European Security in Health Data Exchange

Deliverable D4.1

Privacy by design models and tools: proof of concept

Editor(s):	Stefanie Cox, Mike Surridge
Responsible Partner:	IT Innovation
Status-Version:	Final
Date:	22/12/2017
Distribution level (CO, PU):	PU

Project Number:	GA 727301
Project Title:	SHiELD

Title of Deliverable:	Privacy by design models and tools: proof of concept
Due Date of Delivery to the EC:	31/12/2017

Work package responsible for the Deliverable:	WP4
Editor(s):	Stefanie Cox, Mike Surridge
Contributor(s):	Xabier Larrucea Uriarte, Muhammad Barham, Chris Miles, Ken Meacham
Reviewer(s):	IBM
Approved by:	All partners
Recommended/mandatory readers:	WP4 partners, use case- and tool owners

Abstract:	This initial release enables work to start on the detailed privacy by design analysis of end-to-end test systems and scenarios proposed at M12 in Task 6.1. It covers initial asset-centric security threats and countermeasures, and basic secure design patterns. This deliverable is the result from Task 4.1 – Task 4.3.
Keyword List:	Security Modelling Tools, Secure Knowledge Base, Secure Design Patterns, Privacy by design
Disclaimer	This document reflects only the author's views and neither Agency nor the Commission are responsible for any use that may be made of the information contained therein

Document Description

Document Revision History

Version	Date	Modifications Introduced	
		Modification Reason	Modified by
v0.1	07/08/2017	First draft version	Stefanie Cox (IT Innovation)
v0.2	25/08/2017	Second draft version	Xabier Larrucea Uriarte (Tecnalia)
v0.3	16/11/2017	Third draft version	Stefanie Cox (IT Innovation)
v0.4	23/11/2017	Fourth draft version	Stefanie Cox (IT Innovation)
v0.5	29/11/2017	Fifth draft version	Stefanie Cox (IT Innovation)
v0.6	13/12/2017	Version for internal review	Stefanie Cox (IT Innovation)
v1.0	22/12/2017	Final version	Stefanie Cox (IT Innovation)

Table of Contents

List of Figures

List of Tables

Terms and abbreviations

CRUD	Create, Read, Update and Delete
DoS	Denial of Service
KPI	Key Performance Indicator
NCP	National Contact Point
PbD	Privacy by Design
PN	Participating Nation
SPARQL	SPARQL Protocol and RDF Query Language

Executive Summary

This is the accompanying report for WP4's D4.1 software deliverable. It describes the work done in tasks T4.1, T4.2 and T4.3, and:

- describes the current proof-of-concept prototypes (i.e. describes D4.1 itself); and
- outlines the future development plan for the incremental updates to the SHiELD WP4 deliverables.

T4.1 (Security modelling tools) creates design-time ("offline") modelling tools to support the modelling of health data being transferred as required by the use cases described in WP6, D6.1. This report describes the existing tool including some generic improvements and initial versions of the extensions to support modelling of regulatory compliance.

This version of the "System Modeller" tool enables the user to create design-time models of IT systems describing healthcare applications. Additionally to basic functionality such as signing in and out, performing CRUD (Create, Read, Update, Delete) operations on models and import/export of models, it supports:

- validating a model, i.e. generating a threat catalogue by matching pre-defined patterns from the knowledge base in the system
- asserting controls directly on assets or applying control strategies to block threats
- accepting threats, for example when they don't have a control strategy

System Modeller relies on the security knowledge base in order to perform any of these tasks.

T4.2 (Security knowledge base) captures potential security and compliance threats in a knowledge base. The initial threats are described by tool owners, and explain how the tools can help to manage the threats. The set of threats covered in this deliverable also serves as an example to help use case owners describe the threats they are typically confronted with.

In its initial version, the security knowledge base contains generic security threats, including but not limited to

- remote exploits, such as denial of service attacks, remote injections or snooping attacks
- software bugs, causing a host to become unreliable or unavailable
- unauthorised local access, where an attacker gains physical access to hardware, enabling them to steal data or alter processes or hardware

Furthermore, secondary threats are covered, i.e. threats that appear when a precondition exists. These secondary effects cause other assets to misbehave. This means that they can be chained into "secondary effect chains", where a set of root causes can cause a whole tree of secondary effects and misbehaviours in related assets.

T4.3 (Secure design patterns) devises architectural design patterns on two levels:

- low-level Java design patterns that can protect software against common software vulnerabilities; and
- high-level architectural patterns that assist a system designer to create pre-approved, secure systems.

Both types of patterns can be linked by representing Java design patterns as controls in a system model that – when applied – protect against certain threats.

1 Introduction

This report accompanies the D4.1 initial software release. It describes the current proof of concept software implementation and how we propose to add the results from T4.1 – T4.3.

T4.1 focuses on the Security modelling tools, which will be extended for SHiELD during the course of the project. We show the existing functionality of the System Modeller Tool, which will be the starting point for future developments.

T4.2 defines the Security knowledge base, including basic asset classes, threats and security measures. It draws from the use case scenarios as defined in WP6 and detailed in D6.1. Based on this asset model, we added some initial threats to illustrate how we will extend the knowledge base which at this point consists of generic security threats as explained in the executive summary.

T4.3 devises architectural design patterns on two different layers. High-level patterns help users of System Modeller to design systems that comply with legislation in different jurisdictions by default. They correspond to patterns of assets in the security model, and the security measures that should be included with each such pattern. Low level design patterns give a fine-grained description of software architecture on a code level. They correspond to code structures needed to correctly implement sets of security measures. Both types of patterns can therefore be related to the system security model, and in future the tools will provide support for both system designers and software developers to develop systems that contain privacy by design.

Section 2 covers work from task T4.1 and provides an initial version of the user guide for the proof of concept version of the “System Modeller” tool. The starting point was work from previous projects (the UK ASSURED project and the H2020 5G-ENSURE project), so much of the user guide relates to work done in previous projects, but it has been included here for SHiELD partners who need to use the tools. The main additions to this tool made in the SHiELD project are related to network connectivity and control inference, and the use of threat models to represent compliance with regulations. Additionally, we have improved the usability of the GUI and made some performance improvements in anticipation of dealing with extended end-to-end (cross-border) network models. Section 2 provides a brief description of improvements in the back end components of System Modeller, which covers the bulk of work done in SHIELD. This is followed by a description of the user interface from a user’s perspective, much of which is taken from previous projects. We included this material to provide a complete user guide, which will be needed by SHIELD partners wishing to experiment with the proof of concept tool.

Section 3 covers the models developed in SHiELD in task T4.2 to represent relevant systems and associated component assets and SHiELD-specific threats. These models are not based on previous projects, although basic asset and threat types have been included where they are relevant to SHiELD.

Section 4 describes the work on secure patterns from task T4.3. It describes how low-level Java patterns can be linked to high-level architectural patterns to protect against security threats during different phases in the system lifecycle.

Section 0 provides a summary of results achieved, their implications and next steps. This is followed by a brief list of key references.

2 Initial Security Modelling Tools

2.1 Introduction

System Modeller is a graphical tool that enables a user to create models of complex systems such as computer networks, then analyse (validate) these in order to identify security threats and take mitigation actions. The output of validation is a modified system model, enriched by features that were not captured by the initial design. The original idea was devised some time ago in the SERSCIS project for run-time threat analysis [1], and first applied to design-time analysis in the OPTET project [2]. Since then it has been refined and now forms the basis for our tools for design-time analysis of end-to-end risks in SHIELD.

2.2 Terminology

In describing the software, we use a set of terms for explaining various features:

- *Core Model* – the core ontology, defining common vocabulary and relationships used in all higher-level models.
- *Domain Model* – an ontology defining the typology of Assets, Threats and Controls (security measures) for a given domain (e.g. computer networks).
- *Domain Modeller* – a software tool for defining a generic domain model.
- *Design-Time System Model* – an abstract model of a particular system, described in terms of relationships between system specific assets. The design time model can be enriched by specifying Security Controls to protect the assets from potential threats to the system.
- *Runtime System Model* – a model using instances of Assets, Threats and Controls for describing what is known about the current state of the system.
- *System Modeller* – a software tool for defining a design-time system model in terms of assets and other elements from a suitable generic model.
- *Asset* – is an element of the socio-technical network described by a system model.
- *Involved Asset* – an asset which forms part of a pattern, the presence of which is necessary for a threat to occur or to be managed. Pattern here means a set of assets that are connected to each other user a set of relationships. A pattern is a directed graph that can be used for other purposed, such as described in section 4.1.2.
- *Stakeholder* – can be a person or an organization, i.e. a special type of asset that has motives for participating and initiating actions within the modelled system
- *Process* – usually represented by software, is a type of asset that can transfer or process data.
- *Network asset* – a type of asset representing an element of the infrastructure or environment.
- *Misbehaviour* – represents a way in which an asset may be compromised as a consequence of an active threat.
- *Control Strategy* is a set of controls located at one or more assets that block or mitigate a threat.

2.3 Backend components

The system modeller consists of a front-end (web hosted) graphical user interface, and a back-end service that allows network models to be stored and analysed. Most of the SHiELD-related additions we have implemented in SHIELD T4.1 are not visible in the UI yet. However, these changes represent important functionality which will be required to complement the results of T4.2 and T4.3 in the following releases.

2.3.1 Remote attacks and control inference

Our starting point for T4.2 is a common domain model, covering generic threats to IT networks and their countermeasures. The user can assert hosts and logical subnet assets, and create a design-time system model in which these assets are connected to each other. The model thereby captures the logical network topology in terms of logical subnets and gateway hosts. This logical connectivity is then used to infer potential remote attacks on networked assets from each logical subnet, including:

- remote attacks against devices, e.g. exploiting bugs in the operating system or unnecessary services running on devices that have not been properly secured;
- remote attacks against services that are needed as part of an application, by exploiting bugs or using other methods such as client spoofing;
- more complex attacks such as ‘confused deputy’ attacks in which a trusted service is induced to take harmful actions against back end resources.

To model such threats properly, one must determine the possible paths from the subnet where the attack is launched to the host (i.e. device) that is the target or hosts the target service. In order to manage such threats, one option is to use firewall restrictions to block paths that might be taken by an attacker’s messages. However, while it is sensible to consider such paths as assets in the network (since they enable communication beyond a local logical subnet), they are not the kinds of assets where a user could implement a security measure like firewall restrictions. In practice, firewall restrictions can only be implemented at a host, and can only restrict the flow of messages to, from or (in the case of a gateway) through the host.

Network segmentation and firewalling is very important in health care networks and associated regulations. Most regulations include a requirement that health data not be stored on a public network. Whether a network segment is considered public depends on the network topology and the firewall restrictions in place.

To handle this, remote attacks are now modelled in terms of ‘path’ assets representing routes from the attacker to the target, and control strategies included representing blocking the path. Users cannot assert a firewall control to block a path, but can assert firewall controls at network interfaces (controlling messages to or from a host) and at gateway hosts (controlling messages routed between subnets through the host). Whether a path is blocked is now inferred from the asserted firewall rules present at interfaces and gateways.

The inference procedure depends on the type of attack, or rather, the target of the attack. An attack against a host can be prevented by blocking the path from the attacker. Attacks on services can’t always be addressed so easily because one must not block the path from their legitimate clients. Moreover, the use of network address translation (NAT) means that to allow messages from a client to pass through a firewall restriction, one may have to allow messages from one or more entire subnets. Threat models representing remote attacks therefore refer to different types of path assets, each using a different inference procedure to determine whether or not the attack is blocked. This new feature is triggered whenever a control is asserted.

2.3.2 Compliance threats

Another new feature in the context of SHiELD is the addition of compliance threats.

Particularly when designing systems that span multiple jurisdictions, it becomes more important to be aware of compliance with various regulations. These regulations are encoded in a domain model and consist of "ComplianceSets". Each compliance set represents compliance with a certain set of rules, and consists of a collection of threats (sometimes called 'compliance threats') that shall be treated. These threats typically fall into three broad classes:

- threats describing potential attacks that the regulations require implementers to address, e.g. rules requiring wireless networks to be protected from snooping attacks;
- threats describing a pattern of interacting assets that is not permitted under the regulations, e.g. rules prohibiting back end services with an Internet accessible user interface being used also by users from a secure network (due to the risk that they will upload sensitive data);
- threats describing sets of security controls that should be used irrespective of whether they are needed to control any specific security threat.

The last of these may seem spurious, but represents a 'precautionary principle' approach to deal with security threats that were outside the scope of a threat analysis, or for which no means of implementation is currently known. A good example might be to mandate the use of encrypted storage even for servers in secure environments. Threats describing prohibited patterns are also slightly unusual in the sense that they have no control strategy, since one cannot address them by adding security measures, but only by changing the design of the network.

To capture a set of regulations, one simply specifies in the domain knowledge base which threats are members of the corresponding compliance set. This may involve adding threats to represent prohibited design patterns or precautionary security features, so they can be included in the compliance set. During the validation, system-specific threats are discovered based on all threats in the knowledge base. The system can be checked for compliance with a given set of rules by checking if all system-specific threats from the relevant compliance set are addressed.

This feature has currently been implemented on the backend side only, but in future releases we plan to make this information accessible to the user by displaying it in the GUI.

2.3.3 Performance improvements

The validation is computationally expensive and can take minutes or even hours for very large models. While this is still a great improvement over the manual threat identification process, we continue to work on speed optimisation. The changes we've made include:

- Optimisation of SPARQL (SPARQL Protocol and RDF Query Language) queries through reordering of statements and extensive use of graph filtering options
- Re-architecting of the validator component to minimise the number of queries executed
- Optimisation of the front-end components to reduce the number of REST calls

2.4 User Guide

The following sections provide details of the System Modeller functionality. We have included material produced in previous projects in this section so it provides a complete description of the user interface for project partners wishing to use the proof of concept release. This covers:

- Getting started

- Model management
- Model editing
    - Stage 1: defining assets and relationships which provide the initial model of a network
    - Stage 2: validation and auto-generation of threats
    - Stage 3: defining threat management strategy (selecting controls for assets or control strategies for threats)
- Model outputs (e.g. export)

The modelling process has three stages that may be repeated several times. First, the user constructs a model by placing assets onto the modelling canvas and creates links (or relations) between them (these user-defined entities assets are called "*asserted assets*" and "*asserted relations*"). The validation process in Stage 2 automatically generates *inferred assets/relations*, *threats* and *security controls* to counteract these threats. The validation process also determines whether the information provided about the assets and relationships is consistent and complete. If the validation fails (i.e. the model gets marked as 'invalid') then the user should go back to Stage 1 and update the model, so that it contains sufficient/correct information for a successful validation. In Stage 3, the user addresses threats by selecting or modifying the set of security controls that protect the assets in the system. The aim is to eliminate or at least mitigate the threats.

2.4.1 Getting started

The software can be accessed at https://shield.it-innovation.soton.ac.uk/system-modeller/. User accounts will be created on request.

2.4.1.1 Main page

On the main (welcome) page of System Modeller (Figure 1) there are several links (also available on other pages): *System Modeller*, *Home* and a drop-down menu under *Account*. The *System Modeller* and *Home* links return the user to the main page of System Modeller. The options under the *Account* dropdown are *Sign In, Register and Forgot Password.*

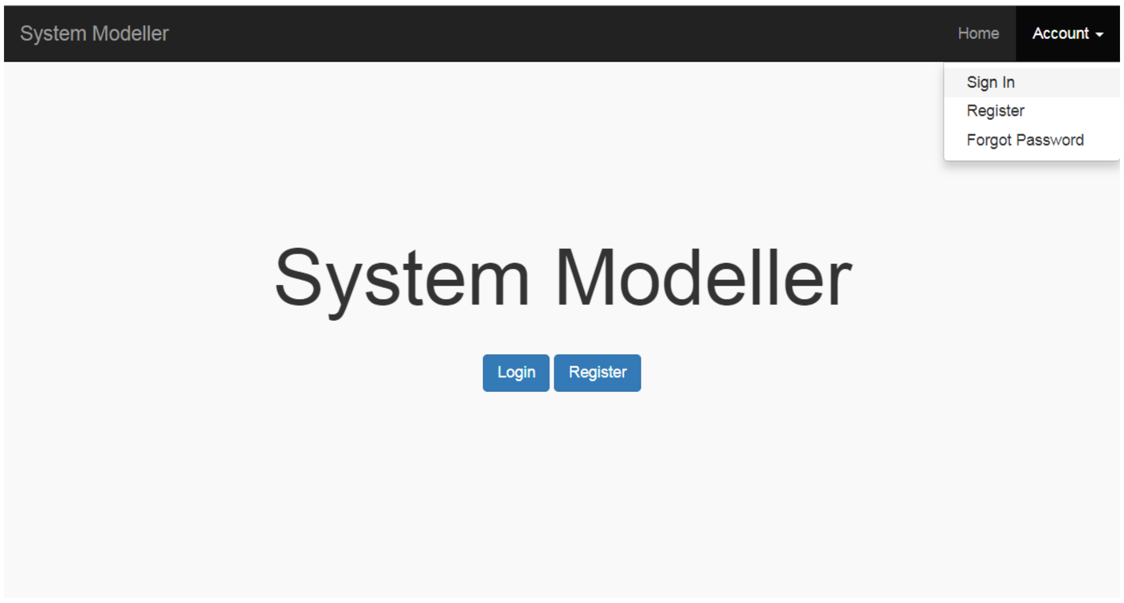
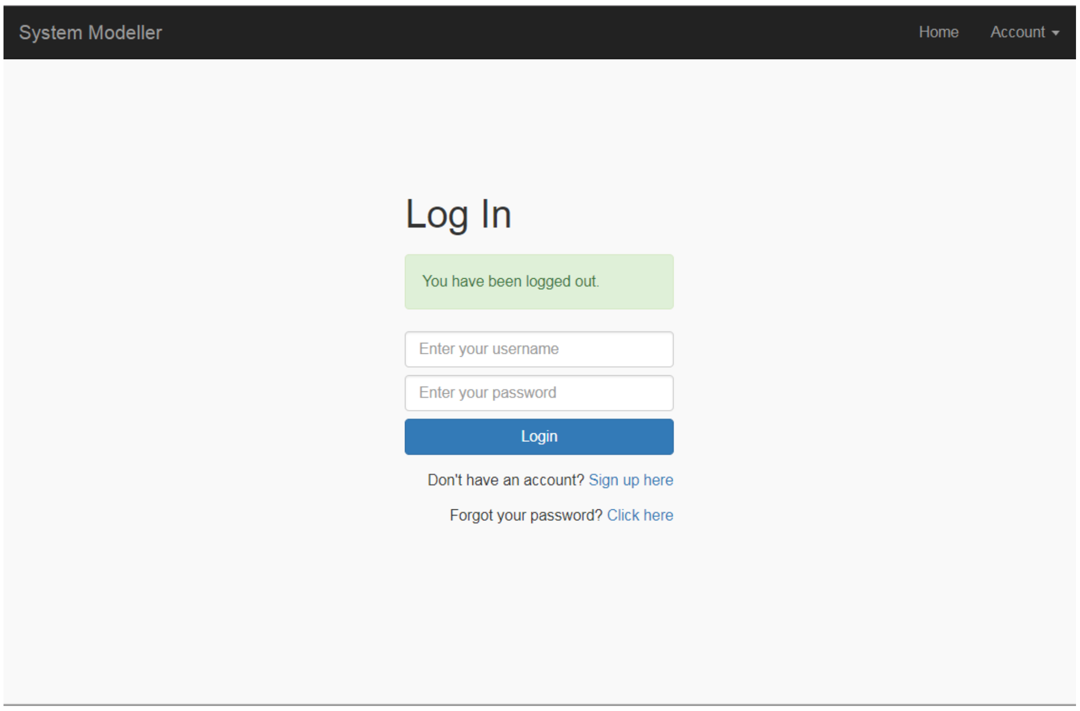


Figure 1 – System Modeller main page

2.4.1.2 *User login*

The login page of System Modeller is activated either by clicking on *Sign In* link in the dropdown menu or by clicking on the *Login* button (see Figure 2). The user must enter their username and password. These are case-sensitive.



You have been logged out.

Don't have an account? Sign up here

Forgot your password? Click here

Figure 2 – User login page

2.4.1.3 *Logout*

Logout is activated by clicking on the *Sign Out* link in the dropdown menu on the main page under the currently logged in user.

2.4.2 Model management

The term "*Model Management*" incorporates several functions such as:

a) Listing models
b) Creating models
c) Importing/exporting models
d) Deleting models

2.4.2.1 *List models*

After a successful login, the user is presented with a list of their models (see Figure 3).

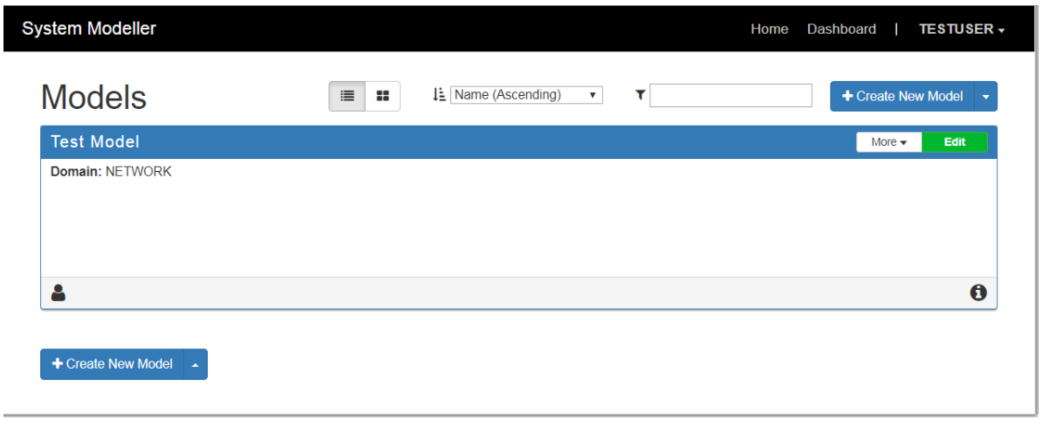


Figure 3 – Listing the models

In the model details, the **Domain** used by the model is labelled. There is also a description of the model, which can be edited via the *Edit Details* function (described in Figure 4). At the bottom of the model panel, there are several icons that reflect the current status of the model (see Table 1).

Icon	Description
👤	last modification of the model
ℹ	when the model was created

Table 1 – Model status icons

The drop-down menu in the top right corner of the model window offers several functions, these are: *Delete*, *Export* and *Edit Details* (see Figure 4).

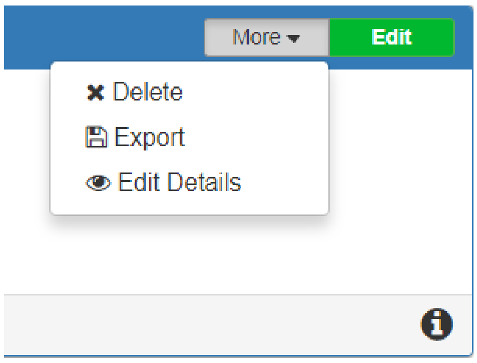


Figure 4 – Model drop-down menu

2.4.2.2 Create model

By clicking on the "*Create New Model*" button, the user can create a new model (see Figure 5). The "*Domain Model"* drop-down allows the user to choose which domain model to use. The new model is added to the models list, as described in the previous section.

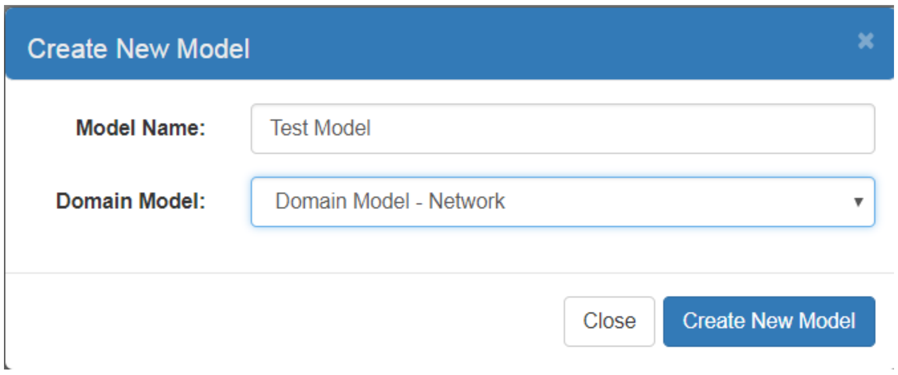


Figure 5 – Creating a new model

The “*Import Existing Model*” option (available on the “*Create New Model*” drop-down list) allows the user to import a model from a file (see next section).

2.4.2.3 Export/Import

Once we have constructed the model, it may be exported into a file. Using the *Export* option in the model drop-down menu (see Figure 4), the model is exported into the user’s “*Downloads*” directory.

Similarly, the *Import* operation allows the user to upload a previously saved file into a new model. In order to do this, the user clicks on the “*Create New Model*” control and selects the “*Import Existing Model*” option. A dialog appears (see Figure 6). The user may choose to *Import asserted facts only* (e.g. asserted assets, relations). If restoring a previous version of a model, the user can check *Overwrite existing model*. Attempting to import the same model without this being checked will result in an error. A user can re-import an existing model with a different name by checking *New Name.*

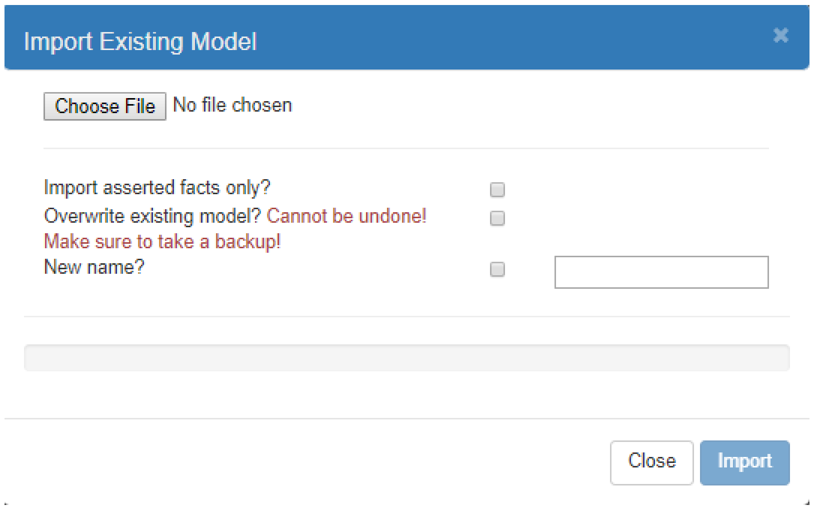

Import Existing Model
Choose File
No file chosen
Import asserted facts only?
Overwrite existing model? Cannot be undone!
Make sure to take a backup!
New name?
Close
Import

Figure 6 – Importing a model from a file

2.4.2.4 Delete model

The delete action removes the model along with any associated data items (e.g. assets, relationships).

2.4.3 Model construction

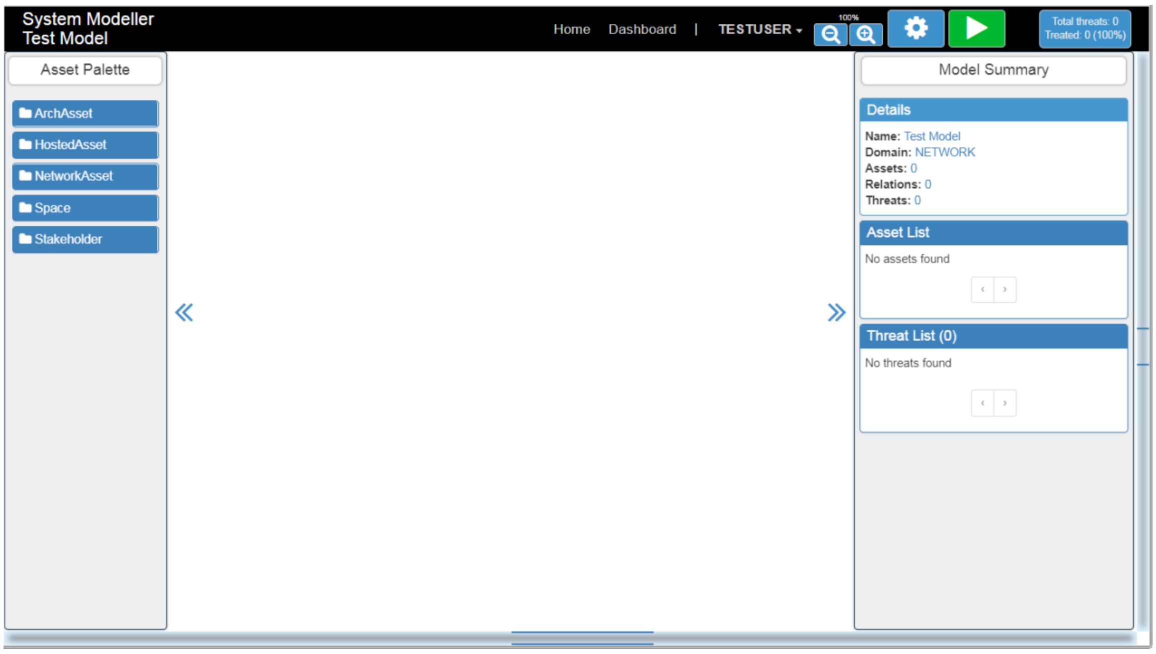
Clicking the *Edit* button (on a model in the models list) opens the model editing page that consists of three main parts (see Figure 7).

On the left side is the "*Asset Palette*", in the middle the *Model Construction Canvas* and, on the right, the *Model Summary*. These are described below.

Asset Palette: contains icons representing the asset types (e.g. "Host") that may be added to the model

Model Construction Canvas: the main area for designing/displaying the model

Model Summary: summarizes details about the model and shows a lists of assets, threats, etc. (these are empty initially).



Name: Test Model
Domain: NETWORK
Assets: 0
Relations: 0
Threats: 0

No assets found

No threats found

Figure 7 – Model editing

The top right corner contains four buttons, these are shown in Table 2 below:

Icon	Description
▶	Process (validate) the model
⚙	Configure the model
100% 🔍 🔍	Zoom controls

Table 2 – Model editing controls

2.4.3.1 Select and add asserted asset

The Asset Palette contains various assets; these fall into five main categories:

a) *ArchAsset*
b) *HostedAsset*

c) *NetworkAsset* (for illustration see Figure 8)
d) *Space*
e) *Stakeholder*

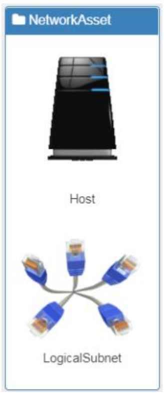


Figure 8 – Selecting items from Network Assets

An asset can be added to the model by selecting an icon in Asset Palette and dragging it onto the model canvas (see Figure 9), showing 5 added assets. N.B. these have already been renamed, as described in section 2.4.3.5).

By clicking on the asset, the right-hand panel updates to show the following asset details:

a) Name and description of the asset
b) Incoming relations
c) Outgoing relations
d) Inferred relations (once the model has been validated)
e) Control sets (once the model has been validated)
f) Threats (once the model has been validated)

For illustration purposes, we analyse a typical use case representing the threats associated with accessing a web page hosted on a remote server. Constructing a security model involves placing assets onto the canvas and establishing connections between them. The model itself consist of two hosts connected to the Internet. The user of Host1 uses a Browser to access a WebService deployed on Host2. This is a simple model that represents the scenario of downloading a web page from the remote web server. The reasons for selecting this simple model are as follows:

a) Frequently occurring case, typical for all web applications
b) Simplicity
c) All types of inference can be well demonstrated, these are:
   - Inferred assets
   - Inferred relationships
d) The threats inherent to the model are well understood
e) The effect of controls and threats can be easily interpreted

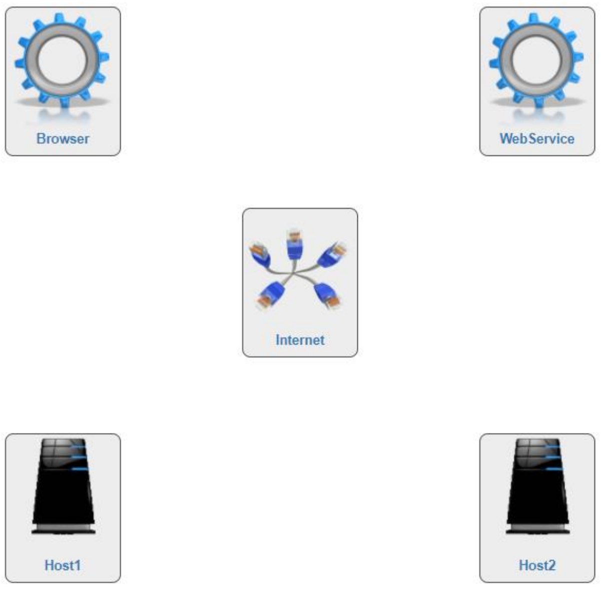


Figure 9 – Adding assets to the model canvas

2.4.3.2 Add relationship between assets

Once the assets have been put onto the modelling canvas, the user can connect pairs of assets by establishing links between them, by clicking on the green cross that appears in the top left corner (Figure 10).

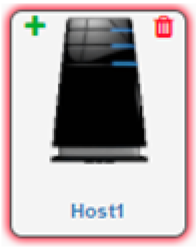


Figure 10 – Connecting assets

After clicking on the green **cross** (on Host1 in our example) a blue tick sign will appear on several other assets, indicating that a link can be made to these assets (see Figure 11).

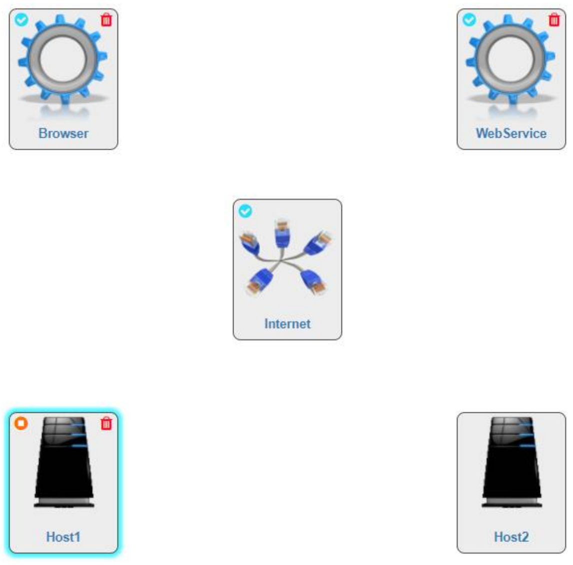


Figure 11 – Target assets for making connections

By clicking on one of the blue tick icons, we can establish a connection between the two assets. Here, we continue making connections (or "*asserted relations*") between assets until the model appears as in Figure 12.

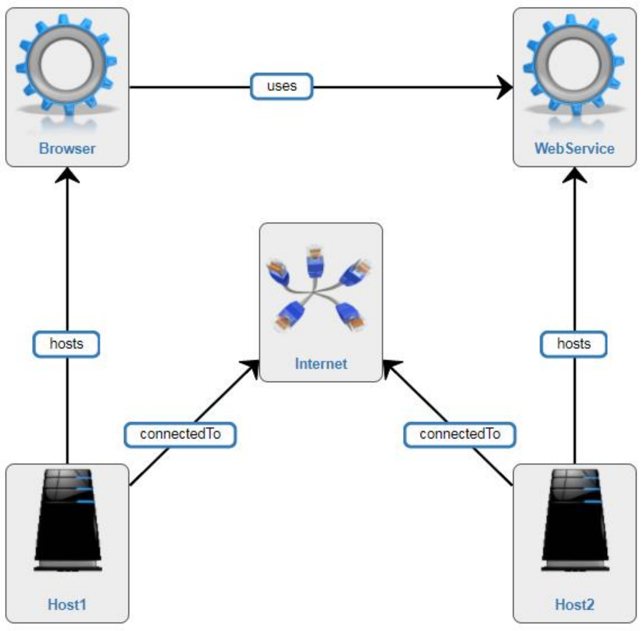


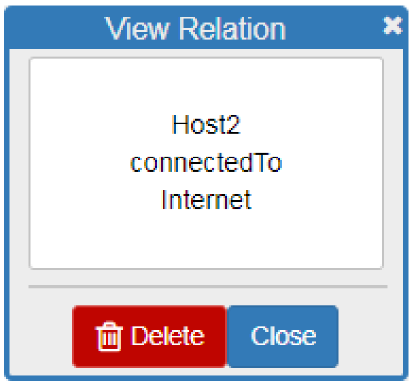
Figure 12 – Connecting assets

2.4.3.3 *Delete asset*

Assets can be deleted by clicking on the red trash icon of the asset in the top right corner (see Figure 10). The delete operation also removes all links between the selected asset and other assets.

2.4.3.4 Delete relation

By right clicking on the connection between two assets a dialog with a delete button pops up (see Figure 13). The delete applies only to the relationship itself; the assets remain on the canvas.



Host2
connectedTo
Internet

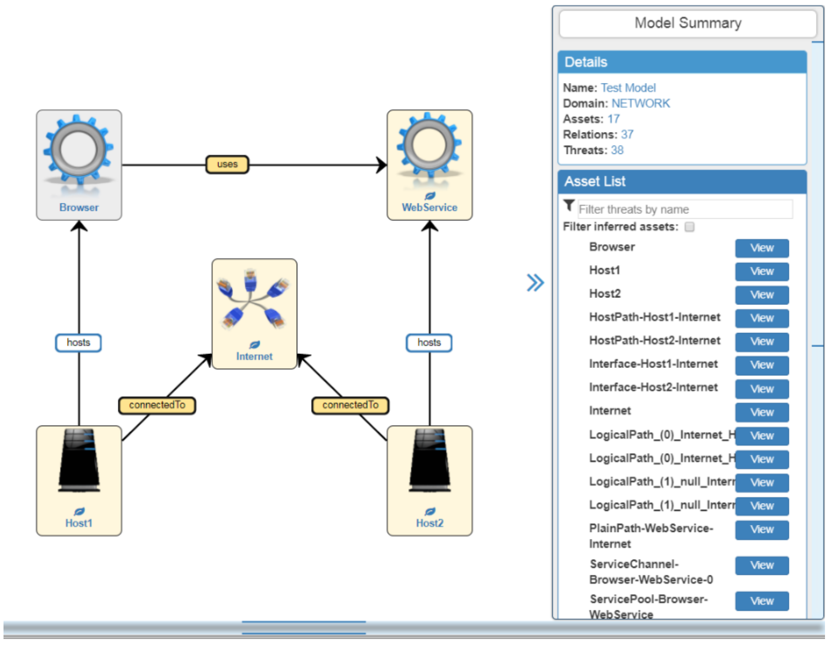
Figure 13 – Delete relation

2.4.3.5 Rename asset

The user can rename an existing asset by editing the asset's name under the corresponding icon. N.B. by changing the name, the asset's connections will stay unaffected. All asset names must be unique.

2.4.4 Model validation

Once the model has been constructed, it then needs to be validated. This operation is initiated by clicking on the red "play" button (see Table 2), which indicates that the model is currently invalid (for a valid model, this button would be green). The validation operation runs semantic reasoning that generates inferred assets and relations that are added to the model automatically, and produces a list of threats that can be associated with the given model. This operation guarantees that the inferred assets are consistent with the asserted assets and relationships. The validation operation can take some time, depending on the complexity of the model. On completion, the updated model is presented to the user (see Figure 14).



Model Summary

Details

Name: Test Model
Domain: NETWORK
Assets: 17
Relations: 37
Threats: 38

Asset List

Filter threats by name
Filter inferred assets:

Browser	View
Host1	View
Host2	View
HostPath-Host1-Internet	View
HostPath-Host2-Internet	View
Interface-Host1-Internet	View
Interface-Host2-Internet	View
Internet	View
LogicalPath_(0)_Internet_H	View
LogicalPath_(0)_Internet_H	View
LogicalPath_(1)_null_Intern	View
LogicalPath_(1)_null_Intern	View
PlainPath-WebService-Internet	View
ServiceChannel-Browser-WebService-0	View
ServicePool-Browser-WebService	View

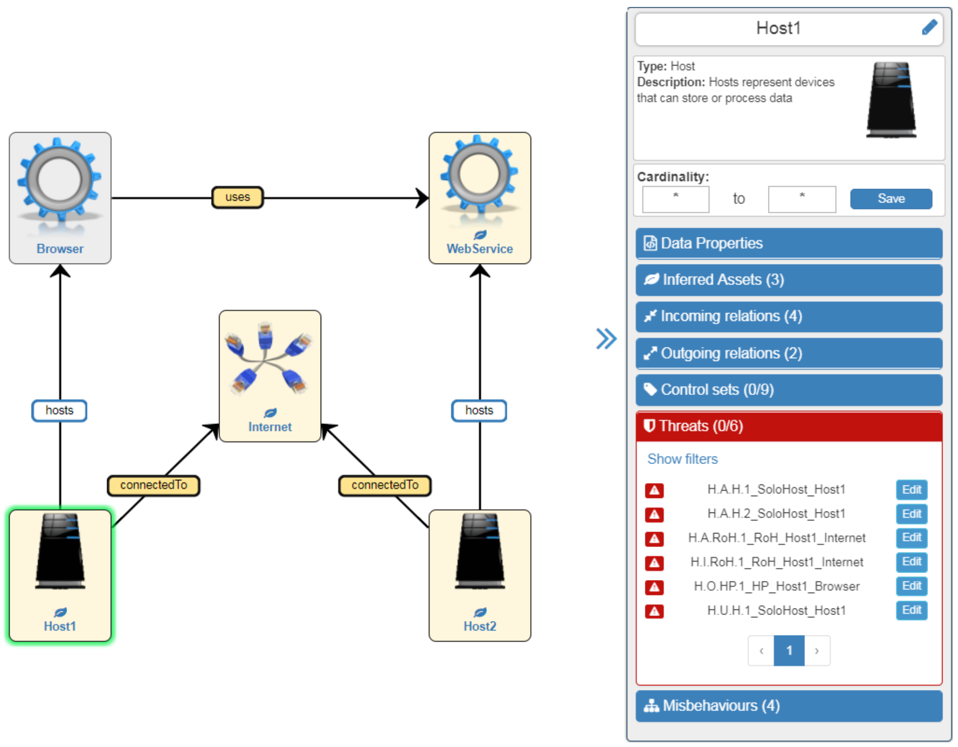
Figure 14 – Model after validation

Here, you will see that certain assets are now highlighted in yellow, which indicates that there are now associated (invisible) inferred assets and relations connected to this asserted asset (these may be viewed via the asset details panel on the right). Similarly, a relation marked in yellow indicates that there are newly added (inferred) assets and relations in the model, connected to this relation. By selecting this relation, the user can view information related to the inferred asset, via the asset details panel.

2.4.5 Threat management

2.4.5.1 Threats associated with a given asset

In order to view the threats associated with a given asset, the user must first select this asset on the canvas then click to expand the "*Threats*" panel within the asset details panel (see Figure 15).

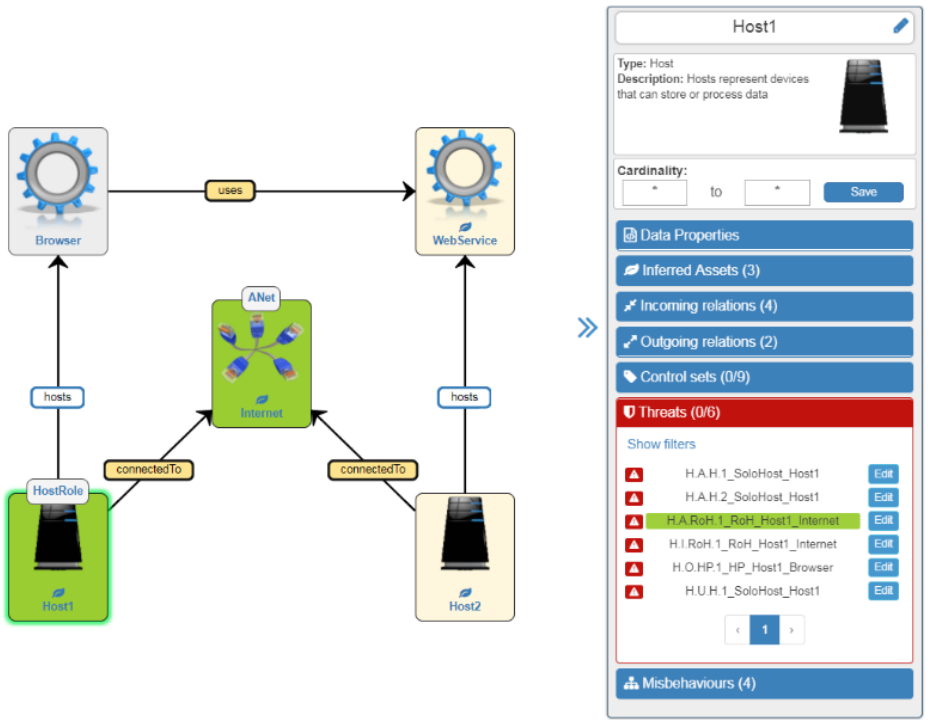


Type: Host
Description: Hosts represent devices that can store or process data

H.A.H.1_SoloHost_Host1 Edit
H.A.H.2_SoloHost_Host1 Edit
H.A.RoH.1_RoH_Host1_Internet Edit
H.I.RoH.1_RoH_Host1_Internet Edit
H.O.HP.1_HP_Host1_Browser Edit
H.U.H.1_SoloHost_Host1 Edit

Figure 15 – List of threats associated with Host1

In our example, none of the threats have yet been addressed, so each status icon is red. The user may hover over a threat in the list, which highlights the threat in green, along with its associated pattern of assets and relations on the canvas. For example, Figure 16 shows the highlighted threat to Host1, "*H.A.RoH.1_RoH_Host1_Internet*". The code stands for a threat applying to a host ("H"), causing a loss of availability ("A") in a remote attack on a on host ("RoH") involving the Host1 and Internet assets. Clicking the adjacent *Edit* button brings up the *Threat Editor* that allows the user to view threat details, and mitigate the threat in various ways (this will be described further in Section 2.4.6).



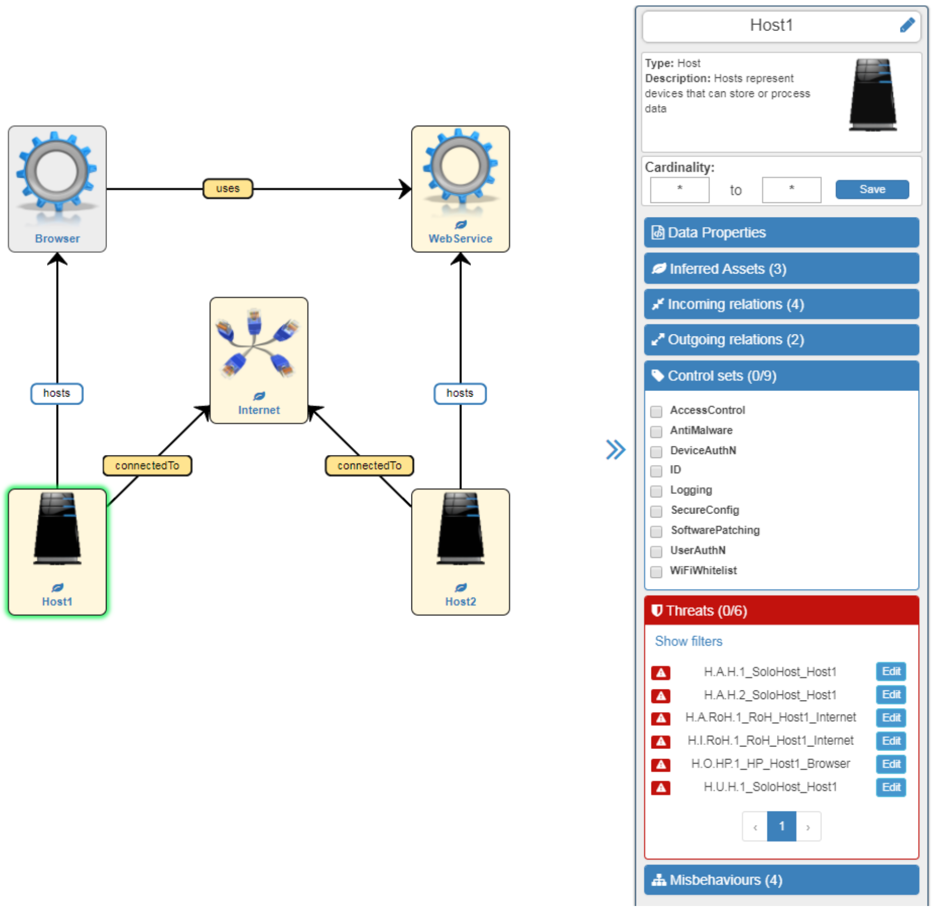
Type: Host
Description: Hosts represent devices that can store or process data

H.A.H.1_SoloHost_Host1 Edit
H.A.H.2_SoloHost_Host1 Edit
H.A.RoH.1_RoH_Host1_Internet Edit
H.I.RoH.1_RoH_Host1_Internet Edit
H.O.HP.1_HP_Host1_Browser Edit
H.U.H.1_SoloHost_Host1 Edit

Figure 16 – Threat highlighting and selection

2.4.5.2 *Selecting controls in the Control sets panel*

Threats may be resolved by selecting one or more *controls* within the *Control sets* panel (N.B. the exact control(s) required depend on each threat). Each *control set* represents a control on this asset. For example, the controls that are available for Host1 are shown in Figure 17; to expand the *Control sets panel*, simply click on it.



Host1

Type: Host
Description: Hosts represent devices that can store or process data

Cardinality: * to * Save

Data Properties
Inferred Assets (3)
Incoming relations (4)
Outgoing relations (2)
Control sets (0/9)

- AccessControl
- AntiMalware
- DeviceAuthN
- ID
- Logging
- SecureConfig
- SoftwarePatching
- UserAuthN
- WiFiWhitelist

Threats (0/6)

Show filters

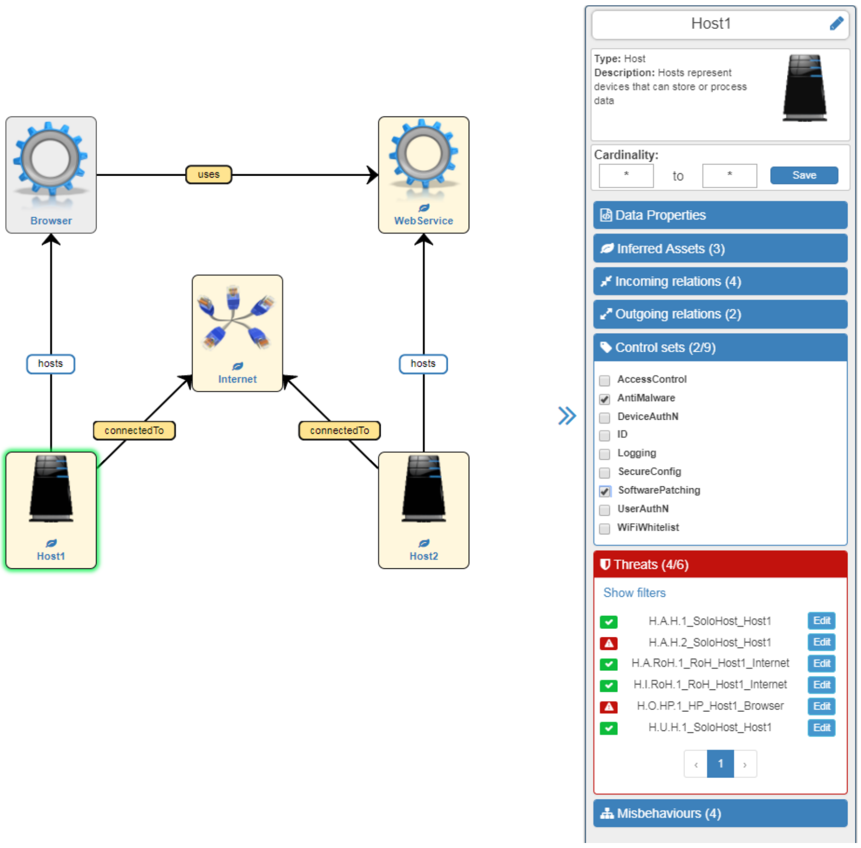
H.A.H.1_SoloHost_Host1	Edit
H.A.H.2_SoloHost_Host1	Edit
H.A.RoH.1_RoH_Host1_Internet	Edit
H.I.RoH.1_RoH_Host1_Internet	Edit
H.O.HP.1_HP_Host1_Browser	Edit
H.U.H.1_SoloHost_Host1	Edit

‹ 1 ›

Misbehaviours (4)

Figure 17 – Selecting Control Set properties for Host1

For the purpose of demonstration, we can select two control options (*Software Patching*, *Anti Malware*) and see which threats will be "resolved". These threats are then indicated by a green colour (see Figure 18).



Host1

Type: Host
Description: Hosts represent devices that can store or process data

Cardinality: * to * Save

Data Properties
Inferred Assets (3)
Incoming relations (4)
Outgoing relations (2)
Control sets (2/9)

- [ ] AccessControl
- [✓] AntiMalware
- [ ] DeviceAuthN
- [ ] ID
- [ ] Logging
- [ ] SecureConfig
- [✓] SoftwarePatching
- [ ] UserAuthN
- [ ] WiFiWhitelist

Threats (4/6)

Show filters

✓	H.A.H.1_SoloHost_Host1	Edit
⚠	H.A.H.2_SoloHost_Host1	Edit
✓	H.A.RoH.1_RoH_Host1_Internet	Edit
✓	H.I.RoH.1_RoH_Host1_Internet	Edit
⚠	H.O.HP.1_HP_Host1_Browser	Edit
✓	H.U.H.1_SoloHost_Host1	Edit

Misbehaviours (4)

Figure 18 – Resolving threats

Figure 18 shows that four threats have been resolved as a result of selecting *Software Patching* and *AntiMalware* from the *Control sets*. Resolving the threats is an iterative process; the user needs to go through the assets one by one, selecting options from the *Control sets* and checking which threats have been eliminated (resolved). Threats may also be resolved via the *Threat Editor*, as we shall see in the next section.

This User Guide describes the threat resolution steps for one asset (Host1) but the same steps are applicable to all assets. By following these steps, the user should be able to resolve most (if not all) threats associated with the given model.

2.4.6 Threat Editor

2.4.6.1 Using the Threat Editor to select controls

The previous section described how to address threats by applying controls to assets directly. This is particularly useful for experienced users who know about the effects that the controls have on the assets. Inexperienced users, however, need some guidance on how a threat can be addressed. This is done using *Control Strategies*, which are essentially collections of *Control Sets*. An active *Control Strategy* manages a threat. Whilst a *Control Set* describes a *Control* located at a particular asset, a *Control Strategy* contains one or more *Control Sets*. Semantically, this means that for the threat to be managed (blocked or mitigated), all *Control Sets* within the *Control Strategy* must be applied. The validated system model contains mappings between Threats and Control Strategies that are made visible in the *Threat Editor*, see Figure 19.

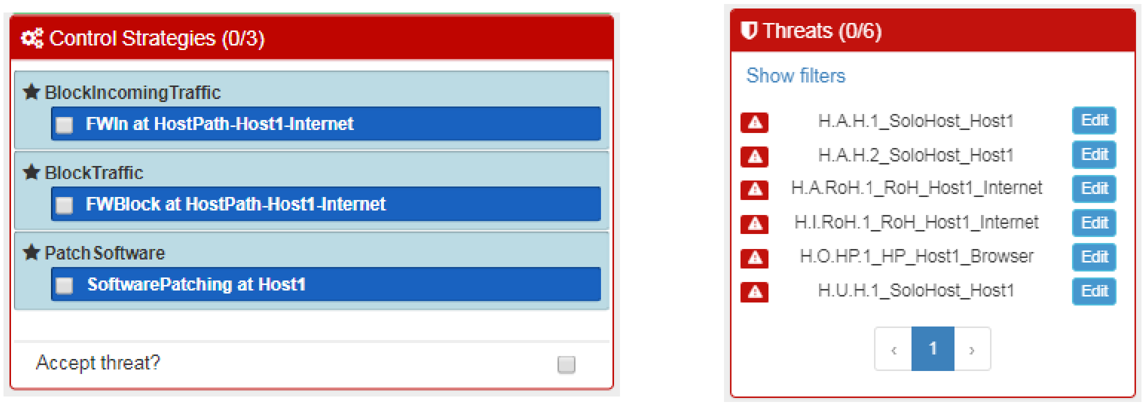


Figure 19 – Control Strategies in the Threat Editor

If any of the *Control Strategies* for a threat are active, it means that the threat is managed. Managed threats appear in green. Hovering over the threat icon inside the panel shows the management type (e.g. “blocked”), see Figure 20.

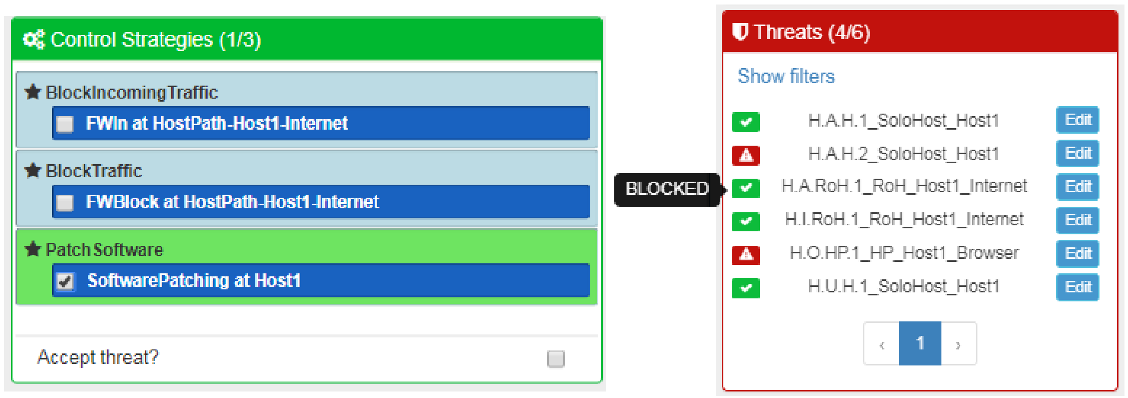


Figure 20 – Active Control Strategy

2.4.6.2 *Accepting a threat*

In cases where no control strategy exists for a given threat (e.g. see Figure 21), or the control strategy would be difficult or expensive to implement the user can instead “accept” the threat.

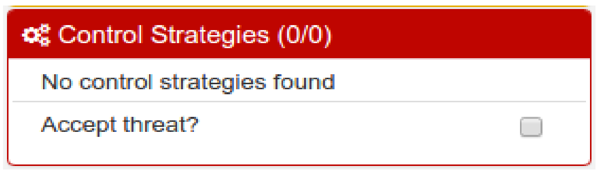
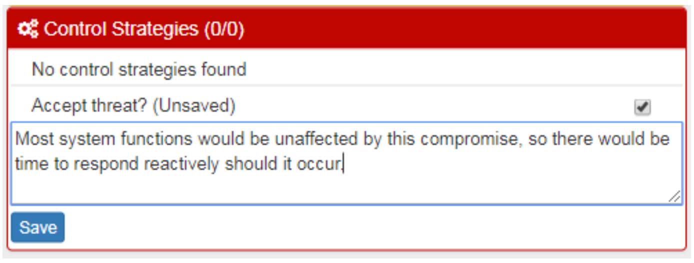


Figure 21 – Accepting threats

Accepting a threat means that the user accepts the risk posed by the threat (see Figure 22). The user must also type in a reason for accepting the threat.



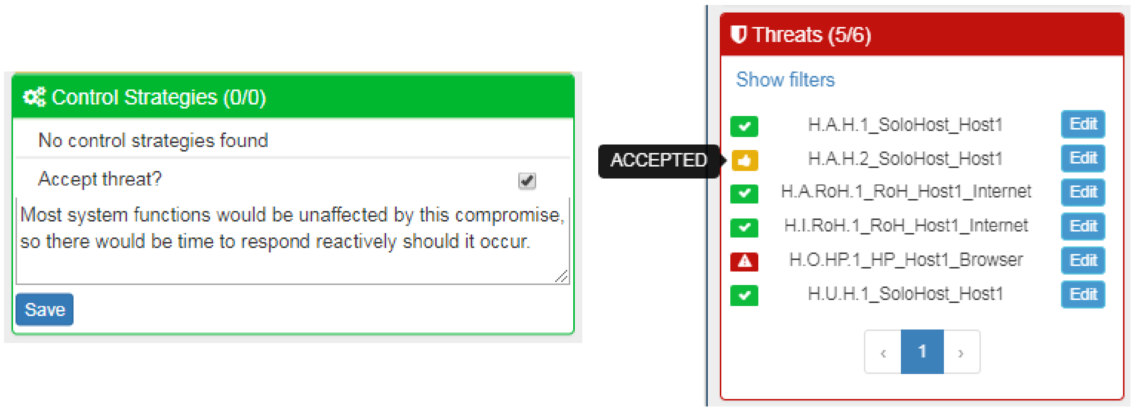
No control strategies found

Accept threat? (Unsaved) ✓

Most system functions would be unaffected by this compromise, so there would be time to respond reactively should it occur

Figure 22 – Accepting a threat

Upon saving, the icon for the accepted threat will change to indicate the new status (see Figure 23). "Accepted" threats are differentiated from other resolved threats, as they are highlighted in yellow.



No control strategies found

Accept threat? ✓

Most system functions would be unaffected by this compromise, so there would be time to respond reactively should it occur.

✓	H.A.H.1_SoloHost_Host1
👍	H.A.H.2_SoloHost_Host1
✓	H.A.RoH.1_RoH_Host1_Internet
✓	H.I.RoH.1_RoH_Host1_Internet
⚠	H.O.HP.1_HP_Host1_Browser
✓	H.U.H.1_SoloHost_Host1

Figure 23 – The UI after accepting a threat

3 Initial knowledge base

System Modeller as described in section 2 is a graphical threat modelling tool, which draws the information required for its reasoning capabilities from a knowledge base. Technically, this knowledge base is a graph store that contains:

- a core model describing assets, threats and controls modelled based on the ISO 27000 series of standards which forms the basis for all threat modelling
- domain models that describe a particular domain, such as "networking", or "consent". These domain models make up the building blocks from which system models can be composed and contain all pattern matching rules and threat definition in order to validate a system model
- system models that depend on the domain models

The knowledge in this knowledge base is captured based on the SHiELD use cases defined in WP6 and the tools developed in WP5. Figure 24 shows a consolidated overview of the architecture across all SHiELD use cases as detailed in D6.1. It is divided into different spaces, representing different jurisdictions with a common space that is shared by all of them. Each threat in the knowledge base contains a subset of these assets.

It is important to note that the use case diagram depicts a system at design-time, i.e. all assets in the diagram represent classes of things rather than individuals. The underlying model specifies the asset types used in such a diagram as well as the relationship types with which assets can be linked. We call this the asset model, which provides the building blocks of the knowledge base.

The asset model is then used to model patterns, i.e. groups of assets connected using specific relationship types. These patterns for the basis of threats. Every threat applies to a pattern, meaning it only occurs when the assets from its pattern exist in this exact constellation in a system model. A threat may cause any involved asset to misbehave. To block or mitigate such misbehaviours, it may define control strategies whereby controls get deployed on the asset to protect it from the effects of the threat.

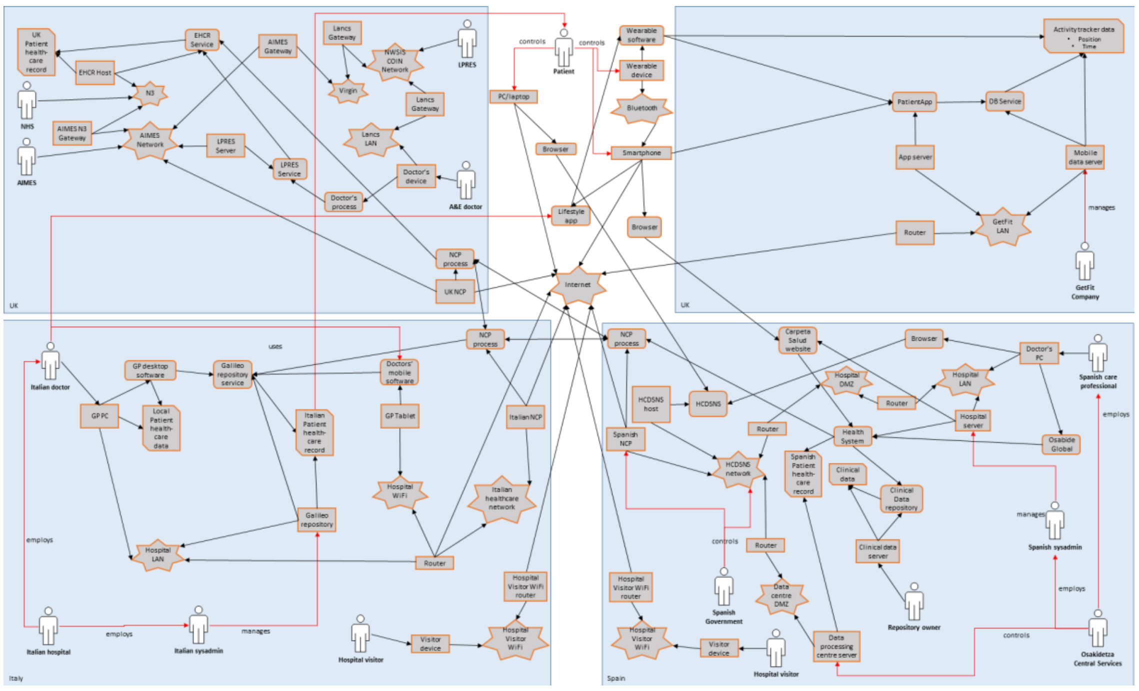


Figure 24 – Architectural overview

Figure 24 shows three jurisdictions, covering all use cases defined in D6.1.

FCSR on the Italian side is shown in detail in Figure 25. Doctors here use tablets to connect to the Galileo repository to retrieve patient data. This network has restricted access and is separate from the public WiFi.

Figure 26 shows Osakidetza and the Spanish part of the system. Doctors access patient data via their desktop machines from different repositories, which Basque as well as Spanish patient data. Devices need to have their MAC address registered before they're allowed to connect to the hospital LAN.

Figure 27 shows the UK side containing LPRES and their connection to the NHS. The N3 connection is provided by AIMES but does not currently use OpenNCP.

Figure 28 shows a fictional company called GetFit, which represents commercial providers of lifestyle and healthcare applications and tracking systems. The architecture is deliberately kept simple and contains only a data server which is connected to the internet. Depending on the size and business model of the commercial provider, this part of the architecture diagram may be much more complex.

The patient is shown in Figure 29, which is outside any jurisdiction, representing the fact that the patient is mobile and should be able to access their data independently of their physical location. The diagram covers end-to-end threats to data privacy and consent as well as common security threats arising from the use of networking.

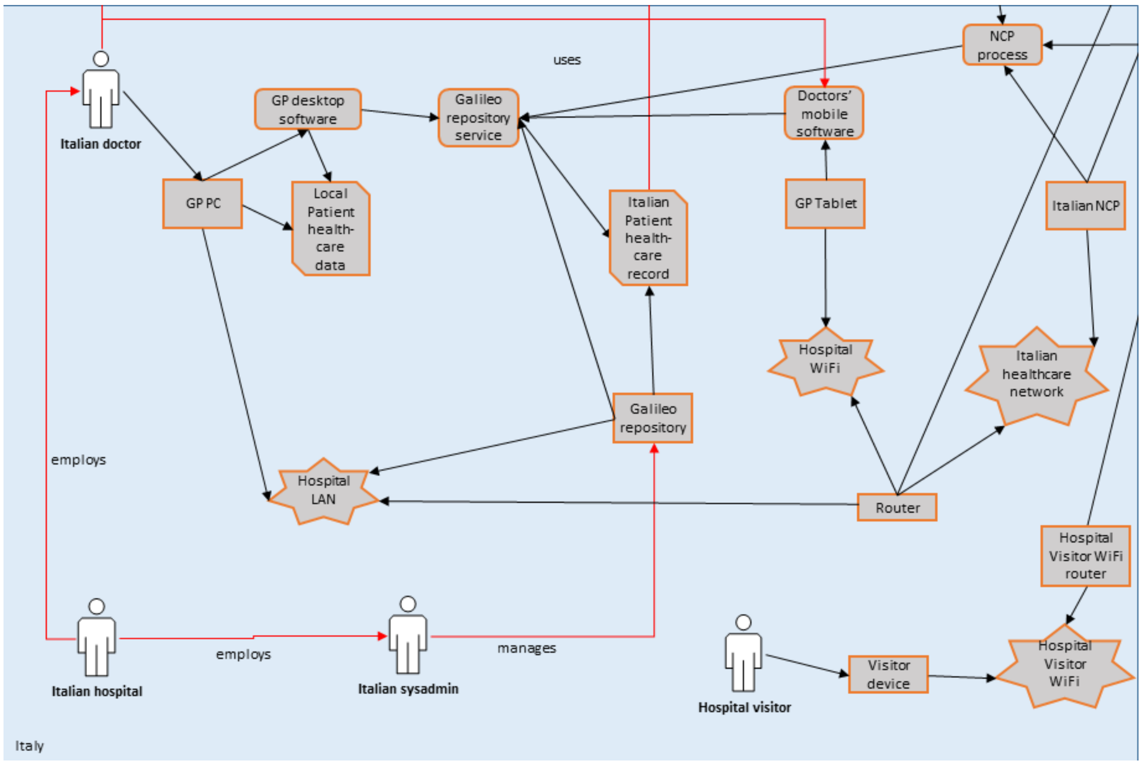


Figure 25 - Italy high-level architecture

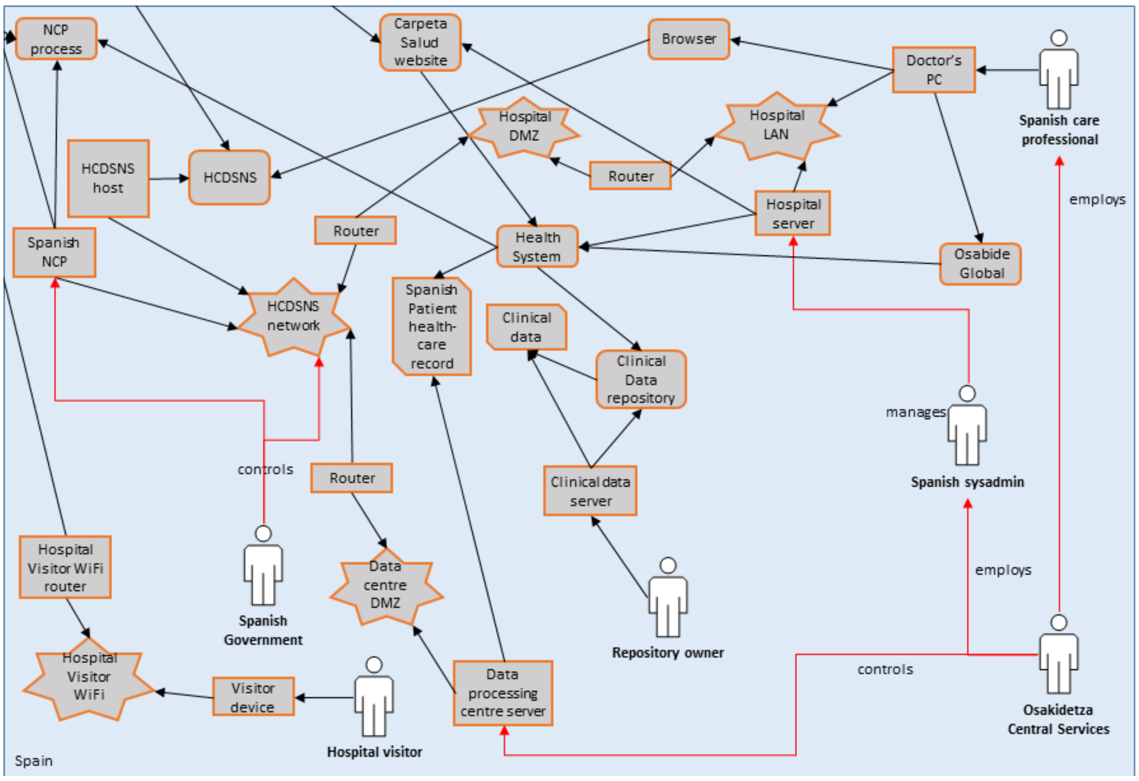


Figure 26 - Spain high-level architecture

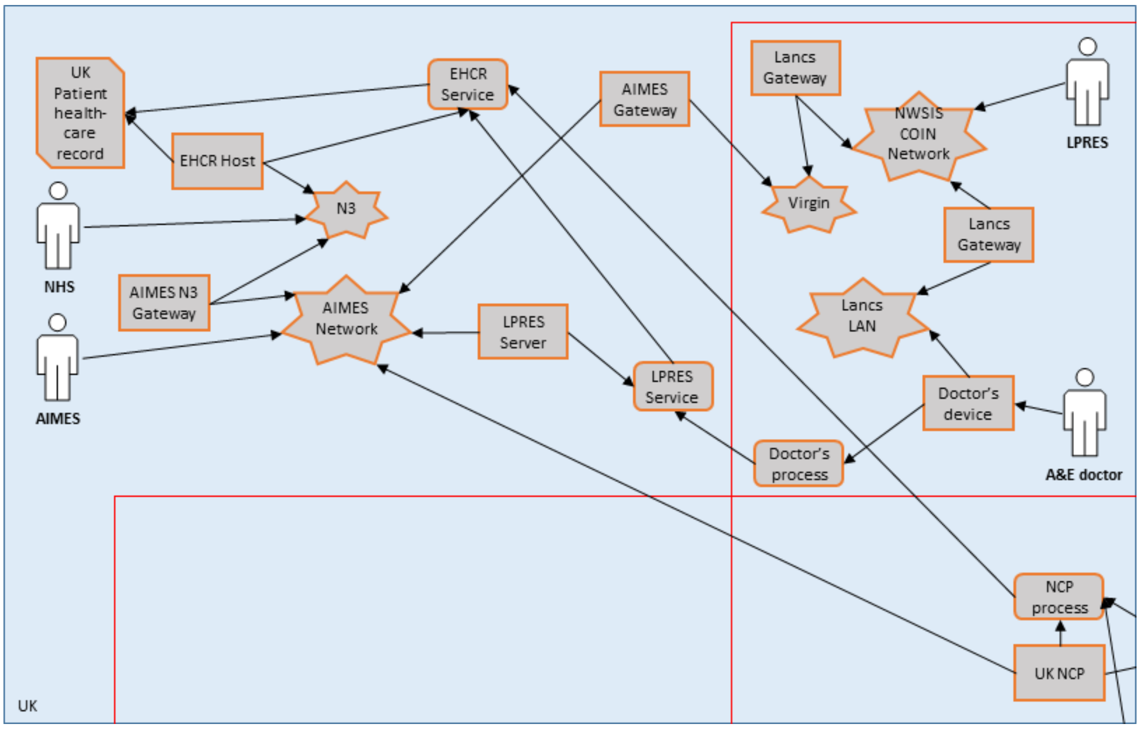


Figure 27 - UK high-level architecture

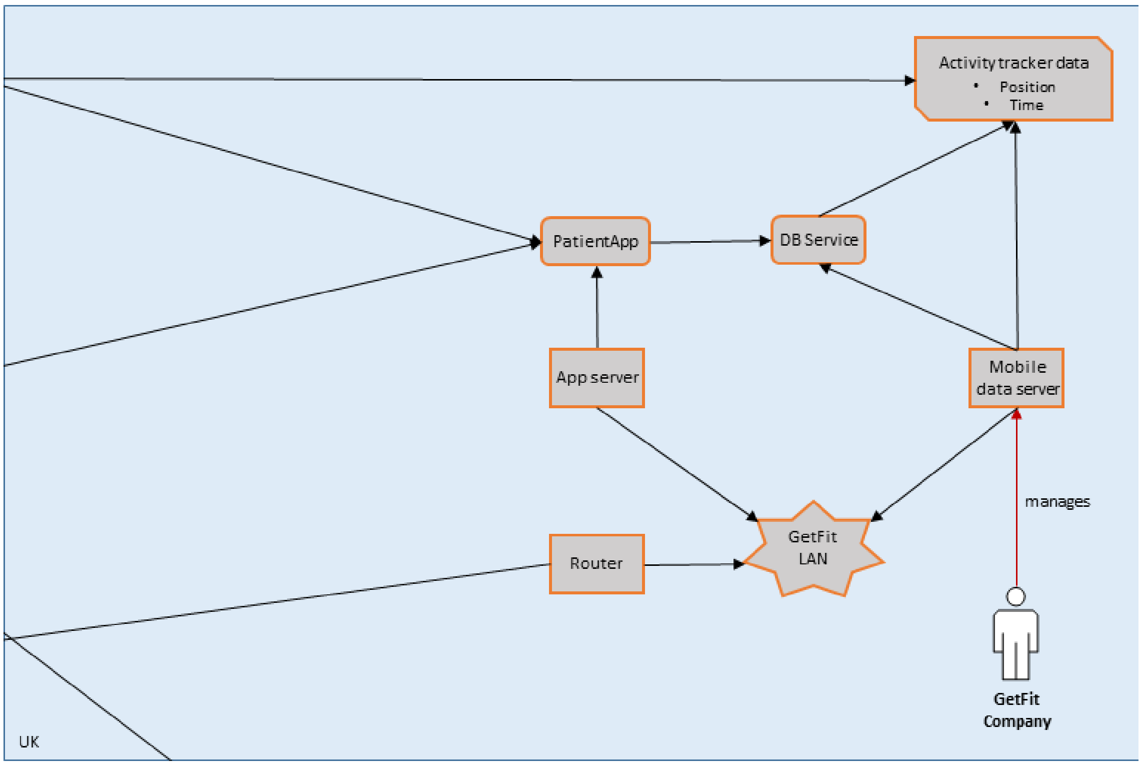


Figure 28 - Commercial provider high-level architecture

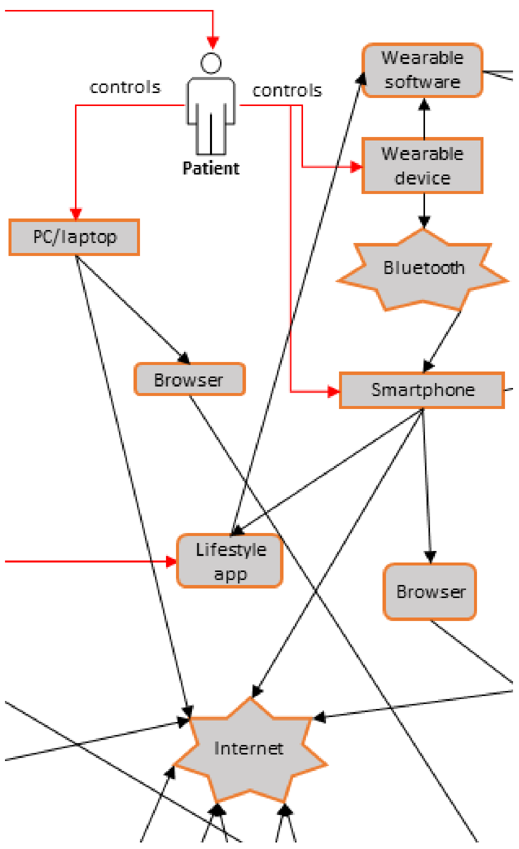


Figure 29 - Patient high-level architecture

Use Case 1 describes a "Break glass" circumstance, where an Italian patient has a stroke while on holiday in Spain. The access to patient data is not subject to consent as the patient's life depends on it.

Use Case 2 is about a Spanish patient travelling across Europe within two months of a surgical intervention. During his trip, sudden symptoms cause him to visit an Italian A&E department, where the doctors require access to data about his recent surgery. Due to the nature of the use case, the patient was able to give consent for sharing data prior to the trip for time-constrained, localised access.

Use case 3 deals with chronic conditions and remote monitoring. In this example, an Italian woman with diabetes, who currently lives in Spain travels to the UK for 3 months for work. She uses a Spanish remote monitoring app with access to her historic Italian data. Following a collapse at work, she is taken to A&E, where various data needs to be accessed, including third party monitoring data that originated from a wearable device and a healthcare/lifestyle mobile application.

3.1 Generic security threats

Apart from SHiELD-specific threats, initial versions of which we have identified in 3.2 and 3.3, the initial knowledge base prototype contains a range of "generic" security threats that apply to IT systems in general. The main categories covered are described in this section.

3.1.1 Remote Exploits

These threats describe an attacker gaining access through a remote subnet. This is usually achieved by exploiting a software vulnerability. Once the attacker has access, they can then access, modify or delete data and cause misbehaviours in various assets. We cover the following categories:

- **Anonymous remote exploits**
  where an anonymous attacker exploits a software vulnerability to gain access to a host or a process or data on a host
- **Network DoS attacks**
  where an attacker sends an excessive number of requests to overload a service, host or interface
- **Remote injections**
  where an attacker injects SQL code in user input fields to gain access to or corrupt a database
- **Remote service exploits**
  where an attacker conducts a malicious exchange using open service ports to corrupt, crash or deny access to the service or data it uses
- **Snooping attacks**
  where an attacker intercepts messages resulting in loss of confidentiality
- **Spoofing attacks**
  where an attacker impersonates a legitimate client or service in order to gain access to data or resources or to overload the system

3.1.2 Software Bugs

These threats relate to the low-level patterns in section 4.1.3. They model how software bugs can make hosts unreliable or unavailable. They can typically be treated by applying software patching to the hosts to fix the vulnerabilities.

3.1.3 Unauthorised Local Access

These threats describe situations in which an attacker gains access to a physical location, for instance a hospital server room, and uses console access to read, alter or corrupt stored data or processes, deny access to the host or its data or processes for legitimate users.

3.2 Mobile devices (UC Metrarc.1)

3.2.1 Use case description

In this use case a mobile device accesses health data, using a key that is stored and accessible on a device. A mobile device then connects to a data exchange service which includes a signing process with a stored private key that is accessible on the device.

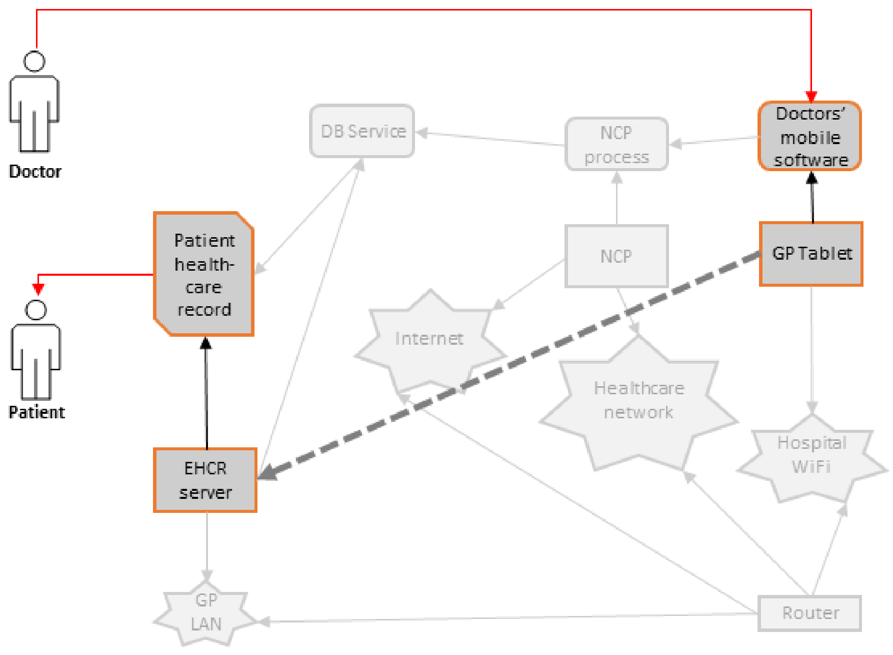


Figure 30 – UC Metrarc.1

For this scenario, it doesn't matter in which jurisdiction the data is stored or the mobile device is located. As long as there is a physical connection between the mobile device and the data, the identified threats for this use case pattern apply.

3.2.2 Identified threats

3.2.2.1 Interception of login credentials (T Metrarc.1.1)

The doctor is impersonated by an attacker, who acquired the doctor's password to log into the system. The attacker gains access to the patient record, resulting in a loss of confidentiality on the data as well as loss of trust at the patient whose data was leaked.

The mitigation strategy is to employ ICMetrics. This will require additional metrics on top of the correct password, which are derived from the doctor's use of the device so the attacker can't get into the system even with the stolen password.

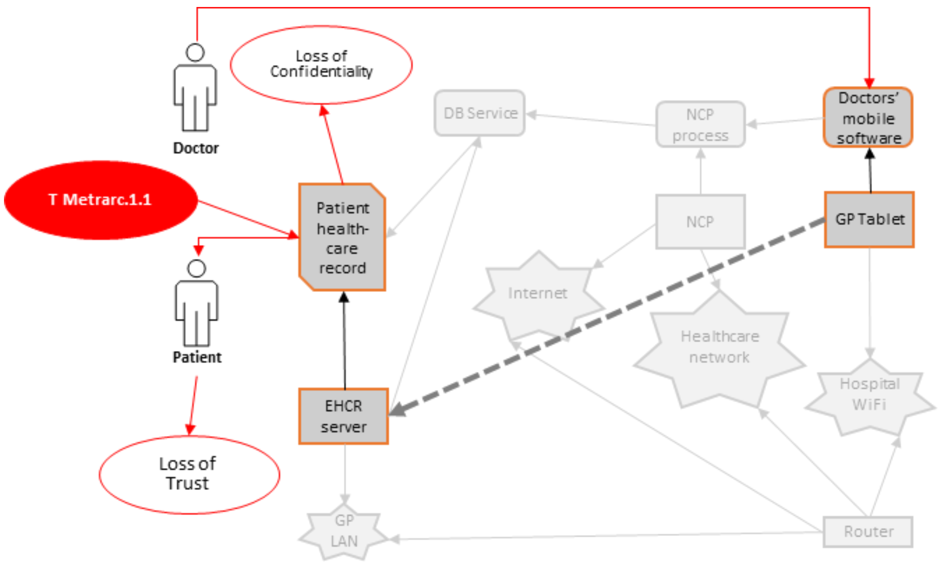


Figure 31 – T Metrarc.1.1

This threat can be controlled by applying ICMetrics controls to the relevant assets as shown in Figure 32.

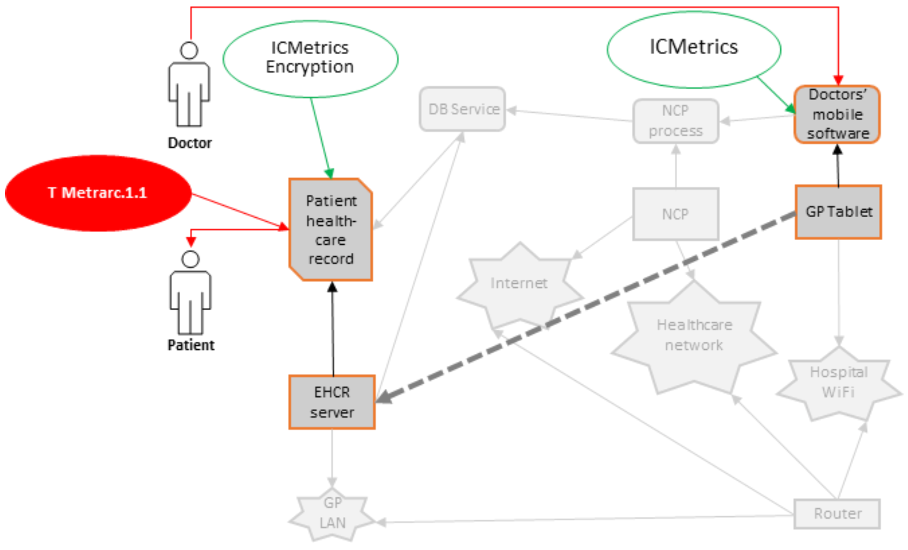


Figure 32 – T Metrarc.1.1 Control Strategy

3.3 Use case title (UC IBM.1)

3.3.1 Use case description

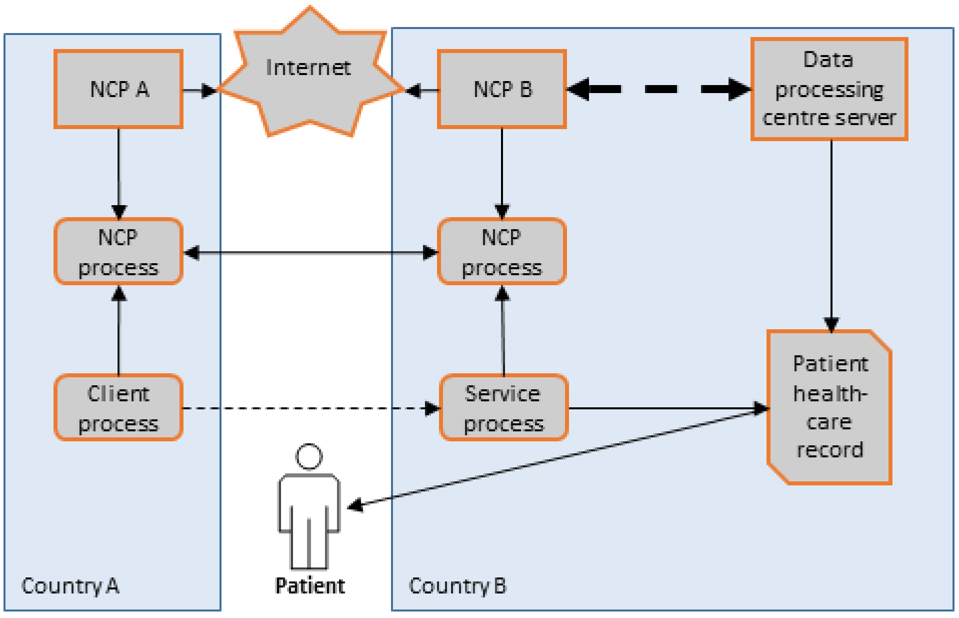


Figure 33 – Cross-jurisdictional data transfer

Figure 33, shows personal health data relating to a patient from country 'B', which resides on a health provider's servers in country 'B'. It is subsequently moved to another health provider in country 'A', caused by one of SHiELD's use cases. The data passes through the countries' national contact points.

3.3.2 Identified threats

3.3.2.1 Lawfulness (T IBM.1.1)

According to GDPR, all personal data must be collected/processed lawfully and fairly. Moving health data between different countries may violate the regulations of the involved countries as well as the EU's. This threat is a compliance threat that doesn't necessarily originate from malicious behaviour but rather from a misconfiguration or erroneous design.

As with most compliance threats, no specific control strategy exists to address the entire threat. A specific control may need to be applied to address any specific compliance issue. For example, modifying masking policies or redesigning the system data flow may be needed to prevent the compliance threat from occurring.

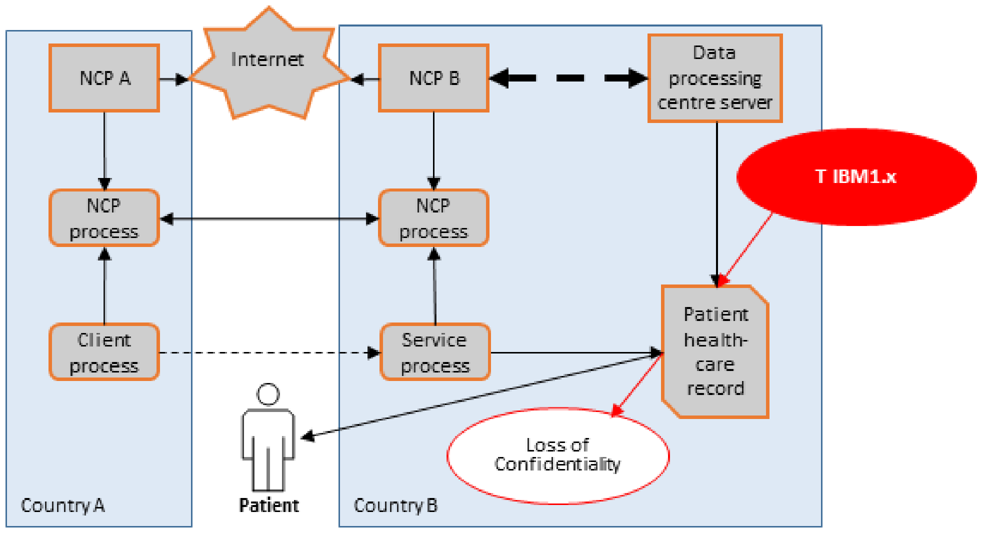


Figure 34 - Sensitive Data leaking

Control strategy:

The masking tool and the consent management tool can provide some mitigation to this threat by enforcing consent and addressing several privacy issues.

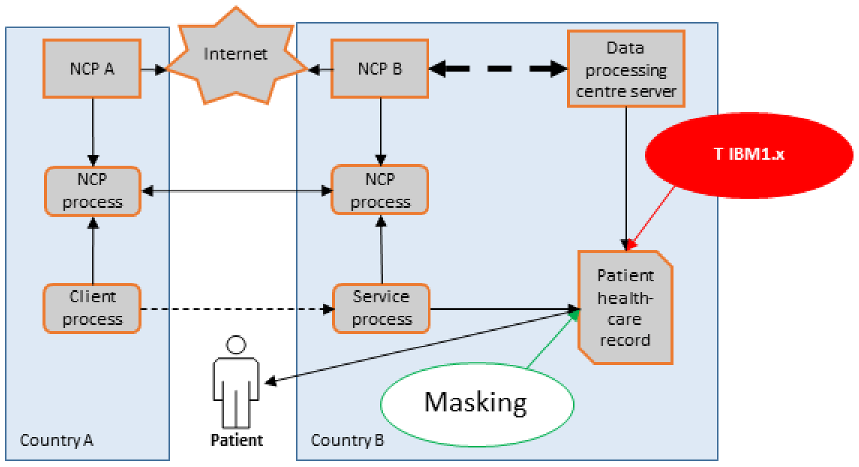


Figure 35 - Masking as a control strategy to prevent sensitive data leaking

4 Initial Secure Design patterns

A Secure Design pattern is a fuzzy concept which can be understood differently depending on the reader. As stated by G.Booch in [4]:

"Similarly, all well-structured software intensive systems are full of patterns. Architectural patterns serve the same role as song structure; design patterns and musical motifs are at the same level of abstraction; programmatic idioms and musical rhythms and scales are isomorphic." Therefore, when we are talking about a design pattern we are referring to architectural patterns. Obviously, there are different abstraction layers in any system, and the SHIELD project is not an

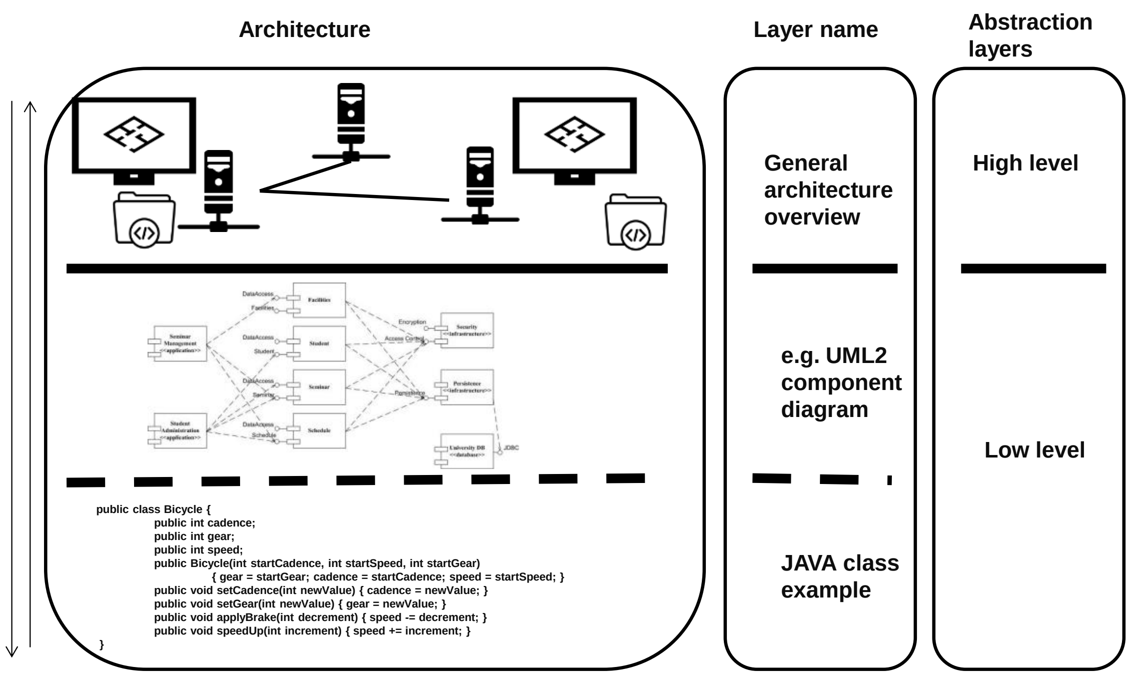
exception. Each abstraction layer has a specific purpose, and it represents specific elements on each layers. Obviously, each layer abstracts and does not represent other elements which are not included in the layer's purpose. For example, the high abstraction layer does not represent JAVA classes' structures. In addition, each person involved on each layer has different skills. For example, the higher abstraction layer can specify that a specific server/machine encrypts a connection between two servers. However, this layer omits which Java libraries are used to encrypt this communication. This approach has been widely used such as in [5] for describing service oriented architecture patterns.

Figure 36 represents different abstraction layers for describing a specific architecture. All these layers are intertwined and therefore implicit connections are set among these layers. These connections can be manually or automatically handled, but the SHIELD project is just focused on those manually made. The development of automatic mechanisms between these layers that can be reused [6] requires a software intensive development approach which is out of the scope of this project. The drawings in Figure 36 are just examples for each layer, and their goal is just illustrative.

Our purpose is to identify a set of architectural patterns for secure design on each layer. Each identified pattern has a specific purpose, and users should identify which ones apply to their systems and which ones are good enough [7] to be used. A recent study on design patterns [8] reveals that "*Pattern Development, Pattern Mining, and Pattern Usage are the most active topics in the field of design patterns*" [8]. In fact, each architectural pattern resolves a recurring problem, and this kind of situations can be dealt with a problem frame approach [9]. In this sense, we need to identify which security contexts are going to be solved. There is a vast amount of academic and industrial papers related to security and privacy in health care sector such as [10]. In fact, electronic health information systems are complex, and they raise several security and privacy issues [11]. In this sense we are going to deal with privacy aspects, and we are going to identify which patterns are stemming from SHIELD scenarios and use cases. As result of SHIELD WP2 D2.1 "eHealth security challenges" there is a set of eHealth security challenges. For each of these challenges a pattern should be identified. However, each challenge is related to several high-level patterns.

- Interoperability: there is no a specific pattern to deal with this aspect within the core Security Patterns Catalogue. Instead we are relying on HL7 (health Level 7) for data structure which is a standards Developing Organisations accredited by ANSI (American National Standards Institute). There is
- Confidentiality: data protection mechanisms such as encryption are associated to this challenge. From the pattern catalogue, "Secure Pipe" and "Secure Message Router" are two relevant patterns to be added.
- Privacy: [KR07] Privacy tool
- Regulations: ([KR06] Data protection mechanisms, [KR08] Legal recommendations Report), and [KR07] Privacy tool (to enforce regulation)
- eHealth related data: specific health related data may be identified appropriately ([KR06] Data protection mechanisms).
- PKI: OpenNCP relies on a specific PKI infrastructure. This challenge can be taken into account when defining the resulting architecture ([KR04] SHiELD open architecture and open secure interoperability API). We will be focused on Open NCP, and some PKI considerations and experiences should be taken into account in order to define an appropriate model.

It is relevant to highlight and to stress that security, including privacy, is not achieved only through technical means [10]. We need to include processes at management and organisational levels which allow and/or avoid specific practices for preserving privacy aspects. These levels include personal and behavioural aspects. However, our approach is focused on our abstraction layers which roughly are high level abstraction and low level abstraction layers. Therefore, we need to identify process patterns at each abstraction layer which will be considered as good practices. From a technical point of view and in order to preserve security and privacy in personal health records (PHR) [12] there are several approaches such as flexible attribute based encryption approaches to secure personal healthcare records in electronic health information systems [13] which can be used. Our approach is to include privacy preserving activities during design time of the resulting product/platform. Therefore we consider the Privacy by Design (PbD) approach [14] to be applied at high and low abstraction layers in order to deal with privacy issues.


Architecture
Layer name
Abstraction layers
General architecture overview
High level
e.g. UML2 component diagram
Low level
JAVA class example
DataAccess
Facilities
Seminar Management <<application>>
DataAccess
Student
Student
DataAccess
Seminar
Seminar
Student Administration <<application>>
DataAccess
Schedule
Schedule
Encryption
Access Control
Security <<infrastructure>>
Persistence
Persistence <<infrastructure>>
University DB <<database>>
JDBC

```
public class Bicycle {
            public int cadence;
            public int gear;
            public int speed;
            public Bicycle(int startCadence, int startSpeed, int startGear)
                        { gear = startGear; cadence = startCadence; speed = startSpeed; }
            public void setCadence(int newValue) { cadence = newValue; }
            public void setGear(int newValue) { gear = newValue; }
            public void applyBrake(int decrement) { speed -= decrement; }
            public void speedUp(int increment) { speed += increment; }
}
```

Figure 36: Example of the abstraction layers[1]

4.1 Patterns

4.1.1 Security patterns

There are several approaches for identifying security patterns. Chris Steel, Ramesh Nagappan and Ray Lai published a referenced book related to security patterns [15], and they have become a reference for security patterns. From this reference study we have adopted the same logical tiers.

[1] We are using examples extracted from different sources such as http://agilemodeling.com/artifacts/componentDiagram.htm for describing UML2 components.

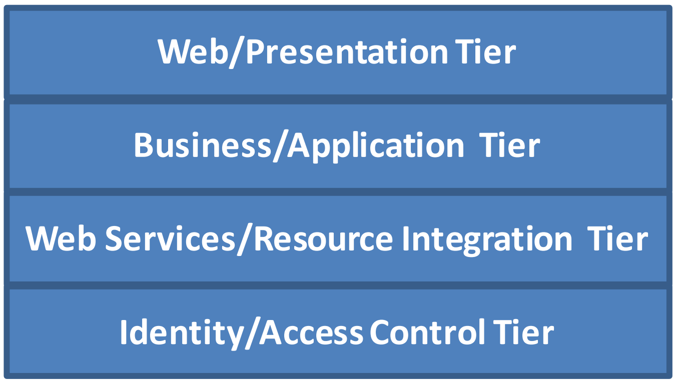

Web/Presentation Tier
Business/Application Tier
Web Services/Resource Integration Tier
Identity/Access Control Tier

Figure 37: Different Tiers from which we can identify patterns

Each pattern follows the same template [15]:

- *Problem*: Describes security issues tackled by the pattern.
- *Forces*: Describes main motivations and constraints related to a problem.
- *Solution*: Describes the approach and the associated mechanisms in detail.
- *Consequences*: Describes the outcomes of using the security pattern
- *Security Factors and Risks*: Describes factors and risks to be considered while applying the pattern.
- *Reality Checks*: Describes a set of review items to identify the feasibility and practicality of the pattern.
- *Related Patterns*: Lists other related patterns from the Security Patterns Catalogue or from other related sources.

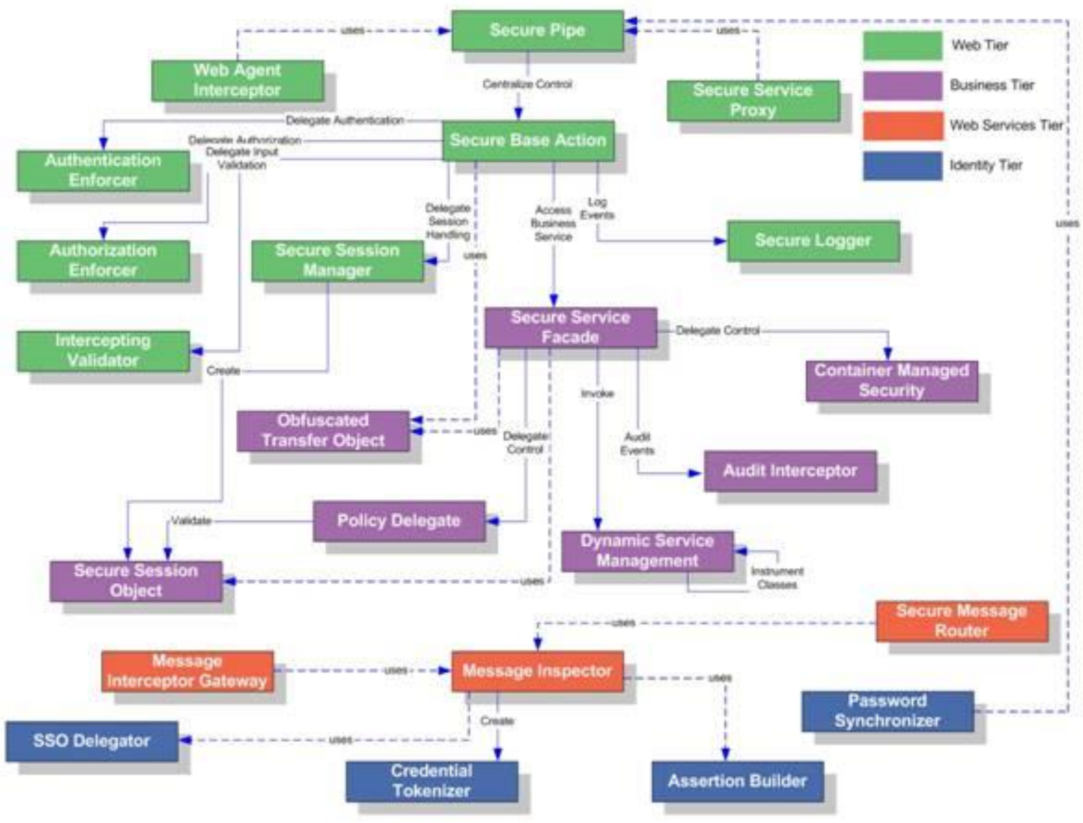


Figure 38: Class diagram security pattern [15]

Basically, for each Tier we have identified the following patterns:

- Web tier:
  - Authentication Enforcer
  - Authorization Enforcer
  - Intercepting Validator
  - Secure Base Action
  - Secure Logger
  - Secure Pipe
  - Secure Service Proxy
  - Secure Session Manager
  - Intercepting Web Agent
- Business tier:
  - Audit Interceptor
  - Container Managed Security
  - Dynamic Service Management
  - Obfuscated Transfer Object
  - Policy Delegate
  - Secure Service Facade
  - Secure Session Object
- Web services tier:
  - Message Inspector
  - Message Interceptor Gateway
  - Secure Message Router
- Identity management:

- Assertion Builder
- Credential Tokenizer
- Single Sign-on (SSO) Delegator
- Password Synchronizer

4.1.2 High level patterns

Patterns as described in section 3 can be used not only to define threats. A pattern describes a group of assets and how they are connected. During the design of a system in System Modeller, users will effectively define their own "system patterns" by dragging and dropping new assets onto the canvas and connecting them. The system will then be validated, i.e. threat patterns will be matched against it to find any potential threats. These threat patterns are invisible to the user, meaning they only see threats after the validation has completed. This then starts an incremental process of revising the system model, revalidating, analysing the threats etc. This process can be accelerated by providing "pre-approved" patterns to the user. Such patterns could be classified as compliant with regulations in one or more jurisdictions or excluding certain security threats by default. The user would have the option to start from a pre-defined pattern as opposed to a blank canvas. Such high-level patterns can be obtained by analysing the compliance regulations. This will be done in cooperation with WP3 and WP6.

Once these high-level patterns have been identified, they need to be mapped to the low-level patterns. This will ensure that the benefits of using low-level patterns propagates through to the system designer. Mapping can be implemented in two ways or a combination thereof:

- We can introduce new asset types as subclasses of "Process" that imply the process uses certain design patterns. When using these new asset types, they might form part of different patterns or exclude matches of processes based on their secure design.
- We can create new controls that represent low-level design patterns. If a threat has been found in the system, these controls could then be applied to processes to manage the threats.

4.1.3 Low level patterns

This low level is tightly related to the epSOS platform, and on its implementation. Basically, the programming language used on this platform is JAVA, and therefore we investigate its Java Application Programming Interface (API). Our aim is to identify which JAVA patterns are used on this platform and which security mechanisms can be put in place for mitigating the identified privacy threats. As shown previously, Table 3 represents a summary of the CWE used during our analysis. These implementations are our low level patterns.

4.2 Preliminary benefits from using patterns

The main benefit is that we are able to identify what CWEs threats are within the current OpenNCP. A secondary benefit is related to technical debt and code smell. If we are able to identify these "issues" during the development phase, we will reduce the efforts of rework in later stages.

This is an iterative process which should be carried out during the whole development process. From an empirical point of view. We need to analyse in depth OpenNCP, and we need to set up OpenNCP instances without these issues.

4.3 Threat identification tool

There is no specific use case for application security. However, we are considering and gathering the main recommendations described by the community such as OWASP (https://www.owasp.org/images/7/72/OWASP_Top_10-2017_%28en%29.pdf.pdf), SANS (https://www.sans.org/top25-software-errors/) and MITRE (http://cwe.mitre.org/top25/).

4.3.1 Identified threats

The identified threats are based on what the community has already identified as potential threats in Java based applications. OWASP (http://www.owasp.org) is a free and open software security community, and the OWASP contributors have dedicated efforts on defining the Ten Most Critical Web Application Security Risks, and they have been ordered as:

- A1:2017 - Injection
- A2:2017 - Broken Authentication
- A3:2017 - Sensitive Data Exposure
- A4:2017 - XML External Entities (XXE)
- A5:2017 - Broken Access Control
- A6:2017 - Security Misconfiguration
- A7:2017 - Cross-Site Scripting (XSS)
- A8:2017 - Insecure Deserialization
- A9:2017 - Using Components with Known Vulnerabilities
- A10:2017 - Insufficient Logging & Monitoring

MITRE Corporation (https://www.mitre.org/) is a US not-for-profit company that operates multiple federally funded research and development centres. This organisation maintains the CWE (Common Weakness Enumeration - https://cwe.mitre.org/) web site, which is one of the most well-known database of threats. They receive of the US Department of Homeland Security's National Cyber Security Division, and they describe in detail the top 25 Software errors. They include more than 700 additional Software errors, design errors and architecture errors that can lead to exploitable vulnerabilities. SHIELD is not going to deal with all these threats.

Table 3: CWE identified for SHIELD based on Sonar Java analyser [2]

CWE ID	CWE Name	Implementing Rules
CWE-20	Improper Input Validation	S2077 SQL binding mechanisms should be used
CWE-78	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')	S2076 Values passed to OS commands should be sanitized
CWE-88	Argument Injection or Modification	S2076 Values passed to OS commands should be sanitized
CWE-89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')	S2077 SQL binding mechanisms should be used
CWE-90	Improper Neutralization of Special Elements used in an	S2078 Values passed to LDAP queries should be sanitized

[2] (https://tinyurl.com/y9wfwsor)

CWE ID	CWE Name	Implementing Rules
	LDAP Query ('LDAP Injection')	
CWE-102	Struts: Duplicate Validation Forms	S3374 Struts validation forms should have unique names
CWE-190	Integer Overflow or Wraparound	S2184 Math operands should be cast before assignment
CWE-259	Use of Hard-coded Password	S2068 Credentials should not be hard-coded
CWE-284	Improper Access Control	S3369 Security constraints should be defined
CWE-293	Using Referer Field for Authentication	S2089 HTTP referers should not be relied on
CWE-326	Inadequate Encryption Strength	S2278 Neither DES (Data Encryption Standard) nor DESede (3DES) should be used
		S2245 Pseudorandom number generators (PRNGs) should not be used in secure contexts
CWE-327	Use of a Broken or Risky Cryptographic Algorithm	S2278 Neither DES (Data Encryption Standard) nor DESede (3DES) should be used
		S2277 Cryptographic RSA algorithms should always incorporate OAEP (Optimal Asymmetric Encryption Padding)
		S2258 "javax.crypto.NullCipher" should not be used for anything other than testing
		S2257 Only standard cryptographic algorithms should be used
		S2070 SHA-1 and Message-Digest hash algorithms should not be used
CWE-328	Reversible One-Way Hash	S2070 SHA-1 and Message-Digest hash algorithms should not be used
CWE-330	Use of Insufficiently Random Values	S2245 Pseudorandom number generators (PRNGs) should not be used in secure contexts
CWE-338	Use of Cryptographically Weak Pseudo-Random Number Generator (PRNG)	S2245 Pseudorandom number generators (PRNGs) should not be used in secure contexts
CWE-369	Divide By Zero	S3518 Zero should not be a possible denominator
CWE-374	Passing Mutable Objects to an Untrusted Method	S2384 Mutable members should not be stored or returned directly
CWE-375	Returning a Mutable Object to an Untrusted Caller	S2384 Mutable members should not be stored or returned directly
CWE-382	J2EE Bad Practices: Use of System.exit()	S1147 Exit methods should not be called
CWE-391	Unchecked Error Condition	S2142 "InterruptedException" should not be ignored
CWE-395	Use of NullPointerException Catch to Detect NULL Pointer Dereference	S1696 "NullPointerException" should not be caught
CWE-396	Declaration of Catch for Generic Exception	S2221 "Exception" should not be caught when not required by called methods
		S1181 Throwable and Error should not be caught

CWE ID	CWE Name	Implementing Rules
CWE-397	Declaration of Throws for Generic Exception	S00112 Generic exceptions should never be thrown
CWE-412	Unrestricted Externally Accessible Lock	S2445 Blocks should be synchronized on "private final" fields
CWE-413	Improper Resource Locking	S2445 Blocks should be synchronized on "private final" fields
CWE-459	Incomplete Cleanup	S2095 Resources should be closed
		S2222 Locks should be released
CWE-476	NULL Pointer Dereference	S2225 "toString()" and "clone()" methods should not return null
		S3655 Optional value should only be accessed after calling isPresent()
		S2447 Null should not be returned from a "Boolean" method
		S2259 Null pointers should not be dereferenced
		S2637 "@NonNull" values should not be set to null
CWE-477	Use of Obsolete Functions	CallToDeprecatedMethod "@Deprecated" code should not be used
CWE-478	Missing Default Case in Switch Statement	SwitchLastCaseIsDefaultCheck "switch" statements should end with "default" clauses
CWE-481	Assigning instead of Comparing	AssignmentInSubExpressionCheck Assignments should not be made from within sub-expressions
CWE-483	Incorrect Block Delimitation	S2681 Multiline blocks should be enclosed in curly braces
CWE-484	Omitted Break Statement in Switch	S128 Switch cases should end with an unconditional "break" statement
CWE-486	Comparison of Classes by Name	S1872 Classes should not be compared by name
CWE-489	Leftover Debug Code	S2589 Boolean expressions should not be gratuitous
		S2583 Conditionally executed blocks should be reachable
		S2653 Web applications should not have a "main" method
CWE-493	Critical Public Variable Without Final Modifier	ClassVariableVisibilityCheck Class variable fields should not have public accessibility
CWE-500	Public Static Field Not Marked Final	S1444 "public static" fields should be constant
CWE-501	Trust Boundary Violation	S3318 Untrusted data should not be stored in sessions
CWE-546	Suspicious Comment	S1135 Track uses of "TODO" tags
		S1134 Track uses of "FIXME" tags
CWE-563	Assignment to Variable without Use ('Unused Variable')	S1854 Dead stores should be removed
CWE-564	SQL Injection: Hibernate	S2077 SQL binding mechanisms should be used
CWE-568	finalize() Method Without super.finalize()	ObjectFinalizeOverridenCallsSuperFinalizeCheck "super.finalize()" should be called at the end of "Object.finalize()" implementations

CWE ID	CWE Name	Implementing Rules
CWE-570	Expression is Always False	S2583 Conditionally executed blocks should be reachable
CWE-571	Expression is Always True	S2589 Boolean expressions should not be gratuitous
		S2583 Conditionally executed blocks should be reachable
CWE-572	Call to Thread run() instead of start()	S1217 Thread.run() should not be called directly
CWE-579	J2EE Bad Practices: Non-serializable Object Stored in Session	S2441 Non-serializable objects should not be stored in "HttpSession" objects
CWE-580	clone() Method Without super.clone()	S1182 Classes that override "clone" should be "Cloneable" and call "super.clone()"
CWE-581	Object Model Violation: Just One of Equals and Hashcode Defined	S1206 "equals(Object obj)" and "hashCode()" should be overridden in pairs
CWE-582	Array Declared Public, Final, and Static	S2386 Mutable fields should not be "public static"
CWE-583	finalize() Method Declared Public	S1174 "Object.finalize()" should remain protected (versus public) when overriding
CWE-584	Return Inside Finally Block	S1143 Jump statements should not occur in "finally" blocks
CWE-586	Explicit Call to Finalize()	ObjectFinalizeCheck The Object.finalize() method should not be called
CWE-594	J2EE Framework: Saving Unserializable Objects to Disk	S1948 Fields in a "Serializable" class should either be transient or serializable
CWE-595	Comparison of Object References Instead of Object Contents	S1698 "==" and "!=" should not be used when "equals" is overridden
CWE-597	Use of Wrong Operator in String Comparison	S1698 "==" and "!=" should not be used when "equals" is overridden
CWE-600	Uncaught Exception in Servlet	S1989 Exceptions should not be thrown from servlet methods
CWE-607	Public Static Final Field References Mutable Object	S2386 Mutable fields should not be "public static"
CWE-609	Double-Checked Locking	S2168 Double-checked locking should not be used
CWE-614	Sensitive Cookie in HTTPS Session Without 'Secure' Attribute	S2092 Cookies should be "secure"
CWE-754	Improper Check for Unusual or Exceptional Conditions	S899 Return values should not be ignored when they contain the operation status code
CWE-780	Use of RSA Algorithm without OAEP	S2277 Cryptographic RSA algorithms should always incorporate OAEP (Optimal Asymmetric Encryption Padding)
CWE-783	Operator Precedence Logic Error	S864 Limited dependence should be placed on operator precedence rules in expressions
CWE-798	Use of Hard-coded Credentials	S2068 Credentials should not be hard-coded

CWE ID	CWE Name	Implementing Rules
CWE-807	Reliance on Untrusted Inputs in a Security Decision	S2254 "HttpServletRequest.getRequestedSessionId( )" should not be used
		S2089 HTTP referers should not be relied on
CWE-835	Loop with Unreachable Exit Condition ('Infinite Loop')	S888 Equality operators should not be used in "for" loop termination conditions
CWE-943	Improper Neutralization of Special Elements in Data Query Logic	S2077 SQL binding mechanisms should be used

SANS (https://www.sans.org) is a non for profit entity which maintains a series of assessments of secure coding skills in three languages along with certification exams that allow programmers to determine gaps in their knowledge of secure coding (https://www.sans.org/top25).

All of them are applicable to SHIELD use case scenarios, but we are going to be focused on the top threats identified by these entities.

4.3.2 Design of the solution

This subsection proposes a small tool for identifying some of these threats from the code we are generating. For example, this tool helps programmers during the development of the OpenNCP platform. Tools are helpful during code development, review and testing phases. However, we consider tools can generate some false positives or issues that should be solved by humans. This means that despite this tool this approach requires human based supervision.

Next Figure 39 represents our prototype. We consider that the development environment is the Eclipse platform and our SHIELD analyser reads JAVA projects and identifies threats identified by MITRE, SANS and OWASP. This tool summarizes these metrics and provides a Metrics Score Card, and a Pattern visualizer containing a set of security patterns.

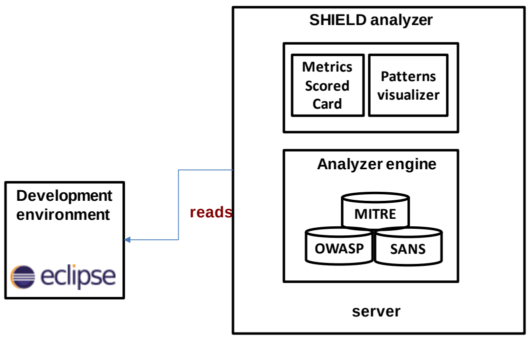


Figure 39 – SHiELDanalyzer architecture

5 Conclusions

This accompanying report for the software deliverable D4.1 describes the work done on WP4 in M7-M12.

We present the initial prototype of “System Modeller” and the underlying knowledge base.

The tool itself has been extended in the past six months, focusing mainly on performance and usability improvements. In parallel, we have been working on concepts for extensions to the software to support some of the key features SHiELD tries to address. Specifically we will be working on visualising compliance in the user interface.

The knowledge base continues to be updated as the project as a whole makes more progress towards modelling the use cases and understanding the requirements to correctly represent the infrastructure as well as model

- security threats that can affect cross-border health data exchange;
- controls representing tools developed in WP5 that block threats;
- regulatory compliance in different jurisdictions with input from WP3 and WP6.

Furthermore, we describe our plan for the creation of a secure design pattern catalogue and how we intend to link the concepts of low-level patterns and high-level patterns. This plan will be implemented starting with the next release (D4.2) which is due at the end of M18 and contain some low-level design patterns linked to controls as well as high-level patterns to support users in building compliant systems more easily.

6 References

1. Surridge, M., Nasser, B., Chen, X., Chakravarthy, A., & Melas, P., Run-Time Risk Management in Adaptive ICT Systems. In Eighth International Conference on Availability, Reliability and Security (ARES), 2013, (pp. 102-110). IEEE (2013).
2. Chakravarthy, A., Wiegand, S., Chen, X., Nasser, B. and Surridge, M., Trustworthy Systems Design using Semantic Risk Modelling. Procs 1st International Conference on Cyber Security for Sustainable Society, Coventry, UK, 2015, (pp. 49-81). Digital Economy Sustainable Society Network. (2015).
3. The MITRE Corporation: Adversarial Tactics, Techniques, and Common Knowledge (ATT&CK™). https://attack.mitre.org (accessed 11/12/2017)
4. Booch, G.: The Well-Tempered Architecture. IEEE Softw. 24, 24–25 (2007).
5. Rothenhaus, K.J., Michael, J.B., Man-Tak Shing: Architectural Patterns and Auto-Fusion Process for Automated Multisensor Fusion in SOA System-of-Systems. IEEE Syst. J. 3, 304–316 (2009).
6. Kallel, S., Tramoni, B., Tibermacine, C., Dony, C., Kacem, A.H.: Generating reusable, searchable and executable “architecture constraints as services.” J. Syst. Softw. 127, 91–108 (2017).
7. Booch, G.: Goodness of Fit. IEEE Softw. 23, 14–15 (2006).
8. Bafandeh Mayvan, B., Rasoolzadegan, A., Ghavidel Yazdi, Z.: The state of the art on design patterns: A systematic mapping of the literature. J. Syst. Softw. 125, 93–118 (2017).
9. Michael Jackson: Problem frames: analyzing and structuring software development problems. Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA.
10. Cavalli, E., Mattasoglio, A., Pinciroli, F., Spaggiari, P.: Information security concepts and practices: the case of a provincial multi-specialty hospital. Int. J. Med. Inf. 73, 297–303 (2004).
11. Bertino, E., Deng, R.H., Huang, X., Zhou, J.: Security and privacy of electronic health information systems. Int. J. Inf. Secur. 14, 485–486 (2015).
12. Qian, H., Li, J., Zhang, Y., Han, J.: Privacy-preserving personal health record using multi-authority attribute-based encryption with revocation. Int. J. Inf. Secur. 14, 487–497 (2015).
13. Qin, B., Deng, H., Wu, Q., Domingo-Ferrer, J., Naccache, D., Zhou, Y.: Flexible attribute-based encryption applicable to secure e-healthcare records. Int. J. Inf. Secur. 14, 499–511 (2015).
14. Schaar, P.: Privacy by Design. Identity Inf. Soc. 3, 267–274 (2010).
15. Steel, C., Nagappan, R., Lai, R.: core Security Patterns. Pearson Education (2005).