Security Engineering and Modelling of Set-top Boxes

Jose Fran. Ruiz, Andre Rein
Fraunhofer Institute for Secure Information Technology
Darmstadt, Germany
{jose.ruiz, andre.rein}@sit.fraunhofer.de

Marcos Arjona, Antonio Maña
Computer Science Department
University of Malaga
Malaga, Spain
{marcos,amg}@lcc.uma.es

Antoine Monsifrot, Michel Morvan
Technicolor, Security & Content Protection Labs
Cesson Sévigné, France
{antoine.monsifrot, Michel.Morvan}@technicolor.com

Abstract— **This paper presents a security engineering process for the development of secure systems focusing on the specification and development of the Set-top Boxes. The paper describes the Set-top Box characteristics and functionalities and, using the process and its secure artefacts, models what we call a Domain Security Metamodel that defines all the security properties of that domain and implements them using Security Building Blocks. This security artefact can be used by system engineers when modelling their system model in order to fulfil its security requirements and, as a result, create a secure system that has security naturally integrated in its architecture and functionality.**

Keywords— security engineering; security building blocks; security modelling

I. INTRODUCTION

A Set-top Box (STB) is a device that extends the capability of a television, allowing it to become a user interface for PayTV, Video on Demand, internet navigation, buy and load applications from an app store, receive streaming from several events and use it as a multimedia system (photos, personal movies, etc.). Although it started with minimal capabilities and limitations, the functionalities of the STB has evolved and transformed from a digital television decoding artefact to an internet multimedia system, which actually is more like a personal computer than a television artefact.

With the evolution and expansion of the functionalities over Internet the threats of the STB has grown exponentially. There exist now data that is streamed over the Internet, the users can buy applications and install them in the STB, store critical and confidential information in the system, etc. The development of this device is a very complex task due to several reasons. First, STBs are embedded devices with real time constraints and their development must be cheap due to market pressure. Second, the STBs provide new features such as Internet browser, games or the possibility to buy movies. These functionalities use very different constraints from a security point of view. The distribution of paying movies, as we commented before, requires some special security features in the system because it works with copyright protected data that A content leakage exposes business of content owners and distributors (broadcasters or service providers). For that reason, all the critical assets must be protected and the possible security threats secured with domain-specific security properties. For example, as Jon Brodkin [1] explains, a web browser potentially introduces a lot of vulnerabilities in the system.

Due to all these special characteristics and functionalities, we have to define some trade-off between security, features, cost and performance. In this context, it is important that system engineers and architects understand the security risks of the system in order to accept the implementation of the appropriate protections. Building strong security should therefore rely on a structured and rigorous security process through the development lifecycle.

We present in this paper a security engineering process for the development of secure systems and a security artefact created using this process that provides domain-specific security properties of the STBs along with the description of its constraints, characteristics, usual threats, attacks, assumptions, etc. The process allows to develop and to use security solutions in order to satisfy the security requirements of a target domain. It provides a framework composed of different security artefacts and processes that, each of them, aim at different objectives with different responsibilities. Its main objective is to help developers and engineers in the management of security aspects and its use in System Models. Due to size limits we describe briefly the security engineering process and the rest of its elements. The approach and work presented have been developed in the SecFutur project [2]. The reader can find more information of this process and the description and composition of the artefacts in [3]. The security aspects are defined as Domain Security Metamodels, which contain the information and security properties of specific domains (smart metering, forest control systems, Set-top Box systems, etc.).

We focus in this paper in the description of the DSM created for the Set-top Box domain presented in Section 2 and its security properties. After describing the DSM we focus in a particular security property: "Secure Data Storage". We present an extract of the DSM containing this security property and describe in depth its elements, the Security Building Block Solution, the elements of the system where it can be applied, etc.

II. SET-TOP BOX DOMAIN DESCRIPTION

The STB is a mass-market consumer device that is connected to the user's television and controlled by the user through a simple user interface. However, the STB functionalities have changed since its first version.

In the early days, its role was limited to receiving protected television signals, decrypting and decoding them to deliver a video signal to be displayed on the television. The emergence of Internet-based services has allowed service providers to propose various additional functionalities such as video on demand, catch-up television, Internet browsing, applications store, subscriber account management, customer private data (e.g., family pictures), etc. It also led to the introduction of Digital Rights Management (DRM) systems.

These changes have completely modified the security protection needs. In previous generations, STBs used dedicated processors and operating systems. Their security was mostly edicted by the Conditional Access vendors (CAS) and mainly based on the use of smartcards. These devices operated in a closed and controlled environment, which facilitate its security requirements and needs. With the recent integration of an IP connection, the STB architecture evolved towards a more common architecture using processors with standard cores and embedded Linux as operating system. Functionally, a modern STB is equivalent to a standard computer.

Common hacker knowledge now applies to STB. Moreover, STBs rarely implement security mechanisms, except the mandatory media access control and digital rights management ones: there are no antivirus, no firewall, no intrusion detection system and middleware runs with root privilege.

These security flaws make the STB vulnerable to many security attacks. The security of the STBs has become more complex and it has increased the need of a rigorous security engineering process.

A. *Functional description*

We focus in this paper in the following five main services provided by a STB:

- Decrypting and decoding broadcasted content coming from satellite. The STB is plugged to a satellite dish that allows receiving protected content. A control word system that changes several times per minute is used to scramble the content. The control words arrive ciphered with the content and are deciphered by a smartcard contained in the STB. All the security of this process is under the responsibility of the Control Access System (CAS).
- Browsing Internet.
- Playing DRM content coming from Over The Top (OTT) services.
- Buying and executing of applications from an Applications Store (appstore).
- Storing personal content (video or photo).

B. *Security problem*

The security of the STB is very important for several reasons. One example is the PayTV distribution. A broadcaster wants to supply early a new movie that it is still being played in the movie theatres. Thus, the content has a very high value and the protections of the subscription and distribution functionalities are very important. If a malicious user access the movie clear data stored in the STB she can stream it over the internet or save it for illegal distribution. Another example is the maintenance of the STB. If a PC company sells a computer its security is not the company's problem because it is not anymore the owner of the machine. If the customer installs a malicious application on her computer this is her own responsibility. On the contrary, the broadcaster buys the STBs to the manufacturer and rents them to the users. If a user installs a malicious application on a STB the responsibility scheme is very different from the PC one. First, the provider has to fix the problem and, second, she may have to do this quickly because the user could not able to buy more content or her critical information could be sent to malicious users.

Security is very important in the STB systems but, as we have previously said, it becomes very complex due to all the different characteristics, functionalities and assets. All these applications have to share the same RAM and the same processor, which increases the security threats of the system and the risk of unauthorized accesses. Moreover, the STB is an embedded device so we have to provide all these security solutions with very limited resources and a very small impact on the performance of the system.

The addition of an appstore is a very challenging task. From a security point of view, using an appstore in the STB allows unknown users to install unknown software developed by unknown developers. Indeed, the most frightened applications are the ones that are developed by targeting the system and the applications developed by bad developers that create security flaws allowing exploits on the system.

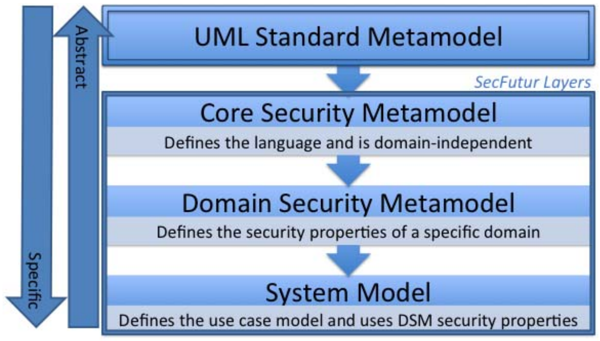
Furthermore, actually there exist no good techniques for detecting malicious applications. Most techniques rely on signature systems. Unfortunately, we know an application is malicious when it is too late. Signature systems only reduce the access range of malicious applications. As Oberheide and Miller [4] show in their Bouncer study, the current means implemented are insufficient.

III. Security Engineering Process and Security Artefacts

The main objective of the SecFutur Security Engineering Process is to help developers and engineers in the treatment of security aspects and in the use of security elements in order to enhance system models and fulfil their security requirements. The Security Engineering Process integrates security solutions into a framework and can be incorporated into an existing design process with a minimum amount of changes. The framework covers all the different phases of the life cycle such as the creation of the security artefacts, security building blocks and the creation of system models using security properties.

A. *Security Engineering Process Architecture*

In order to describe and use the security properties of the domains we have created three different layers that describe the language used to define the security properties, the security properties themselves and their application in the models of the use cases. Due to size restrictions we cannot explain in depth the different layers, creation processes and elements of the security engineering process. We just focus here in the creation of a DSM and the description of one used to model the security properties of the Set-top Box domain. The reader can find more information about the security engineering process in [4]. Each layer is built on the previous one and defines a more specific model of security. Figure 1 shows the dependencies between the different layers. They are organized top to down from the more abstract one (the Core Security Metamodel) to the more specific one (the System Model of the target scenario). It is important to clarify that we use the term domain to refer to specific application domains (e.g. wireless sensor networks embedded systems, smart metering, ad-hoc networks, etc.) while in the field of modelling it is normally associated with the notion of Domain Specific Language (DSL) and focused on code generation.



UML Standard Metamodel
SecFutur Layers
Core Security Metamodel
Defines the language and is domain-independent
Domain Security Metamodel
Defines the security properties of a specific domain
System Model
Defines the use case model and uses DSM security properties
Abstract
Specific

Figure 1. SecFutur Layers

The first layer is the Core Security Metamodel (CSM). It is based on the UML Standard Metamodel and is materialized as a metamodel that contains elements (in the form of UML metaclasses) and relations to represent relevant security concepts such as properties, requirements, threats, attacks, assumptions, domains, actors, tests, etc. It is important to note that this layer deals only with concepts (e.g. “attack”) and not with instances of these concepts (e.g. “illegal memory access). Thus, this layer is domain-independent and defines the common language (rules, relations, etc.) used to express security-related information. The central element of the CSM is the concept of security property. In fact, one of the main objectives of the CSM is to serve as a basis for the definition and description of domain-specific security properties.

Figure 2 shows an extract of the CSM. The attributes of the elements have been hidden in order to improve its visibility. Due to the complexity of the metamodel, its attributes and relations, we have divided the CSM into six different expertise sub-metamodels. Each one focuses in a specific field, allowing users with experience and knowledge in that field to fulfil those classes and use them in the creation of DSMs. The different expertise aspects of the CSM and their descriptions are:

- Properties Model: focuses on the definition of the security property and its characteristics.
- Requirements Model: describes the relations between a security property, the security requirement it fulfils and the solution that implements it.
- Threat Model: defines and describes the security threats of the security properties, the different attacks that implement the threat and the attackers that execute them.
- Domain Model: describes the domain of the DSM, the elements of the real world that can be found in the domain and the list of known actors or roles of the domain.
- Assurance Model: defines the assurance and certification of the security property. It is used to check if the property and its structure are valid and secure.
- Validation & Verification Model: describes the validation and verification mechanisms of a security property.

The specification of the security knowledge for a specific domain is done in the second layer of the framework by creating a Domain Security Metamodel (DSM). DSMs are created using as basis the CSM, allowing experts to capture security knowledge (properties, solutions, threats, etc.) in compliance with the environment of their applications (company policies, standards, etc.) for a specific domain. The DSMs are instances of the CSM in a specific domain. System engineers use the DSMs in order to enhance with security their system models. For that reason, DSMs are created by security experts who, after analysing a domain and thanks to their expertise and knowledge of that domain, can model the different domain-specific security properties and its attributes. The solutions (implementations) of the security properties are captured in the form of what we call Security Building Blocks (SBBs). They provide a hardware and/or software solution that is developed by a SBB expert. The DSM expert selects one for each security solution according to its characteristics, functionality, domain, etc. This process is out of scope of the paper so we just describe it briefly.

Figure 3 presents an extract of a DSM. The main elements are the domain, which describes the domain of the DSM, and the security property. Although a DSM is composed of many security properties we only show one in this example for a better visual representation. As we can see in the Figure, this DSM example defines the domain (STB Domain), some security properties (Process Isolation, Secure Boot, Kernel Modules Integrity, etc.), some security solution patterns that implement the solution of the security properties and some threats (Code Injection, Denial of Services, etc.) for the security properties. We explain better a security property in the example of the secure storage property of the Set-top Box DSM.

The System Model (SM) describes the target scenario where we apply the security properties of the DSM in order to security-enhance it by fulfilling its security requirements. Its details come from the analysis of the scenario and the expertise of the system engineer. Using DSMs implies that the system engineer does not need a high expertise or knowledge of security, as each security property has all the required information for its implementation, information of threats and attacks, tests, V&V methods, etc.

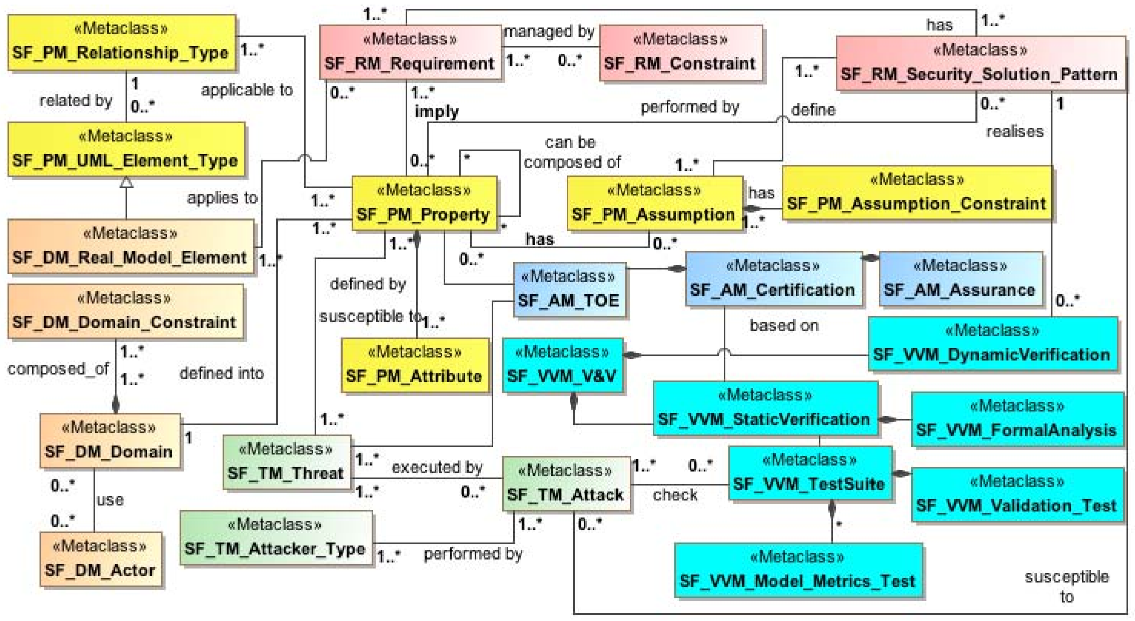

«Metaclass» SF_PM_Relationship_Type
«Metaclass» SF_RM_Requirement
«Metaclass» SF_RM_Constraint
«Metaclass» SF_RM_Security_Solution_Pattern
«Metaclass» SF_PM_UML_Element_Type
«Metaclass» SF_PM_Property
«Metaclass» SF_PM_Assumption
«Metaclass» SF_PM_Assumption_Constraint
«Metaclass» SF_DM_Real_Model_Element
«Metaclass» SF_AM_TOE
«Metaclass» SF_AM_Certification
«Metaclass» SF_AM_Assurance
«Metaclass» SF_DM_Domain_Constraint
«Metaclass» SF_VVM_DynamicVerification
«Metaclass» SF_PM_Attribute
«Metaclass» SF_VVM_V&V
«Metaclass» SF_VVM_StaticVerification
«Metaclass» SF_VVM_FormalAnalysis
«Metaclass» SF_DM_Domain
«Metaclass» SF_TM_Threat
«Metaclass» SF_TM_Attack
«Metaclass» SF_VVM_TestSuite
«Metaclass» SF_VVM_Validation_Test
«Metaclass» SF_DM_Actor
«Metaclass» SF_TM_Attacker_Type
«Metaclass» SF_VVM_Model_Metrics_Test
managed by
has
related by
applicable to
imply
performed by
define
realises
can be composed of
applies to
has
defined by
susceptible to
based on
composed_of
defined into
executed by
check
use
performed by
susceptible to

Figure 2. Extract of the CSM

As this process is performed in the modelling phase of the system, security is naturally integrated. This implies a more secure system and makes it easier to evolve.

B. *Security Building Blocks*

In order to provide the solution for a security property (such as confidentiality), the SBBs use different software and/or hardware components that interact with each other. The definition and relation of these relevant components we use a Security Building Block Metamodel (SBBMM). It provides a general approach of how to model and relate the different SBBs and external/internal elements in the system. The SBBMM defines the grammar and language for the definition of the Security Building Block Models.

The SBBM consists of instances of the artefacts defined in the SBBMM and their interactions. The SBBMs can be composed of several or just one SBB. The solution of a security property can be defined by using just one SBB or several ones. This means that a SBBM must include at least one SBB, which provides a concrete solution for a specific security property of the use case. The SBB is referenced in a Security Pattern as one possible solution for a specific security property.

A SBBM may consist of different SBBs that provide the same solution but using other components. This way, the solution could be language-specific (solution in different programming languages), dependent of the energy consumption, dependent of the number of devices, etc.

Although the description of the SBBMM is out of the scope of this paper, we present in Figure 5 a SBBM of the secure storage property. The description of its elements and relations are:

- Security Building Block (SBB): The main artefact of the model. It encapsulates a secure functionality that implements a security property. The SBB can be single or composed of various SBBs.
- SBB Interface: Defines the methods that can be used in the system implementation to interact with the SBB.
- SBB Datacontainer: It defines and encapsulates any input/output data related with the functionality of the SBB.
- SBB Precondition: Defines a precondition that must be satisfied in order to apply successfully the SBB. If the precondition is not meet the SBB cannot assure that provides enough level of security or any security at all. For example, the SBB may require a 512 bits key for encryption/decryption. If the system does not provide this element the system will not be secure.
- SBB Postcondition: It provides information of the security properties or characteristics that the SBB provides when successfully applied to the System Model. Following with the previous example, if the SBB is applied correctly the system will have the security communications property.

C. *Description of the Set-top Box DSM*

This Section describes the Set-top Box DSM. First, we describe the DSM creation process briefly. Following, we present the security analysis of the domain along with the most important assets, possible security requirements and functionalities of the system. Last, we present an in depth description of the Set-top Box DSM, focusing later in one specific security property.

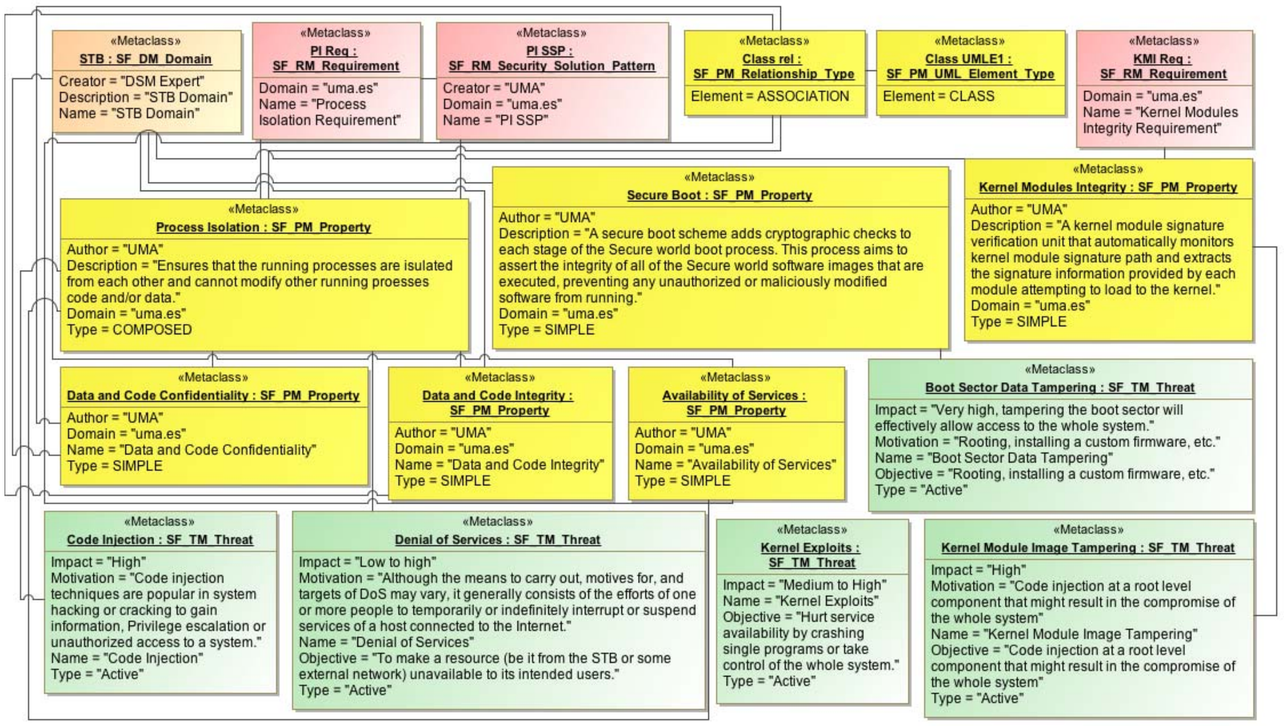

«Metaclass»
STB : SF_DM_Domain
Creator = "DSM Expert"
Description = "STB Domain"
Name = "STB Domain"
«Metaclass»
PI Req : SF_RM_Requirement
Domain = "uma.es"
Name = "Process Isolation Requirement"
«Metaclass»
PI SSP : SF_RM_Security_Solution_Pattern
Creator = "UMA"
Domain = "uma.es"
Name = "PI SSP"
«Metaclass»
Class rel : SF_PM_Relationship_Type
Element = ASSOCIATION
«Metaclass»
Class UMLE1 : SF_PM_UML_Element_Type
Element = CLASS
«Metaclass»
KMI Req : SF_RM_Requirement
Domain = "uma.es"
Name = "Kernel Modules Integrity Requirement"
«Metaclass»
Process Isolation : SF_PM_Property
Author = "UMA"
Description = "Ensures that the running processes are isulated from each other and cannot modify other running proesses code and/or data."
Domain = "uma.es"
Type = COMPOSED
«Metaclass»
Secure Boot : SF_PM_Property
Author = "UMA"
Description = "A secure boot scheme adds cryptographic checks to each stage of the Secure world boot process. This process aims to assert the integrity of all of the Secure world software images that are executed, preventing any unauthorized or maliciously modified software from running."
Domain = "uma.es"
Type = SIMPLE
«Metaclass»
Kernel Modules Integrity : SF_PM_Property
Author = "UMA"
Description = "A kernel module signature verification unit that automatically monitors kernel module signature path and extracts the signature information provided by each module attempting to load to the kernel."
Domain = "uma.es"
Type = SIMPLE
«Metaclass»
Data and Code Confidentiality : SF_PM_Property
Author = "UMA"
Domain = "uma.es"
Name = "Data and Code Confidentiality"
Type = SIMPLE
«Metaclass»
Data and Code Integrity : SF_PM_Property
Author = "UMA"
Domain = "uma.es"
Name = "Data and Code Integrity"
Type = SIMPLE
«Metaclass»
Availability of Services : SF_PM_Property
Author = "UMA"
Domain = "uma.es"
Name = "Availability of Services"
Type = SIMPLE
«Metaclass»
Boot Sector Data Tampering : SF_TM_Threat
Impact = "Very high, tampering the boot sector will effectively allow access to the whole system."
Motivation = "Rooting, installing a custom firmware, etc."
Name = "Boot Sector Data Tampering"
Objective = "Rooting, installing a custom firmware, etc."
Type = "Active"
«Metaclass»
Code Injection : SF_TM_Threat
Impact = "High"
Motivation = "Code injection techniques are popular in system hacking or cracking to gain information, Privilege escalation or unauthorized access to a system."
Name = "Code Injection"
Type = "Active"
«Metaclass»
Denial of Services : SF_TM_Threat
Impact = "Low to high"
Motivation = "Although the means to carry out, motives for, and targets of DoS may vary, it generally consists of the efforts of one or more people to temporarily or indefinitely interrupt or suspend services of a host connected to the Internet."
Name = "Denial of Services"
Objective = "To make a resource (be it from the STB or some external network) unavailable to its intended users."
Type = "Active"
«Metaclass»
Kernel Exploits : SF_TM_Threat
Impact = "Medium to High"
Name = "Kernel Exploits"
Objective = "Hurt service availability by crashing single programs or take control of the whole system."
Type = "Active"
«Metaclass»
Kernel Module Image Tampering : SF_TM_Threat
Impact = "High"
Motivation = "Code injection at a root level component that might result in the compromise of the whole system"
Name = "Kernel Module Image Tampering"
Objective = "Code injection at a root level component that might result in the compromise of the whole system"
Type = "Active"

Figure 3. Extract of a DSM

1) *DSM Creation Process*

The DSMs, as explained in Section 2, are completely independent from the System Models that can be defined in a domain. This means that the DSM defines the security properties using as basis the information of the domain and not the information of a single or several use cases of a domain. A security domain expert with knowledge and expertise in security domains models the domain-specific security properties in the DSM. The security properties are described (modelled) along with their different characteristics such as its attributes, threats, attacks, assumptions, etc. using the language defined in the CSM. We have developed a MagicDraw [5] tool that can help security domain experts in the creation, definition and use of the DSMs. It allows also security domain experts to modify or update a DSM created by another expert.

The first task of the security domain expert is the analysis of the domain. This analysis provides several results:

- A list of the possible security requirements the systems working in this domain could have.
- A system threat list.
- A list of domain elements that must be protected.
- A list of real model elements that are typically used in the domain such as methods (name, parameters), elements (attributes, type), etc.
- The list of roles that interact with any element or functionality of the domain.

These results allow the security domain expert to define the security properties of the domain and all their characteristics (assumption, threats, attributes, constraints, real model elements, etc.). This DSM can be enhanced with more security properties, new or updated elements, etc. by using the feedback of the users that work with the DSM. In this way, DSMs will evolve better and faster the more users use it.

Although many security properties can be defined as they exist (confidentiality, authentication, availability, etc.) we have defined a term called "composed security property" that allows us to create complex or domain-specific security properties. The term composed defines a relation between two or more security properties that are related in the way "A implies B" or "A + B implies C". This way, we can define a complex or domain-specific security property (secure transmission) and describe it as been "composed" of the confidentiality and integrity properties (they have specific attributes and characteristics in order to provide the required functionality for the secure transmission property). Following the previous methodology, we define this relation as "confidentiality + integrity implies secure transmission". The main security property of these relations is the one that defines the solution. This security property defines several other elements (threats, assumptions, etc.) and the ones that compose it describe only its functionality in the composed property. This methodology has demonstrated to be very effective and potent and has allowed us to define many domain-specific security properties that we would not be able to define in other way. We use this methodology in the description of the security properties of the Set Top Box DSM.

2) *Domain Analysis*

The domain analysis presented in this subsection does not present all the information we obtained of the security threats, possible requirements and security properties. Due to size limits we present only some security properties, assets and functionalities that are valuable for the definition of the DSM.

The analysis of the domain will be used to create and describe the security properties of the domain. These security properties are modelled, as we said before, in one DSM without dependencies of any scenario. The scenario provides the security requirements while the domain provides the security properties and its threats, attributes, assumptions, etc.

The description of the domain focuses in five main functionalities with specific security needs. Following, we briefly present some of the requirements that emerge from the domain analysis.

The content broadcasted and received by the STB must be protected. The CAS is responsible for ensuring the protection of the incoming communication. The transmitted data and the control words need to have a secure space to be stored and all the running processes need to have an isolated memory range in the STB.

The web browser must be always available and every transmission must guarantee a secure communication. Besides, other services that needs to communicate with external components such as movies functionality (buy, rent, etc.), select live content to download, application store, etc. must be protected against malicious users. Last, it must provide parental control functionality and authentication mechanisms.

Playing DRM content assumes that there exists a valid and trusted built-in unit in the STB. The signatures must be verified in the kernel modules before any DRM element can be used. Besides, the OTT content availability depends on the integrated STB privileges and features. In the server side, any incoming connection has to be authenticated. Finally, the communication and data storage must be protected.

The execution of Third Party applications needs a secure running environment due to the untrusted nature of any downloaded software. Every application signature must be verified before installation and also before any execution. Finally, the memory space of the running applications must be separated from the other processes executed in the STB.

Finally, the storage of personal content like photos or videos must be secured.

This analysis of the domain gives many possible security requirements, which, as we said before, is used together with the assets, etc. to generate the list of security properties and characteristics of each one. We present in the next subsection a general view of the Set-top Box DSM. It was created using the information and analysis presented here and following, due to the size of the DSM, we focus in one particular security property, secure storage, and describe its components.

D. DSM Description

Following we describe the different security properties defined in the Set-top Box DSM. Due to size restrictions we could not show the full diagram in the paper so we present here the different security properties it has (with their descriptions) and then we focus in a specific security property (secure storage) showing its architecture and relations with the elements that describe it.

The Data Secure Communication property is a composed property defined by the Communication Integrity property, Communication Confidentiality property and Communication Availability propertie. All of them are modelled to ensure that the transmission of data is made in a secure way. The Data Secure Communication property has many threats defined such as Data Theft (attempting to obtain private data or assets from a user). The communication may suffer different types of attacks, such as denial of services, spoofing or disturbance in the data transmission.

The Secure Data Storage property ensures that the data stored in the STB is protected against unauthorized accesses. This property is composed of the Data Storage Confidentiality property, Data Storage Integrity property and Data Storage Availability property. Among all the possible threats this property is susceptible of the data destruction threat is the most important, because it could destroy user sensitive information and this could have repercussions in the trust of the user in the system. We explain more in depth this security property in the next Subsection 3.2.3.

The Secure Boot property is partially showed in the Figure 3. It provides a way to check each stage of the boot process, preventing any unauthorized or maliciously software to modify it. One of the most important threats is the Boot Sector data tampering. It tries to fake the system in order to obtain root privileges and use them to manipulate the system. Others threats like the reverse engineering can reveal the boot code and use it to attack the STB with malware or adding backdoors to the system. Finally, it’s important to consider other threats such as the fake verification threat, which tries to verify an external module of the system as own. Possible assumptions are the pre-boot authentication, which ensures a signature for all the necessary files in the boot initialization, and the id matching verification, which assumes that the encrypted boot loader identification matches the processor id number.

The Secure Kernel and Kernel Module Integrity (partially showed in Figure 3) properties check that the kernel of the system is secured. They contain a signature verification unit that automatically monitors the kernel module signature path and extracts the information that each module attempts to load in the kernel. Each property is focused in different security aspects and, due to that, they have different threats with different objectives. The first security property refers only to the kernel code integrated in the STB and it can be threatened by undiscovered kernel exploits, which provide to the attacker the capability to damage the service availability by crashing single programs or taking control of the system and its functionalities. On the other side, the Kernel Module Integrity property can be targeted by the Image Tampering threat. These threats try to inject code in the root level in order to compromise the whole system. There are many attacks related with the tampering threat like the interception, hijacking or inconsistency, changing the normal course of events defined by the kernel or just oriented to wastage resource to make the STB unusable. These kernel threats must be fixed as soon as possible, so we assume that the kernel is upgradeable and allow us to make modifications in case of attacks.

The Process Isolation property is showed in Figure 3. It ensures that the processes are isolated from each other and cannot modify or interfere with other processes code and/or data. This property is composed of the Availability of Services property, the Data and Code Confidentiality property and the Integrity property. Among the different threats of this property we have the denial of service threat, which tries to make the resources of the system unavailable. The code injection threat, tries to gain unauthorized access in the system. Also, it exists the data injection threat, focused to serve specific faked information to the running processes. Both threats are implemented by many attacks that try to obtain privileges, add malware to the system or steal sensible information from the web browser application.

The Secure Shopping property assures the functionality of a valid trusted store. It checks that the access and payment is secure and all the information transmitted is confidential. This property is composed of several properties, they are the Store Server Authentication property, User Identification property, Authentication property, Secure Payment property, User Data Confidentiality property and Integrity property. As this security property works with external servers its threats are related to the data transmission or the authentication functionality. For example, the data theft threat tries to obtain critical information of the user (e.g. the payment information) and the website phishing threat describes malicious sites that act as login servers and can steal the user credentials. There exist many attacks that implement the previously described threats. This property assumes that the system uses a SSL certificate that protects the system against those threats.

The Authorized Access Retrieval and Parental Control properties ensure that a user is only able to access or download the content she is allowed to. These security properties are composed of the User Identification and the Authentication properties, which check the validity of the user credentials. These properties define an assumption that trust any application downloaded from the app-store. There exist various threats for these security properties such as data theft, impersonation, etc.

The Application Verification property ensures that every application installed in the STB is trusted by verifying its signature each time the STB boots, installs or execute it. This security property is composed of the application integrity property and the verification property. One assumption of this property is that it trust the keys provided by the STB. Therefore there exist a threat of data theft on every attempt to use the verification module. Another usual threat is forging signed data or misuse of the signature if the attacker obtains root privileges in the STB. Many of the attacks focus in the verification modules trying to get the signature keys or the complete algorithm, using for instance a cryptographic attack based on brute force.

These security properties where defined and modelled using the expertise of the security domain expert and the different analysis of the domain. They fulfil all the possible security requirement of the domain described in Section 2.

Once the DSM is completed it is validated and stored in a DSM repository where any system engineer will access and use it in her system model in order to fulfil her security requirements. The feedback of the system engineers is very important because they can provide information about modifications of existing security properties or the necessity of new ones.

E. Secure Data Storage property

This subsection describes the characteristics and architecture of the Secure Data Storage property. The rest of the security properties defined in the DSM follow the same architecture and specification.

Figure 4 shows an extract of the Secure Data Storage property. We do not show the Certification or the V&V elements due to size restrictions. The elements we describe are the core ones and represent the specification of a security property.

The domain analysis described in Section 3 shows that the Secure Data Storage property can be used in several scenarios. Two of them use the property directly and define the real model elements where it can be used. The first functionality is the content broadcast, where the traffic data and the control words need to be securely stored and the second one is the secure storage of personal data (e.g. photos, videos, etc.). In a common STB scenario this property is used in order to secure several information such as downloaded watermarked content, paid applications or signature keys.

The Secure Data Storage property has to ensure the protection of the data containments (e.g. databases, key stores, etc.) or control the functions that store/access the different elements (e.g. variables, attributes, etc.). The security property is composed of the Data Storage Confidentiality, Data Storage Integrity and Data Storage Availability properties. The confidentiality property checks that the stored data is accessed only by the allowed users, the integrity property checks that the data is not modified by unauthorized users and the availability property ensures that the data is always available for the users.

Figure 4 shows also some assumptions of the security property. The first one asserts that the STB was not manipulated during the manufacturing and the other one assumes that there exists an encryption hardware module in the STB. The trust in these characteristics is critical for the functionality of the security property. If some of them are not met in the system model where the security property is applied then the system will not be secure.

The Security Data Storage property defines some threats. The first one is data modification. Its objective is to modify user data in the STB. The objective of the malicious user is to manipulate critical information or spoof user's identity. It is an active threat, because malicious users have to actively try to access the system. The way to achieve this is described in the attacks elements related to this class. Finally, the impact of this threat in the system depends on the data that is manipulated. If it is trivial data then it does not impact the user but if it is critical data (such as the personal data, app-store data, etc.) then the impact is very high and can produce serious consequences.

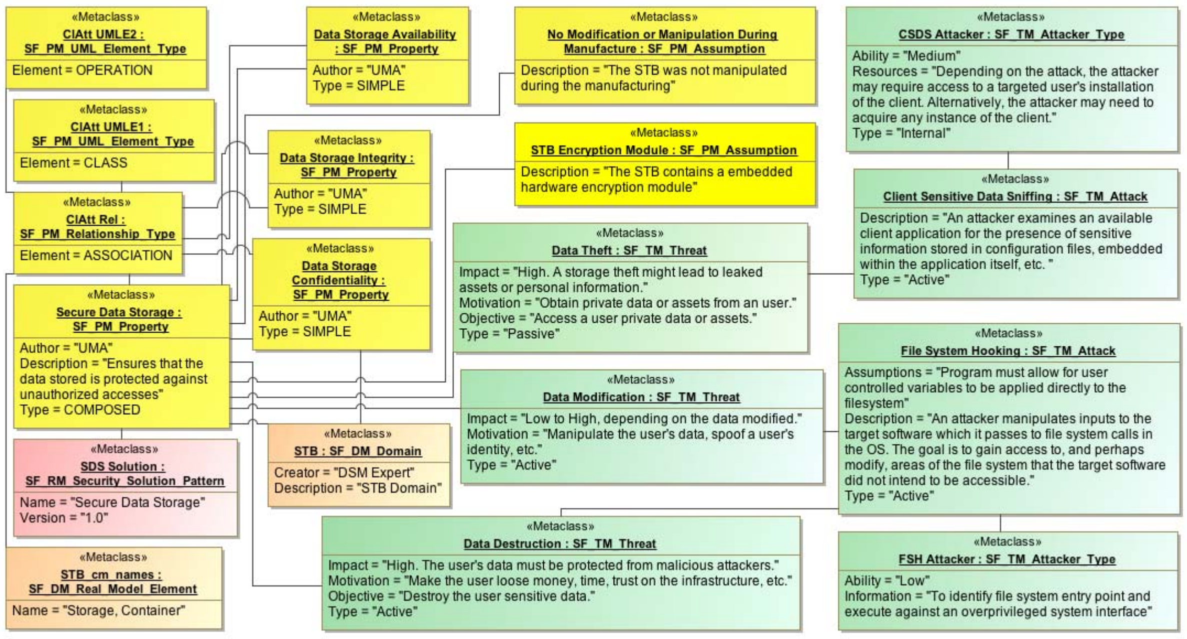

«Metaclass»
CIAtt UMLE2 : SF_PM_UML_Element_Type
Element = OPERATION
«Metaclass»
CIAtt UMLE1 : SF_PM_UML_Element_Type
Element = CLASS
«Metaclass»
CIAtt Rel : SF_PM_Relationship_Type
Element = ASSOCIATION
«Metaclass»
Secure Data Storage : SF_PM_Property
Author = "UMA"
Description = "Ensures that the data stored is protected against unauthorized accesses"
Type = COMPOSED
«Metaclass»
SDS Solution : SF_RM_Security_Solution_Pattern
Name = "Secure Data Storage"
Version = "1.0"
«Metaclass»
STB_cm_names : SF_DM_Real_Model_Element
Name = "Storage, Container"
«Metaclass»
Data Storage Availability : SF_PM_Property
Author = "UMA"
Type = SIMPLE
«Metaclass»
Data Storage Integrity : SF_PM_Property
Author = "UMA"
Type = SIMPLE
«Metaclass»
Data Storage Confidentiality : SF_PM_Property
Author = "UMA"
Type = SIMPLE
«Metaclass»
STB : SF_DM_Domain
Creator = "DSM Expert"
Description = "STB Domain"
«Metaclass»
No Modification or Manipulation During Manufacture : SF_PM_Assumption
Description = "The STB was not manipulated during the manufacturing"
«Metaclass»
STB Encryption Module : SF_PM_Assumption
Description = "The STB contains a embedded hardware encryption module"
«Metaclass»
Data Theft : SF_TM_Threat
Impact = "High. A storage theft might lead to leaked assets or personal information."
Motivation = "Obtain private data or assets from an user."
Objective = "Access a user private data or assets."
Type = "Passive"
«Metaclass»
Data Modification : SF_TM_Threat
Impact = "Low to High, depending on the data modified."
Motivation = "Manipulate the user's data, spoof a user's identity, etc."
Type = "Active"
«Metaclass»
Data Destruction : SF_TM_Threat
Impact = "High. The user's data must be protected from malicious attackers."
Motivation = "Make the user loose money, time, trust on the infrastructure, etc."
Objective = "Destroy the user sensitive data."
Type = "Active"
«Metaclass»
CSDS Attacker : SF_TM_Attacker_Type
Ability = "Medium"
Resources = "Depending on the attack, the attacker may require access to a targeted user's installation of the client. Alternatively, the attacker may need to acquire any instance of the client."
Type = "Internal"
«Metaclass»
Client Sensitive Data Sniffing : SF_TM_Attack
Description = "An attacker examines an available client application for the presence of sensitive information stored in configuration files, embedded within the application itself, etc. "
Type = "Active"
«Metaclass»
File System Hooking : SF_TM_Attack
Assumptions = "Program must allow for user controlled variables to be applied directly to the filesystem"
Description = "An attacker manipulates inputs to the target software which it passes to file system calls in the OS. The goal is to gain access to, and perhaps modify, areas of the file system that the target software did not intend to be accessible."
Type = "Active"
«Metaclass»
FSH Attacker : SF_TM_Attacker_Type
Ability = "Low"
Information = "To identify file system entry point and execute against an overprivileged system interface"

Figure 4. Secure Data Storage example

The second threat is data destruction. Its goal is to destroy the maximum amount of information from the STB. The motivations can be various. From disable the STB to make users lose confidence in the company that sold them the STB. This threat, as the previous one, is active. The malicious users have to actively try to access the STB in order to destroy the data. The impact of this threat is high, as the information deleted can be critical for the user or disable the STB completely.

The last threat is the data theft. Its goal is to obtain critical data of the user. This threat is very important as the malicious user can be obtaining important info of the user for a long amount of time before it is detected. The motivation of this attack is, as we said before, to obtain data from the user, being a one time specific theft or just access the data for a long amount of time. It is a passive threat, because once the malicious software or connection to the system is done, the hacker will have complete access to the data until the security flaw is detected. The impact of this threat is high, because the critical information theft can be used against the user or used for illegal activities.

Figure 4 shows the attacks that implements the threats described in the previous paragraphs. The File System Hooking (manipulating input to file system calls) implements Data Destruction and Data Modification and the Client Sensitive Data Sniffing (lifting sensitive data from the client) implements Data Theft. Each attack has defined an attacker type that is able to execute the attack.

The File System Hooking attack manipulates inputs to the target software, which the target software passes to file systems calls in the system. The goal is to gain access/modify areas of the file system that the target software did not intend to be accessible. An assumption of this attack is that programs allow user controlled variables to be applied directly to the file system. The attack is active, as the malicious user has to be directly attacking the system to gain access. The attacker type element describes the information of the attacker that can perform this attack. The ability required for the attacker is low, as they usually use applications that perform this type of attack. The only information the malicious user needs is to identify the file system entry point.

The Sensitive Data Sniffing attack examines an available client application for the presence of sensitive information. This information may be stored in configuration files, embedded within the application itself or stored in other ways. Sensitive information may include long-term keys, passwords, etc. This attack must be executed directly by the malicious user because she must obtain a client application for the sniffing. The attacker needs a medium ability, because she must know the data structure of the client application. That way, depending on the attack the attacker needs access to the target user's installation of the client or an instance of the client.

The security property, as we can see in the Figure, can be applied in a UML class or operation element. The model provides also information about the known domain real model elements usually used in the use cases that work in this domain. This information will be very useful when the system engineer uses the DSM containing this security property.

The security solution that implements the security property defines a solution pattern that links to SBBs. The SBBs implement the security property by using hardware and/or software elements. The Security Domain Expert selects the security solution that fits better the security property and imports it to the model. The next section presents a description of an extract of the SBB Model that implements the security property.

F. Secure Data Storage SBB Model

The selection of solutions that can implement the Secure Storage property is supported by the MagicDraw tool developed in the project. The SecFutur official repository contains all the SBBs created by developers. Each SBB includes the implementation, the functional components and some information about its capabilities. The selection procedure is out of the scope of this paper but some key characteristics are: semi-automatic search of SBBs using the info of the domain, the security property implemented, resources and energy requirement, etc.; it checks of compatibility between the selected SBB and the existing ones of the model; provides information of the implementation, etc.

Figure 5 depicts an example extract of the SBB Model of the SBB Secure Storage. The model is composed of the different artifacts described in Section III-B. The operation seal of the SBB Secure Storage encrypts any kind of given arbitrary data (plain video data, user data, etc.) and saves it on the device specified as output in the model. For each encrypted data an *Identifier* is returned, which can be used to *unseal* and thus retrieve the decrypted original data. The SBB Secure Storage applies a cryptographic algorithm too, which handles the encryption and decryption functionality of the input data. For that reason, the system (device) must provide an encryption key. The “Internal_Keymanager” component provides the operation getkey(), which is used by the encryption algorithm to retrieve the encryption key for both the encryption and decryption processes. The encrypted data is stored and retrieved from the device by the “Internal Storage”. The “Internal Keymanager” and “Internal Storage” component and operations must be provided by the device.

In order to apply the SBB successfully and thus provide the ”Confidentiality Property” the Confidentiality Precondition must be valid. It assumes that the cryptographic key received from the device is also confidential, accessible only by the “Internal Keymanager” and additionally bound to the specific device. If this precondition is met and the SBB is applied, the Confidentiality Postcondition assures that the sealed data is stored encrypted and can only be decrypted by the device This means that all data stored on the device by using the SBB Secure Storage can be considered confidential.

IV. State of the Art

Although there exist many security engineering processes that could be used to model the STB domain no one integrates naturally security since the initial phases of the development.

One basic standard for security engineering is the Systems Security Engineering Capability Maturity Model (SSE-CMM) [6]. The model highlights the relationship between security engineering and systems engineering, regarding the former as an integral part of the latter and not an end in itself.

Most of the existing approaches are not comprehensive enough in the sense that they focus either on some special stage of development, e.g. on design or implementation, which implies that security is introduced in a latter stage of development, or on a specific security aspect such as access control. As each domain can contain many different security properties (some of them simple and other complex), the required security engineering process and security artifacts for the definition of the security characteristics of the domain must be open and allow users to define the specific attributes, requirements, etc. of the domain.

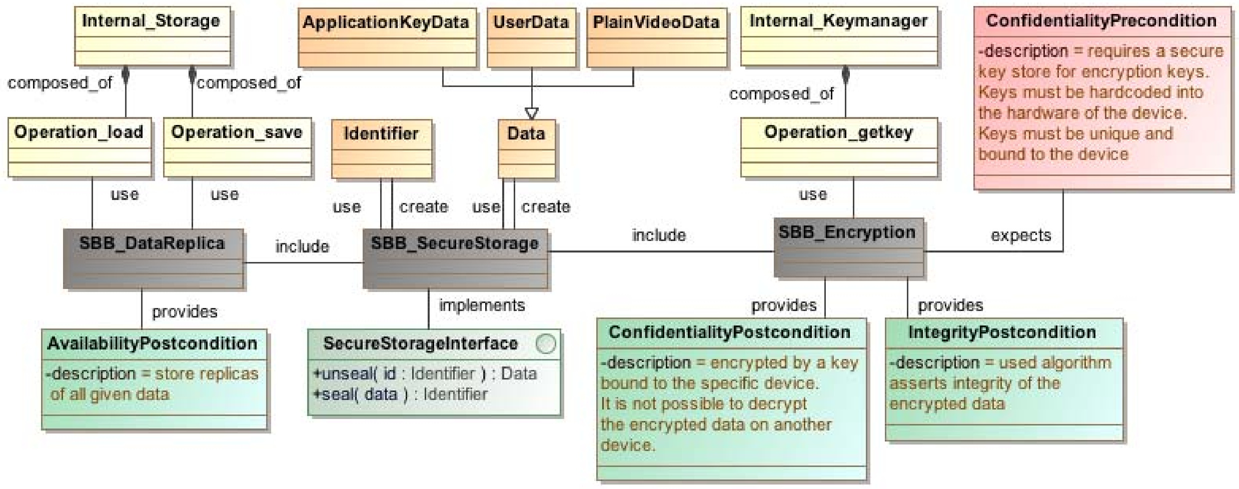
UMLSec [7], a standard extension mechanism, introduces new semantics into UML models and presents a rigorous secure software engineering approaches tailored for using in particular designs, those with a powerful tool-suite for security verification. However, UMLSec only addresses a few specific security requirements. These characteristics make it not enough powerful to describe all the security properties and solutions we have showed in Section 3.

One of the most interesting approaches for introducing security in the development cycle is Model-Driven Security (MDS) [8, 9, 10, 11]. This is a specialization of the model driven architecture approach that suggests a modular approach combining languages for modelling systems with languages for modelling security and by using transformations to enforce security properties from high-level designs to implementation artefacts [12]. Among the drawbacks in these approaches, we highlight the need for developers to define new security modeling languages, which is a difficult task requiring expertise.

SPACE [13] is a Model-Driven Engineering approach in which system functionality is specified using collaborative building blocks. These elements are essentially template-like patterns. One of the most relevant drawbacks of this approach is that no guidance is provided in choosing particular security building blocks. Nevertheless, the most important disadvantage is that security is integrated after, not during, functional modeling.

A different way to model the security properties of the domains is the use of patterns in security engineering. Security Engineering Process with Patterns (SEPP) [14, 15] is a rigorous security methodology with a strong focus on the problem domain and detailed approach for selecting appropriate security solutions. The development process follows an analysis model in order to find a solution but it enforces a set of possible solutions, which implies that some issues such as building an optimal software architecture is not considered. The methodology defined by Fernandez et al. [16, 17] relies on coherent catalogues of security patterns, which can cover a range of security concerns. However, its main drawback is that there is a lack of tool-support for pattern selection, implying that developers must choose then manually and it is hard to carry out at runtime.

As conclusion, there not exist a security engineering process that allows us to define and use specific security properties of domains, due to the number of requirements, assets, functionalities, etc. and the complexity of the different security domains, which are composed of complex models and elements. Besides, the security engineering process presented here helps system developers in the modeling of the system integrating security since the beginning of the development phase.



Internal_Storage
composed_of
composed_of
Operation_load
Operation_save
use
use
ApplicationKeyData
UserData
PlainVideoData
Identifier
Data
use
create
use
create
Internal_Keymanager
composed_of
Operation_getkey
use
ConfidentialityPrecondition
-description = requires a secure key store for encryption keys. Keys must be hardcoded into the hardware of the device. Keys must be unique and bound to the device
SBB_DataReplica
include
SBB_SecureStorage
include
SBB_Encryption
expects
provides
implements
provides
provides
AvailabilityPostcondition
-description = store replicas of all given data
SecureStorageInterface
+unseal(id : Identifier) : Data
+seal(data) : Identifier
ConfidentialityPostcondition
-description = encrypted by a key bound to the specific device. It is not possible to decrypt the encrypted data on another device.
IntegrityPostcondition
-description = used algorithm asserts integrity of the encrypted data

Figure 5. Secure Storage SBB

V. Conclusions

The creation of a security model with all the possible security properties (and its characteristics) of a domain is a very interesting and useful element that greatly facilitates the work and understanding of the system models related with that domain. Currently, the process is been applied to several use cases of different domains, showing good results in each of them.

The DSM presented here was developed using the information of the STB domain along with several meetings with security developers of STBs. They provide us many security assets, threats, security properties that would be useful in the use cases of the domain, etc. We have specified a use case in this domain and have applied the DSM to fulfil its security requirements. Besides, we are currently developing a virtual STB using the security-enhanced system model obtained thanks to this process.

Acknowledgment

This work was partially supported by the E.U. through FP7 project SecFutur (IST- 25668).

References

[1] Jon Brodkin. http://www.networkworld.com/news/2007/100407-web-site-vulnerabilities.html.October 2007.

[2] Design of secure and energy-efficient embedded systems for future internet applications (SECFUTUR), IST-25668. Seventh Framework Programme.www.secfutur.eu.

[3] Jose Fran. Ruiz, Rajesh Harjani and Antonio Maña, "A Security-focused Engineering Process for Systems of Embedded Components", SD4RCES'11, Naples.

[4] J. Oberheide and C Miller, "Dissecting the Android Bouncer," SummerCon2012, New York, June 2012.

[5] MagicDraw, UML Modelling Tool. NoMagic Inc. http://www.nomagic.com/products/magicdraw.html

[6] Systems Security Engineering Capability Maturity Model, http://www.sse-cmm.org/model/ssecmmv2final.pdf

[7] Jurjens, J., "Towards development of secure systems using UMLSec", Lecture Notes in Computer Science, 2001

[8] Dimitrakos T, Raptis D, Ritchie B, Stolen K., "Model-based Security Risk Analysis for Web Applications", In Proc. Euroweb, 2002

[9] Lodderstedt, T., Basin, D., Doser, J.: "SecureUML: A UML-based modeling language for model-driven security"; In UML 2002 - The Unified Modeling Language : 5th International Conference, Dresden, Germany, September 30 - October 4, 2002, Springer-Verlag, (2002), 426-441.

[10] Lang, U., Schreiner, R.: "A Flexible, Model-Driven Security Framework for Distributed Systems"; In Proceedings of the IASTED International Conference on Communication, Network, and Information Security (CNIS 2003), New York, USA, (2003).

[11] Basin, D., Doser, J., Lodderstedt, T.: "Model Driven Security"; ACM Trans. Softw. Eng. Methodol. 15(1), (2006), 39-91.

[12] Fernandez, E.B.: "Security patterns and a methodology to apply them"; In Spanoudakis, G., Maa, A., Kokolakis, S. (Eds.), Security and Dependability for Ambient Intelligence, Boston, MA: Springer Verlag, (2009), 37-46.

[13] Sánchez, O., Molina, F., García-Molina, J., Toval, A.: "ModelSec: A Generative Architecture for Model-Driven Security"; J. Univers. Comput. Sci. 15(15), (2009), 2957-2980.

[14] Hatebur, D., Heisel, M., Schmidt, H.: "A security engineering process based on patterns"; In Procs. of the 18th International Workshop on Database and Expert Systems Applications (DEXA'07), Regensburg, Germany: IEEE Computer Society, (2007), 734-738.

[15] Hatebur, D., Heisel, M., Schmidt, H.: "A Pattern System for Security Requirements Engineering"; In Procs of the 2nd International Conference on Availability, Reliability and Security (ARES'07), Los Alamitos, CA, USA: IEEE Computer Society, (2007), 356-365.

[16] Fernandez, E.B.: "A methodology for secure software design"; In Proceedings of the 2004 International Conference on Software Engineering Research and Practice (SERP'04), (2004), 21-24.

[17] Fernandez, E.B., Sorgente, T., Larrondo-Petrie, M.M.: "A UML-based Methodology for Secure Systems: The Design Stage"; In Proceedings of the Third International Workshop on Security in Information Systems (WOSIS'05), Miami, FL, (2005).