D5.1

Specification of PSA and associated data and interfaces

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	D5.1
Version (date)	v1.1 (18/7/2014)
Work package(s)	WP5
Due date	30/6/2014 – M9

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

Abstract	This deliverable describes the requirements and the solutions for modelling technical and architectural aspects of PSA, considering its functionalities, interactions with other components of SECURED and its execution. The current version of the interfaces and related operations (API) are provided.
Keywords	PSA, architecture, specification, API

Editor
Marco Vallini (POLITO)

Reviewers
Adrian L. Shaw (HPLB)
Antonio Lioy (POLITO) – quality control

Contributors
Jarkko Kuusijärvi (VTT)
Jouni Hiltunen (VTT)
Antonio Pastor (TID)
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	09.07.2014	Base version	M.Vallini
v0.1	18.07.2014	Quality control	A.Lioy

Executive Summary

This document defines the general architecture for a SECURED Personal Security Application (PSA), which processes all network traffic of a specific user and executes on a Network Edge Device (NED). Starting from the requirements, the document proposes the solutions for modelling technical and architectural aspects of PSA, highlighting functionalities (e.g. security capabilities) and interactions with other components of SECURED.

Finally, it proposes the current version of the APIs, which are exposed to external components of SECURED, a set of development requirements and procedures for creating, deploying and integrating security applications in SECURED project. However, the specification of APIs may subject to change in future versions.

Contents


SECURED

1 Introduction

The SECURED project proposes to achieve the protection from Internet threats by offloading execution of security applications into a programmable device at the edge of the network (NED), such as a home gateway or an enterprise router. Within the NED, each user will be provided with a Personal Security Controller (PSC), a controller that will control and manage the user's security applications at the edge of the network. The PSA is the entity that implements security features by processing user's traffic and typically performing operations at different network layers (e.g. blocking a packet depending on its content, or MAC/IP addresses). Therefore, to perform the security offloading in a complex architecture, which is built upon several other components, the design of the PSA must be addressed carefully.

The objective of this document is to define the requirements and a set of solutions for modelling the technical and architectural aspects of PSA, considering its functionalities, interactions with other components of SECURED and its execution.

First of all, we need to design a PSA architecture that suits the NED specifications, i.e. the environment to execute, manage (e.g. configure, forward traffic) the security applications. The details on these specifications are available in D3.1.1 [1]. Considering the PSA architecture, we identify three main requirements:

- Generality: must be suitable for all the types of security controls (e.g. firewall, web-proxy);
- Flexibility: needs to support complex designs, for example traffic chaining (to forward traffic among different PSAs following a particular order);
- Security: must be secure, the management operations must be separated from user's traffic to maintain adequate segmentation and isolation.

Second, considering the offloading of security, we analyse the following requirements: identification of security functionalities (named security capabilities) to process (e.g. forward, discard) and protect (e.g. encrypt, decrypt) the user's traffic; identification of PSAs and related security controls, i.e. the set of mechanisms to implement a security capability (e.g. a specific tool) to enforce security policies; the definition of a process to transform MSPL (Medium-level Security Policy Language) into specific configurations for the security controls. Considering security and network operations, several capabilities are available, therefore a classification (e.g. access control, malware detection) is needed. More in details, each security capability is implemented in a PSA by using a security control (e.g. netfilter/iptables for packet filter capability). Therefore, each security control must be configured to correctly enforce security policies. Often, a wide set of PSAs (with different security controls) could implement a security capability, therefore we need to define a set of criteria to choose amongst them. A typical criterion is performance, e.g. network throughput, which could affect traffic latency. For example, in a corporate scenario, the network throughput plays a fundamental role while in a home scenario it is typically less important.

Third, the usage of PSAs in SECURED introduces additional requirements. One of the most important requirements is the dependencies on the execution environment (EE), i.e. the set of parameters and components needed to run a PSA inside the EE. Examples of these parameters are hardware/software requirements (e.g. operating systems, required libraries), mobility support, security (e.g. remote attestation). Sometimes a PSA could stop working, therefore, the adoption of monitoring functionalities will help to identify the problem and propose a remediation (e.g. restarting the application).

Therefore, to define and organize security functionalities, features of an application, environment parameters, a "PSA manifest" is proposed.

Finally, two other requirements must be examined: the interfaces between PSA and external world (e.g. PSC) and the development of new applications. The interfaces that a PSA exposes to external components of SECURED must be defined according to the general architecture of the project. One of the aims of SECURED is to encourage external actors to develop new applications.

This document is organized as follows:

Section 2 introduces the PSA concept providing its definition and a general architecture considering different aspects. In particular, it discusses the role of the PSA and related approaches to generate specific configurations for a security control from MSPL.

Section 3 discusses the computational models to execute a security application, considering Virtual Machine PSA, Container PSA Model and Application PSA models.

Section 4 defines the security capabilities, organized by categories, e.g. privacy, audit.

Section 5 highlights the interactions of a PSA with the general architecture of SECURED, specifying the available operations which can be performed to a PSA.

Section 6 defines the "PSA manifest" that models PSA at a conceptual/abstract model using a semi-formal structured language. The manifest contains the information to describe functionalities, PSA execution model, PSA configuration, etc.

Section 7 specifies the API which is exposed to external components of SECURED, for example to the PSC.

Section 8 discusses a tentative guide to PSA developers for creating, deploying and integrating security applications in the SECURED architecture.

A PSA is a fundamental component of SECURED that processes all network traffic, it is executed by the NED and must interact with other components (e.g. PSC). Therefore, the operations to handle the traffic and to control the PSA (e.g., to enforce a configuration) must be designed accordingly and implemented by using a set of interfaces. In addition, to ensure a secure design, management operations must be separated from end-to-end traffic to maintain an adequate segmentation and isolation. The PSA architecture must also be flexible enough to support complex design, for example, the aggregation of security controls (e.g. to optimize performance).

To protect users' network traffic a PSA adopts set of security controls, e.g. netfilter/iptables. Each of them implements one or more capability, i.e. the ability to perform some task (e.g. filter packets at layer 4 of the ISO/OSI stack). In SECURED, security policies are defined by using High-level Security Policy Language (HSPL) for expressing concepts related to end-point protection. This language is suitable for capturing the user requirements but cannot be directly implemented by security controls. Therefore we perform a transformation step to create a Medium-level Security Policy Language (MSPL) which conveys the same information in a format suitable for configuring security controls, typically an ordered sequence of permit and deny actions related to matching packets or payloads. This transformation, named *policy refinement*, generates an abstract configuration (represented by using MSPL) for a generic PSA. However, PSAs have different capabilities (e.g. packet filtering, deep packet inspection), therefore, we need a precise mapping to bind together abstract security requirements (HSPL), PSAs security capabilities and abstract security configurations (MSPL). In particular, during the refinement we need a mechanism able to identify the related set of security capabilities needed to enforce the policies by using a set of PSAs (e.g. packet filter PSAs). 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 state, identifying the MIME type of a payload), but expressed in a generic syntax. In order to use it for configuring an actual security control, we need one more translation step (performed by M2L service) to the actual language used to write the configuration of the specific target control. This can be seen as a low-level security policy language, but actually it is not a single language, rather the collection of all the languages used for configuration by the various security controls.

Finally, another objective of SECURED is to create an open-source architecture for hardware manufacturers and software developers. This makes the creation of marketplaces possible, where applications can be download on-demand from a repository. This can be personal, corporate, provider, or public. Therefore, to support the advancement of marketplaces, the project will provide the necessary development environments (e.g. IDE tools) and services to create and deploy the applications.

1.1 Definition of the PSA

A PSA is the component within the SECURED architecture which actually processes the network traffic of a SECURED user. It is responsible to enforce a security policy as specified by the user.

A PSA is composed by one or more security controls, where each of them implements one specific security capability. Security capabilities include access control, malware detection, privacy protection, monitoring, legal interaction, and forensic analysis. For more details see Section 4.

The order in which different security controls, within a PSA, are interconnected is predefined and known to the SECURED architecture. The configuration for each security control is generated by the SECURED architecture according to the users' security policy.

To enforce the desired security policy of a user the SECURED architecture may need just one PSA or must interconnect multiple ones. The decision on how many and which PSAs are needed is taken by the SECURED architecture.

For example the security policy *Parental Control* can be enforced by one PSA developed specificity for this task (*Parental Control PSA*) or by interconnecting multiple PSAs. The *Parental Control PSA* is composed of three security controls (packet filter, log and web proxy), otherwise three interconnected PSAs which expose the same security controls are needed. The packet filter drops everything other then web traffic, the log keeps track of the connection attends and the web proxy filters all websites not present on a white-list.

Different PSAs of a single user are isolated to each other and coordinated by the PSC of the user. A PSA is implemented for one specific execution environments and therefore, one single user may have multiple execution environments associated to him in order to satisfy all the needed PSAs.

The current state of each security control in a PSA is accessible by the SECURED architecture. This is necessary to know which of the user-defined security policies are in effect, and to allow mobility of the user by being able to migrate the state of the security control, if necessary. When a user changes NED, the state of all his PSAs is retrieved from the current NED and applied to the newly connected one.

The SECURED architecture is open for developers to implement new PSAs by using the SECURED public APIs. Finally, PSAs can be developed from scratch or based on existing security applications. In both cases, the developer need to implement the SECURED-specific components and interfaces. For more details see Section 7 and Section 8.

1.2 General PSA architecture

2 Description, functionalities and requirements

As introduced before, a set of requirements for the PSA architecture has been identified. First of all, the architecture must be general enough to support different types of PSAs. For example, must be suitable for modelling a wide set of security applications, such as firewall, web-proxy, anti-virus. Second, the architecture must be flexible enough to support complex design, i.e. interconnection of several PSAs in different order. For example, it must support the traffic chaining to forward traffic among different PSAs following a particular order. Finally, its design must be secure, i.e. the management operations must be separated from user's traffic to maintain an adequate segmentation and isolation.

With these requirements in mind, the PSA can be divided into two planes: "Data Plane" (PSA_D), which handles the traffic and performs operations on it, and "Control and Management Plane" of the PSA itself (PSA_C), which enforce configurations settings and also offer additional Management interface of the specific PSA. This is a common model accepted and standardized (ITU X.805 Security planes [2]) to deploy applications with a secure design, where management, control and end-to-end traffic are separate planes to maintain a adequate segmentation and isolation.

The Control and Management plane (PSA_C) will cover all OAMP (Operations, Administration, Maintenance, and Provisioning) functions including low level configuration of policies in the applications (for security controls), and also guarantee correct supervision and efficient operation of the PSA. One typical example can be a firewall PSA where PSA_C will enforce user policies translated into netfilter/iptables commands and also offer a channel to check status, integrity and correct operation.

Data Plane (PSA_D) handles in general the service traffic, that is, the end-to-end traffics data flows. This data plane is composed of a "Online User Traffic" interface, which is the user's traffic flowing through the network and passing through the PSA to be manipulated by the logic, and "Offline Traffic" interface, which is the auxiliary traffic for supporting PSA operations. The "Offline Traffic" interface can be: unidirectional, typically for update operations(e.g. software updates) or bidirectional, typically to perform complex operations by invoking external service or forwarding traffic to another PSA. One example can be an IDS PSA where the user's traffic is inspected by an engine (e.g. Snort) through "Online User internal/external" traffic interface (OnUIT, OnUET) in the PSA_D plane, and the "Offline Traffic" interface (OffT) represents updates of signatures rules of the IDS. Considering the malware detection, two solutions are available. In the first case, a PSA can demand to an external service the analysis of signatures. This scenario has at least two benefits: the execution of the PSA requires less computational resources at NED, the PSA does not require a signature database and related updates. On the other hand, the PSA relies on external service, increasing the bandwidth and potentially the latency of analysis. However, the adoption of caching techniques could improve performance and reduce bandwidth. In the second case, a PSA is connected, by using "Offline Traffic" interface, to other PSA that performs malware analysis. For example, several PSAs can be installed to provide analysis for different types of traffic. This solution requires more computational resources but increases performance. Finally, another advantages is that the PSA(s) for malware analysis could be shared among users (e.g. corporate scenario). Therefore, an "Offline Traffic" interface can be connected only to external services (e.g. update center) or to another "Offline Traffic" interface.

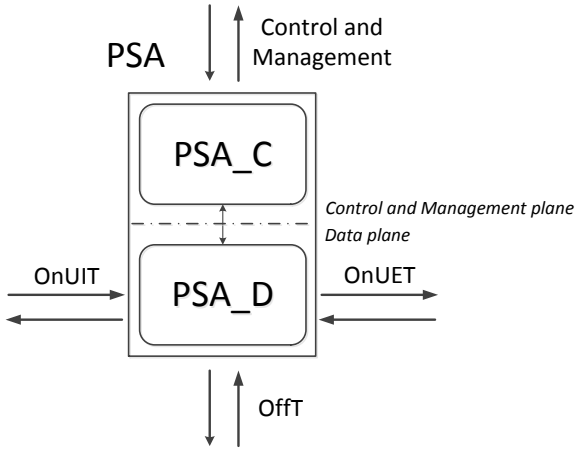


Figure 1: PSA architecture

2.0.1 Examples

Figure 2 shows different instantiations of the general architecture for different purposes. The example in Figure 2a is used for copy, audit, parsers and packet decryption. All the traffic coming from the user terminal (OnUIT) is copied, one flow is redirected to the Offline Traffic interface (OffT) and the other one is redirected to the Online External Traffic Interface (OnUET). The example in Figure 2b is used for IDS. All the traffic coming from and going to the user terminal is copied, one flow is redirected to the Offline Traffic interface and the other one is redirected to the Online External Traffic Interface. The example in Figure 2c is used for access control, IPS, anti-malware and phishing protection. All the traffic coming from the user terminal is categorized and according to the configured policy is blocked on the Online External Traffic interface. The example in Figure 2d is used for data cipher, data anonymizer and VPN. All the traffic coming from the user terminal is encrypted on Online External Traffic interface, and all traffic going to the user terminal is decrypted on Online External Traffic interface. The example in Figure 2e is used for malware detector. Files are send over the Offline Traffic interface to be analysed.

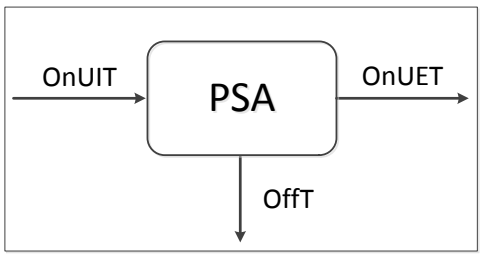
2.0.2 Complex PSA and chaining

One of the main advantages of this PSA architecture is that PSA logic can have a flexible complexity. We can design an advanced logic to manage the user traffic offering security capacities or defining simple logic gates to enforce direct access policies, or even reuse legacy security software as the logic of PSA. A PSA with a single security control is defined as *simple PSA*. A PSA with more than one security control (e.g. netfilter/iptables and squid-cache proxy) is defined as *complex PSA*.

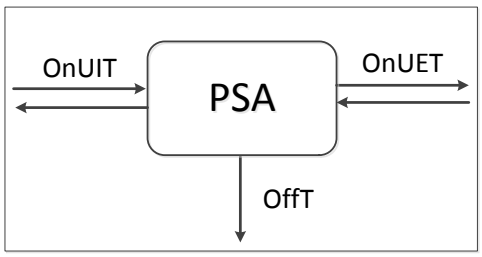
Therefore, when more than a security control is required to enforce the policies, two solutions are available: create a complex PSA or perform a PSA chaining (built upon a set of simple PSA).

In models with simple logic PSA, a complex security policy or advance security service can be achieve with PSA packing (*complex PSA*), where the PSA integrates more security controls. This solution has less overhead and typically requires less computational resources.

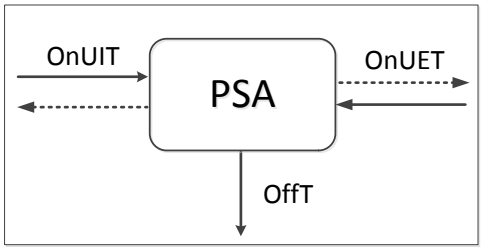
Otherwise, a set of PSAs can be chained in different ways, typically connecting PSAs by following serial or parallel model. This solution requires more computational resources but typically has better performance (e.g. the processing of flows can be parallel). This is the accepted concept of Service Chaining (IETF SFC) and usually it is represented as forwarding graphs.



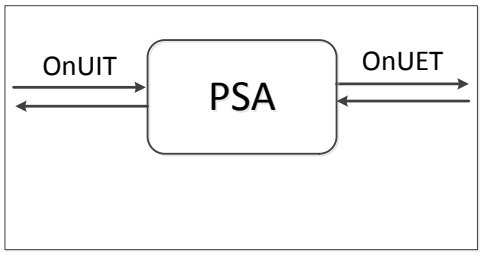
(a) copy, audit, etc.



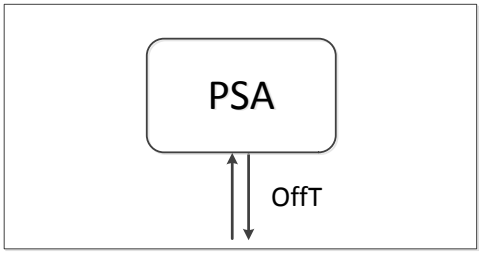
(b) IDS



(c) access control, IPS, anti-malware, etc.

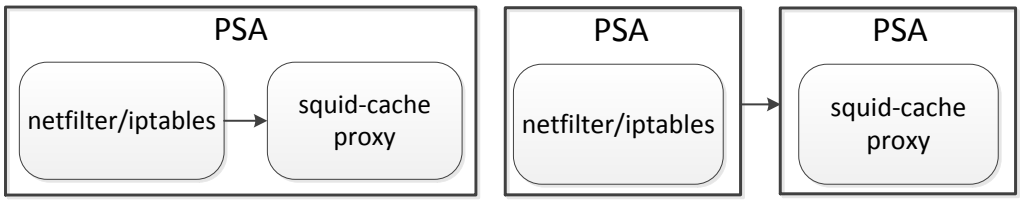


(d) data protection (e.g. anonymizer, VPN)



(e) malware detector

Figure 2: Examples



(a) complex PSA (b) chaining of simple PSAs

Figure 3: complex PSA vs chaining PSAs

Let's assume that a policy requires a packet filter (e.g. netfilter/iptables) and a web proxy (e.g. squid-cache proxy) to be enforced. Figure 3 depicts the available solutions. In the first case, a complex PSA is provided (Figure 3a) i.e. both security controls are integrated in the same PSA. In the second case, the chaining of simple PSAs (Figure 3b) highlights that each security control is installed in a single PSA.

2.1 PSA-specific translation

In SECURED, an abstract security configuration (MSPL) must be translated into a concrete security configuration for a specific PSA. Since a concrete configuration depends on the settings used by each specific PSA (e.g. two packet filters offer the same security capability but may be configured via two completely different languages) a translation operation is required.

This is done by invoking the M2L service (MSPL to low level configuration) associated to each PSA: it transforms a statement expressed in MSPL to configuration settings required by the security control (e.g. netfilter/iptables).

The configuration files will contain *back pointers* to statements written in MSPL and HSPL policies (and the related policy stack elements) so that at any time during activity of a PSC the user will be able to know which policies are in effect and who (beyond himself) required them. This mechanism ensures the tracking of the entire policy stack from HSPL to real configurations.

The translation process from MSPL to specific PSA configuration is simpler than the previous one from HSPL to MSPL because it uses a "syntax mapping" approach to translate a MSPL configuration into a security control specific language without adding any information.

2.1.1 Translation service

To support a wide set of low-level security controls, the translation must be designed to support different languages (e.g. netfilter/iptables or PF for a stateful firewall). A feasible approach is to develop a general module (with related API) that takes as input MSPL statements, optionally the type of security control (if it supports more than one security control) and invokes a plugin to generate the specific configuration. Each plugin must contain the logic to perform the translation for a specific security control implementation.

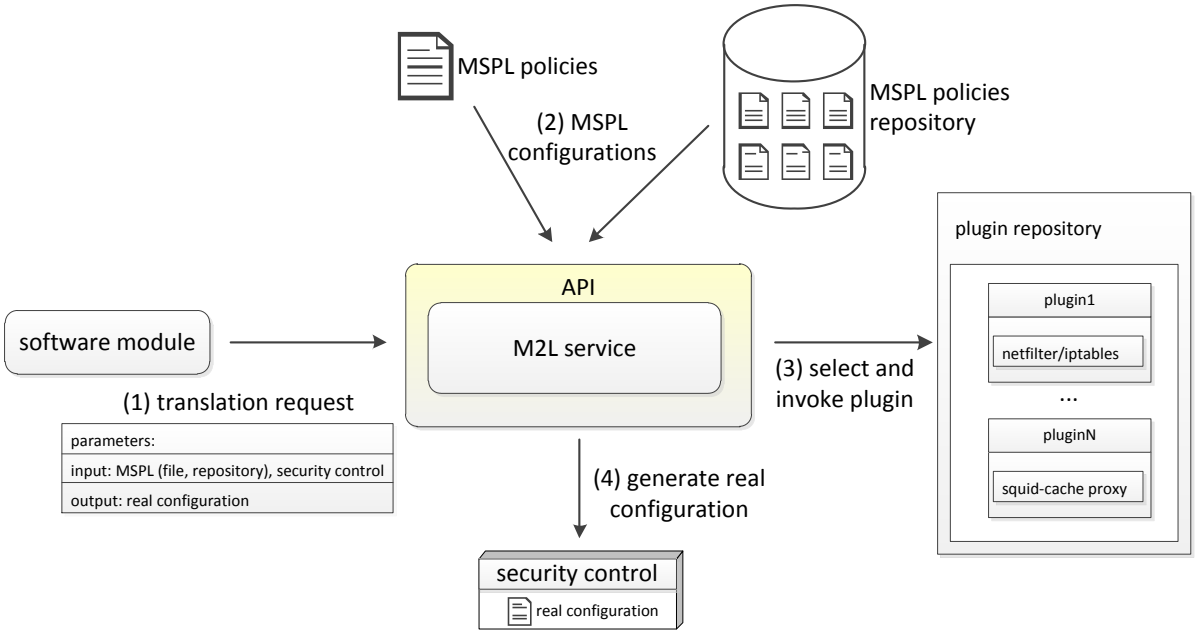


Figure 4: Translation process logic

In SECURED, this task relies on the M2L service, where its logic is depicted in Figure 4. A software module invokes the service by specifying the MSPL configuration and the security control. The M2L selects and invokes the corresponding plugin to perform the translation and returns the configuration.

In detail the M2L service offers an interface to provide the translation. This interface exposes the following operation:

- **translate** that takes as inputs: the MSPL (given by file or by a reference to MSPL policies repository) and the target security control (e.g. netfilter/iptables). The M2L contacts the plugin repository and downloads the plugin to perform the translation from MSPL to configuration settings for security control. Then, M2L returns the generated configuration.

Therefore, the repository offers an interface with the following operation:

- **getPlugin** that takes as input the type of security control and returns the plugin to perform the translation.

A plugin is designed to be compact and efficient, therefore it offers a single interface with the following operation:

- **getConfiguration** that takes as input the MSPL (given by file) and returns the generated configuration as a file.

For example, if a specific PSA contains the implementation of two security controls (netfilter/iptables for packet filter and Openswan for IPsec) two plugins are required to translate packet filter MSPL and IPsec MSPL into the implementation specific languages for netfilter/iptables and Openswan. When the PSA is instantiated and needs to be configured, the M2L service receives two requests: the former with a MSPL configuration for packet filter and the latter with a MSPL configuration for IPsec. In the

first case the M2L service invokes the packet filter plugin of the PSA and creates the configuration for netfilter and, in the second case, it invokes the IPsec plugin to create the configuration for OpenSwan.

Initially we use a RESTful service to implement the operations of M2L service and plugin repository.

By following this approach, a developer that want to create a new PSA, must only develop the plugin if not yet available, to translate MSPL to the low level configuration for the PSA's security abilities. A plugin can be implemented as binary file, script or by using XML transformation techniques.

2.1.2 Versions of translation service

As the API for M2L, plugin and repository will likely evolve over time, the various versions are maintained on-line at the following page:

https://www.secured-fp7.eu/ref/M2L_translation/.

3 Categorization by computational model / execution environment

As introduced before, a PSA is the entity that process the user's traffic. However, several execution models can be followed to run a PSA. This section discusses the utilization of three most prominent PSA models: Virtual Machine PSA model, Container PSA model and Application PSA model. These models are considered in the alpha specification of the NED (see D3.1.1 [1]). Another model that could be supported in the future is Para-virtualization PSA model. Para-virtualization PSA model could be seen as enhancement to Virtual Machine PSA model which offers an custom interface to virtual machine to enable more optimal performance in cost of a need to tailor virtual machines to support the custom interface. Application PSA model in turn could provide performance enhancements by reducing the size of the PSA (and help mobility/migration of PSAs in some deployment scenarios), but at the same time make the PSA more complicated to develop and possibly to use in different NEDs.

3.1 Virtual Machine PSA model

In Virtual Machine PSA model the PSAs, which contain one or more security controls, are running within isolated execution environments that are each facilitated by a virtualized OS instance. Usage of common virtualized OSs will ease porting of security legacy applications to SECURED architecture as long as the execution environment supports SECURED management and control interfaces between PSC and PSA as well as chaining of the execution environments.

Each PSA will have a set of software and hardware dependencies towards the VM they are running in. The hardware dependencies could include, CPU, storage and memory requirements while the software dependencies include compatibility with hypervisor.

Thus the hypervisor to PSA interface has a major importance in setuping, starting and stopping the execution environments for PSAs in Virtual Machine PSA model. The setup requirements are provided by the PSA, e.g. in a manifest file, that essentially defines the minimum requirements that PSA needs in order to be executed. Giving each PSA the minimum requirements is not obviously the definite goal for NED resource allocation since allocating more resources could lead to better PSA performance e.g. in terms of processing delays. Therefore, dynamic resource allocations among PSA in the Virtual Machine PSA model have benefits in terms of overall NED performance. The hypervisor is naturally responsible of making such allocations which stipulates also real-time monitoring of the virtual OS resource usages. The dynamic runtime resource allocation is not something that state-of-the-art virtual

machine hypervisors take for granted (although heavy cloud computing platforms offer this feature), which indicates a complexity problem not readily to be implemented in NED. However, the dynamic resource allocation can simply be overcome by stop-setup-start procedure, although PSA service continuity restricts the frequency and thus granularity of the achievable dynamic allocations.

Reference implementations for enabling the Virtual Machine PSA model exist in open-source domain that give a hint of best-practices that could be followed. Nowadays the standard approach to such reference implementations is through terminology used in cloud computing in which resources such as processing power, storage, network and software are abstracted and provided as services that can be accessed remotely. From the high-level NED perspective, a reference is needed for Infrastructure as a Service (IaaS) model that enables facilitating PSAs in terms virtual machine disk image libraries, file storages, virtual networks, load balancers and so on. On the other hand, what PSA needs is basically a Platform as a Service (Paas) that provides the needed execution environment without interaction with hardware resource allocations processes.

One of the popular options for such reference implementation is OpenStack [3] that supports both IaaS and PaaS approaches. OpenStack is a collection of open-source projects that enable setting and running cloud computing infrastructure. There are five main service families provided, namely Compute (Nova), Storage (Swift), Imaging (Glance), Authentication (Keystone) and UI (Horizon) Services. The overview of the OpenStack infrastructure is presented in the Figure 5

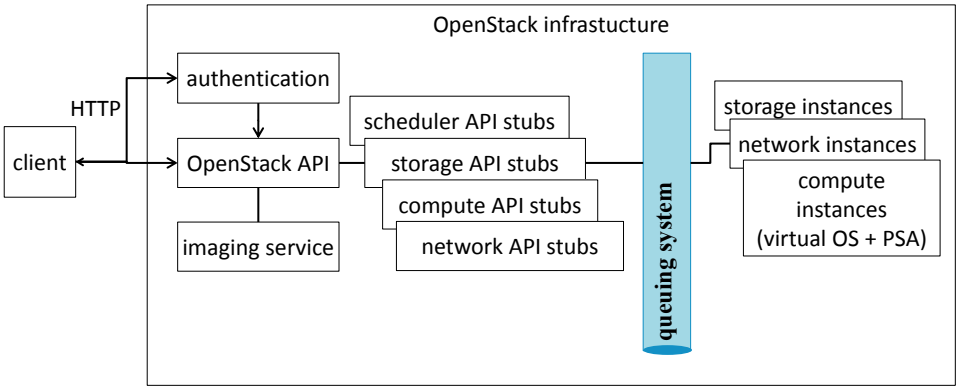


Figure 5: OpenStack infrastructure overview

From the Virtual Machine PSA model perspective the most interesting OpenStack Service family is the Compute Service since it handles all activities needed to support the life cycle of virtual machine instances including computing resources, networking and authorization needs. Like other OpenStack Services, it exposes its capabilities through a web services API which is compatible with EC2 API of Amazon Web Service but also a custom OpenStack API is provided. Under the skin, it provides the virtualization functionalities through libvirt API that enables interaction with various hypervisors such as KVM, UML, Xen, XenServer, VMware vSphere and Hyper-V.

3.2 Container PSA model

In contrast to Virtual Machine PSA model where execution environment of each PSA utilizes fully-isolated software resources including OS kernel, the Container PSA model enables multiple isolated user space instances using a single kernel. In this approach the overhead of kernel space duplication is minimized in cost of reduced isolation among PSAs. In addition, the PSAs development must take

account that each execution environment for each PSA is essentially same running on the same OS and with same kernel.

In the simplest form the Container could be implemented with the standard chroot mechanism available in Unix-like systems which changes the apparent directory structure of the process included in the Container while restricting access outside the designated directory structure. However, PSA would need more advanced isolation, e.g. in terms of providing resource management to limit the impact of one Container's hardware utilization and shared software usage on the other Containers. The more advance free Containers include Docker, Linux-VServer, OpenVZ, LXC and FreeBSD Jail of which Docker is a modern approach that is described as a reference implementation below.

Docker [4] an open-source project initially released in March 2013. It extends Linux kernel functionality (e.g. LXC and cgroups), that enable resource isolation and separate namespaces, with high-level API, easing the setup and deployment of application inside the Containers. The Container features included enable assigning memory limits, CPU quotas, isolating file systems and networks while lacking features include root privilege isolation, disk quotas and I/O rate limiting.

The Docker infrastructure overview is depicted in Figure 6. It mainly consist of Docker Engine, which is a portable, lightweight runtime and packaging tool, and the Container which includes the PSA and its software dependencies.

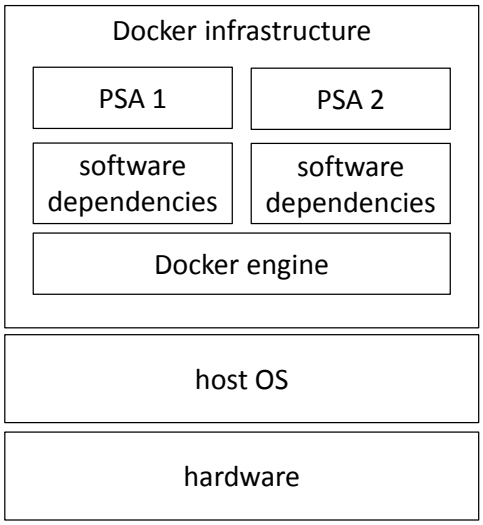


Figure 6: Docker infrastructure overview

3.3 Application PSA model

In the Application PSA model the PSA is an application running inside an execution environment (VM, container, etc.). In this case the EE will isolate the PSAs from each other, so that they can not interfere with each other. Similarly, the PSA chaining has to be provided by the EE. The PSA chaining can of course be set up by either taking the outputs of the PSAs and forwarding it to the next PSA on the chain; or it can be done by configuring the PSA to send the result to the next PSA.

In the Application PSA model we consider, e.g. a Java application (running in JVM) to be executed in the Execution Environment. SECURED could, e.g., forward packets coming in to the EE from the specified network interface to the first PSA and finally after the last PSA forward the packet outside

of the EE. With this model it is also harder to achieve the communication with the SECURED platform (PSC, for example). The developer can either make one single application, which includes the CTRL+MGMT part and also the desired security control, controlled by the CTRL+MGMT part. The second option is to provide these in separate applications, where the SECURED part would configure the security control running in the EE, but this causes problems in terms of security and isolation.

4 PSA Security capabilities

This section provides the description of security capabilities that should be supported by a PSA. As introduced before, in SECURED, a security capability is a functionality to process and protect the user's traffic.

4.1 Structure of security capabilities

With the aim of creating a hierarchical structure of the capabilities of the PSAs, we have defined a model tree of these features with two targets:

- a better description of the capability by looking for it on the tree hierarchically when a policy must be defined;
- a more flexible definition model where a PSA can be identified with different capabilities.

Figure 7 shows a classification of the categories that are in the scope of SECURED and the capabilities of the PSAs that fit in each category.

In particular, seven categories have been identified:

- **Access control** that 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. as 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. Belong to this categories the Traffic anonymizer (e.g. TOR [5]), 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.

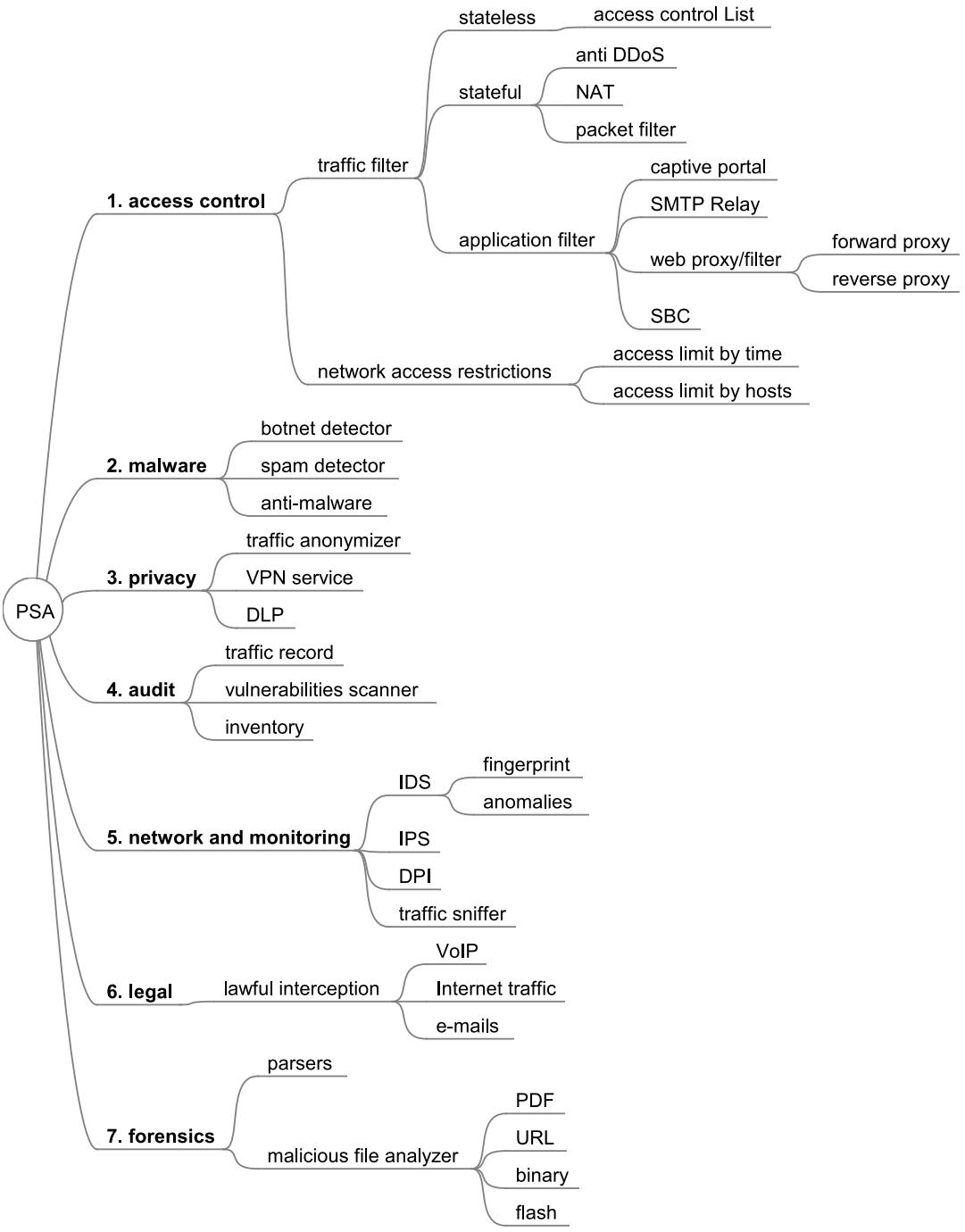


Figure 7: PSA categories

4.1.1 Access control

Here are PSAs that have the capability of filter and restrict the traffic of the network 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, or external conditions defined by the user, e.g. as day or time restrictions, content filtering.

- Traffic filter: in this category we decide to block or allow traffic packets based on different conditions. The complexity of the traffic analysis makes possible three options of development:
  - Stateless filter, it means that the security capability offered works based on filtering by packet and does not have any context information. It compares each packet against the restrictions defined per fields to allow or deny it. A typical example is a router access control list (ACL).
  - Stateful filter, that analyses, not only a packet in isolation, but as part of a flow of a communication. This capacity permits to act on traffic packets linked between them by different protocols. A common example is the FTP protocol where there are FTP control and FTP data packets. This security capability covers applications like NAT or classic firewall and devices that are usually limited to IP or Layer 4 protocols. Also complex features like Distributed Denial of Service (DDoS) detection and mitigation products will belong to this category.
  - Application level, that has the capability to manage a communication at a ISO/OSI layer 7 and filter or modify each packet in-line, based on several conditions. In some cases, it is able to generate a new packet from scratch. Depending of the protocols or the class of traffic there are, in turn, several security capabilities sub-families per protocols: Web proxy (direct or reverse) for HTTP, Session Border Controllers (SBC) for SIP, SMTP relays or captive portal for HTTP and DNS.
- Network access restrictions: can be considered restrictions according to date/time or name/number of hosts.

Examples of applications that implement some of these capabilities are pfSense [6] (stateful packet filter, NAT, captive portal), zeroshell [7] (stateful packet filter, captive portal, forward proxy with antivirus), Privoxy [8] (non-caching web proxy, filtering capabilities), Squid-cache [9] (application layer filtering), NGINX [10] (HTTP reverse proxy, SSL terminator, IMAP/POP3 proxy server), netfilter/iptables [11] (stateful paket filter) and PF [12] (stateful packet filter).

4.1.2 Malware

This category offers the detection and eradication of malware. We identify the following capabilities:

- Botnet Detector: capabilities to detect whether a user is infected by a Botnet, mainly performed by behavioural analysis.
- Spam Detector: it embraces tools used to detect unsolicited, unwanted mail traffic and to prevent this traffic. Can be incoming (spam filter) or outgoing (spam generation).
- anti-malware detector. Anti-malware products (evolution of anti-virus) which work either directly on the network traffic and on data passed explicitly by another PSA through the management interface.

4.1.3 Privacy

This category are listed security capabilities that offer mechanisms to provide privacy to the userstraffic:

- Traffic anonymizer: useful to preserve users' identity during the communications. TOR [5] is an example of application that has the capability of encryption, privacy and anonymity.
- VPN service to set secure channels and tunnels between different points. There are many applications offering this service as OpenVPN [13], IPsec-Tools [14] and strongSwan [15].
- Data Leakage Prevention (DLP) to prevent potential data breach by monitoring, detecting and blocking sensitive data while in-use (endpoint actions), in-motion (network traffic), and at-rest (data storage). An example of DLP application is myDLP [16].

4.1.4 Audit

This section embraces the audit capabilities family based on different technologies. Some of subcategories are: traffic record, vulnerabilities scanner or software inventory that helps the users to discover threats in their SECURED accessing device. Tcpdump [17] is an open source example of a traffic analyzer and OpenVAS [18] is a vulnerability scanner.

4.1.5 Network and monitoring

Network category contains the capabilities that implement functionalities at layer 3 of the ISO/OSI stack. In SECURED, we are interested in routing, NAT and VLAN. In GNU/Linux and *BSD operating systems, these capabilities are typically provided by the kernel or by tools available as part of the operating system.

Network Monitoring category implements capabilities that analyse the traffic flowing through the PSA looking for threats:

- Network Intrusion Detection System (IDS) that analyses network traffic searching security breaches (e.g. by using fingerprint, anomalies). Some known examples of IDS are Snort [19], Suricata [20] and Bro [21].
- Network Intrusion Prevention System (IPS), that identifies malicious activity, log information about this activity, attempt to block it, and report it. Most open source IDS also can behave as IPS.
- Deep Packet Inspection (DPI), which capabilities will include applications identification and classification by traffic and security problems associated. DPI can be done for all the traffic or for selective protocols.
- Traffic sniffer, that analyses selectively the traffic non exclusively for security. In this category we could include open sources examples as tcpdump and ntop [22].

4.1.6 Legal

Network Operators, service providers or companies may be subject to legal requirements. Therefore, a Legal capacity is an opportunity for PSA developers. Maybe the most interesting capability related to Legal aspects is the Lawful Interception (LI) in different implementations. It goes from simple capabilities associated with LI that intercept any Internet traffic from the SECURED device, to complex and selective interception models where, for example, VoIP transmission are intercepted and decoded or emails are extracted. In general, all previously mentioned information is delivered to LEA (Law Enforcement Agencies).

4.1.7 Forensics

This category collects the capability of carry out forensics techniques looking for threats considering online and offline traffic:

- Parsers to scan, for example, packets or URLs to determine if the packet forwarding is permitted or if there is malicious traffic to block.
- Malicious File analyser (PDF, URL, binary, flash, java, etc.), analyses files searching malware presence by signatures or behavioural. One example is REMnux [23], that includes several documents/files analyser or anubis [24].

4.2 Identification of standards and compliance requirements

The security capabilities defined for SECURED are based on the analysis of different security standards, regulations and frameworks. SECURED focuses on a wide range of end users and therefore the analysis includes documents related to different stakeholders such as: private persons, schools, public internet access, corporations, service providers or network operators. All those stakeholders have regulations to comply with security standards to follow as part of their policy. Also, several initiatives are focused on the protection of end user (mainly children) to the risk of being exposed on Internet.

The analysis is divided into four categories: Legal, Security related Government frameworks, Security standards, and Good practices and initiatives. The *Legal* categories focuses on service providers or network operators. The *Security related Government frameworks* focuses on corporations in general. The *Security standards* focuses on security standards for corporations. The *Good practices and initiatives* are guidelines for private persons, schools, and public internet access.

4.2.1 Legal

Network operators and service provider in Europe are bound to comply with lawful interception and data retention national laws. These laws translate the technical requirements defined by ETSI LI group [25], and require the deployment of applications to collect and provide information to LEAs (Law Enforcement Agents) related to intercepted communications. As a consequence, a potential LI compliant PSA could it be of great interest.

Additional laws can be apply to network operators which are obliged to use filtering or blocking traffic applications by court orders, for example, in case of terrorism or child pornography. For example, in US the Children's Internet Protection Act (CIPA) [26] is a federal law that require technology protection measures as the filtering and blocking of content that address offensive material issues in schools and libraries.

4.2.2 Security related Government frameworks

Several frameworks and good practice guides exists for the IT government and management that are strongly related with security. All of them require a set of technical controls that can only be achieved with security applications. Some of the PSA categories can cover these requirements. Some examples are:

- Control Objectives for Information and Related Technology (COBIT) [27]. Center in IT management and government include controls related to security.
- SANS 20 Security Critical Controls [28]. Several tools are needed for this control like: malware defense, vulnerability assessment, audit logs, limit network traffic, etc.
- Information Technology Infrastructure Library (ITIL) [29]. Also for IT management, including security policies definition.

4.2.3 Security standards

One of the relevant use case of security applications are the standard compliance. PSA can fulfil these needs in several security standards. Some of them are:

- Payment Card Industry Data Security Standard (PCI_DSS) [30]. Electronic payment systems require strong security controls are in place, based on applications like firewalls, network monitoring, access control, etc.
- ITU-T Recommendation X.805 [2] defines a network security architecture for providing end-to-end network security. Allows to use it as a reference model of security to identify threats and deploy security measures that can be achieve thanks to PSAs.
- ISO 27000 [31] includes a set of security standards, with strict controls to implement around the ICT solutions. Several of this controls are put in place with security policies.

4.2.4 Good practices and initiatives

Finally, there are specialized groups and initiatives related with security. These include recommendations and protections that network operators, companies or particular Internet user also needs to apply for security. Most of that enforcements relay in security applications that can be provided by PSAs or policy controls. Some of these groups are listed below:

- RIPE Anti-Abuse Working Group [32]. Includes the best current practices and tools recommendations for mail spam and other kind of abuse for service providers an network operators.
- Messaging, Malware and Mobile Anti-Abuse Working Group (MAAWG) [33]. Develops effective model to combat online threats like spam, botnet, phasing, denial-of-service attacks, etc. Several of the proposals include the use of security policies based on security applications.
- Internet Watch Foundations (IWF) [34]. Offers a specific black list of child sexual abuse content. This list are used by several organizations like GSMA Alliance that covers most mobile network operators. The list is typically applied in a web proxy filter application as transparent mode to the end user.

- Self-regulations for a better internet kids [35]. Based on EU promotion of self-regulations, several initiatives include technical requirements related to access control mechanisms, e.g. filtering services. PSA offers a strong options to develop this kind of services.
- ITU Child Online Protection (COP). Also includes several guidelines that recommend enabling control parental services for mobile operator and service providers in general.

5 PSA interactions

This section analyses the interaction between the PSA and the general SECURED architecture. An interface and a set of operations to manage a PSA are provided.

5.1 PSA in the SECURED architecture

The NED architecture[1] depicted in Figure 8 is organized in two parts: user-agnostic and user-specific. The user-agnostic manages the Trusted Virtual Domain(s) (TVDs), one for each user, by using the TVD manager (TVD MGR). This configures TVD networks to maintain isolation among users' traffic. The user-specific part is composed by users' TVD, i.e. an isolated portion of the NED that contains the execution environments. Every EE is a virtual machine composed by a set of PSAs. A privileged part, for management and control (EE MGMT and EE CTRL), is used for PSA management.

A NED also contains two network planes: Control and Management plane for the infrastructure networks and Control and Data plane for user networks. The former is used to manage PSA configurations (e.g. to send a configuration to the PSC), to restart EE or PSA (asked from PSC to TVD MGR) and to ask the integrity status of an EE. Considering the latter plane, the red network is dedicated to user's traffic, the black network is used for sending notifications from a PSA to the PSC and the grey network is used by the PSC for sending notifications to an user.

5.2 PSA management

This section introduces the operations to manage the PSA. First of all, two types of PSA are considered: stateless and stateful. A stateless PSA manages each request as an independent operation that is not related to any previous request, therefore, the communication consists of independent pairs of requests and responses. A stateless PSA does not need to maintain session information or state.

On the contrary, a stateful PSA requires to keep the internal state (e.g. network sessions). The states for stateless and stateful PSAs are depicted respectively in Figure 9 and Figure 10.

To run and stop PSA, we define the following operations:

- **Initialize** permits to configure the PSAs, by using the specific low-level configuration obtained through the M2L service. For example, in a firewall PSA, the initialization consists of receiving the rules (e.g. netfilter/iptables rules) that must be applied to traffic. The initialization moves the PSA in the `Ready` state, where the PSA becomes runnable.
- **Start** permits to run the PSA after the initialization or the stop operations. If the PSA is stateless, there is no difference between starting after the initialization or after the stop. On the other hand,

[1] More details about the NED architecture are available in D3.1.1 [1]

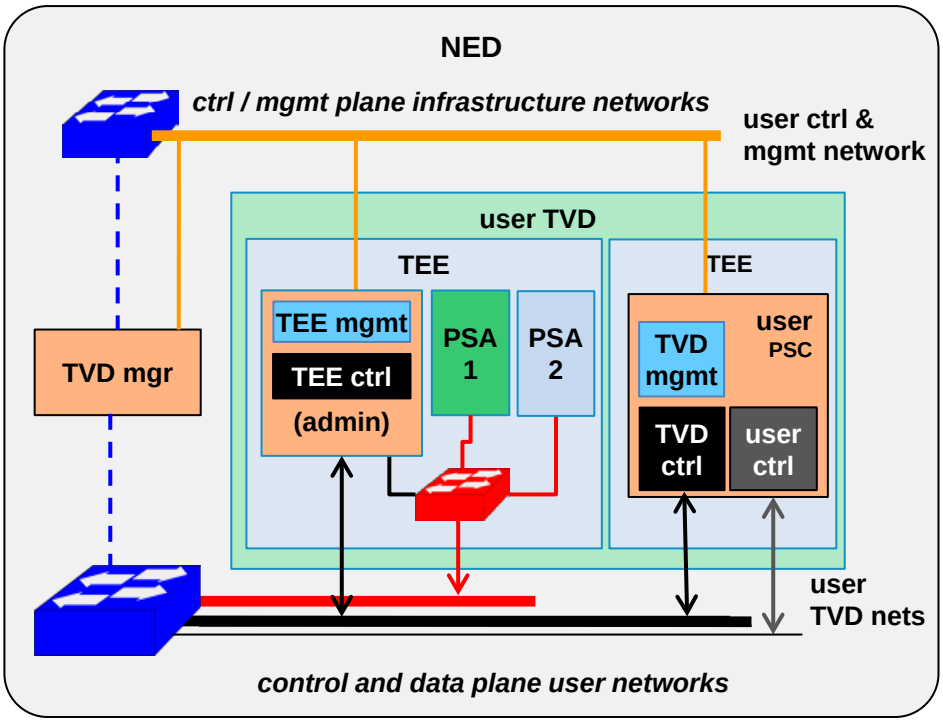


Figure 8: NED architecture.

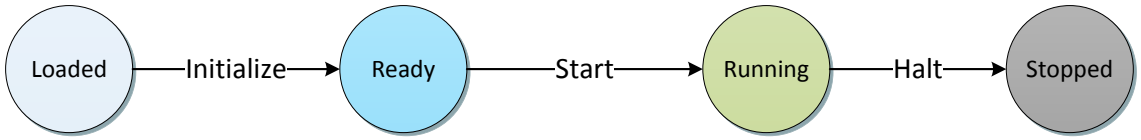


Figure 9: State diagram of stateless PSA

for stateful PSA, "start" from `Paused` runs the PSA considering the current state, frozen in the `Paused`.

- **Reset** as well as the operation of "start", runs the PSA after the initialization or executes the PSA after the stop. If the PSA is stateless, starting after the initialization is the same of starting after the stop, then the operation of "reset" is equivalent with the "start" operation. On the other hand, in a stateful PSA, when we restart from the `Paused` state, we resume the current state that was frozen and saved previously.

- **Pause** permits to stop the PSA, i.e. it freezes the state of the PSA. Especially, this operation may be required when the PSA cannot update its configuration or migrate at runtime. In this case, the update of the configuration can be only performed when the PSA is stopped. The *Pause* command can be useful to save computational resources when the user is idle.

- **Halt** turns off the PSA. The PSA can be stopped either from `Running` or `Paused` state. If the PSA is stateless the `Stopped` state is equivalent to `Paused`.

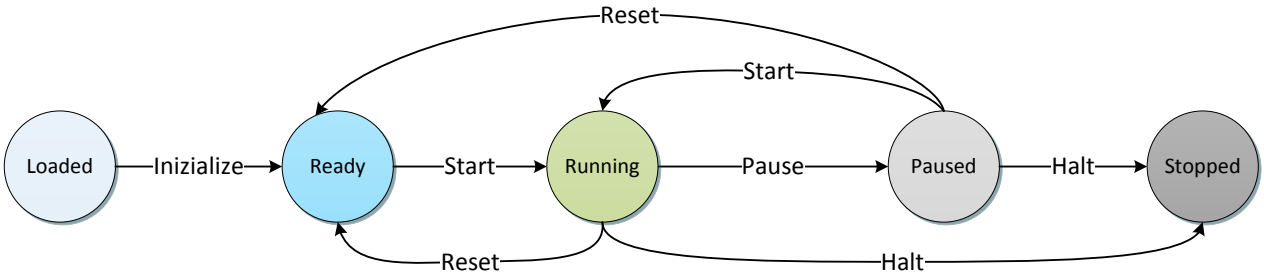


Figure 10: State diagram of stateful PSA

Besides the *state* of a PSA, we are also interested in the internal state. This is the set of information (a BLOB), managed at runtime, and typically stored in volatile memory of a PSA, e.g. the network connections for a firewall. The internal state information is useful to achieve live migration, i.e. when PSA data and traffic is moved to different NED without disruption. In this case, to save system and network resources (e.g. bandwidth), only the internal state could be transferred, without migrating the entire PSA. On the contrary, for cold or static migration, the internal state is not required.

The operations to configure and manage the information of a PSA are the following:

- **Set/Update configuration** replaces the entire configuration file defined in the initialization phase. Whereas the operation of *update configuration* allows to modify or insert new configurations (e.g. firewall rules) into a PSA. The PSA could be updated at runtime or after the "pause" operation. Depending on how the update operation is implemented in the PSA, it should be possible to perform a runtime update.
- **Get configuration** retrieves the configuration of a PSA (e.g. firewall rules). Depending on how this operation is implemented, it is possible to receive the entire configuration file or its path. Furthermore, if the PSA supports different security capabilities (e.g. packet filter and VPN) there will be two or more configuration files.
- **Get internal state** retrieves the internal state of the PSA (e.g. network connections for a firewall). Depending on the PSA, it is possible to receive heterogeneous types of information with different details.
- **Set internal state** modifies the internal state of the PSA. An use case of this operation happens when if we would prefer to migrate the configuration and the internal state of a PSA into another PSA, rather than migrate the entire PSA, in order to have a more lightweight task.
- **Get monitoring information** retrieves the information to monitor a PSA, e.g. CPU, memory and disk utilization, logs.

Furthermore there are a set of operations used to define and manage the network configuration. Namely it is possible to configure the *Online User Internal/External Interface* and the *Offline Interface*. More details are reported in Section 7.

6 PSA manifest

The aim of the "PSA Manifest" is to model the functionalities, properties and requirements to manage a PSA. This manifest will include all the information needed by PSC and NED to execute, configure and

monitoring the PSA. The following sub-sections will describe the high-level schema and its relevant sections.

The manifest will consist of five sections:

- **general info** contains data human readable section with general description of the PSA;
- **functionality** contains the features offered by the PSA;
- **execution model** describes requirements needed to execute the PSA;
- **configuration** describes how the PSA must be configured and behave expected;
- **monitoring** describes which values must be supervised in the control interface.

Finally, a manifest must be associated to a PSA and available to users. Therefore the best location to store the manifest is the PSA Repository (PSAR).

6.1 General info section

This information will be used by PSAM and PSAR to offer a detailed description to SECURED users that allows them to decide if it is what they needs. Therefore it contains general and human readable information on who provide the PSA (e.g. author, vendor), type of licence, version, etc.

6.2 Functionality section

This section will describe features of the PSA which explain what the PSA can do with the user's data. In particular, the supported security capabilities (e.g. packet filter) are listed by using a meta-language in a hierarchical form. The categories of security capabilities are described in Section 4. As previously discussed, the security policies defined by users will be translated to MSPL. These statements are not directly applicable to the PSA configuration (see Section 2 for details). It will require specific PSA configuration, applied by using specific commands of a PSA. Therefore, this section must include the type of a security control (e.g. netfilter/iptables) and a reference, e.g. a unique identification number, name or path, to the associated plugin(s) that execute the M2L translation.

6.3 Execution model section

This section describes the values that must be guaranteed to select, download and execute the PSA in the EE, controlled and managed by the PSC. As introduced before, a PSA could be implemented in different ways, e.g. by using virtualization/isolation techniques (e.g. virtual machine), or as common application (i.e. standard process). Therefore the infrastructure requirements to execute the PSA are fundamental. In particular, we identify three types of requirements: hardware (e.g. CPU architecture), Execution Environment information (e.g. libraries) and restrictions to interoperability (e.g. a PSA only can be deployed on GNU/Linux Ubuntu 12.04 or FreeBSD 9). Additionaly, a section on security and availability requirements is provided. It will include several parameters that will help to accomplish security requirements: confidentiality, integrity, authentication, non-repudiation and remote attestation, that the PSCM could use. This will also include additional information related to the security offered by the PSA that could be of interest for the user. Mobility, is key concept in SECURED, therefore, this

section must address it. In particular, the supported type of migration (cold or live). Also the performance of a PSA should be considered. This section of the manifest will include a set of parameters that will offer information related with performance of the PSA. This will help the policies manager system to select the most suitable PSAs to assign to a user. In case there is a competitive market for PSAs, this information could help the user to select the security application, based on performance. Finally, additional information is needed for describe dependencies with PSC, e.g. a PSA chaining graph or PSC related to parameters.

6.4 Configuration section

This section describes how the PSA must be configured to be executed. We have several common information for any implementation of a PSA. First of all the adopted formats, i.e. packing and compression format, binaries or execution environment related format. Then the configuration files to use, paths to executables, security controls (e.g. files, directories permissions) and environment variables. As alternative to automatic script or API to provide configurations. Considering the adoption of "complex PSA", i.e. when more security controls are integrated in the same PSA, the configuration order of the controls must be defined. For example, considering the Internet traffic flowing from an user to Internet we can place a stateful firewall control and then a web proxy. Finally, the configuration of network parameters are required to have a correct execution. Some examples are the number of network cards, their configuration properties (e.g. MAC and IP addresses), routing and DNS settings. Additionally, it can be useful consider the enabled network services (e.g. SSH), kernel options (e.g. traffic forwarding). All data of the configuration section may depend on the PSA implementation model used for execution environment (see Section 3). These details will be provided in future works when the PSA execution models will be chosen.

6.5 Monitoring section

Finally, this section embraced all the values that are required to be monitored by the control interface. It may be necessary to know the status of certain values and generate alerts for the PSC to be aware of abnormal situation. In particular we have identified:

- bandwidth: current input and output bandwidth, unit: kbps;
- delays: current delay, unit: ms;
- number of users sharing the PSA;
- characteristics values of each PSA, e.g. number of blocked/allowed packets, alerts to presence of malware or data leakage, keys used to encrypt/decrypt traffic, logs for errors and warning messages, CPU and memory (RAM and disk) utilization.

6.6 Custom-extensibility section

We define a custom section for future use. This makes it possible to cover situations not foreseen at the specification phase and leave it open to new requirements.

6.7 Versions of PSA manifest

As the manifest format will likely evolve over time, the various versions are maintained on-line at the following page:

https://www.secured-fp7.eu/ref/PSA_manifest/

7 PSA API

This section describes the PSA APIs used to communicate with the SECURED considering different PSA execution models (e.g. if PSA is a full VM or container or an application running in a VM, see Section 3 for more details).

7.1 API general information

The PSA API described here reflects the internal specification of the SECURED architecture (WD2.1 [36]) and the alpha specification of the NED (D3.1.1 [1]) and is subject to change.

7.1.1 Full API reference information

This section describes the initial high-level API specification, whilst the full API specification will be iteratively updated throughout the project.

7.1.2 PSA API status

This section represents the status of the PSA API. The possible statuses for API references are:

- Current - the current stable version, which should be used (can receive future versions).
- Experimental - the current version under development, and very likely to change.
- Deprecated - previous version that is still supported, but could be removed in future releases.

7.1.3 Different options for PSA API implementation

The following options on PSA development relate to trade-off between quality (e.g. fidelity of security levels) and effort on the implementation of the PSA_C - PSA_D interface. This related mainly to the control and management (CTRL+MGMT) interface of PSA and how it can control the PSA_D. The options for this interface implementation are following:

- Generic PSA_C: PSA_D must implement a common mechanism for configuring PSA_D (e.g. a file describing setup options and how these setup options relate to predefined PSC policies and management options). PSC may modify (through available generic PSA_C) the PSA_D setup by modifying these options automatically based on PSC commands.
- Simple Custom PSA_C: PSA_C component readily available to all PSA types that maps the PSC policies and management options to a set of relative (not absolute) values to offer a simple API. PSA developer maps the options levels to actions in PSA_D (with custom PSA_C).

- Custom PSA_C: PSA developer implements interface components based on PSC policies and management API.

From the above choices, the first option is the easiest for the PSA developer, but also provides the worst quality in terms of versatility. The other options require more developer effort, but on the other hand, provide better quality in terms of versatility. All the above options relate to how easy it is for a developer to create the configuration for a specific security control based on the security policies provided to it via the CTRL+MGMT.

For a Custom PSA, a developer must implement the API for CTRL+MGMT and network configuration described in Section 7.2. Initially we use a RESTful service to implement the API.

7.2 Current PSA API references

This section describes the current API specification for the PSA.

7.2.1 CTRL+MGMT API

The API offered to the SECURED MGMT+CTRL component to control the PSA are implemented by the operations described in Section 5, and are the following:

- `status initialize(configuration)`: initializes the PSA passing the configuration and returning the status;
- `status start(void)`: runs the PSA, no input parameter is needed, returns the status;
- `status restart(void)`: restarts the PSA, no input parameter is needed, the output returns the status;
- `status pause(void)`: pauses the PSA, no input parameter is needed, returns the status;
- `status halt(void)`: turns off the PSA, no input parameter is needed, returns the status;
- `status setConfiguration(configuration, securityControlType)`: sets the configuration settings of a security control (both passed as arguments) and returns the status;
- `status updateConfiguration(configuration, securityControlType)`: modifies the current configuration settings of a security control (both passed as arguments) and returns the status;
- `configuration getConfiguration(securityControlType)`: returns the configuration settings of the security control;
- `status setInternalState(state)`: sets the internal state (e.g. actual network connections) passed as input parameter and returns the status;
- `state getInternalState(void)`: returns the internal state, no input arguments are required;
- `info getMonitoringInformation(type)`: returns the monitoring information, the type (e.g. CPU, memory, disk utilization or logs) is provided by passing an argument.

The `status` represent the return code, an integer value, equal to zero in case of success or greater than zero for errors.

7.2.2 Network configuration API

Considering the network configuration of a PSA, we identify the following operations:

- `status setOnlineUserInternalInInterface(interfaceId, flowId)`: sets the input "Online User Internal Traffic" interface by specifying the reference to interface (interfaceId) for a particular network flow (i.e. a set of packets). It returns the status;
- `interfaceId getOnlineUserInternalInInterface(void)`: returns the input "Online User Internal Traffic" interface, no input parameter is needed;
- `status setOnlineUserInternalOutInterface(interfaceId, flowId)`: sets the output "Online User Internal Traffic" interface by specifying the reference to interface (interfaceId) for a particular network flow (i.e. a set of packets). It returns the status;
- `interfaceId getOnlineUserInternalOutInterface(void)`: returns the output "Online User Internal Traffic" interface, no input parameter is needed;
- `status setOnlineUserExternalInInterface(interfaceId, flowId)`: sets the input "Online User External Traffic" interface by specifying the reference to interface (interfaceId) for a particular network flow (i.e. a set of packets). It returns the status;
- `interfaceId getOnlineUserExternalInInterface(void)`: returns the input "Online User External Traffic" interface, no input parameter is needed;
- `status setOnlineUserExternalOutInterface(interfaceId, flowId)`: sets the output "Online User External Traffic" interface by specifying the reference to interface (interfaceId) for a particular network flow (i.e. a set of packets). It returns the status;
- `interfaceId getOnlineUserExternalOutInterface(void)`: returns the output "Online User External Traffic" interface, no input parameter is needed;
- `status setOfflineInInterface(interfaceId, flowId)`: sets the output "Offline Traffic" interface by specifying the reference to interface (interfaceId) for a particular network flow (i.e. a set of packets). It returns the status;
- `interfaceId getOfflineInInterface(void)`: returns the input "Offline Traffic" interface, no input parameter is needed;
- `status setOfflineOutInterface(interfaceId, flowId)`: sets the output "Offline Traffic" interface by specifying the reference to interface (interfaceId) for a particular network flow (i.e. a set of packets). It returns the status;
- `interfaceId getOfflineOutInterface(void)`: returns the output "Offline Traffic" interface, no input parameter is needed;
- `status setControlMgmtInInterface(interfaceId, flowId)`: sets the input interface for management and control by specifying the reference to interface (interfaceId) for a particular network flow (i.e. a set of packets). It returns the status;
- `interfaceId getControlMgmtInInterface(void)`: returns the input interface for management and control, no input parameter is needed;

- `status setControlMgmtOutInterface(interfaceId, flowId)`: sets the output interface for management and control by specifying the reference to interface (interfaceId) for a particular network flow (i.e. a set of packets). It returns the status;
- `interfaceId getControlMgmtOutInterface(void)`: returns the output interface for management and control, no input parameter is needed;

The `interfaceId` is an identifier for referencing an interface and depends on the PSA execution model (see Section 3). For a Virtual Machine PSA the interface is a virtual network interface or a virtual switch port. For the other types (e.g. PSA application), the interface could be a network socket or another type of IPC mechanism. The `flowId` is an identifier for a set of packets, used to specify that a particular set of packets must be processed by an interface.

7.3 Invoking API

Initially we use a RESTful service to implement the API that must be available on management and control plane of a PSA (PSA_C). Finally, to guarantee security and integrity of the PSA, the interface must be implemented by using secure protocols (e.g. SSL/TLS).

7.4 Versions of API

As the API will likely evolve over time, the various versions are maintained on-line at the following page:
https://www.secured-fp7.eu/ref/PSA_API/

8 Tentative guide to PSA development

This section is a tentative guide for a PSA developer that will be enriched during the project. It describes the development steps to create a PSA from scratch or by importing existing security controls. It also introduces the interoperability with other frameworks and discusses the PSA development lifecycle.

8.1 Introduction

One of the aims of the SECURED project is to create an open-source architecture for hardware manufacturers and software developers. Furthermore, the project will support the creation of marketplaces providing the technical foundation (a repository and the associated management system) and will foster their birth through the exploitation actions of the business project partners. The creation of marketplaces permits to download on-demand applications (after the user authentication), from a repository that can be personal, corporate, provider, or public. Application marketplaces can be created in several ways based on the different use and business models, like: open or closed, public or private repository; anonymous or signed, plain or certified applications.

In order to support the advancement of marketplaces and to encourage the PSAs development, the project will make available the necessary development environments with also several services needed to actually deploy the applications.

Reconsidering the example of Parental Control, several techniques exist and these aim to create parental controls to block websites, mobile devices, video games, computer and software. Every technique

has its own advantages and disadvantages; furthermore each of them can be the best solution for one purpose but the worst for another. Then a goal of this project is to create all the materials and supports to help developers to evaluate all security risks and problems in the PSA implementation.

Finally with respect to the integration with the SECURED components, we will create an interface and an application API (Section 7) for the individual applications. Those elements are sufficiently high-level to make application development straightforward enough for no-expert developers, in order that it can result a code that runs in an efficient manner from a resources consumption point of view.

The PSA SDK will provide a set of instructions and templates for a PSA developer to create a new PSA and to port an existing security control into a PSA that can be run in the SECURED infrastructure. This includes providing templates for the PSA Manifest (see Section 6), providing the necessary API(s) and sample software codes for selected programming languages in order to configure, run and manage the PSA in the defined PSA models (see Section 3). These will be explained in more detail in the following sections.

The PSA SDK will be developed throughout the project and it would be desirable (but optional) that in the end we would have an SDK that includes an integrated development environment (IDE) with tools that allow the developer to validate the manifest and the PSA and even emulate the NED in a Virtual NED inside the SDK on a selected execution platform. By having a real SDK and IDE the user is able to create PSAs more quickly and efficiently.

8.2 General PSA developer steps

This section describes the general steps that a PSA developer has to do in order to design, develop and deploy a PSA into SECURED. In particular, the following list presents the steps shortly:

- select the PSA execution Model;
- implement the security controls;
- implement migration-aware features;
- implement required interfaces;
- create M2L plugin and PSA manifest;
- test the PSA;
- deploy the PSA.

Select the PSA execution Model The developer can choose the PSA execution model from the list of available ones. Depending on the selection, the SDK provides guides on how to use it and provides sample codes and packages for implementing the SECURED interfaces. The implementation of the interfaces may be depend on the selected execution model. Additionally, some security controls might require a direct access to the network interface (possibly with privileged rights), which might affect in the selection of the execution model. Depending on the type of PSA execution model different templates are provided.

Implement the security controls One of the most important parts of a PSA development cycle is the implementation of the security controls. In the case the PSA should contain multiple security controls the developer must choose a execution order and ensure that the traffic is processed correctly. The SDK assists the developer during the selection of the security capabilities and the definition of the execution order. This informations are later provided to the developer when he creates the manifest.

Implement migration-aware features If the PSA is migration-aware (defined in the PSA manifest, see Section 6), the migration process can be implemented by the EE or the PSA itself or as a combination of these. If EE does not provide all necessary functionalities for migration the developer has to implement missing ones, i.e. save the state of the PSA and restore it. The SDK can later provide a test-suite to test this functionality.

Implement required interfaces Depending on the PSA execution model all the required interfaces have to be implemented. In particular this interfaces include the control and management interfaces on the control plane and the traffic interfaces of the data plane. The control and management interfaces are already included in the template as stubs and must be implemented correctly. The traffic interfaces on the other hand are configured in the template and the developer is responsible to connect them to the right security control.

Create M2L plugin and PSA Manifest The developer has to create a M2L plugin for his/her PSA. Guides will be provided on how to create the plugin. See Section 2.1 for more details on the translation service.

The SDK will provide a manifest template for different categories of PSAs (explained in Section 6). Furthermore the SDK will assist the developer at modifying and validating the manifest information.

Test the PSA This describes the steps how a developer can test the PSA. In particular, we identify the following requirements:

- testing the PSA can be done inside a Virtual-NED (when available) in the specified platforms;
- SECURED could provide a playground in the internet, where PSA's can be tested;
- debugging the PSA, possibly via a developer-specific interface allowing connection directly to the PSA;
- certifying the PSA, i.e. tests that the PSA is compliant with the manifest and functions correctly (also by using digital signatures); specific certification procedures can be defined to match providers requirements (e.g. mobile operators).

Deploy the PSA For testing purposes, the PSA might be possibly deployed into a local installation of NED. The software developers use the PSA Managemer (PSAM) web page to send their application to validation and verification to publish a PSA in a private or public repository, called PSA Repository (PSAR).

The SDK should help the developer in deploying the PSA into a specific SECURED environment or publishing the PSA into a PSAR. The PSAR can be public or private and the process of publishing an

application to a certain PSAR might have different steps that has to be followed. These steps include, e.g. PSA verification, signing, and updating.

Then the manifest for the new PSA will be uploaded to PSAR and associated to the security application. Finally the M2L plugins are uploaded to the related repository (e.g. by using a web interface).

8.2.1 Existing security control (also named legacy)

The SECURED architecture is designed so that existing security controls (like the ones mentioned in Section 4) can be converted into SECURED PSAs with very little effort. The SECURED architecture supports different execution environments, which are able to execute existing code of different security controls (see Section 3 for details). The original code which implements the security control(s) can be reused and only SECURED specific components must be created from scratch. SECURED specific components include the manifest, the M2L service, SECURED interfaces.

In order to convert an existing security control into a SECURED PSA, a developer must take in consideration the following key points:

Compatibility The first requirement for converting existing code into a SECURED PSA is to verify that the needed execution environment is supported by the SECURED architecture. For example, pfSense requires as execution environment a BSD styled operating system, e.g. FreeBSD [37], OpenBSD [38], NetBSD [39]).

Required components The second requirement is to identify the required components needed for the execution of the code. This includes the executable of itself, static resources, static and dynamic libraries. If the code depends on static or dynamic libraries the developer must verify that the SECURED execution environment includes them, otherwise they must be included in the PSA.

MSPL compatibility The third requirement is to identify the security controls provided by the code and ensure that SECURED has MSPL models for each of them. This is fundamental otherwise, during the policy refinement, the missing MSPL policy will not be generated and the security control can not be configured accordingly.

Control and Management Plane The forth requirement is to implement the control and management plane of the PSA in order to be able to configure and monitor the security controls. The control management interface must translate the SECURED commands to the security control specific ones.

SECURED interfaces The fifth requirement is to implement the SECURED interfaces of the PSA in order to give access to the security controls. The interfaces include the traffic and the control/management interfaces. The three traffic interfaces (Online User Internal Traffic, Online User External Traffic, and Offline Traffic) must be connected to the appropriated component within the PSA. The control/management interfaces must be connected to the control and management plane of the PSA.

M2L service plugin The sixth requirement is to implement the plugin for M2L service for each security control contained in the new PSA. For example pfSense includes three security controls (packet filter, NAT, and captive portal) and uses as configuration interface a XML-file. Therefore three M2L

plugins must be created and each of them must return a part of the XML-file according to the MSPL-policy.

Manifest The final requirement is to create the manifest for the new PSA and insert all the necessary informations. The needed information include a general description of the PSA, the list of security controls, the performance parameters, the execution environment parameters, security parameters, mobility parameters, and other dependencies.

8.3 Interoperability with other frameworks

Interoperability with existing frameworks that enable development and deployment of network applications could offer to SECURED beneficial infrastructures, in terms of developer communities, software repositories, compatible hardware/software platforms and interest groups. This could speed up the adoption of our new technology, especially in case of split-NED (where network element/SECURED App will redirect user network traffic to computing elements with, e.g. SDN and NFV) architecture in which SECURED computing elements could be deployed rapidly within existing infrastructures.

The main downside of the interoperability support is the lifecyle management of the supported interfaces. In scope of SECURED project a low level interoperability is not feasible but high level interoperability in terms of offering compatible SECURED computing elements would be possible. In practice, the available options include offering a virtual-NED implementation that would run whole NED, e.g. in a public cloud or a remote execution environment implementation that would run only EE and PSAs remotely.

From the PSA developer perspective the aforementioned approaches are transparent as compared to hardware NED deployments. However, the SDK could facilitate the use of remote deployment of Virtual Machine PSA Model, e.g. in OpenStack architecture in:

- creating virtual OS instances;
- setting up additional resources such as storages and databases;
- setting up network interfaces, traffic handling and EE/PSA chaining;
- setting up authentication for used SECURED services;
- setting up performance enhancements such as load balancing.

Another approach to support interoperability is to utilize existing package managers and software repositories to enable fast deployment on local hardware. Specifically, the SDK could support, e.g. the Docker infrastructure in:

- finding relevant elementary and developer contributed containers;
- creating custom containers;
- managing resources within container;
- chaining containers;
- deploying containers.

8.4 PSA Development lifecycle

Naturally, SDK will also integrate security in the PSA development lifecycle just as in best-practice software development projects. The security related steps in the life-cycle are summarized below.

1. Engineering requirements:
   - defining general security objectives;
   - defining PSA manifest;
   - needed for applying relevant design guidelines, helping setting of code review objectives and execution of security testing.
2. Design requirements:
   - selecting PSA Execution Model that meets the required PSA isolation levels;
   - following identified design guidelines for developing secure PSAs;
   - perform PSA threat modelling to identify threats, attacks, vulnerabilities and countermeasures;
   - reviewing the design in relation to execution environment and security policies, critical functionalities and used software dependencies.
3. Development requirements:
   - checklist based security code review;
   - PSA unit testing.
4. Testing requirements:
   - integration testing in Virtual-NED.
5. Deployment requirements:
   - SECURED conformance automation.
6. Maintenance requirements:
   - handle interoperability with updated software libraries.

9 Conclusions

This document provides the specifications for a PSA which contain the general architecture, PSA functionalities and interfaces with other SECURED components. In particular, the proposed architecture is applicable to all the types of security applications (e.g. firewall, web-proxy), it supports complex design (e.g. traffic chaining) and it provides isolation between management operations and user's traffic.

The offloading of security, by moving the security controls from user's devices to the PSAs, has driven the design process that identified a set of security capabilities (organized by categories) and related low-level controls. The transformation from MSPL to configuration for specific security control is performed by M2L service. A generic M2L service and a set of specific plugins (available on PSAR),

that effectively perform the translation, are designed. Following this approach, the PSA developer should only develop the plugin to perform the conversion. Also a set of API for M2L, PSAR and plugin are provided.

The usage of PSA within the SECURED project, in particular to manage (e.g. run, configure) the application, requires the introduction of models to describe its functionalities and requirements. To define all the information needed by PSC and NED, useful to manage the application, the "PSA manifest" is provided. The manifest, expressed by using a formal representation, contains the properties related to functionalities (e.g. security capability), execution model (e.g. software requirements), configuration (e.g. order of security controls) and monitoring (e.g. bandwidth utilization).

References

[1] SECURED consortium, "D3.1.1 - NED specification (Alpha preview)", June 2014

[2] International Telecommunication Union, "ITU X.805", [online] https://www.itu.int/rec/T-REC-X.805-200310-I/en

[3] "OpenStack" [online] https://www.openstack.org/

[4] "docker" [online] https://www.docker.io/

[5] "TOR" [online] http://www.torproject.org

[6] "pfSense" [online] http://www.pfsense.org

[7] "Zeroshell" [online] http://www.zeroshell.net

[8] "Privoxy" [online] http://www.privoxy.org

[9] "Squid-cache" [online] http://www.squid-cache.org

[10] "NGINX" [online] http://nginx.org

[11] "Netfilter/iptables" [online] http://www.netfilter.org

[12] "PF" [online] http://www.openbsd.org/faq/pf/

[13] "OpenVPN" [online] http://openvpn.net

[14] "IPsec-Tools" [online] http://ipsec-tools.sourceforge.net/

[15] "strongSwan" [online] http://www.strongswan.org

[16] "myDLP" [online] http://www.mydlp.com

[17] "tcpdump" [online] http://www.tcpdump.org

[18] "OpenVAS" [online] http://www.openvas.org

[19] "Snort" [online] http://www.snort.org

[20] "Suricata" [online] http://www.suricata-ids.org

[21] "Bro" [online] http://www.bro.org

[22] "ntop" [online] http://www.ntop.org

[23] "REMnux" [online] http://zeltser.com/remnux

[24] "Anubis" [online] https://anubis.iseclab.org/

[25] "ETSI Lawful Interception group" [online] http://www.etsi.org/technologies-clusters/technologies/security/lawful-interception

[26] "Children's Internet Protection Act" [online] http://www.fcc.gov/guides/childrens-internet-protection-act

[27] "COBIT 4.1: Framework for IT Governance and Control" [online] http://www.isaca.org/Knowledge-Center/cobit/Pages/COBIT-online.aspx

[28] "Critical Security Controls for Effective Cyber Defense" [online]. http://www.sans.org/critical-security-controls/

[29] "Information Technology Infrastructure Library" [online] http://www.itil-officialsite.com/

[30] "PCI Data Security Standard (PCI DSS)" [online] https://www.pcisecuritystandards.org/security_standards

[31] International Standard Organization, "ISO/IEC 27000:2014" [online].
http://www.iso.org/iso/home/store/catalogue_ics/catalogue_detail_ics.htm?csnumber=63411

[32] "Anti-Abuse Working Group" [online] http://www.ripe.net/ripe/groups/wg/anti-abuse

[33] "Messaging, Malware and Mobile Anti-Abuse Working Group" [online]. https://www.m3aawg.org

[34] "Internet Watch Foundation (IWF)" [online] https://www.iwf.org.uk/

[35] "Self-regulation for a Better Internet for Kids" [online] http://ec.europa.eu/digital-agenda/self-regulation-better-internet-kids

[36] SECURED consortium, "WD2.1 - Internal specification of the SECURED architecture," 2014.

[37] "FreeBSD" [online] http://www.freebsd.org

[38] "OpenBSD" [online] http://www.openbsd.org

[39] "NetBSD" [online] http://www.netbsd.org