – to intercept and block specific traffic (e.g. related to advertisement or tracking techniques);
- *reduce(s)* – to limit the usage of bandwidth;
- *check(s) over* – to analyse network vulnerabilities and threats;

- *count(s)* – to trace and limit the number of connections (e.g. DNS packets);
- *protect(s) confidentiality* – to ensure confidentiality of a particular data flow;
- *protect(s) integrity* – to ensure integrity of a particular data flow;
- *protect(s) confidentiality integrity* – to ensure confidentiality and integrity of a particular data flow.

### 2.3.3 Object

The object represents the entity controlled by the action (e.g. particular type of traffic, particular resource). To simplify the creation of a security policy, a predefined set of objects is provided.
In particular, the HSPL supports the following objects:

- *Internet traffic*, i.e. the traffic involving IP addresses different from private addressing;
- *DNS traffic*, *DNS-request*, *DNS-response*, i.e. respectively the traffic of DNS, the DNS requests or DNS responses;
- *intranet traffic*, i.e. the traffic involving IP addresses of the intranet;
- *VoIP traffic* i.e. the traffic regarding VoIP (e.g. SIP protocol);
- *3G/4G traffic* i.e. the traffic regarding 3G/4G;
- *all traffic*, i.e. every type of traffic (Internet, intranet, VoIP, etc.);
- *public identity*, i.e. to masquerade the identity of the user;
- *resource "x"*, i.e. the traffic regarding a particular resource ("x"), better specified by using a field condition;
- *basic parental control*, i.e. parental control with basic features that cannot be customized;
- *advanced parental control*, i.e. parental control with advanced features that can be customized by the user;
- *antivirus*, i.e. antivirus features;
- *logging*, i.e. to trace user's activities (e.g. visited web sites);
- *IDS*, *IPS*, i.e. intrusion detection and prevention features;
- *DDos attack protection*, i.e. protection against distributed DoS attacks;
- *email scanning*, *file scanning*, i.e. specific objects like e-mail messages or files;
- *tracking techniques*, *advertisement*, i.e. common techniques used to track users or advertisements;
- *bandwidth*, i.e. to control bandwidth (e.g. to reduce it);

- *connection*, i.e. to control the number of connections;
- *security status*, this feature is required by industrial partners to check the security status of the network that belongs to the NED (for example, to verify that the software configuration is up to date and known vulnerabilities are not present).

### 2.3.4 Field condition

A set of optional parameters is available to better specify the scope of an object (e.g. to define that an action is applied only for a particular type of traffic). These parameters are implemented by using field conditions. A field is organized in two parts: `field_type` and `value`. The following fields and available values are identified:

- *time* – to restrict the application of a HSPL policy statement to timing conditions, e.g. date range or particular days. The value part of a time range contains starting date/time, ending date/time and a time zone following the format `{[yyyy:mm:dd]hh:mm[:ss]-[yyyy:mm:dd]hh:mm[:ss], ..., GMT+1 || Europe/Rome}`, e.g. `{8:00-9:00, GMT+1}`. The first selector is the starting date/time (8:00), the second is the ending date/time (9:00) and the third is the time zone (GMT+1), expressed by using GMT compact or extended notation. It is possible to define more intervals by using a comma separation. To define particular days, the field supports the use of the following keywords: `weekend`, day of week, e.g. `monday`.
- *specific URL* – to restrict the policy considering a specific URL or website (e.g. www.facebook.com). In this case the value part contains URLs in the format `{url1, ...}`, e.g. `{http://www-.facebook.com, https://www.facebook.com}`;
- *type of content* – to restrict the policy considering specific contents, the supported types are social networks (e.g. Facebook, Twitter), illegal content, gambling, explicit sexual content, etc. The adopted format is `{content1, ...}`, e.g. `{gambling, social networks}`;
- *traffic target* – to specify the traffic of a particular user, organization or company e.g. corporate traffic (`{corporate network}`), user1's traffic (`{user1}`). Obviously the keywords adopted for the target must be defined by the user (who creates the policy) for each organization;
- *purpose* – to specify the type of analysis, the supported types are malware detection, spam detection by using the format `{purpose1}`, e.g. `{malware, spam}`.

## 2.4 Combining actions, objects and fields

In the previous sections we presented subjects, actions, objects and fields. However, an action does not support every objects and fields, but a restricted set of them. For example, the "enable" action can be applied to antivirus, basic parental control but not to DNS traffic.

In order to clarify the usage of actions, objects and fields, the Table 1 shows the available combinations between actions and objects while Table 2 illustrates the available fields for each object.

| **Object** / **Action** | is/are not authorized to access | is/are authorized to access | enable(s) | remove(s) | reduce(s) | check(s) over | count(s) | protect(s) conf. | protect(s) integr. | protect(s) conf.integr. |
|---|---|---|---|---|---|---|---|---|---|---|
| VoIP traffic | + | + | | | | | | + | + | + |
| P2P traffic | + | + | | | | | | + | + | + |
| 3G/4G traffic | + | + | | | | | | + | + | + |
| Internet traffic | + | + | | | | | | + | + | + |
| intranet traffic | + | + | | | | | | + | + | + |
| DNS traffic | + | + | | | | | + | + | + | + |
| all traffic | + | + | | | | | | + | + | + |
| public identity | | | | + | | | | | | |
| Resource 'X' | + | + | | | | | | | | |
| file scanning | | | + | | | | | | | |
| email scanning | | | + | | | | | | | |
| antivirus | | | + | | | | | | | |
| basic parental control | | | + | | | | | | | |
| advance parental control | | | + | | | | | | | |
| IDS/IPS | | | + | | | | | | | |
| DDos attack protection | | | + | | | | | | | |
| tracking techniques | | | | + | | | | | | |
| advertisement | | | | + | | | | | | |
| bandwidth | | | | | + | | | | | |
| security status | | | | | | + | | | | |
| connection | | | | | | | + | | | |
| logging | | | + | | | | | | | |

Table 1: Combining actions and objects.

| Object / Field | time period | traffic target | specific URL | type of content | purpose | bandwidth value | resource value |
|---|---|---|---|---|---|---|---|
| VoIP traffic | + | + | | | | | |
| P2P traffic | + | + | | | | | |
| 3G/4G traffic | + | + | | | | | |
| Internet traffic | + | + | + | + | | | |
| intranet traffic | + | + | + | + | | | |
| DNS traffic | + | + | + | + | | | |
| all traffic | + | + | + | + | | | |
| public identity | + | + | + | | | | |
| Resource | | | | | | | + |
| file scanning | | | | | + | | |
| email scanning | | | | | + | | |
| antivirus | | + | | | | | |
| basic parental control | | | | | | | |
| advance parental control | + | | | + | | | |
| IDS/IPS | | + | | | | | |
| DDos attack protection | | + | | | | | |
| tracking techniques | | | + | | | | |
| advertisement | | | + | | | | |
| bandwidth | + | | + | | | + | |
| security status | | | | | | | |
| connection | + | | + | | | | |
| logging | + | + | + | + | | | |

Table 2: Combining objects and fields.

## 2.5 Examples of HSPL

This section proposes some examples of HSPL. In particular, Section 2.5.1 describes a policy to deny access to illegal web sites, Section 2.5.2 describes a policy to permit Internet traffic only on a particular time slot, Section 2.5.3 describes a policy to detect malware in emails, Section 2.5.4 describes a policy to remove tracking techniques of Facebook's website, and Section 2.5.5 describes a policy to encrypt corporate network traffic.

### 2.5.1 Deny access to illegal web site

A typical security policy, suitable both for a company and a home environment, is to block access to illegal contents. The proposed policy denies access to web sites (that contains illegal contents) for Alice. The corresponding HSPL statement is:

```
Alice is not authorized to access Internet traffic
      (type of content, {illegal websites})
```

where:

- the subject is Alice, i.e. the policy is only applied to this user, regardless of the NED where she connects;
- `is not authorized to access` denies the access;
- `Internet traffic` is the object, i.e. data flow related to Internet;
- `(type of content,{illegal websites})` specifies that the action (block) is applied only to "illegal websites" (a keyword that refers to a specific set of web sites).

### 2.5.2 Permit Internet traffic for a specific time slot

A common policy is to permit access to Internet (e.g. to visit social network websites) only on predefined time slots, e.g. during lunch time or at the end of every work day. This policy permits Internet traffic only from 18:30 to 20:00 considering the timezone of Rome.

The corresponding HSPL statement is:

```
is authorized to access Internet traffic
(time period, {18:30-20:00 Europe/Rome})
```

Where:

- the subject is omitted, i.e. the policy is applied to the current user, i.e. who is specifying this policy;
- `is authorized to access` permits the access;
- `Internet traffic` is the object, i.e. all Internet traffic;
- `(time period, {18:30-20:00 Europe/Rome})` specifies that the action (allow) is applied only for the time slot 18:30-20:00 in the timezone of Rome.

### 2.5.3 E-mail malware detection

Another common policy, both for home and corporate environment, is to protect e-mail from malware. This policy requires that each e-mail must be analysed to detect malware.

The corresponding HSPL statement is:

```
Bob enables email scanning (purpose,{malware})
```

Where:

- `Bob` is the subject i.e. the policy is applied to Bob;
- `enables` activates the control(s) to perform the operation specified by the object;
- `email scanning` is the object, i.e. scan email;
- `(purpose,{malware})` specifies which is the purpose of the analysis.

### 2.5.4 Remove tracking techniques from Facebook's website

Social networks are the most visited websites but some features could decrease the user's privacy. For example, the use of cookies for gathering information or propose customized data to a user (e.g. custom advertisement). The objective of this policy is to remove (only for the users of the network protected by SECURED) the tracking techniques (e.g. cookies) of Facebook's website.

The corresponding HSPL statement is:

```
remove tracking techniques (specific URL, {http://www.facebook.com})
```

Where:

- the subject is omitted, i.e. the policy is applied to the current user;
- `remove` performs a remove operation on the object;
- `tracking techniques` is the object, i.e. common techniques used to track users;
- `(specific URL, {http://www.facebook.com})` specifies the website on which the action is applied. In this case, the tracking techniques are removed only for the traffic related to http://www.facebook.com.

### 2.5.5 Encrypt traffic of corporate network

Finally, the encryption of business information, trade secrets, etc. are quite important to avoid information disclosure. This policy aims to encrypt all traffic from a user protected by SECURED to a specific portion of his corporate network.

The corresponding HSPL statement is:

```
Bob protects confidentiality intranet traffic
    (traffic target, {marketing-subnet})
```

Where:

- `Bob` is the subject, i.e. the policy is applied only to user Bob;
- `protects confidentiality` encrypts the object;
- `intranet traffic` is the object, i.e. intranet network traffic;
- `(traffic target, {marketing-subnet})` the action is applied only on marketing-subnet traffic. Note that the rest of Bob's traffic (e.g. access to Facebook) is not covered by the policy.

## 2.6 HSPL templates

A complex policy, for example to define parental control, is composed by several sub-policies, i.e. to limit Internet access by considering: timing conditions (e.g. from 20:00 to 22:00), type of content (e.g. blocking gambling), particular URLs (e.g. blocking URL of a blacklist). Therefore, it cannot be easily defined by using a single HSPL statement. Therefore we defined a set of templates to simplify the definition of complex policies, i.e. to avoid unskilled users to define each HSPL statement. A HSPL template is a group of HSPL statements that together implement a complex policy. These policies are predefined (i.e. a user can use a template as is) but can be customized to satisfy particular requirements (e.g. timing constraints or adding specific URL to a blacklist). Again, to simplify the definition of a security policy, the template is invoked as follow:

```
[ sbj ] enable(s) template_name
```

Where:

- `sbj` identifies the subject (i.e. the user that undergoes the policy) and can be omitted, in which case the template is applied to the current user;
- `enable(s)` is the action to activate the template;
- `template_name` is the name of a predefined or user defined template.

The HSPL template supports only the action `enable(s)`, also adopted by HSPL statements. Therefore, to distinguish between HSPL statement and template it is necessary to analyse the element that follows the action. When it is an object, i.e. one of the keywords defined in Section 2.3.3, a HSPL statement is processed, otherwise, a template is invoked.

This form of invocation is quite close to natural language and it is suitable for unskilled users.

### 2.6.1 Example: enabling parental control

The parental control demonstrates the importance of HSPL templates. Considering the home environment, a common policy is to limit and monitor the usage of entertainment services (e.g. web TV) or particular applicatio s (e.g. games). Several types of policy can be defined, for example, to limit a particular website (e.g. Facebook) or to define particular time slots. As introduced before, the parental control template is typically composed by several HSPL statements and the most important ones are:

- "`enable antivirus`" to perform antivirus scan on all network traffic;


- "`is not authorized to access Internet traffic (type of content, porn)`" to deny Internet traffic with porn content;
- "`is not authorized to access Internet traffic (specific URL, myblacklist)`" to deny Internet traffic of sites defined into "myblacklist";
- "`is authorized to access Internet traffic (time,{20:00-22:00 GMT+1})`" to permit Internet traffic from 20:00 to 22:00 by considering GMT+1 timezone;
- "`remove advertisement`" to block advertisement from web pages;
- "`enable logging ((type of content, gambling), (traffic target, Internet traffic))`" to trace Internet traffic which contains gambling contents.

Therefore, to avoid the definition of these HSPL statements, a user may enable parental control by selecting the related template as follow:

```
Bob enables basic parental control
```

Where:

- `Bob` is the subject, i.e. the template is enabled only for the user Bob;
- `enables` activates the template specified in the object;
- `basic parental control` is the object, i.e. the basic parental control template, without customization.

This template can be customized for example by modifying the content of "myblacklist", adding or removing HSPL statements. Finally, it is also possible create a new template from scratch or by cloning an existing one to define a new set of HSPL statements (e.g. to define specific requirements for an organization).

## 2.7 HSPL view

The HSPL is an authorization and mandatory access control language structured to be easily interpretable from a machine. However, its structure, in particular when field types need to be specified, may introduce some unmanageable complexity for end-users. This complexity suggests to provide an even simpler way to specify HSPL, by means of sentences close to natural language, for supporting the policy authoring. The basic idea is to model the HSPL by using the Model-View-Controller (MVC) paradigm. By following this approach, the structure of a HSPL statement is the model, the set of sentences, expressed by using natural language, is the view (HSPL view) and the transforms (i.e. the representation from HSPL view to HSPL) is performed by the controller. This represents the natural sentences (passed as input by HSPL view) by using an ad-hoc mapping into the HSPL structure (i.e. the model).

The structure of a natural language HSPL sentence is the following:

```
verb obj [ obj_complement ...  and ...  obj_complement ] [for sbj]
```

Where the verb (i.e. action) is represented by `verb`, the object by `obj`, the object complement (optional) by `obj_complement` and the subject (optional) by `sbj`. A set of conjunctions and prepositions `and`, `for` and `of` are useful to bind together the elements of the sentence.

By using the HSPL view, implemented via a GUI, a user composes and customizes the sentences from a predefined set. First of all, he/she selects the action by a list (e.g. by using a combo box) of available ones. This choice enables the selection of suitable objects and related complements. The object complements can be customized by using a set of parameters, for example to specify timing constraints. Then, the sentences are automatically represented into the model, by the controller, as HSPL statements. Subjects and objects are the same for HSPL and HSPL view, therefore a transformation is not needed. Verbs adopt the same keywords of HSPL or synonyms, for example "Allow" is transformed as "is/are authorized to access". Finally, to be easily managed by a software process, the complement must be transformed into a fixed structure, i.e. the set of field types (`field_type`).

For example, the sentence:

```
Allow Internet traffic from 20:00 to 22:00 tz GMT+1 for Alice
```

is translated into:

```
Alice is authorized to access Internet traffic
      (time period, {20:00-22:00 GMT+1})
```

In this case the object complement `from 20:00 to 22:00 tz GMT+1` is automatically transformed into the field type `time period`.

# 3 Capabilities

Before modelling the abstract configuration of a security control, its functionalities must be defined.

A capability denotes any kind of security functionality that can be provided by a PSA or, generally, by a software module available in a PSA, e.g. filtering, logging or authentication.

The objective of SECURED is to produce abstract configurations that can be consistently transformed into specific settings for the security mechanisms. These abstract configurations will be technology-dependent but vendor-independent.

Based on the catalogue of security capabilities of available PSAs, the policy refinement process discovers and evaluates the possible configurations that fulfils a requested policy.

This section illustrates the abstract model of security capabilities describing the structure and the functionalities useful in the scope of SECURED. The proposed model is defined by using the Unified Modelling Language (UML).

## 3.1 Capability concept and hierarchy

This section presents the capability hierarchy according to the security domains of SECURED.

The class `Capability` denotes any capability supported by an `ITResource` (a specific software module, e.g. netfilter/iptables) that is useful for SECURED purposes. The capability concept is represented as an abstract class whose subclasses describe security abilities and/or potential for use of IT resources (e.g. the capability of establish a secure channel, the capability of encrypting data or the capability of

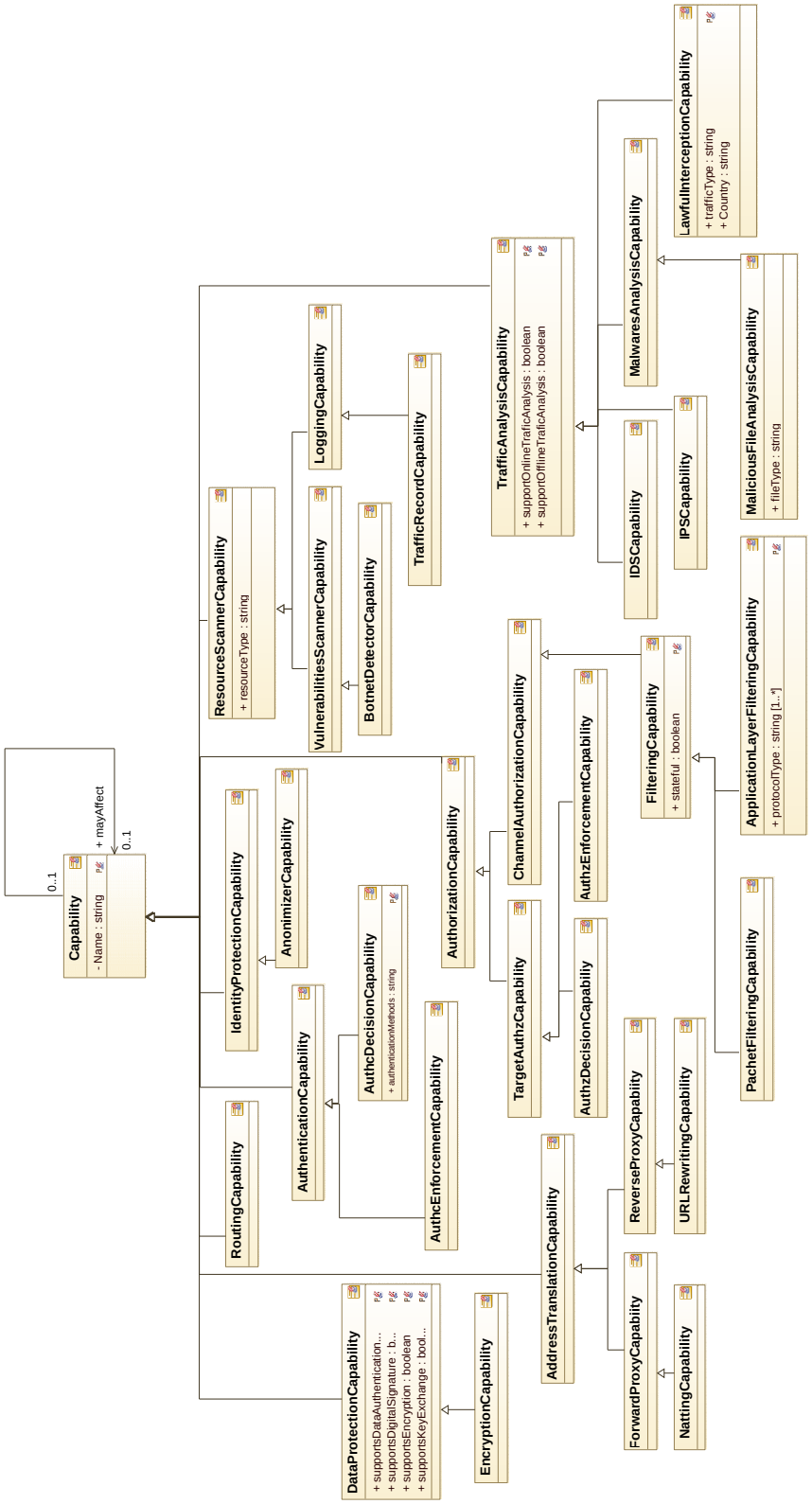


Figure 1: Capability hierarchy.

enforcing authentication and authorization policies). The concept of capability tied to an `ITResource` is crucial and has a particular importance for the policy refinement in SECURED, as it indicates which security functionality can be enforced by using an `ITResource` to satisfy a given high-level policy.

Figure 1 shows the security capabilities defined in the scope of SECURED. First of all, the behaviour of a capability may affect another capability, for example an IPS (Intrusion Prevention System) that drops a packet may affect filtering. This behaviour is described by the association `mayAffect`.

The hierarchy distinguishes, at the first level of sub-classing, between abstract capabilities and specific ones, implemented by security controls (i.e. a set of feature, e.g. packet filter) of a PSA. The abstract capabilities include common security abilities of an IT Resource, such as the following ones:

- address translation (`AddressTranslationCapability`);
- authentication (`AuthenticationCapability`);
- data protection (`DataProtectionCapability`);
- authorization (`AuthorizationCapability`);
- routing (`RoutingCapability`);
- identity protection (`IdentityProtectionCapability`);
- resource analysis (`ResourceScannerCapability`, `TrafficAnalysisCapability`).

The specific capabilities describe the abilities of security controls and are used by the SECURED refinement process (e.g. policy-driven approach) to identify which set of PSAs can be used to enforce a security policy.

More in detail, `AddressTranslationCapability` causes a dynamic mapping of addresses and URLs, features typically implemented by proxies and NAT devices. We will use these data to update rules and configurations and to precisely enforce access control. These features are represented by subclasses `ForwardProxyCapability`, `ReverseProxyCapability` and related `URLRewritingCapability` and `NattingCapability`. The `RoutingCapability` defines the ability to dispatch packets among network nodes which connect network elements.

Although authentication and authorization are much related to each other (i.e. authorization is not possible without a prior authentication) we do distinguish between the two functional models, because each one can be performed (or enforced) by separate entities. The abstract class `AuthenticationCapability` is extended by two subclasses `AuthcDecisionCapability` and `AuthcEnforcementCapability`. The objective of this separation is to support centralized authentication systems where an authentication enforcement point (e.g. a WPA2 Wireless Access Point acting as an IEEE 802.1x authenticator) forwards an authentication request to an authentication decision point (e.g. a RADIUS authentication server). The authentication methods, which could be supported by the `AuthcDecisionCapability` probably, would be symmetric challenge, asymmetric challenge, username/password, zero knowledge, etc.. In general, we distinguish between these two subclasses to be able to represent modern authentication scenarios, where security domains (single-sign-on solutions) manage identities and credentials in one central node to serve different security enforcing entities in the system. The `AuthcDecisionCapability` includes the attribute `AuthenticationMethod` that lists the set of methods the authentication policy decision point supports to authenticate the users. The abstract class `AuthorizationCapability` is extended by two subclasses `TargetAuthorizationCapability` and `ChannelAuthorizationCapability`. These subclasses are needed to distinguish typical authorization systems that control access on resources that are under their direct control (e.g. operating systems, DBMS rules,

XACML data) from devices that make decision on channels, e.g. discard packets like, for instance, the firewalls. The abstract class `TargetAuthorizationCapability` is extended in the two subclasses `AuthzDecisionCapability` and `AuthzEnforcementCapability`. These two subclasses represent the capability of an IT Resource, which is able to enforce access control rules (deny or allow and other privileges) for given principals (roles, users or groups) on a given security object. The intention of this separation, although it is out of the scope of SECURED, is to support commonly used authorization PEP (Policy Enforcement Point) and PDPs (Policy Decision Point) in authorization management.

The class `ChannelAuthorizationCapability` has one subclass: `FilteringCapability`. It indicates if an ITResource can act as a filtering device according to the layer at which it can work on. It includes the `stateful`, a Boolean attribute, that indicates that the filtering device support stateful inspection (e.g. keeps track of connections at transport layer and make decision based on this information). The class has also two sub-classes: `PacketFilteringCapability` useful to manage traffic from layer 2 to layer 4 of the ISO/OSI stack; `ApplicationLayerFilteringCapability` useful to perform inspection at layer 7 of the ISO/OSI stack. The supported protocols are identified by a list of strings specified by the attribute `protocolType`. In particular, application layer filtering comes in two flavours:

- inspection of the protocol header at application layer (e.g. HTTP, FTP) to make decisions based on it;
- inspection of the payload of application layer protocols to make decisions based on it (e.g. SOAP, MIME objects).

The abstract class `DataProtectionCapability` is used to indicate that an ITResource is able to protect data at the application level or to protect the channel communication. It includes a set of Boolean attributes to indicate which "transformation" it can apply on Data. The attribute:

- `supportsEncryption` indicates that an ITResource has the ability to act as a (symmetric) encryption component (e.g. using AES);
- `supportsDigitalSignature` indicates that the ITResources having a capability to digitally sign data or documents (e.g. with RSA, DSS);
- `supportsDataAuthenticationAndIntegrity` indicates that the ITResources is able to provide data integrity and authentication protection using a symmetric algorithm (e.g. using HMAC);
- `supportsKeyExchange` indicates that the ITResources is able to securely exchange a symmetric key (e.g. using asymmetric algorithms like Diffie-Hellman or RSA).

The `EncryptionCapability` is a sub-class of that, useful to specify that an ITResource has the ability to encrypt data.

The `IdentityProtectionCapability` represents the feature to protect the identity of an user or resource. It has the sub-class `AnonimizerCapability` that ensures the anonymity, for example the TOR project [16].

The `ResourceScannerCapability` is an abstract that aggregates the capabilities for scanning and detecting malicious content of a resource (email, file, etc.). This class is specialized into the following sub-classes:

- `VulnerabilityScannerCapability` useful to scan a network for identify vulnerabilities (e.g. analysing network services);
- `LoggingCapability` can be used in SECURED to trace network connections or visited URLs (e.g. for parental control);
- `TrafficRecordCapability` to capture and store network traffic;
- `BotnetDetectorCapability` to analyse network and related service detecting botnet.

The `TrafficAnalysisCapability` performs analysis of the network traffic for statistics or in order to take a decision, for example dropping a malicious packet, recognized by using a signature (e.g. IPS). Traffic analysis can be online or offline, in the first case it may permit, drop or modify network traffic (e.g. dropping a packet). This class has the following specialization:

- `IDSCapability` that implements classical Intrusion Detection System (IDS) feature, and typically operates offline;
- `IPSCapability` that implements classical Intrusion Prevention System (IDS) feature and typically operates online;
- `MalwareAnalysisCapability` to analyse, detect malware and viruses on a resource. The specific class `MaliciousFileAnalysisCapability` is defined to analyse file (e.g. PDF);
- `LawfullInterceptionCapability` contains the features to analyse and dropt traffic in order to obey to legal requirements. The type of traffic to analyse is specified by the attribute `trafficType` and the reference country by the attribute `Country`.

Finally, the given capability hierarchy can be extended by sub-classing the `Capability` class or any of the existing subclasses.

## 3.2 Capability use cases

In this section, we give an overview of possible use of SECURED security capabilities according to different network layers or software components.

### 3.2.1 Authorization Capabilities

This capability indicates if an IT Resource is able to enforce access control rules for given principals (roles, users or groups). Possible use of authorization capabilities subject to different layers or software components could be:

- Network (User Personal Network - UPN), e.g. restricted access to specific network services based on the IEEE 802.1X Port Authentication (VLAN, NAC, bandwith restriction, filters, time limits, etc.);
- Access control (on different network layers): mostly done by restricting system access to authorized users according to user privileges via different Access Control mechanisms (e.g. group membership, Role-Based AC, Context-Based AC, Context-Aware AC provisioned by Identity and Access Management Systems (IAM);

### 3.2.2 Filtering Capabilities (channel authorization)

Filtering devices designed to permit or deny network transmissions based upon a set of rules can operate on different layers, inspect different packets, protocols and even their content. The following list shows some examples of firewall rules on different layers and levels:

- MAC filtering;
- Ethertype filtering (e.g. ARP);
- Address filtering (e.g. IP address, IP range);
- Protocol filtering (e.g. ICMP, UDP, TCP);
- Port filtering (including the direction of the connection establishment, port range, etc.);
- HTTP filtering (e.g. squid-cache, apache, web application firewalls);
- Content filtering (application level firewalls, proxies, etc.).

### 3.2.3 Authentication Capabilities

Authentication capabilities can be summarized to some standard authentication schemes on different layers related to the strength requested by a supplied IT policy:

- Authentication based on different methods:
  - shared secret based authentication;
  - private/public key based authentication;
  - biometric based identification
- one-way or two-way (mutual authentication);
- Single or multifactor authentication.

### 3.2.4 Data Protection Capabilities

Data protection can be enforced by different techniques subject to intended target:

- Secure channel communication: Based on the authentication protocol a key agreement results on the crypto algorithms supported by both sides. Dependent to the chosen cipher suite the communication channel will be authenticated and protected accordingly. (WPA2, SSL/TLS, IPSec, SSH, etc.);
- Message protection: Message protection is primarily used on application layer to protect independently content of data as in e-mails using S/MIME or PGP or XML message protection with XML-ENC or XML-DSIG.

### 3.3 Mapping security capabilities onto PSAs

As widely discussed in D5.1 [17], each PSA implements one or more security capability. This sections discusses a mapping of capabilities onto the different PSA categories.

Figure 2 depicts the PSA categories, organized by using a hierarchical structure.

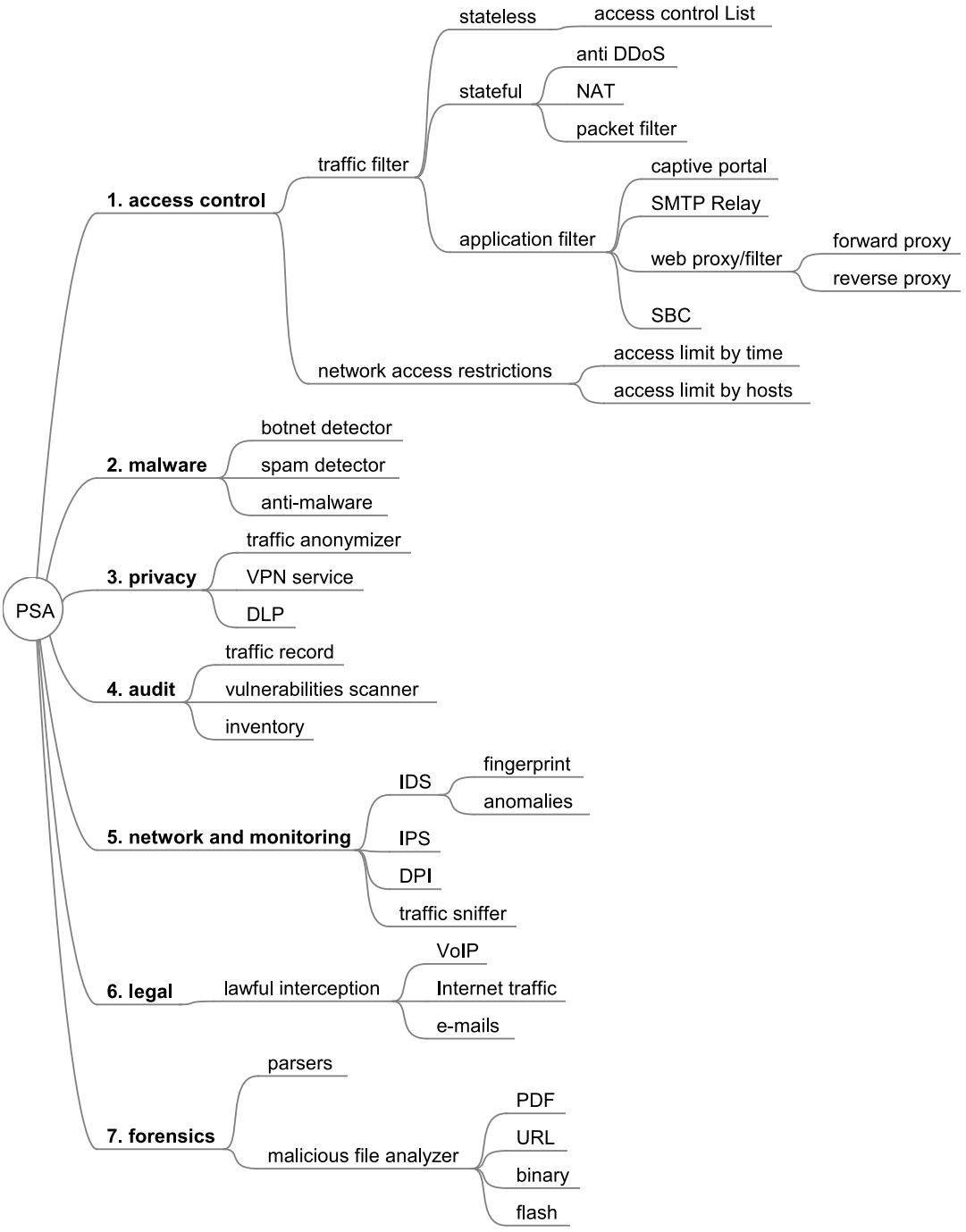


Figure 2: PSA categories.

For the sake of clarity we summarize the contents of these categories:

- *access control* contains the set of capabilities to filter and restrict the traffic based on a list of conditions. Conditions for filtering can be based on any field of the Internet protocols layer starting from Layer 2 to Layer 7 of the ISO/OSI stack, or external conditions defined by the user, e.g. day or time restrictions, content filtering.
- *malware* for detection and eradication of malicious traffic. Three types of capabilities are identified: botnet detector (to discover bot infections), spam detector (to detect unsolicited traffic) and network anti-malware (to detect malware on network traffic).
- *privacy* that offers mechanisms to provide privacy to the users traffic, such as the Traffic anonymizer (e.g. TOR [16]), Virtual Private Network (VPN) and Data Leakage Prevention (DLP).
- *audit* that contains capabilities to help the users to discover threats in their SECURED accessing device. Examples of capabilities are traffic record, vulnerabilities scanner or software inventory.
- *network* and *monitoring*, network category contains the capabilities that implement functionalities at layer 3 of the ISO/OSI stack, e.g. routing, NAT, VLAN. Network monitoring category implements capabilities that analyse the traffic flowing through the PSA looking for threats, e.g. Network Intrusion Detection System (IDS), Network Intrusion Prevention System (IPS), Deep Packet Inspection (DPI) and sniffers.
- *legal* that contains capabilities to help network operations and service providers to obey to legal requirements. The most interesting capability is Lawful Interception (LI).
- *forensics* contains capabilities for looking for threats considering online and offline traffic, e.g. parsers to scan, or malicious file analyser.

In order to clarify which security capabilities (shown in Figure 1) are required by a PSA, we propose the following mapping:

- *access control* is implemented by using the class `FilteringCapability`. In particular, `PacketFilteringCapability` covers *stateless* and *stateful* inspections, `ApplicationLayerFilteringCapability` covers *application filter* inspection. *Forward* and *reverse* proxies are mapped by using `ForwardProxyCapability` and `ReverseProxyCapability`. The *network access restriction* is available on `PacketFilteringCapability` and `ApplicationLayerFilteringCapability`;
- for *malware* related aspects, we distinguish among *spam detector* and *anti-malware* (that are implemented by using the generic class `MalwareAnalysisCapability`) and *botnet detector* (implemented by the specific class *BotnetDetectorCapability*);
- *privacy* is implemented by using different types of security capabilities. In particular, we map *traffic anonymizer* to `AnonimizerCapability`, *VPN service* to the general class of `DataProtectionCapability`. The prevention of data leakage, i.e. *DLP*, is implemented by the class `ApplicationLayerFiltering`;
- *audit* items are implemented by `ResourceScannerCapability` (in particular, *traffic record* by `TrafficRecordCapability`, *vulnerability scanner* and *inventory* by `VulnerabilityScannerCapability`);

- *network* and *monitoring* contain different sub-classes: *IDS* and *IPS* are implemented respectively by using `IDSCapability` and `IPSCapability`, *traffic sniffer* is covered by `TrafficRecordCapability` and *DPI* is a generic functionality, covered by `ApplicationLayerFiltering` and `TrafficAnalysisCapability`;
- *legal* category and in particular, *lawful interception* is implemented by `LawfulInterceptionCapability`;
- finally, *forensics* is generally mapped on `TrafficAnalysisCapability` and *malicious file analyzer* on `MaliciousFileAnalysisCapability`.

# 4 Medium-level Security Policy Language

The base idea of the Medium-level Security Policy Language (MSPL) is to describe the configuration settings of a class of security controls. More in details, the MSPL is an abstract language with statements related to the typical actions performed by various security controls (e.g. matching patterns against packet headers, keeping track of connection status, identifying the MIME type of a payload), but expressed in a generic syntax.

In order to guarantee the flexibility of the framework, MSPL is also directly accessible to expert users, i.e. the ones which have better knowledge about security and networking.

As discussed in Section 2, HSPL is suitable for capturing the user requirements, however it cannot be directly implemented by security controls. This requires the definition of a medium-level language able to express the same information into operational policies by using a format suitable for configuring security controls. This type of language is typically an ordered sequence of actions (e.g. permit and deny for a firewall) related to matching packets or payloads. In particular, the MSPL is an abstract language, with statements related to the typical actions performed by various security controls (e.g. matching patterns against packet headers, keeping track of connection status, identifying the MIME type of a payload), but expressed in a generic syntax. Similarly to capabilities, the proposed model is defined by using the Unified Modeling Language (UML). In addition, the introduction of MSPL, which uniforms the representation of policies for every PSA, is useful to perform conflict analysis.

Finally, the design of MSPL is based on a set of configuration models related to the previous EC-funded project PoSecCo [4].

## 4.1 Requirements for MSPL

The definition of MSPL should satisfy the following requirements:

**Abstraction.** MSPL must contain abstract security-related configurations, independent from a given vendor or product specific representation and storage. The reason for this requirement is that configuration semantics are independent from the actual representation. In fact, the same configuration settings can be represented and enforced in different PSAs.

**Diversity.** MSPL must allow the description of configurations for a variety of security capabilities (confidentiality, filtering, etc.). The configuration meta-model must furthermore support the configuration of different security capabilities, which follow different policies and concepts (e.g. communication protection parental control), apply to different types of PSA.

**Flexibility and extensibility.** MSPL must be flexible and extensible enough to support the introduction of new PSAs.

**Continuity.** MSPL must ensure the continuity of the policy chain, starting from the HSPL down to the security control settings. This is useful to tracking which policy is actually enforced and which user is associated to it.

## 4.2 MSPL meta-model

This section presents the configuration meta-model and explains the main concepts, capability and configuration, and their relationships.

Figure 3 sketches the elements of the SECURED configuration meta-model. First of all, we concentrate on core elements which are `ITResource`, `Capability` and `Configuration`.

An `ITResource`, is the central concept and represents a piece of software that implement a security control. For each `ITResource`, we can assign the capabilities being supported by this software component with the help of the relation `provides`. Each `ITResource` can have zero or more configurations which are assigned to exactly one capability. The assignment of configurations to an `ITResource` is done by means of the relation `configuration`. The association class with the same name allows the qualification of different types of configurations for an `ITResource` instance in the model. It can be used to represent the current configuration of an `ITResource` and the golden configuration as a result of the policy refinement process, at the same time.

As introduced before, a Capability denotes any kind of (security) functionality that can be provided, e.g. filtering, logging or authentication. In SECURED, mainly common security functionalities will be considered, i.e. capabilities that are typically supported by a class of software products. A complete description of the `Capability` class and its subclass hierarchy can be found in Section 3.

A `Configuration` corresponds to abstract configuration settings, which are independent of a given vendor or product. The abstract class `Configuration` can be subclassed by configurations dedicated to deliver the different capabilities, e.g. class `FilteringConfiguration` (shown in Figure 5 Configuration Hierarchy) to describe settings for packet filtering of a `Firewall` component. To ensure the identification of the affected infrastructure element for a given configuration, each `Configuration` is assigned by the relation `configuration` to exactly one `ITResource`. A `RuleSetConfiguration` is a specialization class used to assign a configuration (i.e. a policy) to a PDP. This kind of configurations is the expected outcome of the policy refinement process, that is, a rule set is the representation of a MSPL policy for a PSA. Each `RuleSetConfiguration` consists of a set of `ConfigurationRule`. A `ConfigurationRule` is enforced by `ConfigurationAction` (association `configurationRuleAction`) and can use a set of `ConfigurationCondition`. A `ConfigurationCondition` is a Boolean predicate, that allows the expression of complex conditions using AND and OR connections. The attribute `isCNF` in the `ConfigurationRule` class is used to specify that a rule matches if all conditions match (default attribute `isCNF=true`) or when at least one matches (attribute `isCNF=false`). A `ConfigurationCondition` can be assigned to a `ConfigurationRule` via the association `configurationCondition`. The attribute `isCNF` in the `ConfigurationCondition` class has the same role and semantics as its analogous in the `ConfigurationRule` class. When an event needing a policy-based decision happens at the PEP (e.g. a packet at the firewall interface), the PEP "contacts" the PDP and passes it a set of environment data (e.g. the IP header of the received packet, or the authentication packets). Then the PDP evaluate the conditions of rules against the environment data. If exactly one rule matches then the PDP sends the PEP `ConfigurationAction` specified by that rule (e.g. the

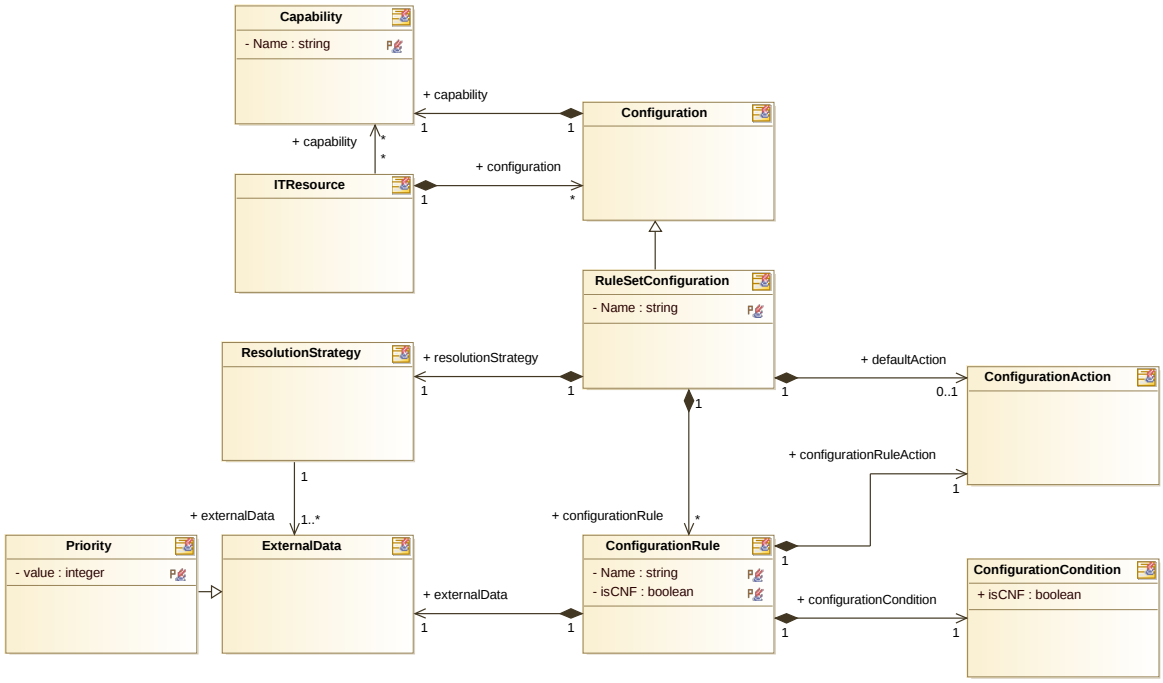


Figure 3: General configuration meta-model.

firewall denies or allow packet according to the rule action). Practically, two additional cases need to be considered when describing a policy using a set of rules: what happens if no rules apply and what to do when more than one rule applies. The latter is often named *policy conflict*. With the association `defaultAction` a default `ConfigurationAction` is linked to the `RuleSetConfiguration` in order to specify what to do when no rule can be applied. If more than one `ConfigurationRule` can be applied, the `ResolutionStrategy` assigned to the `RuleSetConfiguration` via the association `resolutionStrategy` is used to solve the conflict. We present a list of the most common resolution strategies (see Figure 4). The presented specializations of the class `ResolutionStrategy` can be further refined and extended when needed. The most used resolution strategy, e.g. in filtering rules of firewalls, is the First Matching Rule resolution strategy (FMR), represented by the `FMR` class, when rules are assigned to priorities, if more than one rule applies, then the action from the rule at the highest priority is enforced. This means that every `ConfigurationRule` may also be assigned to `ExternalData`, e.g. priority or creation time or last modification, which is used to make decisions. The contrary resolution strategy is the Last Matching Rule resolution strategy (LMR), represented by class `LMR` that respectively select the last applicable rule in an ordered list As an alternative, the Deny Takes Precedence (DTP) can be used, represented by the `DTP` that, in case of conflicts, selects the most restrictive action, or vice versa, the Allow Takes Precedence (ATP) can be used, represented by the `ATP` that, in case of conflicts, selects the most permissive action. Access control for particular a scenario sometimes uses MSTP (Most Specific Takes Precedence), which enforces the action of the rule, which has the most specific condition. Alternatively, the opposite strategy LSTP (Least Specific Takes Precedence) may be adopted. They are represented as `MSTP` and `LSTP` classes.

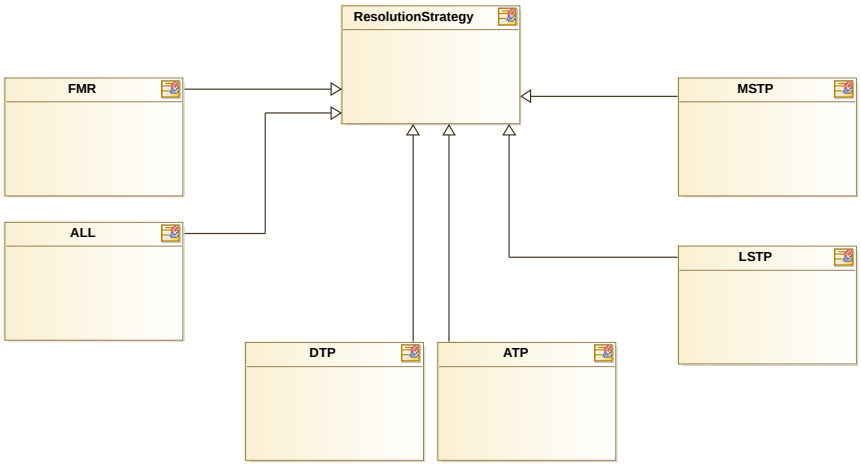


Figure 4: Resolution strategy.

## 4.3 MSPL model specializations

This section provides configuration model specialization that are especially relevant for SECURED, i.e. Filtering (see Section 4.3.1), Parental control (see Section 4.3.2) and Data protection (see Section 4.3.3) configurations. We will describe the concepts (classes) and relations of the configuration model specializations together with illustrative examples.

### 4.3.1 Filtering configuration

Figure 5 presents the Filtering configuration, i.e. the first specialization of the SECURED configuration meta-model. The `RuleSetConfiguration` is a subclass of the `Configuration`. To describe a filtering configuration we specialize `ConfigurationAction` and `ConfigurationCondition`. The other classes belong to the general configuration meta-model presented in Figure 3. A filtering configuration can be represented as a set of `ConfigurationRules` that use conditions that are subclasses of the `FilteringConfigurationCondition` class and enforce (exactly) one action that is subclass of `FilteringAction`. The default action must be taken from the `FilteringAction` represented by the attribute value `FilteringActionType`, which usually allowed filtering in Allow and Deny; some devices also support the Reject action that discards the packet but also sends an ICMP notification to the sender. The resolution strategy for a filtering configuration may be one of the available strategies i.e. FMR, ALL, DTP, ATP, MSTP and LSTP (see Figure 4). Filtering devices are divided according to the ISO/OSI stack at which they can work. Most of the firewalls are able to inspect packets at layer 3 and layer 4, some of them can inspect packets up to layer 7 (application layer firewalls). Application layer firewalls support the possibility of specifying conditions on protocol-specific fields, for instance, Squid, supports filtering on HTTP protocol headers. Recently, some products have been made available able to watch inside the content at the application layer (content filters). To represent this scenario, our configuration meta-model introduces two specialization classes of the `FilteringConfigurationCondition`, i.e. `PacketFilterCondition` and `ApplicationLayerCondition`. The former includes a number of attributes that are needed to select the packets to which to apply the action: the source and destination IP addresses and ports (`sourceAddress`, `sourcePort`, `destinationAddress`, `destinationPort`), the protocol type field in the IP header (`protocolType`), and the direction (`direction`, i.e. inbound or outbound). The latter includes the attributes typically related to the application layer, i.e. URL (`URL`)

and particular methods (e.g. `httpMethod` for the HTTP protocol).

Moreover, filtering devices are categorized according to their ability to maintain state information. Devices that do not maintain state information are named packet filters, while the others are named stateful filtering devices; in particular the functionality they implement is named stateful inspection if the analysis works at transport level or stateful protocol analysis if it works at application layer. Stateful devices keep track of each connection by examining certain values of TCP and other protocols headers and maintain a state table. State tables contain an entry for each of the observed (or to reduce memory allocation only the allowed) connections, usually represented by a five-tuple composed by the IP source and destination addresses, the source and destination port, and the protocol type. Stateful conditions always refer to some packet filter condition. Values in the state table are used to make decisions, e.g. allowing all the TCP packets related to an established connection, and sometimes implicitly, e.g. blocking packets that do not comply with the TCP protocol specification. Stateful inspection sometimes enforces a simple form of bandwidth control, that is, to specify the maximum number of connections allowed to a given destination, and limits the packets rate per destination address base or per port base. An improved type of application firewall is the application-proxy gateway, which enforces an access control policy using a proxy agent. Application-proxy gateway may keep track of authenticated users, to permit the specification of rules limiting the maximum number of allowed users, or the maximum number of connections on a per user base. To model stateful filtering, we introduced the abstract class `StatefulCondition`. Its subclasses permit to describe:

- `State`, to describe conditions on states, for instance, if the RELATED connections should be allowed as well (as for FTP data and connection flows) or ESTABLISHED, used to permit the TCP traffic back for communications that are allowed in one sense and that correctly terminated the TCP three way handshake;
- `limitRuleHits`, allows to describe conditions on bandwidth and other counters.

### 4.3.2 Parental control configuration

In SECURED, parental control can be enforced in two different ways: by using common secuirty controls, e.g. netfilter/iptables and Squid-cache together (packed as two simple PSAs or a complex PSA) or by using a specific PSA, that contains an ad-hoc and embedded logic. In the first case, the meta-model is the same described for filtering configuration (see Section 4.3.1). In the other case, a simplified model is provided. Figure 6 sketches the parental control specific configuration, where `parentalControlAction` class is a specialization of `ConfigurationAction`. The new action contains only an attribute (`enable`) that enable or disable the parental control features. Therefore, this approach does not permit to configure specific MSPL policies or low level rules of a PSA. On the contrary, policies and the related enforcement is managed by a third-party, e.g. the company that provides the application.

### 4.3.3 Data protection configuration

Data protection can be enforced in different ways: channel protection, static data protection and message protection. A secure channel is created agreeing on one or more symmetric cryptographic keys that are then used to cipher or authenticate data (using for instance a HMAC or keyed digest). A challenge is often performed in order to authenticate peers (nevertheless, the challenge is a feature of the

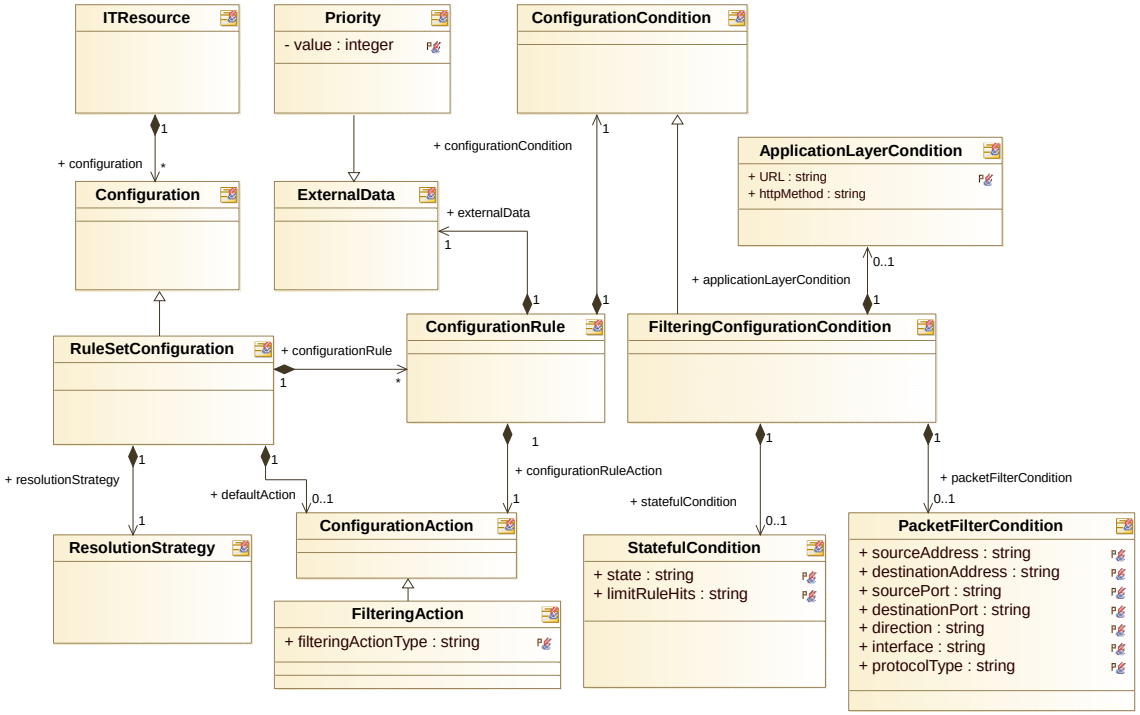


Figure 5: Filtering configuration.

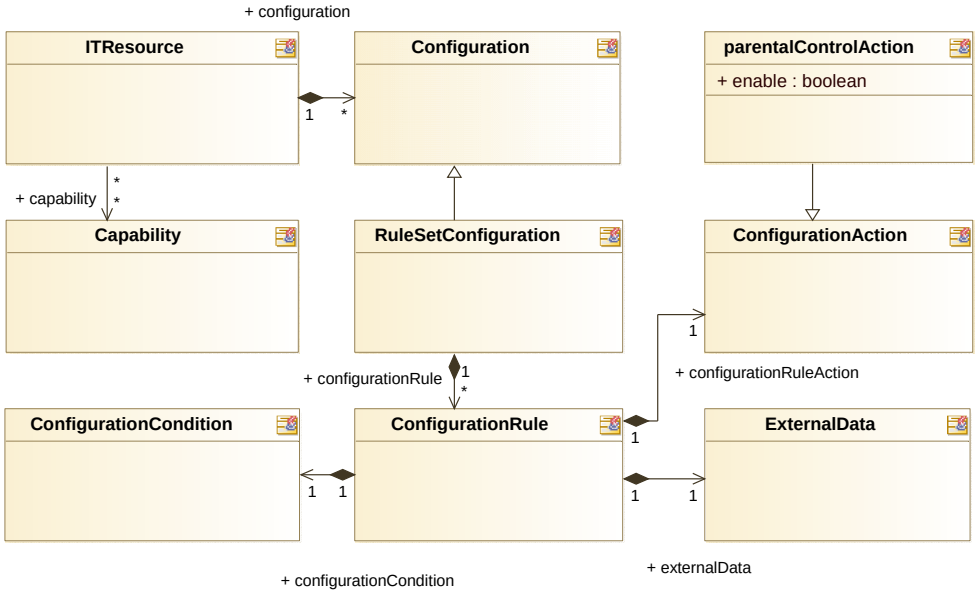


Figure 6: Parental control configuration.

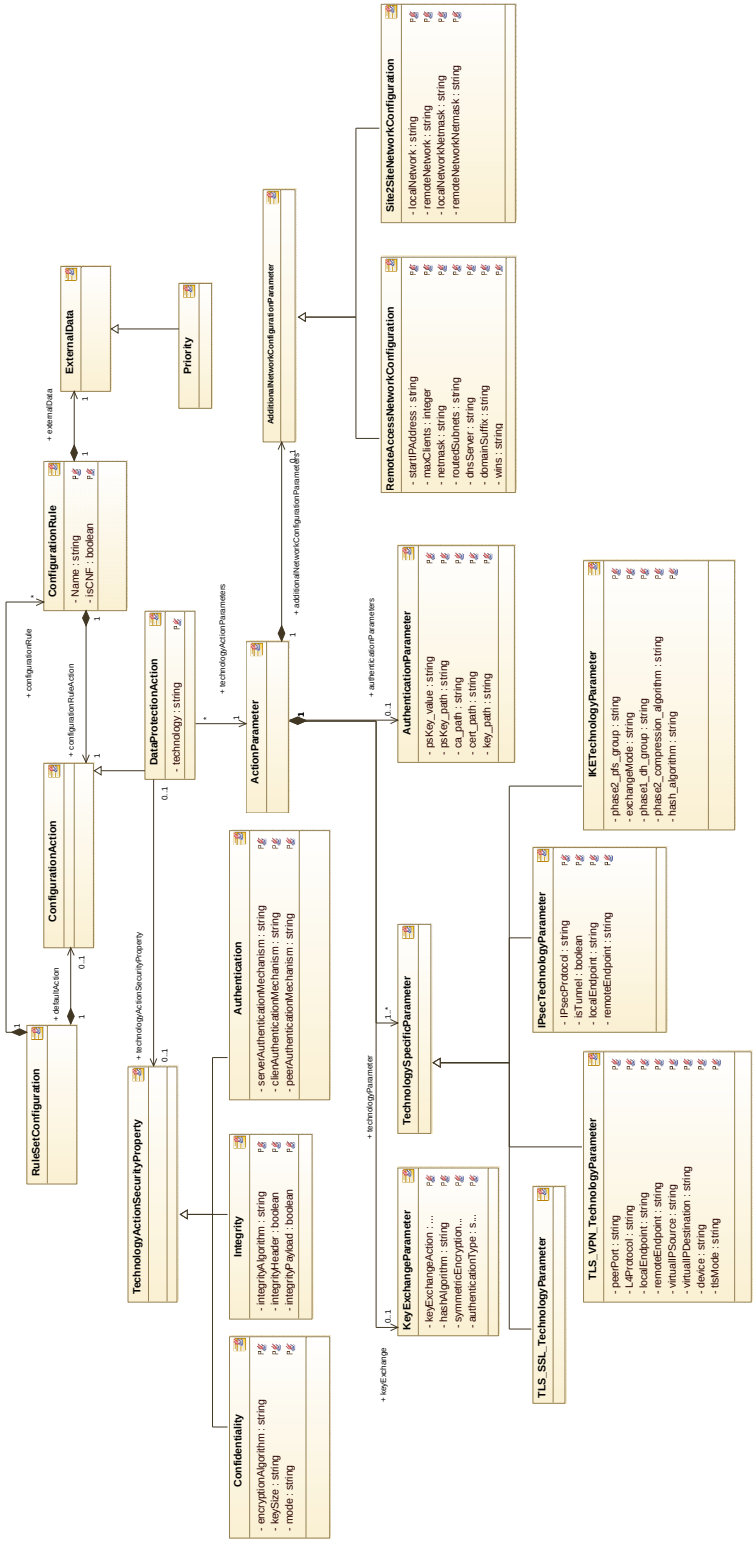


Figure 7: Data protection configuration (part 1).

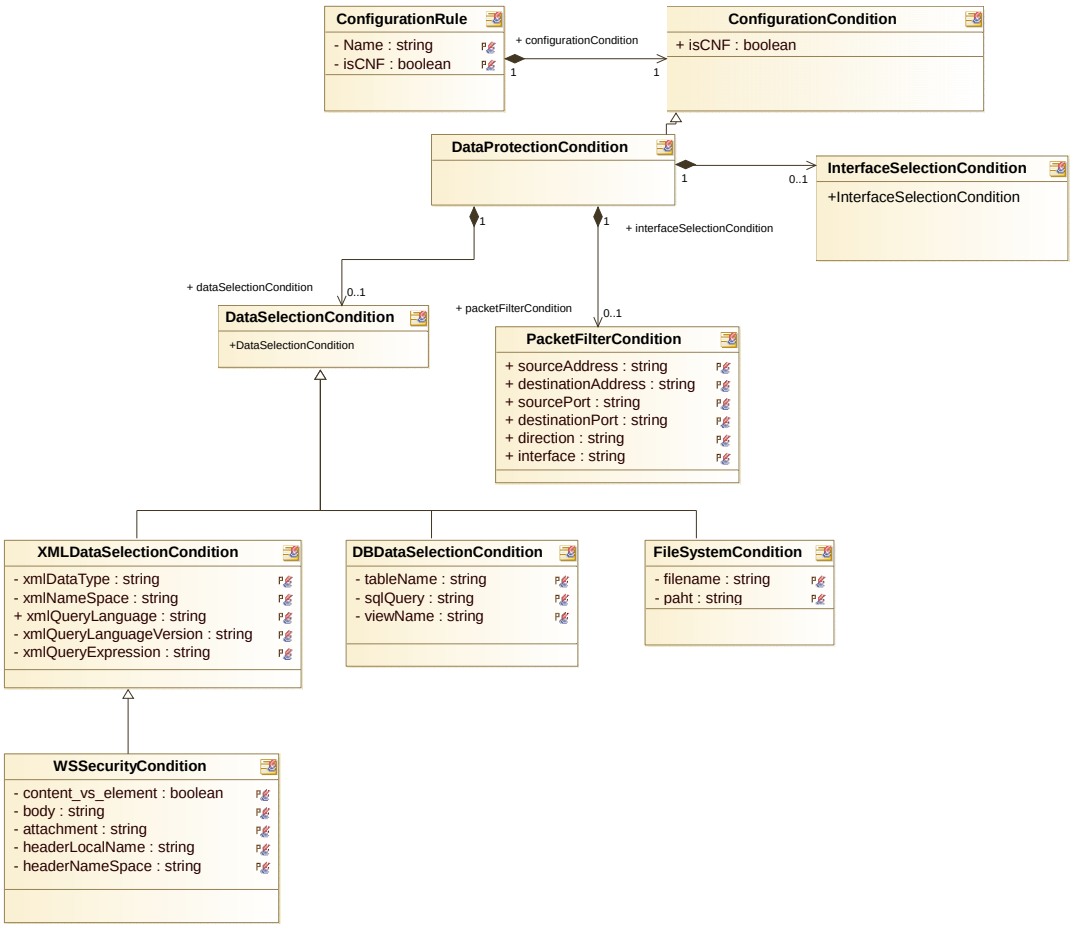


Figure 8: Data protection configuration (part 2).


authentication capability). Examples of protocols and standards able to create a secure channel are IPSec, WPA2, SSL/TLS, SSH. Additionally, there are techniques that can be applied to protect data to be transferred independently from the channel. This process is often named "message protection". Although the data protection model supports message protection, this feature is out of the scope of SECURED.

The SECURED configuration meta-model must be able to describe every type of data protection method in an abstract way. The specialization model defined to this purpose is the `DataProtection` configuration meta-model (Figures 7 and 8). Similarly to filtering configuration, `RuleSetConfiguration` is a specialization of `Configuration` class. To describe data protection configurations it is necessary to subclass the `ConfigurationAction` (Figure 7) and the `ConfigurationCondition` (Figure 8). The `DataProtectionAction` specializes `ConfigurationAction` and contains the attribute `technology`, used to convey precise information about the technology to which the abstract configuration maps to. Every `DataProtectionAction` is associated to other technology specific information conveyed by the class `ActionParameter`. The subclasses of that class are used to better characterize the parameters to apply when the used technology is known. In particular, `KeyExchangeParameter` (by the association `keyExchange`) permits to specify the key exchange action, the authentication type, the algorithms for hash and symmetric encryption. The class `AuthenticationParameters` (by the association `authenticationParameters`) contains information on pre-shared keys (values and path), CA (path), certificate (path) and key (path), typically used for IPSec and SSL/TLS VPNs. The abstract class `TechnologySpecificParamters` contains the specific parameters for different data protection technologies. This class is specialized as:

- `TLS_SSL_TechnologyParameter` that actually does not contain specific attributes;
- `TLS_VPN_TechnologyParameter` specifies attributes for configuring SSL/TLS-VPN, such as the one offered by OpenVPN [18]. In particular the port to adopt (e.g. 1194/TCP), local and remote endpoints (expressed as IP addresses or hostnames), device (e.g. tun0), TLS mode (e.g. tls-server, tls-client).
- `IPsecTechnologyParameter` specifies attributes for configuring IPSec. In particular the protocol (e.g. ESP, AH), tunnel or transport mode, local and remote endpoint (expressed as IP addresses or hostnames).
- `IKETechnologyParameter`, applied only to IPSec configurations, specifies IKE attributes. In particular, the exchange mode (e.g. main), hash algorithm (e.g. SHA1) and attributes for initialization phases.

The configuration of VPNs often requires information related to the network. By using the abstract class `AdditionalNetworkConfigurationParameters` (and the association `additionalNetwork-ConfigurationParameters`) two specializations are provided to describe remote access and site to site scenarios. The class `RemoteAccessNetworkConfiguration` specifies the attributes to configure remote access scenarios, including the IP addresses for clients, the routing information to reach other networks, DNS servers, etc.. Similarly, the class `Site2SiteNetworkConfiguration` specifies the network information on local and remote network, including IP addresses and netmasks.

The class `DataProtectionAction` also has a set of security properties (by using the association `technology/ActionSecurityProperties`) related to integrity, confidentiality and authentication. The class `Technology/ActionSecurityProperties` contains the subclasses of security properties, `Confidentiality`, `Integrity` and `Authentication`. The first class contains the details on integrity

algorithm (e.g. HMAC-SHA1), and Boolean value for integrity of header and payload (e.g. true when a property is required). The Confidentiality class has attributes to specify encryption algorithm (e.g. AES), the mode (e.g. CBD) and the key length in bit (e.g. 256). The third class, `Authentication`, specifies the authentication mechanism (e.g. pre-shared key).

Figure 8 describes the class `DataProtectionCondition`. In some cases, we need to explicitly select the data to protect. This can be done using explicit conditions ranging on the instances of the `DataSelectionCondition` class. Its subclasses, although out of the scope of SECURED, permit to select particular data types:

- `DBDataSelectionCondition`, that permits to select a portion of the database to protect using the attributes `tableName`, `ViewName` or directly via a SQL query using the `sqlQuery` or the viewName attribute;
- `FileSystemCondition`, that permits to select files, and directories using the `filename` and `path` attributes;
- `XMLDataSelectionCondition`, that includes a consistent set of attributes (`xmlDataType`, `xmlNamespace`, `xmlQueryLanguage`, `xmlQueryLanguageVersion` and `xmlQueryExpression`), to precisely select an entire XML file or a portion. This class is further refined subclassed to `WSSecurityCondition` to introduce more attributes (`headerNameSpace`, `headerLocalName`, `attachment`, `body`, and `content_vs_element`) useful to select data according to the WSSecurity OASIS standard [19].

In other cases, it could be necessary to protect all the data transferred between a service and all its clients, thus we will protect the interfaces. The (abstract) class used to select the interface to which apply data protection is the `InterfaceSelectionCondition`. This class is only used to group all the condition types that can be used to identify interfaces. Interfaces can be selected using the subclasses of the `PacketFilterCondition`, that could be useful to better select the traffic to protect. The details of `PacketFilterCondition` are available in Section 4.3.1.

# 5 Conclusions

As discussed in previous documents, the aim of SECURED is to offload security to PSAs that run on NED. Since the configuration of security applications is complex and not well understood by end-users, this document proposes the adoption of a user-oriented security policy language. This goal is achieved by defining two policy languages, respectively at high (HSPL) and medium level (MSPL). HSPL is suitable for expressing the general protection requirements of typical non-technical end-users. On the contrary, MSPL is designed for expressing specific configurations by technical users in a device-independent format.

This document contains the first specifications of HSPL and MSPL, and future deliverables can extend them if the need would arise.

# References

[1] E. Rissanen, “eXtensible Access Control Markup Language (XACML) Version 3.0”, 22 January 2013, http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.pdf

[2] B. Moore, E. Ellesson, J. Strassner and A. Westerinen, “RFC-3060 - Policy Core Information Model”, February 2001

[3] The POSITIF project (Policy-based security tools and framework), http://cordis.europa.eu/project/rcn/75115_en.html

[4] The PoSecCo project (Policy and Security Configuration Management), http://www.posecco.eu/

[5] N. Damianou, N. Dulay, E. Lupu and M. Sloman, “Ponder: A Language for Specifying Security and Management Policies for Distributed Systems”, Imperial College Research Report DoC 2000/1, 20 October 2000, http://www-dse.doc.ic.ac.uk/Research/policies/ponder/PonderSpec.pdf

[6] N. Damianou, N. Dulay, E. Lupu and M. Sloman, “The Ponder specification language”, IEEE Workshop on Policies for Distributed Systems and Networks, Bristol (UK), 29-31 January 2001, pp. 18-39

[7] M. Y. Becker, C. Fournet and A. D. Gordon, “SecPAL: Design and semantics of a decentralized authorization language“, Journal of Computer Security, 18(4), June 2010, pp. 619-665

[8] A. Lazouski, F. Martinelli and P. Mori, “Usage control in computer security: a survey”, Computer Science Review, 4(2), 2010, pp. 81-99

[9] J. Lobo, “CIM Simplified Policy Language (CIM-SPL)”, 14 July 2009, http://www.dmtf.org/documents/policy/cim-simplified-policy-language-cim-spl-100

[10] A. Uszok, J. M. Bradshaw, M. Johnson, R. Jeffers, A. Tate, J. Dalton and S. Aitken, “Kaos policy management for semantic web services”, IEEE Intelligent Systems, 19(4), July-August 2004, pp. 32-41

[11] L. Kagal, “Rei: A Policy Specification Language”, May 2005, http://rei.umbc.edu/

[12] L. Kagal, “The Rein Policy Framework for the Semantic Web”, June 2006, http://dig.csail.mit.edu/2006/06/rein/

[13] G. Tonti, J. M. Bradshaw, R. Jeffers, R. Montanari, N. Suri and Andrzej Uszok, “Semantic web languages for policy representation and reasoning: A comparison of kaos, rei, and ponder”, International Semantic Web Conference, Sanibel Island (USA), 20-23 October 2003, pp. 419-437

[14] S. Cantor, “Security Assertion Markup Language (SAML)”, March 2005, https://www.oasis-open.org/standards#samlv2.0

[15] D. Waltermire, S. Quinn and K. Scarfone, “Specification for the Security Content Automation Protocol (SCAP)”, NIST SP 800-126, rev.1, February 2011, http://csrc.nist.gov/publications/nistpubs/800-126-rev1/SP800-126r1.pdf

[16] The TOR Project, “The Onion Router (TOR)”, http://www.torproject.org

[17] SECURED consortium, “D5.1 - Specification of PSA and associated data and interfaces”, September 2014


[18] OpenVPN Technologies, “OpenVPN”, http://openvpn.net

[19] OASIS, “WS-SecurityPolicy 1.3”, http://docs.oasis-open.org/ws-sx/ws-securitypolicy/v1.3/ws-securitypolicy.pdf

# A Netfilter/iptables example

Figure 9 sketches an example of filtering configuration for Netfilter/iptables.

First of all this configuration adopts the first matching rule strategy and the default rule drops each packet. Therefore, to permit specific traffic (e.g. SSH) a specific rule must be defined. More in detail, the configuration contains three rules:

- `rule1` permits any host (also the ones of the public network) to reach port 22/TCP (i.e. SSH) of host 192.168.0.60. The rule is applied on interface WAN. Rule selectors (i.e. source IP address, destination IP address, etc.) are specified by using the attributes of `PacketFilterCondition` class. The `StatefulCondition` specifies that established, related and new connections are permitted. The `Priority` value (equal to 1) indicates that `rule1` is the rule with highest priority.
- `rule2` permits any host of the network 192.168.0.0/24 (i.e. from 192.168.0.1 to 192.168.0.255) to reach port 80/TCP (i.e. HTTP) of host 192.168.0.60. The rule is applied on interface WAN. As for previous rule, the selectors are specified by using the attributes of `PacketFilterCondition` class. The `StatefulCondition` specifies that established, related and new connections are permitted. The `Priority` value (equal to 2) indicates that `rule1` has precedence on `rule2`.
- `rule3` permits ICMP packets from any host to IP 10.10.10.1. The rule is applied on interface WAN. As for previous rule, the selectors are specified by using the attributes of `PacketFilterCondition` class. The `StatefulCondition` specifies that only 20 packets per second are permitted, i.e. to avoid Denial Of Service and ping flooding attacks. The `Priority` value (equal to 3) indicates that `rule2` has precedence on `rule3`.

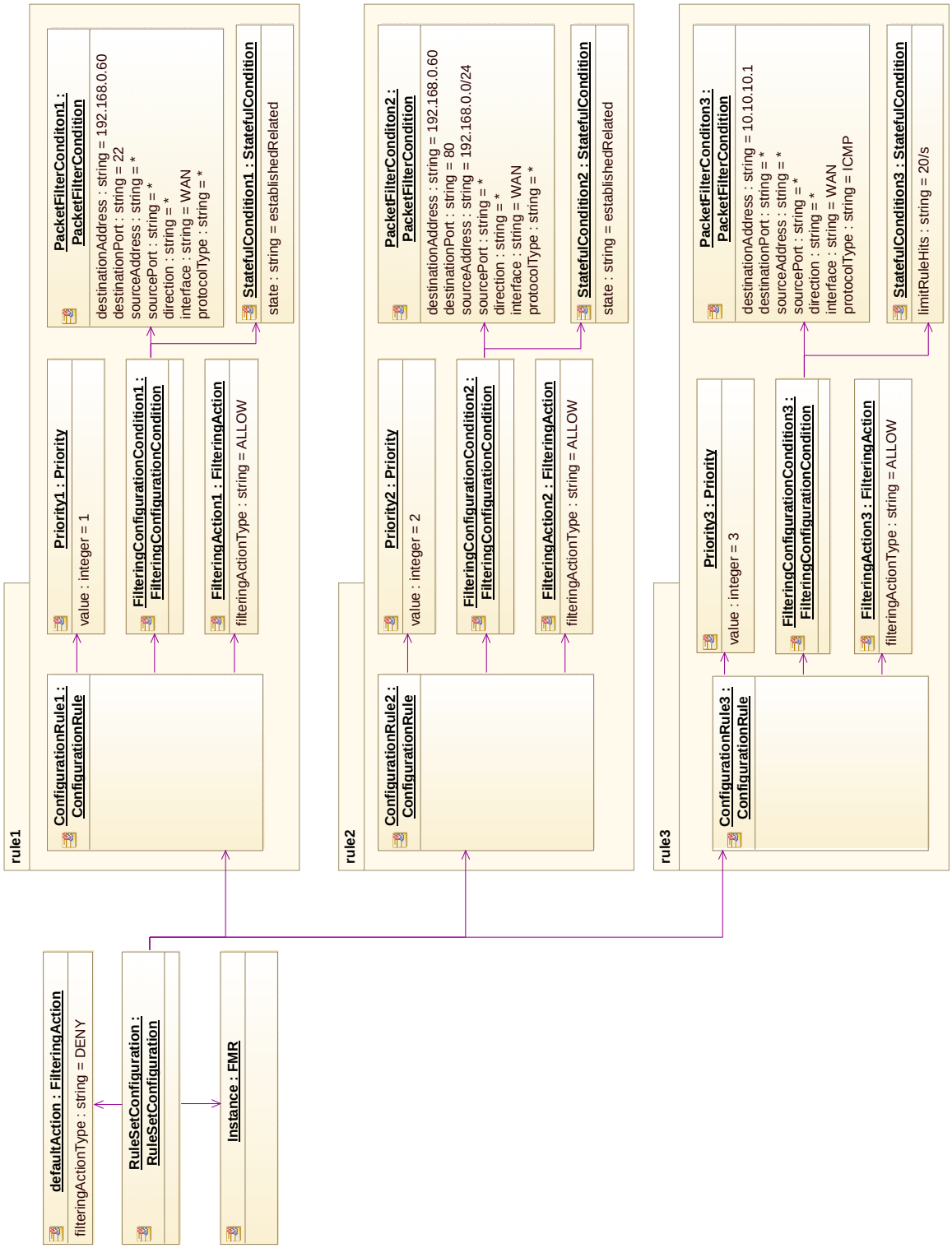


Figure 9: Netfilter/iptables configuration example.

# B Squid-cache example

This example (Figure 10) presents a typical configuration of a border firewall that supports the application level filtering for the HTTP protocol. We suppose that the border firewall is a Squid-cache. As usual, squid uses the FMR strategy; therefore, the rule order is important. The default action is calculated in a very peculiar way: it is set to DENY if the last rule enforces the ALLOW action, vice versa, the default action is ALLOW if the last rule enforces DENY. For better security and to reduce human mistakes, it is always suggested to insert as last rule the DENY all rule. The major difference between netfilter/iptables and Squid-cache configurations is the specification some attributes of class `ApplicationLayerCondition`.

More in detail, the configuration contains four rules:

- `rule1` Allow cache management (port 3128, any protocol type) only from localhost (i.e. 127.0.0.1). Similarly to netfilter/iptables configuration, each information is defined into `PacketFilterCondition` class. In this case, the attributes of class `ApplicationLayerCondition` is applied to any URL and any HTTP methods;
- `rule2` Deny HTTP access via all the ports which are not listed as safe. The safe ports are 21, 70, 80, 210, 240, 443 both TCP and UDP. Again, the attributes of class `ApplicationLayerCondition` is applied to any URL and any HTTP methods;
- `rule3` Allow HTTP access from machines on LAN, i.e. the subnet 192.168.0.0/24. Also in this case, the attributes of class `ApplicationLayerCondition` is applied to any URL and any HTTP methods;
- `rule4` Allow connections to Yahoo Messenger on port 5050 both TCP and UDP. In this case, the class `ApplicationLayerCondition` contains a restricted set of URLs (scs.msg.yahoo.com and cs.yahoo.co.jp) and only CONNECT method is permitted.

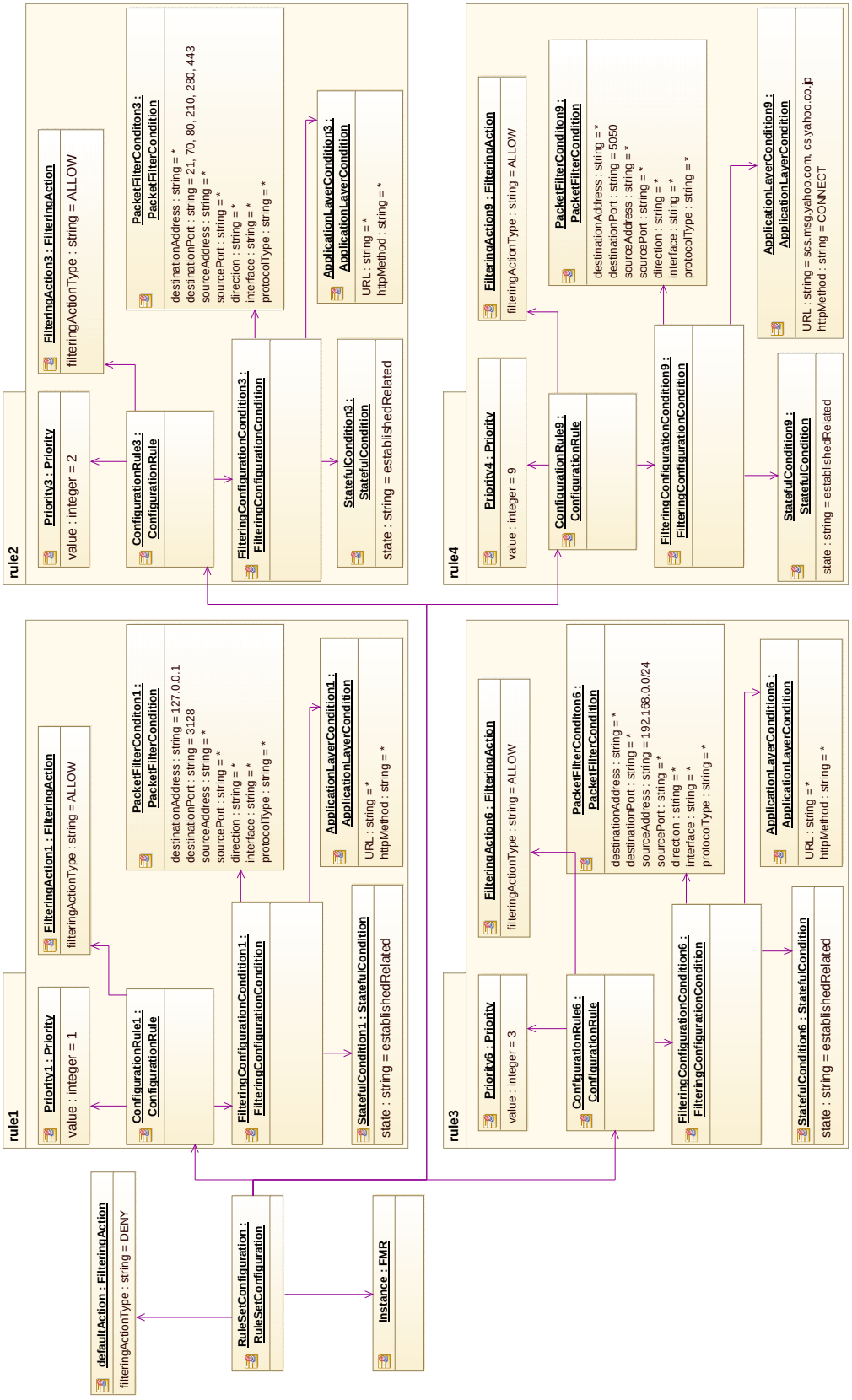


Figure 10: Squid-cache configuration example.

# C IPsec example

Figures 11 and 12 present an instantiation example for IPSec configured in remote-access. This mode is quite useful in SECURED to permit access and protect traffic of nomadic users in a corporate scenario. In particular, host2 identifies the PSA that contains the VPN client and r1 the PSA that contains the VPN terminator for the corporate. Although the VPN terminator is out of the scope of SECURED for the sake of completeness we also propose its configuration.

First of all, the action is to protect confidentiality of the traffic and it is achieved by defining:

- a peer authentication mechanism, in this case a X.509 digital certificate;
- a set of parameters for configuring confidentiality, in this case the use of AES CBC algorithm with a 256 bit key;
- a set of parameters for configuring integrity, in this case the protection of the payload by using HMAC-SHA1 algorithm.

In addition, as described in Section 4.3.3, the action of a DataProtection configuration is composed by a set of parameters. The `IPsectechnologyParameter` defines that the configuration adopts ESP protocol, tunnel mode, the local endpoint IP address (1.1.1.1) and that remote endpoint is anonymous (i.e. the client has a dynamic IP address). The `RemoteAccessNetworkConfiguration` specifies the network settings, in this case the client IP addressing starts from 192.168.1.1 to 192.168.1.10 (i.e. 10 clients), the netmask is 255.255.255.0 (/24) and the local dns server is available on 192.168.0.1. The `IKETechnologyParameter` contains the set of parameters to configure Internet Key Exchange features, typically phase1 and phase2 algorithms (e.g. modp1024, modp768), exchange mode (e.g. main) and hashing algorithm (e.g. SHA1). The `AuthenticationParameter` defines pre-shared key and paths for digital certificates.

Finally, the differences between the configuration of the client (host2) and the VPN terminator (r1) are only in the `IPsectechnologyParameter`. Considering the example of Figure 12 this class contains the remote endpoint IP address (1.1.1.1), the tunnel mode (as for r1) and the protocol (as for r1).

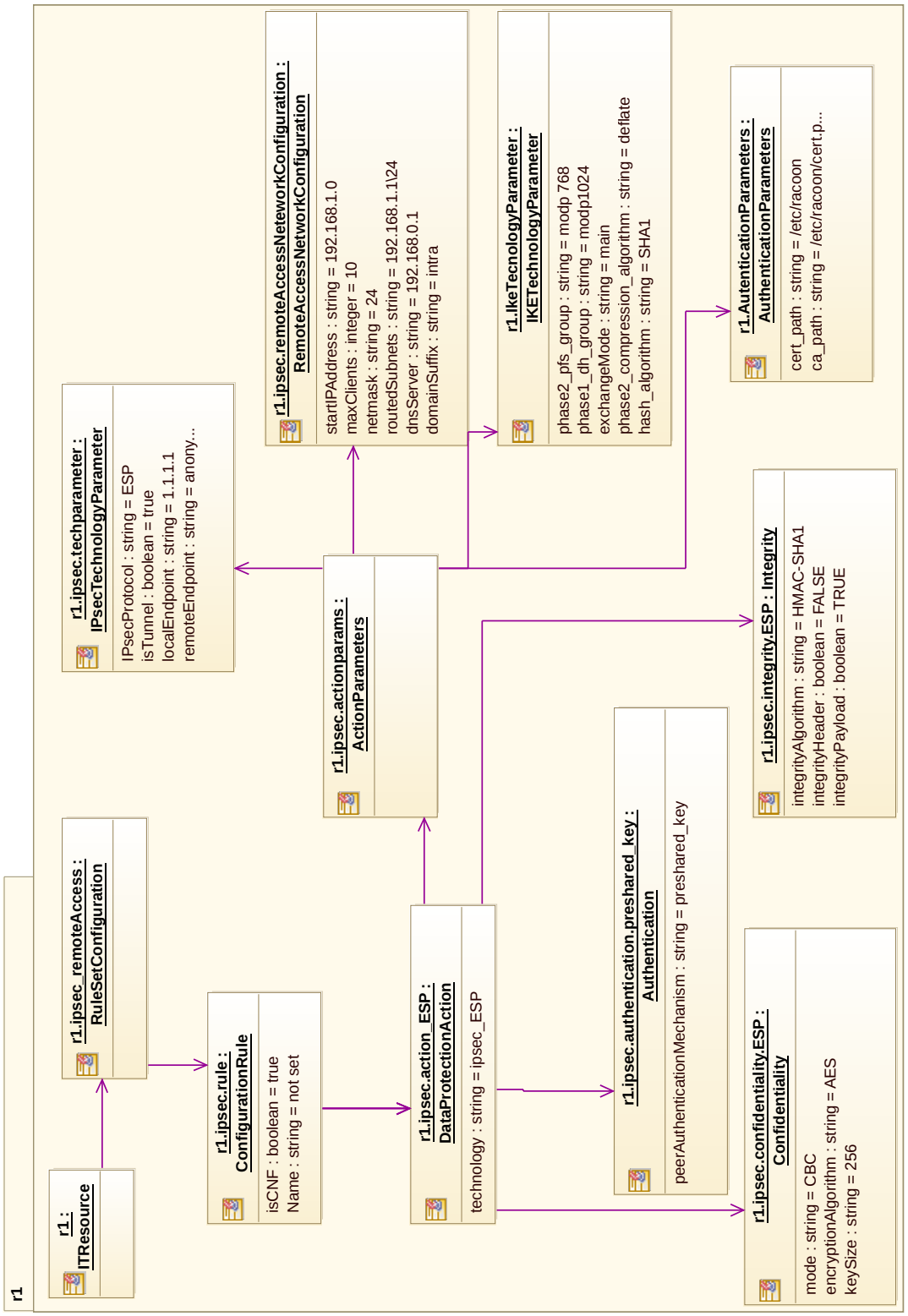


Figure 11: IPSec remote access configuration example for r1.

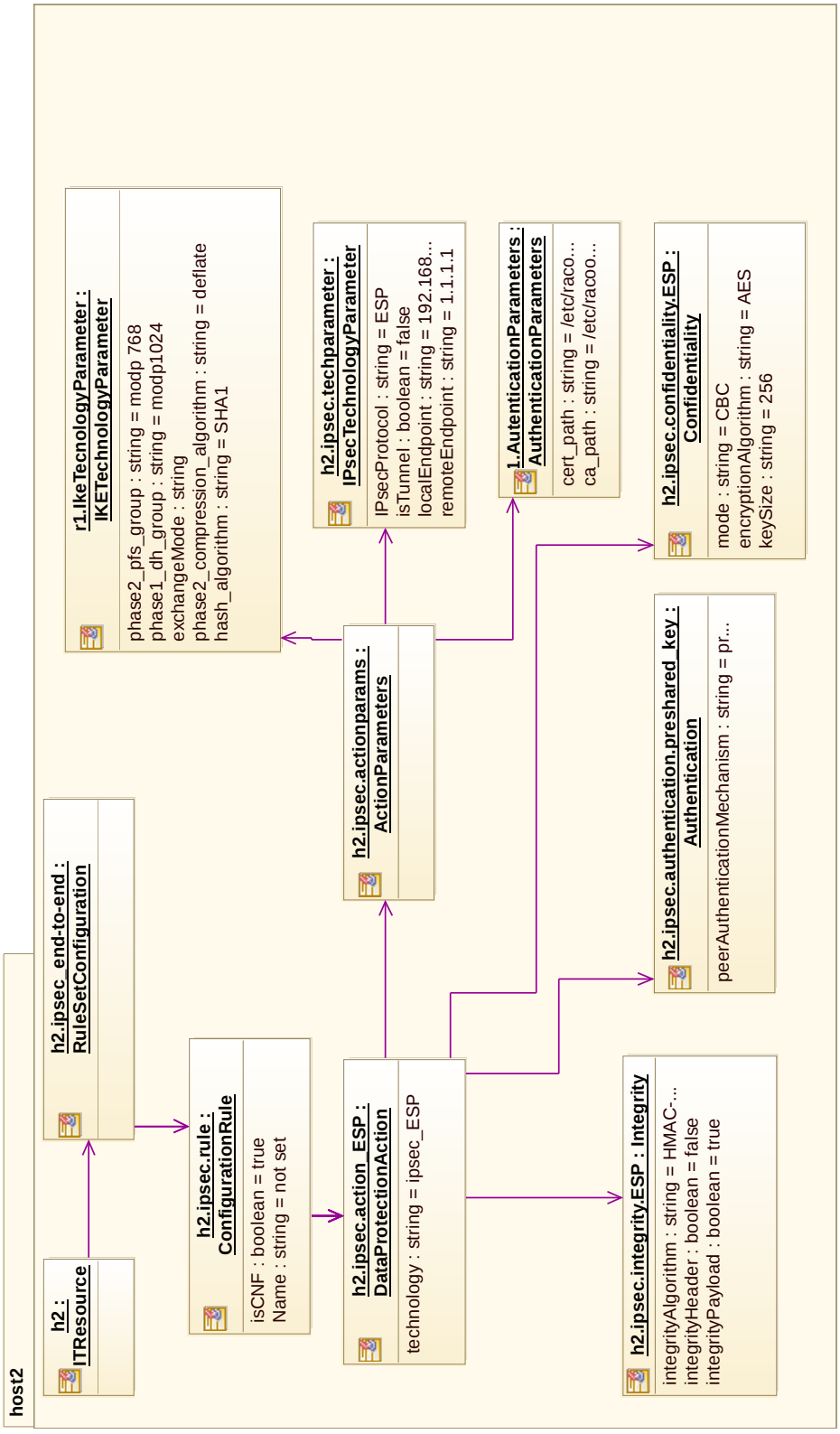


Figure 12: IPSec remote access configuration example for host2.