applied sciences

MDPI

Article

CyberSPL: A Framework for the Verification of Cybersecurity Policy Compliance of System Configurations Using Software Product Lines

Ángel Jesús Varela-Vaca *, Rafael M. Gasca, Rafael Ceballos, María Teresa Gómez-López and Pedro Bernáldez Torres

Departamento de Lenguajes y Sistemas Informáticos, Universidad de Sevilla, 41004 Sevilla, Spain; gasca@us.es (R.M.G.); ceball@us.es (R.C.); maytegomez@us.es (M.T.G.-L.); pedbertor@alum.us.es (P.B.T.)
* Correspondence: ajvarela@.us.es; Tel.:+34-954556238

Received: 25 September 2019; Accepted: 4 December 2019; Published: 8 December 2019

Abstract: Cybersecurity attacks affect the compliance of cybersecurity policies of the organisations. Such disadvantages may be due to the absence of security configurations or the use of default configuration values of software products and systems. The complexity in the configuration of products and systems is a known challenge in the software industry since it includes a wide range of parameters to be taken into account. In other contexts, the configuration problems are solved using Software Product Lines. This is the reason why in this article the framework Cybersecurity Software Product Line (CyberSPL) is proposed. CyberSPL is based on a methodology to design product lines to verify cybersecurity policies according to the possible configurations. The patterns to configure the systems related to the cybersecurity aspects are grouped by defining various feature models. The automated analysis of these models allows us to diagnose possible problems in the security configurations, reducing or avoiding them. As support for this proposal, a multi-user and multi-platform solution has been implemented, enabling setting a catalogue of public or private feature models. Moreover, analysis and reasoning mechanisms have been integrated to obtain all the configurations of a model, to detect if a configuration is valid or not, including the root cause of problems for a given configuration. For validating the proposal, a real scenario is proposed where a catalogue of four different feature models is presented. In this scenario, the models have been analysed, different configurations have been validated, and several configurations with problems have been diagnosed.

Keywords: configuration; variability; software product line; security policies; compliance; feature models

1. Introduction

There has been a significant increase of cybersecurity attacks during the latest years. Frequently, these attacks are caused by an absence of proper security configurations [1], or by inappropriate default configuration values [2,3] of software products and systems that do not follow the desirable cybersecurity policies. The default configuration or the improper configuration of systems can unleash critical damages for the organisations as the leak of information [4]. For instance, the use of default accounts or usernames and passwords is a very common in industry [1,5,6]. In general, the use of default parameters of systems is the first attack vector used for attackers [7]. OWASP project [8] establishes the A6:2017-Security Misconfiguration and A9-Using Components with Known Vulnerabilities, as two of the top-10 vulnerabilities for Web systems.

Likewise, the complexity in the configuration of systems and products can lead to improper configurations or misconfigurations [2]. For instance, in access control systems, the improper

configuration of rules can derive a granted use of unauthorised resources. The complexity in the configuration of software products and systems is a known challenge in the software industry. An example of a configuration tool is KConfig [9], where developers are able to select among more than 12,000, Linux Kernel configuration options. An incorrect or inappropriate configuration can trigger security problems, such as attacks due to the use of a non-secure channel configuration in the communications when the cybersecurity policy establishes a strong security channel. In this paper, Software Product Lines (SPL) [10], more specifically, the use of variability analysis and feature analysis techniques (Feature-oriented Domain Analysis, FODA) [11] are the baseline to propose a solution that facilitates the detection of non-compliance of the cybersecurity policies by the system configuration. The Carnegie Mellon Software Engineering Institute [12] defines SPL as "a set of software-intensive systems that share a common, managed set of features satisfying the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way". Feature models are a specific representation of the products which composes a SPL. Previous works related to cybersecurity have faced up the problem also using SPL, such as extracting and selecting the features from the log file [13], applying machine learning to detect cybersecurity threats, to verify the security mobile application configurations by applying partial model checking [14], or the feature-oriented approach to the modular construction of fault trees, which enables reusing the structures of fault propagation roads [15]. However, the automatic verification of cybersecurity policies according to the possible configuration is still an open problem.

The development of SPL has been frequently analysed by using feature model techniques. In this article, the use of feature model techniques is proposed to fulfil cybersecurity policies by checking the appropriate configurations of the software systems and products. Feature models could be considered as a model-based cybersecurity design, where software products and systems are designed according to the requirements of cybersecurity design policies.

Feature models [11] are the main concept of the functional decomposition of the product line approximation. These models will be used for representing the configuration parameters according to established policies. Moreover, these models allow setting constraints and attributes between the characteristics of the configurations, as it was proposed in a previous work [16], allowing a greater expressiveness of dependencies and relationships between the different characteristics related to cybersecurity.

Once the proper configurations are modelled as a feature model, automatic formal verification techniques [17] can be applied. These formal verification techniques allow us to know whether or not they comply with design policies (*Detection of configuration failures*), and then also identify possible mistakes in system and product configurations (*Diagnosis of configuration failures*). The nature of cybersecurity models is highly context-dependent [18]. Therefore, feature models should be adapted according to the objectives and the contexts of cybersecurity policies [19].

This article aims to offer a framework that covers the following objectives:

- **OBJ1.** Provide users with a set of tools for the definition and the maintenance of a catalogue of cybersecurity policies based on feature models. These policies are associated with different contexts of software products and systems related to cybersecurity.
- **OBJ2.** Provide users with a set of tools to obtain properties of feature models. For example, a property could be the verification of the correctness of the model, that is, to know the conformance between the products and the model. Another example could be the extraction of all configurations of products or systems supported by the cybersecurity policy.
- **OBJ3.** Automate the detection of configuration failures to validate with cybersecurity policies through the description of features of the different cybersecurity contexts.
- **OBJ4.** Provide the diagnosis of the established configurations to isolate and identify the configuration mistakes that cause the non-conformance of the cybersecurity policies.
- **OBJ5.** Validate the proposal through complex use cases. Such as the configuration of cybersecurity mechanisms of a web application server.

In order to verify the proposal, the Design Science Research methodology [20] is used to evaluate it. The methodology has been previously used in research papers that propose a software artifact to solve a business problem [21]. In this case, the artifact (CyberSPL) provides mechanisms for verifying cybersecurity policies according to possible configurations. Design Science Research is formed of six phases used as a guideline of the structure of this paper: problem identification and motivation that contextualise the perceived difficulty to manage configurations in cybersecurity, as is presented in the Introduction; objectives of the proposed solution, analysed comparing previous work and lack of functionalities found; design and development, as a prototype of the artifact presented in [22]; demonstration by means an evolution of the previous prototype with and improvement of the previous implementation; evaluation, applied to a real example, and; communication, as mechanism to be known by the community and possible users, objective of this paper.

The article is organised as follows: Section 2 presents an overview of related works. Section 3 introduces features model concepts and analysis. Section 4 tackles the proposal, including the architecture, the workflow, and the description of operations in a real scenario. Section 5 shows and discuss the results obtained from the experimentation applied to a proposed use case. Finally, conclusions are drawn and future work is proposed.

2. Related Work

Related works have been divided into the four areas of research addressed in the article: how features models can be used to model configuration scenarios and the existing techniques for their analysis and diagnosis; how software product lines have been used in the security field; the solutions that exist related to cybersecurity configurations; and previous software product line tools that support both modelling and repositories.

2.1. Analysis and Diagnosis of Feature Models

Software Product Line (SPL) and Feature-Oriented Domain Analysis (FODA) have become mature fields in the Software arena in the last decades [17]. Several are the scenarios where SPL have been applied [23,24]. Concretely, the use of feature models as a subarea of SPLs provides mechanisms for both modelling and analysis. The automated analysis of feature models has been addressed over the last decade in the area of Software Product Lines [17,24,25]. The goal of the automated analysis is to extract or to infer certain properties of the models. In some works, the analysis is focused on detecting, analysing, or diagnosing errors in feature models, in a design phase [26] or a reconfiguration phase [27].

There are other approaches where the analysis of feature models are applied to other areas. In [28], an extension of a goal-based method (KAOS) is proposed to generate adaptable requirement models from variability models. In [29], feature models are used to analyse the variability requirements and to transform these feature models for developing an architectural model. In [30], feature model analysis is used to provide self-adapting systems by dynamical determination of the best variants for the specific QoS requirements.

2.2. Cybersecurity and Software Product Lines

Security is an understudied field in SPL. Most of the approaches are focused on the application of security requirements or properties into the SPL. Different approaches have been presented to manage the variability and specify security requirements from the early stages of the product line development [31–33]. Similarly, other approaches addressed the idea of including the security variability into an SPL [34]. In [35], the authors established a software architecture as a reference to develop SPL, dealing with information security aspects.

On the other hand, there are approaches focused on the security as a use case, such as in [36] and the methodology SecPL [37] is proposed to specify the security requirements and product-line variability. These are annotated in the design model of any system. This last proposal

is evaluated concerning efficiency, scalability and usefulness and it allows the security analysis of a realistic product line. However, none of the papers in the literature deal with the security analysis using feature models as it is done in this paper.

2.3. Analysis and Verification of Cybersecurity Configurations

In general, the problems of the verification of configurations have been studied using different approaches, such as Machine Learning [38,39]. The challenge in the context of cybersecurity is whether the known techniques about feature models can be applied to cybesecurity or they must be adapted. In [40], an approach is proposed for improving the development of secure software product lines (SPLs) and their derivated products. In the context of verification of security configurations, the *IoTChecker* [41] platform for security analysis in IoT products has been proposed. Since the configuration data of Internet-of-Things (IoT) systems are unstructured, the traditional techniques cannot be automatically applied to a specific IoT configuration such as security. Finally, for policy compliance Bring Your Own Device (BYOD), in [14] a technique for automatic security verification in mobile applications is proposed. In the context of security policy verification, it is necessary to extend the feature models to annotate the requirements specified by the policies.

2.4. Software Product Line Tools

To support both the design of the feature models and the repository to store and query them, it is necessary to provide a tool that supplies these functionalities. From the tool perspective, there exist a vast number of tools such as FAMILIAR [42] , Gears [43], FeatureIDE [44], pure::variants [45], etc. Nevertheless, most of these solutions are stand-alone and their application in online environments, with multiple users, and repository of feature models will require a redefinition of those tools. There are other approaches focused on web-based tools such as SPLOT [46], VariaMos [47] and Glencoe [48]. SPLOT attempts to open the use of software product line by offering a repository of models and it also includes a set of tools for the automated analysis of feature models. SPLOT is a well-know tool in the software product line community since it is a repository which gathers a large number of models. However, the feature model specification is very limited, for instance, it does not support the graphical edition of models. On the other hand, the repository and the analysis are also limited concerning the type of operations available. Recently, other new web-based approaches have emerged such as VariaMos and Glencoe. Both approaches provide an easy-to-use visual editor of feature models and it integrates some analysis features. Although, the visual editing capabilities are enough to define advanced feature models, but in some cases, textual editing could be desirable in huge feature models with several cross-relations. Regarding the analysis, the feature model analysis capabilities provided for both solutions are very limited. For instance, VariaMos analysis focuses on a mere syntactic and some semantic issues. On the other hand, Glencoe emphasises on the visualisation and representation of feature models, but provides analysis on determining all possible configurations, depth feature, etc. However, none of the web-based tools integrates a public neither a private repository of models, and analysis tools that enable the diagnosis of configurations, the exportation of all possible configurations or enable the feature model versioning.

To the best of our knowledge, there are no references on the use of feature models applied in the field of cybersecurity policy compliance, and therefore, the approach presented here is a novel.

3. Background in Feature Models and Validation Mechanisms

The use of feature models is a broadly used technique for analysing SPLs. Feature Models (FMs) involve a model that defines the features and their relationships. There exist various notations to design FMs [11], although the most widely used is that proposed by Czarnecki [17], exemplified in Figure 1. FM diagrams enable six types of relations between a parent feature and its child features:

- *Mandatory*, child feature is required. (cf. in the figure, PS2 is mandatory sub-feature of A, PS2 $\leftrightarrow$ A).

- *Optional*, child feature is optional (cf. in the figure, PS2 optional sub-feature of B, PS2 $\rightarrow$ B).
- *Alternative*, one of the sub-features must be selected (i.e., in general $a_1, a_2, \cdots, a_n$ alternative sub-feature of b, $a_1 \vee a_2 \wedge \cdots \wedge a_n \leftrightarrow \bigvee_{i<j}(a_i \vee \cdots \vee a_j)$).
- *Or-relation*, at least one of the sub-features must be selected (i.e., in general $a_1, a_2, \cdots, a_n$ or sub-feature of b, $a_1 \wedge a_2 \wedge \cdots \wedge a_n \leftrightarrow b$, in the figure $C \leftrightarrow C1 \wedge C2$).
- *Require relation*, a feature requires the existence of other features with non-direct family relation (cf., in the figure A2 $\rightarrow$ C1).
- *Exclude relation*, a feature excludes the existence of other features with non-direct family relation (cf. in the figure, $\neg$(D $\wedge$ E))

Definition 1. ***Feature Model****. Let FM be a feature model which consists of a tuple (F,R), where F is a set of n features* $\{f_1, f_2, \cdots, f_n\}$*, and R is a set of relations* $\{r_1, r_2, \cdots, r_m\}$*.*

There exist extensions to the classical FM representations for the inclusion of attributes and extra-functionalities for features, such as *Cost* attribute attached to feature B in Figure 1.

The model in Figure 1 can represent all possible configurations of a software product or system that meet a certain policies. In this example, features $PS2$ and A are mandatory for the product or system, since $PS2$ is the root, and there is a mandatory relation with A. In spite of that, B and C are optional, therefore, they may or may not appear in the configurations. Optionality can also be expressed, for example, $A1$, or $A2$, or both, may appear whether A is part of the configurations. Other dependencies express cross-constraints between features, for example, if the feature $A2$ appears it implies that the feature $C1$ must also appear. The model also can contain extra properties that include additional information for the model. For example, in Figure 1, the model has three attributes $Cost$, $Benefit$, and $Risk$ associated with feature B. These attributes have an integer domain. If the feature B appears in a configuration, these attributes must be associated with a tuple of value in the range defined in the model.

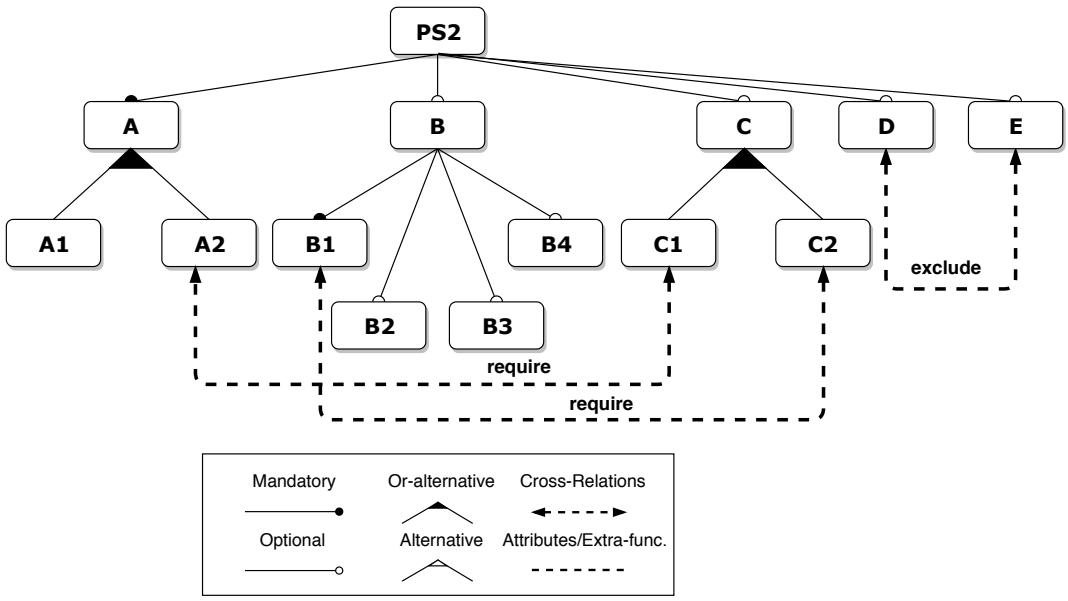

Figure 1. An example of feature model.

For the automated analysis of these models, formal methods are proposed [17], based on propositional logic, description logic or constraint programming. In the literature, tools as FAMA [16,49] enables making direct transformations from the feature models to Constraint Satisfaction Problem (CSP) or Constraint Optimisation Problem (COP) [50]. When the domains of the variables are only Boolean type (i.e., *true* and $false$ values), SAT (propositional satisfiability problem) solvers can be

applied to improve the efficiency of the resolution algorithm [51]. In this work, feature models can be formalised by constraint satisfaction problems.

Definition 2. ***Formalised Feature Model****. Let FMF be a formalised feature model consisting of the tuple (F,R), where F is a set of Boolean variables* $\{f_1, f_2, \cdots, f_n\}$ *that represent the features and R as a set of relations* r_i *that are a tuple (S,BR), where the scope S=(*f_i*,* f_j*) are the features involved in the relation and* $BR \subseteq d_{f_i} \times d_{f_j}$ *are the tuples satisfying the relation and* d_{f_k} *is the domain of the values of the feature* f_k*.*

Given a formalised feature model, FMF, composed of a set of features $F = \{f_1, f_2, \cdots, f_n\}$, and a set of relations, $R = \{r_1, r_2, \cdots, r_k\}$, an assignment a_i is the association of Boolean values to all the variables (features).

$$a_i = \prod_{f_k \in F} d_{f_k}$$

$$a_i = \{f_1 \rightarrow v_1, f_2 \rightarrow v_2, \cdots, f_n \rightarrow v_n\}$$

For instance, a_i=$\{PS2\ true, A\ true, A1\ true, A2\ false, ...\ \}$ represents an assignment for the model in Figure 1, where missed features are assigned to *false* value. Therefore, let A be the set of all the possible assignments of a model FMF.

$$A_{FMF} = \bigcup_{i \in FMF} a_i$$

An assignment is considered a valid assignment when satisfies all relations (R) of the model FMF.

$$a_i = \prod_{f_k \in F} d_{f_k} \rightarrow \{true, false\}$$

$$valid(a_i) = true \iff \{\forall r_i \in R | BR(a_i) \equiv true\}$$

Otherwise, if there exist any relation that cannot be satisfied, the assignment will be considered invalid, hence, valid predicate will return *false* value. For instance, the previous assignment, a_i, represents a valid assignment since satisfies the mandatory relation between $PS2$ and A feature and the or-relation between A and $A1$. The $\{PS2 = true, A = true\}$ assignment is invalid due to the relation between A, $A1$, and $A2$ is unsatisfied.

For our purpose, a configuration represents an assignment of FMF. Thus, the previous examples of assignments represent configurations for the feature model. The configurations can be valid whether the selection of assigned features to *true* satisfies all the relations, invalid otherwise. The set of valid and invalid assignments of FMF can be defined as the union of all valid and invalid assignments, respectively.

$$A_{FMF}^{valid} = \bigcup_{a_i \in A_{FMF}} \{a_i | \quad valid(a_i)\}$$

$$A_{FMF}^{invalid} = \bigcup_{a_i \in A_{FMF}} \{a_i | \quad \neg valid(a_i)\}$$

The union of the sets of valid and invalid assignments represent the domain of assignments for FMF model. Let A be redefined as the union of the sets of valid and invalid assignments.

$$A_{FMF} = A_{FMF}^{valid} \cup A_{FMF}^{invalid} \quad , \quad A_{FMF}^{valid} \cap A_{FMF}^{invalid} = \emptyset$$

The snippet code in Figure 2 presents an example of a CSP model for ChocoSolver tool [52] obtained from the model of Figure 1. Each variable represents a feature that can or cannot participate in the configuration, represented by the value 1 or 0 respectively. Constraints establish the relation between features such as *ifonlyif* restriction which represents the mandatory relation.

```
==== VARIABLES ====
B1 [0, 1], A [0, 1], B2 [0, 1], A2 [0, 1], C1 [0, 1], PS2 [0, 1], B [0, 1], B.cost [0, 10],
B.risk [0, 8], B.benefit [0, 15], B3 [0, 1], A1 [0, 1], B4 [0, 1], C [0, 1], C2 [0, 1],
rel-4_card [1, 2], rel-9_card [1, 2], S-B1 [0, 1], S-B2 [0, 1], D-A [0, 1], S-A2 [0, 1],
S-C1 [0, 1], S-B3 [0, 1], S-B [0, 1], S-B.cost [0, 1], S-B.risk [0, 1], S-B.benefit [0, 1],
S-PS2 [0, 1], S-A1 [0, 1], S-B4 [0, 1], S-C [0, 1], S-C2 [0, 1]
==== CONSTRAINTS ====
ifonlyif({PS2[0,1],cst[1],A[0,1],cst[1]})
implies({B[0,1],cst[0],B2[0,1],cst[0]})
ifonlyif({S-C1[0,1],cst[1],C1[0,1],cst[1]})
ifthenelse({C[0,1],cst[0],C1[0,1]C2[0,1],rel-9_card[1,2],C1[0,1]C2[0,1],cst[0]})
ifthenelse({B[0,1],cst[1],B.risk[0,8],cst[1090519040],
B.risk[0,8],cst[1077936128],B.risk[0,8],cst[0]})
ifonlyif({S-B.benefit[0,1],cst[1],B.benefit[0,15],cst[0]})
ifonlyif({S-B1[0,1],cst[1],B1[0,1],cst[1]})
ifonlyif({S-C2[0,1],cst[1],C2[0,1],cst[1]})
...
```

Figure 2. An example of code to check signatures.

Thus, the formalisation enables inferring information by applying some operations over the model. Modelling a feature model as a CSP enables verifying the satisfaction of the model, and to obtain the number of valid software products.

Definition 3. ***Products***. *Let FM be a feature model, the potential products of FM is the valid configurations of its equivalent FMF.*

$$products(FM) = A^{valid}_{FMF}$$

Definition 4. ***Number of Products***. *Let FM be a feature model, the potential number of products of FM is the solution of its equivalent number of solutions of the FMF.*

$$NumberOfProducts(FM) = |A^{valid}_{FMF}|$$

For example, all the valid configurations obtained from the model of Figure 1 are shown in Figure 3. For each configuration, the selected features are shown. For example, features $PS1$, A, and $A1$ are selected for $Configuration1$. The final product is composed of these features.

```
Configuration1 = "PS2;A;A1;",
Configuration2 = "PS2;C;C2;A;A1;",
Configuration3 = "PS2;C;C1;A;A1;",
Configuration4 = "PS2;C;C1;C2;A;A1;",
Configuration5 = "PS2;C;C1;A;A2;",
Configuration6 = "PS2;C;C1;A;A1;A2;",
Configuration7 = "PS2;C;C1;C2;A;A2;",
Configuration8 = "PS2;C;C1;C2;A;A1;A2;"
```

Figure 3. List of configurations.

As aforementioned, the analysis of a product line enables inferring certain information, whether a feature model or a certain configuration (c) is valid or not.

$$validModel(FM) \iff |A^{valid}_{FMF}| \neq 0$$
$$validConfig(FM, c) \iff c \in A^{valid}_{FMF} \wedge |A^{valid}_{FMF}| \neq 0$$

For instance, given a configuration $c = \{PS2 \rightarrow true, A \rightarrow, A1 \rightarrow true, B \rightarrow false, ...\}$ is a valid configuration whether c belongs to the set of valid assignments of the formalised feature model, FMF.

In this case, the configuration is valid as previously indicated hence it belongs to the set of valid assignments.

In the Table 1, relevant characteristics of the example of Figure 1 are shown, and the results of applying two operations: (1) to verify whether the model is valid (symbol ✓) and (2) to obtain the number of all possible configurations.

Table 1. Relevant characteristics for the example in Figure 1.

Model Characteristics	Number of features	12
	Mandatory	2
	Optional	5
	OR	2
	XOR	0
	Attributes	1
	Cross-Relations	2
Model Analysis	Valid	✓
	Number of configurations	8

When a configuration c is not valid, there are some failures (no correct assignment to features) in that configuration. The minimal diagnosis represents the explanation which makes the configuration invalid changing the value of minimal features. For simplification reasons, also a configuration could be represented as a set of only features that have value true in the assignment. In the previous example, the tuple c could be represented as the set $C = \{PS2, A, A1\}$.

Definition 5. ***Diagnosis of Configurations****. Let $\triangle$ be the minimal diagnosis of an invalid configuration C with regard to FMF is the minimal subset of features of C that must be modified their values such that the resultant configuration C′ is a valid configuration.*

$$\neg validConfig(FM, C) \xrightarrow{\triangle} validConfig(FMF, C')$$

Invalid configurations could have different Δ which satisfies the previous assert. For instance, the configuration $C = \{PS2, A\}$ is invalid according to the Figure 1. The problem is that the features $A1$ or $A2$ must be selected but they were not. Therefore, the minimal diagnosis could be $\triangle_1 = \{A1\}$ or $\triangle_2 = \{A2\}$. In the first minimal diagnosis, C' is $\{PS2, A, A1\}$ and in the second minimal diagnosis, is $\{PS2, A, A2\}$.

4. CyberSPL: Cybersecurity Software Product Line

In this section, the workflow and operation of CyberSPL methodology is introduced as well as the main characteristics and implementation details of the approach.

4.1. CyberSPL Workflow and Operation

As aforementioned, CyberSPL is focused on verifying the compliance of cybersecurity policies according to a configuration problem. The CyberSPL workflow is based on the business process shown in Figure 4. This process enables evaluating different cybersecurity contexts for an organisation. The process, inputs, outputs, stakeholders, and the specification are detailed to describe each activity. NIST 800-181 categories National Initiative for Cybersecurity Education (NICE) Cybersecurity Workforce Framework: https://csrc.nist.gov/publications/detail/sp/800-181/) have been used to represent the stakeholders' areas of responsibility for each activity:

1. There is a *selection of security contexts* (cf., Define SPL context) according to this cybersecurity policy. In this stage, the organisation has to do an effort on the analysis of the enterprise architecture, thus, to identify the assets and the security control in all the stages of every enterprise layer.

Afterwards, the organisation should delimit the scope, the context, by identifying the ecosystems to take into account for the application of the cybersecurity policy.

- *Input*: Cybersecurity policy, Assets.
- *Output*: Cybersecurity Context.
- *Specification*: Analysis of the resources, processes, systems, security controls, and the cybersecurity policy conditions.
- *Stakeholders*: Operate and Maintain (OM), Protect and Defend (PR), Investigate (IN), and Analyse (AN).

2. Once the context is delimited, feature models are selected (cf., Select/Build feature models), if any in the catalogue or constructed. Thus, CyberSPL provides a set of public or private repository of models that can be useful for the analysis of the systems in the selected context. The organisation, through product managers, just should select or define the feature models needed.

   - *Input*: Cybersecurity Context, Feature Model Catalogue.
   - *Output*: Feature models.
   - *Specification*: Selection of feature models or definition of feature models that describe the context.
   - *Stakeholders*: Securely Provision (SP).

3. A customisation of *the features, attributes and constrains* are set (cf., update features, attributes and constraints). The selected or defined feature models must be fully updated with the latest details of the cybersecurity policy established by the organisation. In this case, product managers must adjust certain parameters into the models regarding the subsequent analysis. The adjustments generate new versions of the models that are stored.

   - *Input*: Feature Models.
   - *Output*: Refined feature models store in the catalogue.
   - *Specification*: Include, delete, modify, and update into the models. Thus, update and readjust feature models.
   - *Stakeholders*: Securely Provision (SP).

4. Once the models are adjusted, the organisation is able to analyse the context and possible configurations (cf., Analysis Feature Models). Thus, the organisation can apply different operations: (1) validate the models, thus, validate the context of the organisation; (2) determine the number of possible configurations in the context; (3) determine and export all the possible configurations in the context; (4) validate certain configuration product or service according to the established models that describe the context; (5) diagnose certain configuration with regard a context.

   - *Input*: Refined feature models.
   - *Output*: Depending on the operation different output is obtained: a Boolean value which represents the validation of the model, a Boolean value which represents the validation of a configuration model, a number when the number of configurations is required, etc.
   - *Specification*: Validation model, Valid a configuration, Diagnose configuration, Number of configurations, determine all the configurations.
   - *Stakeholders*: Securely Provision (SP).

The *selection of security context* and the *selection of features and attributes* depend on the established cybersecurity policies, and it will require a previous analysis before defining the feature models, as shown in [19]. For instance, a service deployment context can be selected based on an application

server, like *Apache Tomcat* server. To reconcile the cybersecurity policy communications with the server must be through a secure channel based on Secure Socket Layer (SSL) [53] and Transport Layer Security (TLS) [54,55].

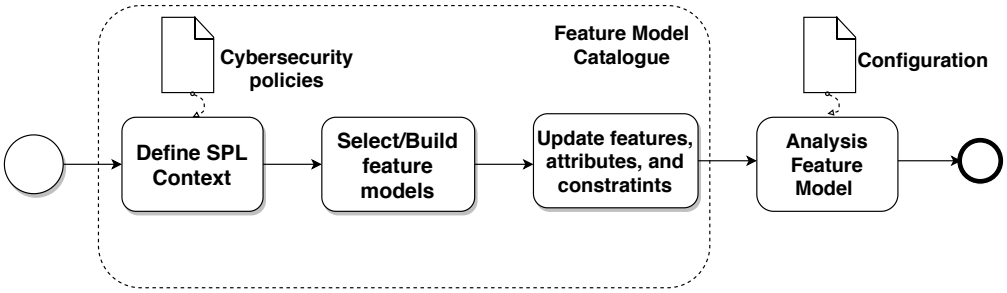

Figure 4. Process for verifying compliance of cybersecurity policies.

CyberSPL provides mechanisms for graphically constructing new feature models and the formal specification FAMA [49] format, and for selecting feature models from a catalogue as shown in Figure 5. CyberSPL also enables the persistence and updating of feature models into a catalogue.

Definition 6. ***Catalogue of models****. Let be CT, a catalogue formed of a set of* $\{FMF_1, FMF_2, \cdots, FMF_n\}$ *formalised feature models that represents certain cybersecurity contexts and policies.*

$$CT = \bigcup_{i=1}^{n} FMF_i$$

The goal of using a catalogue is to reduce the time of development and analysis reusing previous feature models for similar contexts. In a previous work [16], a set of feature models such as security controls are defined to configure business process engines. Nevertheless, those models are out-of-date. As mentioned, this catalogue can be shared for reusing the knowledge stored in the models.

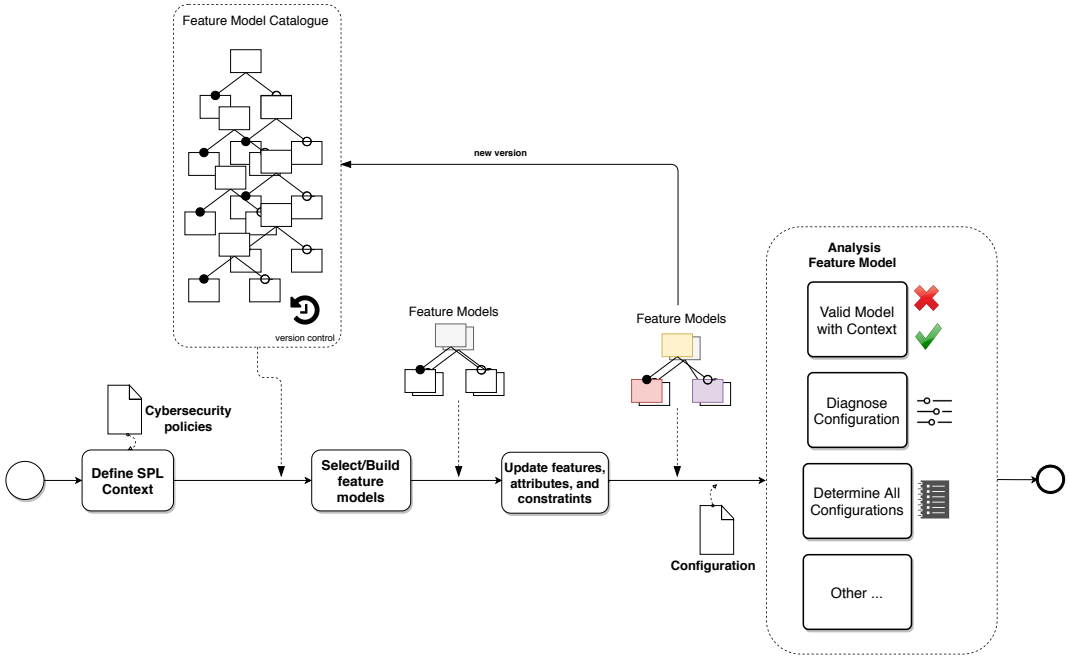

Figure 5. CyberSPL workflow with feature model catalogue.

By using the catalogue, a depth workflow of CyberSPL is shown in Figure 5. First, a model is selected from the catalogue depending on the context and the policy, and then the model is

readjusted (cf., tuning) with appropriate features and attributes of the context. The editions and updates (cf., update) of the existing feature model creates automatically a new version of this model. Thus, given a feature model and a list $M = \{m_1, m_2, \cdots, m_k\}$ of modifications (i.e., updates and tuning) the application of these modifications over the model FMF creates a new model.

$$update(FMF, M) = FMF' \quad | \quad FMF' \neq FMF$$

The update or tuning of feature models creates new versions for this model that are gathered into a historical record.

Definition 7. ***Historical record****. Let be H, a historical record with the set of records* $\{h_1, h_2, \cdots, h_n\}$ *where* h_i *is a triple* (f_i^o, M, f_i^d) *where* f_i^o *is the original feature model, M is the list of modifications applied, and* f_i^d *is the resulting feature model after applying M.*

$$history(FMF) = \{h_1, h_2, \cdots, h_n\}$$
$$\forall_{i=1}^{n-1}(h_i, h_{i+1}) \quad | \quad h_i(f_i^o, m_i, f_i^d), h_{i+1}(f_{i+1}^o, m_{i+1}, f_{i+1}^d) \implies f_i^d = f_{i+1}^o$$
$$update(FMF, M) = FMF' \implies history(FMF') = \{history(FMF) \cup (FMF, M, FMF')\}$$

CyberSPL is able to gather information related to all the changes applied over a model. Thus, CyberSPL presents a track record (i.e., versioning history) to enable a version control of feature models (cf., version control in Figure 5).

This tuned and update model is transformed into a formal constraint-based model, as it was shown in Section 3. Depending on the model, the appropriate solver is selected and the *compliance analysis* is applied. For example, to validate the model according to the context, to diagnose a particular security configuration, to obtain all valid configurations, or other operations.

4.2. CyberSPL Implementation

The CyberSPL architecture is modular, and it includes as main modules a user management module, a platform management module, and a feature model management module as shown in Figure 6. CyberSPL is integrated with the FAMA framework via a REST API (cf., CyberSPL API REST) to integrate reasoning characteristics. In this respect, the FAMA framework uses ChocoSolver (ChocoSolver: http://www.choco-solver.org) reasoner which enables the analysis and reasoning with feature models. CyberSPL and public models are free accessible for the community at https://estigia.lsi.us.es/cyberspl.

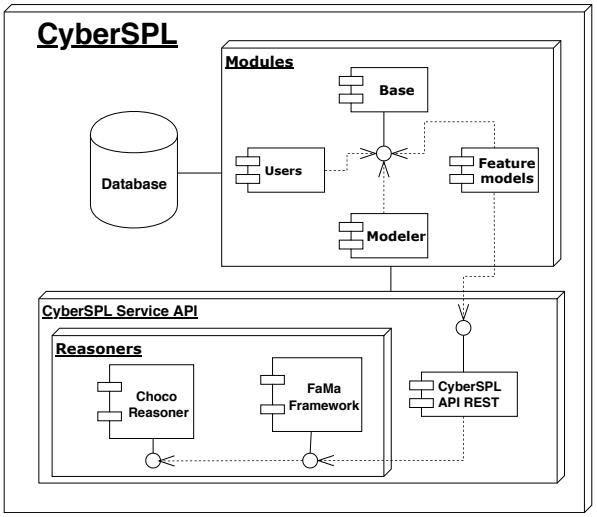

Figure 6. CyberSPL architecture.

CyberSPL is designed as a cross-platform and multi-user web solution. The user only needs a web browser to register and to use it. The public models are available in CyberSPL for all users of the community, and users are able to create their catalogue of models as shown in Figure A1 at the appendices. In the public repository, the feature models can be consulted and analysed but not edited. Moreover, in Figure A1 it is shown how to edit a CyberSPL model in a formal way (i.e., FAMA format) or graphically as is shown in Figure A2. The models of a user can be private and exclusive for his/her, or public and available for every user. In contrast, in the user private repository, each model can be edited, deleted, or analysed. Figure A1 shows a private repository of a user, but some models have been established as public (cf., Toy Model).

As aforementioned, the edition of existing feature model automatically creates a new version of this model. Thus, CyberSPL presents a track record (i.e., versioning history) to enable a version control of feature models (cf., Version History button in Figure A1). At the appendices, Figure A3 shows in background the versioning history with a track record of versions of model shown in Figure 1. The focused window highlighted one of the tracked changes using the *Show Diff* button.

For each model, CyberSPL enables the editing, deleting, and performing an analysis. The analysis section provides for main options as shown at the appendices in Figure A4: (1) to validate model; (2) to diagnose compliance of a configuration and a cybersecurity policy; (3) to obtain all valid configurations for cybersecurity policy, and; (4) diagnosis with regard to the history.

In Figure A4 (cf., Appendix A), the option of diagnosis a configuration is selected, and a specific configuration has been introduced and compared by CyberSPL with the model previously shown in Figure 1. The results of the operations performed on the model are available for the user in a panel (cf., Analysis Console). CyberSPL has an option to save externally in a JSON format, the obtained configurations. This option is named *Export configurations* and it is below the option to obtain all configurations.

Although CyberSPL has been created with the aim of analysing security using SPL techniques, it also provides characteristics that differentiate it from the rest. In general, most of the tools provide a modeller (i.e., in formal or graphical way) and automated analysis similar to CyberSPL. A comparison of CyberSPL with regard to non-commercial tools has been done in Table 2, but the comparison has been focused on the next aspects: (1) *Web*, if a distributed and multi-user tool is provided; (2) *Diagnosis*, if the tool supports diagnosis mechanisms; (3) *Private catalogue*, if the tool supports the development of a catalogue of models per user; (4) *Public catalogue*, if the tool allows users to share their models publicly; (5) *Historical*, if the tool allows the versioning of models and maintain a history of these versions per model.

Table 2. Comparison of SPL tools.

Tool	Web	Diagnosis	Public Catalogue	Private Catalogue	Historical Record
CyberSPL	✓	✓	✓	✓	✓
Variamos	✓	∼			
SPLOT	✓	∼	✓		
FAMILIAR		∼			
FeatureIDE		∼			
Glencoe	✓	∼		✓	

First, there are two main conclusions after the comparison: (1) none of the tools support all the characteristics at the same time; (2) none of the tools consulted to support the feature model versioning and the historical record of models. Regarding diagnosis, all the studied tools provide automated tools to validate and verify certain aspects of the model and configurations, for this reason, they have been stated as partial in diagnosis. Regarding the catalogue aspects, only SPLOT provides a public catalogue of models but not a private catalogue and Glencoe presents similar private catalogue per users, but not a public catalogue of models. The rest of the tools enable creating projects wherein some cases, only one feature model can be stored. This can be seen as a private repository, but in a similar way as shown in CyberSPL.

5. Evaluation Approach

To illustrate and evaluate the advantages for applying CyberSPL, the use case is presented as shown in Figure 7.

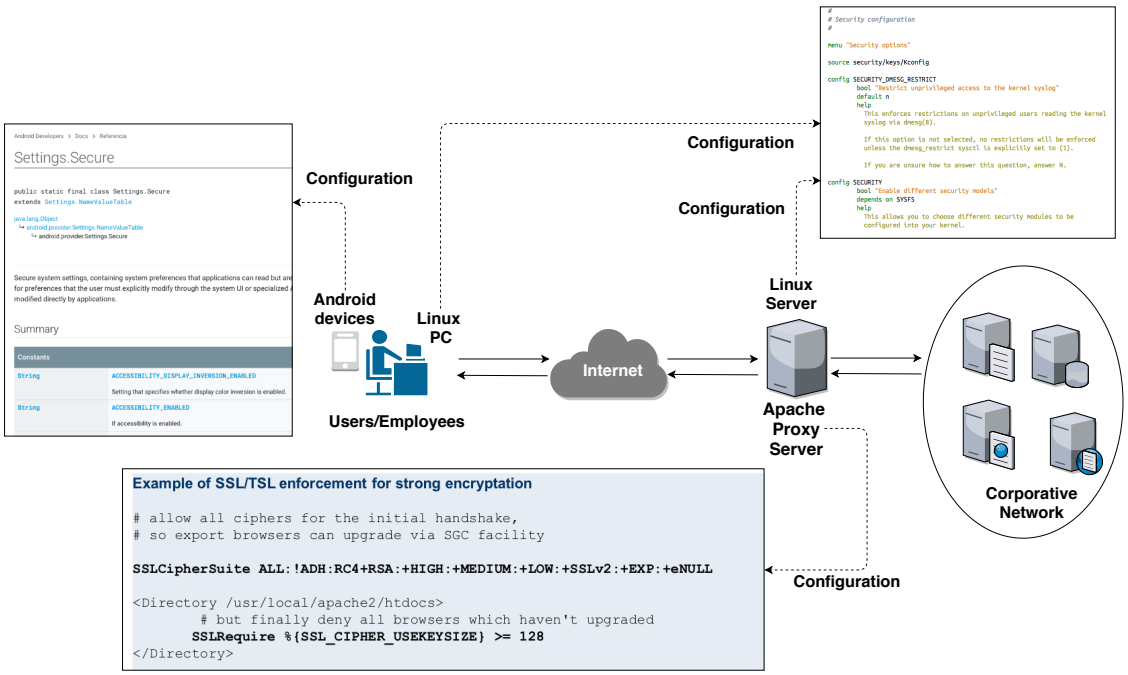

Figure 7. Use case scenario.

5.1. Context Description and Feature Models

The use case represents an enterprise architecture which encompassed several security patterns of the Open Security Architecture (OSA) [56]. The security patterns used in this use case are: 'SP-001: Client Module', 'SP-002: Server Module', 'SP-003: Privacy Mobile Device Pattern', and 'SP-008: Public Web Server Pattern'. This architecture simulates a real cybersecurity context, where an organisation has a set of users and employees connected to corporate network through an *Apache Server* as proxy.

The users and employees can use personal devices such as personal computers and smart phones to connect the corporate network. The cybersecurity policy specify restrictions for users to connect the corporate network by using linux-based operating systems and Android devices in the case of personal computers and smart devices respectively. On the other hand, the cybersecurity policy states that the connections to the corporate network must be secure, for instance, by securing the channel through the use of *SSL/TLS* protocol. In Figure 7, a part of the configuration of an *Apache Tomcat* server is shown, where protocol versions or key sizes are established.

To analyse the use case cybersecurity contexts, four feature models are given as a catalogue: (1) Security *Apache* model; (2) *SSL/TLS* model; (3) Security *Linux* kernel model; (4) Security *Android* model. The two first models are an updated evolution of those presented in a previous work [16], where unsafe features have been removed and other parameters such as key sizes have been added in the case of the *SSL/TLS* model. The *Linux* and *Android* models are the consequence of the analysis of security configuration parameters provided by the Linux kernel and the security settings that provide Android developer guide to secure applications.

The *Apache* proposed model is shown in Figure 8; it is based on security features of an *Apache Tomcat* server. The *Apache Tomcat* server can provide different HTTP connectors (Apache Tomcat Configuration: https://tomcat.apache.org/tomcat-7.0-doc/config/http.html) that enable listening for connections on a specific TCP port number on the server. In this particular case, the *Apache* model represents the attributes needed to configure a HTTP connector with SSL support. Among the parameters to be set up in the *Apache* server, can be found: (1) *Protocol* to establish the version of the protocol to be used in the connections; (2) *Algorithm* to establish the certificate encoding algorithm to be used; (3) *Ciphers* to establish a list of encryption ciphers to be supported by the connections; if specified, only the ciphers that are listed and supported by the SSL implementation will be used; (4) *ClientAuth* to establish the protocol of checking the client certificate in the connection (true value to require a valid certificate chain from the client before accepting a connection, want value to request a client Certificate, but not fail if one is not presented, or a false value for not to require a certificate chain; (5) *keystore* to establish the type and attributes to the store server keys and certificates; (6) *Truststore* to establish the type and attributes to the store for the keys and certificates from other parties that you expect to communicate with. In order to configure SSL connections in Apache, SSL/TLS capabilities and restrictions must be considered since those restrictions can determine the implementation of secure connections to the server. For instance, the use of an specific version of the *Procotol* and *Algorithm* restricts the number of *Ciphers* to be used for the compatibility. From the perspective of security the selection of certain protocols can determine the weakness of the connections, for instance, in the use of old version of TLS. For a better understanding of the ciphers' restriction, SSL/TLS protocol is carried out.

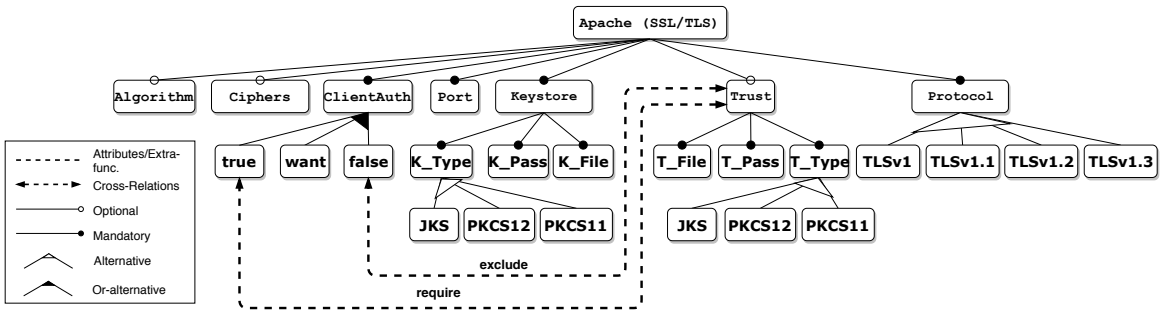

Figure 8. Feature Model of Apache Server.

As aforementioned, there are certain parameters, such as the versions of the protocol, that can be configured in the model. For instance, SSLv3 is discouraged due to vulnerabilities in the protocol that enable the application of exploits such as POODLE attacks. Although SSLv3 is enabled and supported by the different providers, most of the applications discouraged the use. Therefore, *SSLv3*

is discouraged and it was removed in the model. *TLSv1* or *TLSv1.1* are weak versions, but they are still being used because some suppliers and customers support these protocols, although the end of the support is announced for the year 2020. Client authentication (cf., *ClientAuth*) requires certain infrastructure, such as stores (cf., *Keystore* and *Trust*) and digital certificates. As well, certain protocols allow the possibility of setting specific Cipher Suites (cf., *Ciphers*). An example of configuration is shown in Figure 9.

```
<!-- Define an SSL Coyote HTTP/1.1 Connector on port 8443 -->
<Connector
protocol="org.apache.coyote.http11.Http11NioProtocol"
port="8888" maxThreads="250"
scheme="https" secure="true" SSLEnabled="true"
keystoreFile="${user.home}/.keystore" keystorePass="PASSWORD"
clientAuth="false" sslProtocol="TLSv1+TLSv1.1+TLSv1.2"/>
```

Figure 9. Example of HTTP connector in Apache Coyote Server.

The handshake of protocol *SSL/TLS* is based on this sequence of steps: (1) Negotiation of the *Cipher Suite* to be used during the transfer, and generation and exchange of a random number (master key); (2) Set and exchange a session identifier between client and server; (3) Authentication of the server to the client; (4) Authentication of the client to the server. The handshake has been simplified in *TLS* version 1.3. *SSL/TLS* enables the authentication of the client and the server, and anonymous communication. Authentication is done through digital signatures, such as digital certificates or keys. The digital certificates must be vouched for as validated by a Certificate Authority (CA). *SSLv3* and *TLSv1.0* protocols also enable anonymous authentication based on *Diffie-Hellman* key exchange.

The *SSL/TLS* model is shown in Figure 10 and it is a complete model where all the features for configuring the cipher suites are available for versions *TLS1.2* (TLS version 1.2 RFC 5246: https://tools.ietf.org/html/rfc5246) and *TLS1.3* (TLS version 1.3 RFC 8446: https://tools.ietf.org/html/rfc8446). All previous versions are considered unsafe and they have been removed of the model. This model can be linked to *Apache* Model (Figure 8) by the feature *Ciphers*.

The *SSL/TLS* model represents the attributes to configure certificates, key-stores, ciphers, key sizes on the use of any *SSL/TLS* protocol version. The main features to setup into *SSL/TLS* communications are: (1) *CipherSuite* which indicates the suites of key change method, cipher and message authentication code are supported; (2) *Key change method* which indicates that cryptographic algorithms have been employed to generate cryptographic keys; (3) *Cipher* which indicates that conventional cryptographic algorithms are employed to encrypt the message in the transmission; (4) *Message Authentication Code (MAC)* which indicates the algorithm employed to encrypt the message to provide integrity; (5) *Protocol* which indicates the version of protocol to be used; (6) *Session Resumption* which indicates the identification session established in the negation of connection client–server; (7) *Authentication method* which indicates the authentication method to be used (certificates or shared keys); (8) *Digital signature* which indicates other types of signature support instead of certificates (SRP and PSK); (9) *Certificate* which indicates the type of certificates supported (x509 and OpenPGP); (10) *Keysize* which establishes the size to be used for the cryptographic algorithms; (11) *Curves* are the function to be used to produce keys.

It is important to notice that the *SSL/TLS* model is a simplification since constraints are not represented to have a clear and non-complex visual model. However, the constraints have been written down in the *Cross Relations* text box (cf., Figure 10). Some examples are: *AES_128_CBC* is not recommended for *TSLv1.3* protocol and the use of *RSA* implies that the key size (*KeySize*) must be *2048*.

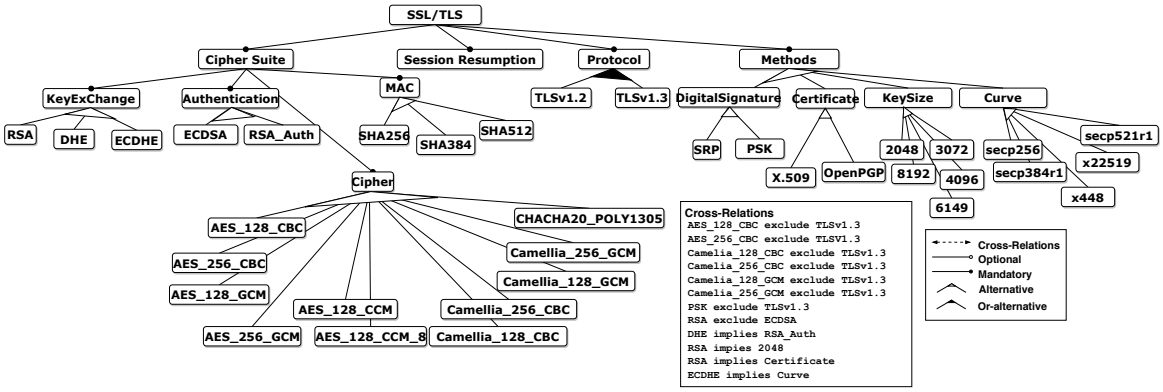

Figure 10. SSL/TLS Feature Model.

Figure 11 is related to the security configuration of the Linux kernel (Linux Kernel Security: https://elixir.bootlin.com/linux/latest/source). This feature model represents the Linux kernel security configuration in version 5.3. This model can help to understand which configurations are needed to develop aspects such as enabling the access control to the file system, restrict the access to *syslog*, even establishing different security modules. Regarding the features at the model represent: (1) *SECURITY* allows you to choose different security modules to be configured into your kernel; (2) *SECURITYFS* enables the *securityfs* file system; (3) *SECURITY_DMESG_RESTRICT* enforces restrictions on the access to privileged or sensitive information of *dmesg*; (4) *PAGE_TABLE_ISOLATION* feature is a countermeasure against security attacks which enables limiting the number of shared information between the users and kernel space; (5) *INTEL_TXT* is used as a security control to provided trusted initialisation of the kernel; (6) *HAVE_HARDENED_USERCOPY_ALLOCATOR* option checks memory transfers from to the kernel wrong memory regions when rejecting memory ranges; (7) *FORTIFY_SOURCE* enables detecting overflows of buffers in common string and memory functions; (8) *STATIC_USERMODEHELPER* is a countermeasure which enables redirecting the calls to kernel functions through a usermode helper as unique entry point to the kernel interface; (9) *LSM* enables the Linux kernel to support a variety of computer security models. This feature enables specifying a list of LSMs in initialisation order.

For instance, in version 5.3 of the Linux kernel, the socket and networking security hooks for a security module can be enabled to implement socket and networking access controls with the *SECURITY_NETWORK* configuration. This configuration depends on the security modules enabled into the kernel, which can be configured via *SECURITY* configuration. The enabled security modules can be also modified in this version of the kernel. The *LSM* configuration is used to manage them, which defines an ordered list of the enabled security modules. This configuration replaces the *DEFAULT_SECURITY* configuration, used in previous versions of the Linux kernel to specify the security module to be used by default.

In order to simplify the model and make it easier to read, the restrictions have not been used in the model to represent the dependency relationships between configurations. Instead, optional relationships have been used so that choosing a configuration gives access to the configurations that depend on it (if any). For instance, the configuration of the feature *HARDENED_USERCOPY_FALLBACK* and *HARDENED_USERCOPY_PAGESPAN* depends on the configuration of *HARDENED_USERCOPY* feature which also depends on the configuration of the *HAVE_HARDENED_USERCOPY_ALLOCATOR* feature.

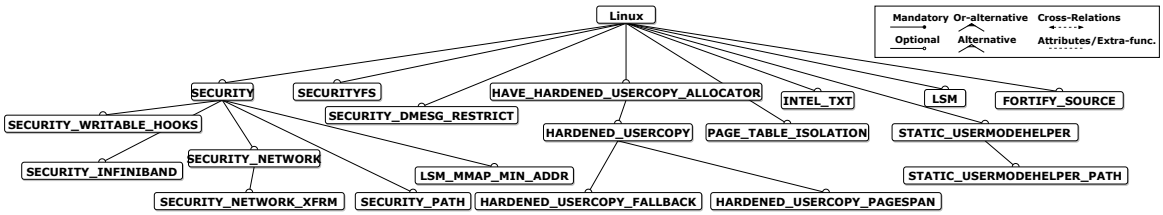

Figure 11. Security Linux Kernel Feature Model.

Figure 12 is related to the security settings provided by the API level of the Android SDK (Android Developer Guide: https://developer.android.com/reference/android/provider/Settings.Secure). This feature model represents the security-related system preferences specified in Android SDK API level 29. These preferences must be modified by the user explicitly through the system's user interface or specialised APIs. It should be borne in mind that this model excludes all preferences that, although related to system security, are obsolete at the API 29 (or earlier) level of the Android SDK.

Through this model, the security parametres that can be configured by users can be consulted through the apps in version 29 of the Android SDK API. For instance, the list of input methods that are currently enabled by the user is defined in the *ENABLED_INPUT_METHODS* configuration while the default input method is recorded into the *DEFAULT_INPUT_METHOD* configuration. Regarding location, most of the available configurations in previous versions of the API level of the Android SDK have been removed from the security settings but in the current version only two of them have remained: (1) *LOCATION_MODE_OFF* which specifies that the location mode is off, and; (2) *ALLOWED_GEOLOCATION_ORIGINS* which defines the origins for which browsers should allow geolocation by default.

Regarding the feature model structure, the preferences have been grouped in four groups to facilitate the readability of the model and the definition of configurations: (1) *ACCESSIBILITY* encompasses preferences related to system accessibility; (2) *INPUT_METHOD* encompasses preferences related to system input methods; (3) *LOCATION* encompasses preferences related to device location; (4) *TTS* groups preferences related to the Text-To-Speech system, and; (5) *MISCELLANEOUS* encompasses preferences not related to any of the groups described above.

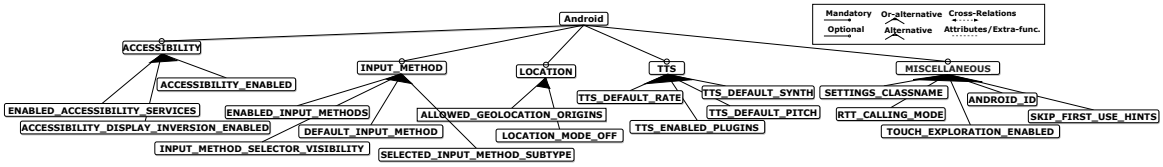

Figure 12. Security Android Feature Model.

5.2. Analysis of Feature Models and Diagnosis of Configurations

The analysis of the previous models is focused on validating them and computing the available configurations. The results are shown in Table 3. The models are valid, therefore, at least one valid configuration would be obtained from each model.

All the models have been validated and it would be reached without the automatic analysis provided by CyberSPL. Regarding the configurations, the number of possible configurations for the *Apache* model is 96, for the *SSL/TLS* model is 1482, for the *Android* is 262,144, and for *Linux* is 56,448. Even limiting the configurations to a few parameters, the complexity of the configuration continues to be very high. The high number of configurations of *Android* and *Linux* models with regard to the *Apache* and *SSL/TLS* models is derived from the type of relations and cross-relations. Thus, a high number of cross-relations will limit the number of solutions that satisfy the relations of the model. Despite this, the configurability problem is intractable whether it would be done manually by a human. The complexity increases by combining the systems, since the number of configurations would increase significantly because of the combinatorial explosion in the number of possible configurations.

Thus, there is a total number of 320,650 configurations among the four models. The automatic resolution of problems by means of CSP solver helps to deal with this complexity. The resolution of constraint-based problems is a well-known problem in the literature [50]. However, this complexity can be dealt with CSP solvers as shown in previous works, [16,57]. In both works, the complexity is studied in terms of performance and scalability of constraint-based problems, and we demonstrated remarkable results from the usability perspective response time in the resolutions.

Table 3. Analysis of the feature models.

Feature Model	***Apache***	***SSL/TLS***	***Android***	***Linux***
Number of features	27	48	24	20
Mandatory	10	8	0	0
Optional	3	0	5	18
OR	1	1	5	0
XOR	3	9	0	0
Cross-Relations	2	12	0	0
Valid	✓	✓	✓	✓
Number of configurations	576	1482	262,144	56,448

One of the advantages of CyberSPL is the possibility of diagnosing configurations as shown in Figure 13. The idea is to check if a configuration conforms to the cybersecurity policy represented in the feature models. In case of invalid configurations, CyberSPL is able to establish the diagnosis, that is, to determine what are the possible failures in them. In Figure 13, a diagnosis is depicted in the console that recommends to the user to select *TLSv1.2* or *TLSv1.3* and deselect *SSLv3*.

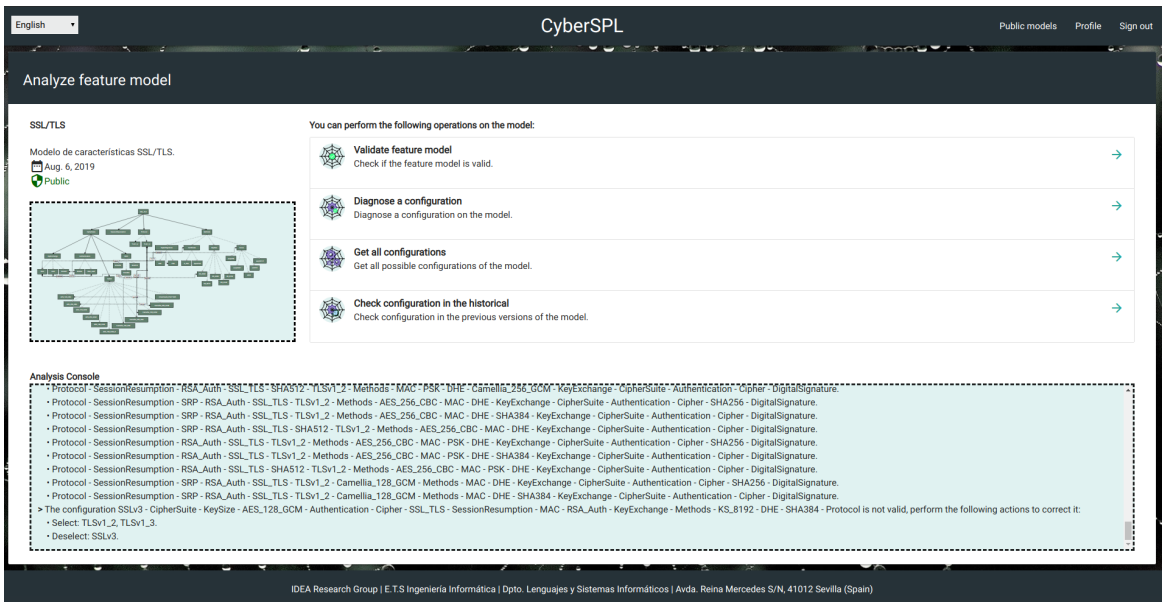

Figure 13. Diagnosis results in CyberSPL.

For each model six configurations have been tested, two valid and four invalid. The invalid configurations have been diagnosed and the explanation of the diagnosis is given. The cases of *Apache* and *SSL* are shown in Tables 4 and 5, respectively. Obtaining the diagnosis is more than determining whether a configuration is valid or not, it is an explanation about why a configuration is invalid according to the established cybersecurity policies. In particular, the diagnosis indicates the features that must be selected and deselected. Following the definition of diagnosis, *Select* an feature means that it had a *false* value in C (invalid configuration) and in C' (valid configuration) it would have the

true value. On the other hand, *Deselect* means that in C (invalid configuration) the concrete feature have a *true* value and in C′ (valid configuration) it would have a *false* value.

Table 4. Diagnosis of *Apache* model configurations.

#	Configuration	Result	Diagnosis
1	Apache, Protocol, TLSv1.1, KeyStore, K_File, K_Pass, K_Type, PKCS11, ClientAuth, false, Port	**Valid**	-
2	Apache, Protocol, TLSv1.2, KeyStore, K_File, K_Pass, K_Type, Ciphers, Algorithm, ClientAuth, want, Port, PCKS12	**Valid**	-
3	Apache, Protocol, KeyStore, K_File, K_Pass, K_Type, ClientAuth, false, Port, Algorithm, Ciphers	**No valid**	**Select**: TLSv1, TLSv1.2, TLSv1.1, TLSv1.3
4	Apache, Protocol, TLSv1.2, KeyStore,K_File, K_Pass, K_Type, ClientAuth, Port, Algorithm, Ciphers	**No valid**	**Select**: want, false
5	Apache, ClientAuth, false, Port, Keystore, K_File, K_Pass, K_Type, PKCS7, Protocol, TLSv1.3	**No valid**	**Select**: JKS, PKCS11, PKCS12 **Deselect**: PKCS7
6	Apache, Ciphers, ClientAuth, want, Keystore, K_File, K_Pass, K_Type, PKCS12, Protocol, TLSv1.3	**No valid**	**Select**: Port

Table 5. Diagnosis of *SSL/TLS* model configurations.

#	Configuration	Result	Diagnosis
1	SSL/TLS, Protocol, TLSv1.2, KeyExchange, DHE, CipherSuite, Cipher, AES_128_GCM, MAC, SHA256, Authentication, ECDSA, Methods, KeySize, 3072, SessionResumption	**Valid**	-
2	SSL/TLS, Protocol, TLSv1.2, CipherSuite, Cipher, AES_128_CCM, Authentication, ECDSA, KeyExchange, DHE, MAC, SHA512, Methods, KeySize, 3072, SessionResumption	**Valid**	-
3	SSL/TLSS, Protocol, TLSv1.2, CipherSuite, Cipher, AES_128_CCM, Authentication, KeyExchange, ECDHE, MAC, SHA512, Methods, KeySize, 3072, SessionResumption	**No valid**	**Select**: RSA_Auth, DHE **Deselect**: ECDHE
4	SSL/TLS, Protocol, CipherSuite, MAC, SHA384, Authentication, ECDSA, Cipher, AES_256_GCM, KeyExchange, DHE, Methods, KeySize , 4096, SessionResumption	**No valid**	**Select**: TLSv1.2, TLSv1.3
5	SSL/TLS, CipherSuite, KeyExchange, SRP, Authentication, RSA_Auth, Cipher, AES_128_CCM, MAC, SHA512, SessionResumption, Protocol, TLSv1.2, Methods, KeySize, 8192	**No valid**	**Select**: DHE **Deselect**: SRP
6	SSL/TLS, CipherSuite, KeyExchange, DHE, Authentication, RSA_Auth, Cipher, AES_128_CCM, MAC, SHA512, SessionResumption, Protocol, TLSv1.2, Methods, KeySize, 1024	**No valid**	**Select**: 3072, 4096, 8192, 6149, 2048 **Deselect**: 1024

In the case of *Apache* configurations (cf., Table 4), the protocol version has not been specified in the first wrong configuration, and the suggested diagnosis is to select a specific version of the protocol.

In the second wrong configuration, the type of *ClientAuth* has not been indicated. In this case, you cannot select *true* because it would imply the selection of the *Trust* feature, so the suggested diagnosis is to select between the options *want* or *false*. In the case of the third invalid configuration, *PKCS7* has been chosen as the format of the certificate keystore. This configuration is invalid because, as indicated in the Apache Tomcat documentation, the current version of Apache Tomcat only works with the *JKS*, *PKCS11* and *PKCS12* keystorage formats. In fact, the configuration diagnosis indicates that in order to correct the configuration, the chosen format must be deselected and one of the three previously mentioned must be selected. In the case of the fourth invalid configuration, the *Port* feature has not been chosen, which corresponds to the *TCP* port number used by the HTTP Connector component of an Apache Tomcat server to create a server socket for incoming connections. This configuration is invalid since, as indicated in the Apache Tomcat documentation, if the Tomcat port is not configured, it will select a free random port, a situation that should only occur in embedded and test applications. In this way, in the configuration diagnosis it is indicated that in order to correct it, this characteristic must be selected.

In the case of *SSL/TLS* configurations (cf., Table 5), there are also four erroneous settings. For the feature of *Authentication*, no mechanism has been selected in the first wrong configuration, and the suggested diagnosis is to select *ECDSA* or *RSA_Auth*. In the second wrong configuration, *ECDHE* has been selected as a feature of *KeyExchange*, but that an elliptical curve (*Curve*) is required, and the configuration also sets the size of the key (*KeySize*) to 3072. The suggested diagnosis includes two possibles solutions: (1) deselect *ECDHE* or (2) to select features *RSA_Auth* and *DHE* since by selecting these features the configuration is compatible with a key size of 3072. In the case of the third invalid configuration, *SRP* has been chosen as the key exchange protocol. This configuration is invalid because, as indicated in the TLS definition (RFC 5246), RSA and DH must be used for key exchange. In this way, the configuration diagnosis indicates that in order to correct it, *SRP* must be deselected and *DHE* must be chosen. In the case of the fourth invalid configuration, 1024 has been chosen as the key size. This configuration is invalid because, although as indicated in the TLS definition (RFC 5246) the use of any key size is allowed, this size is not considered secure. In this way, the configuration diagnosis indicates that in order to correct it, the chosen size must be deselected and one of the recommended ones must be chosen.

The cases of *Linux* and *Android* are shown in Tables 6 and 7 respectively. In the case of the first invalid configuration of *Linux*, the feature *SECURITY_NETWORK_XFRM* is selected. This feature enables the XFRM (IPSec) networking security hooks. Nevertheless, this is not possible to be configured without previously selecting the *SECURITY_NETWORK* flag. The security network flag enables the socket and networking security hooks. Therefore, the diagnosis recommends to deselect *SECURITY_NETWORK_XFRM* and select *SECURITY_NETWORK*. Similarly occurs with the second invalid configuration, this configuration consists of several hardened user copy strategies (i.e., harden memory copies between the kernel and the userspace) such as *HARDENED_USERCOPY_FALLBACK* without previously selecting the feature *HARDENED_USERCOPY*. For the third invalid configuration, *STATIC_USERMODEHELPER_PATH* has been chosen without choosing *STATIC_USERMODEHELPER*. This configuration is invalid because this parameter defines the path to the static user mode auxiliary binary, but you must also include the *STATIC_USERMODEHELPER* parameter, which forces all user mode help calls to pass through a single binary. In fact, the configuration diagnosis indicates that, in order to correct the configuration this parameter must be deselected or *USERMODEHELPER* must be chosen. In the case of the fourth invalid configuration, the parameter *SECURITY_PATH* has been selected without selecting *SECURITY*. This configuration is invalid because this parameter, which enables security hooks for path-based access control, also requires you must choose the *SECURITY* parameter, which enables different security models. In fact, in the configuration diagnosis, it is indicated that in order to correct it, this parameter must be deselected or *SECURITY* must be chosen.

Table 6. Diagnosis of *Linux* model configurations.

#	Configuration	Result	Diagnosis
1	Linux, SECURITY, SECURITY_NETWORK, SECURITY_NETWORK_XFRM, INTEL_TXT, FORTIFY_SOURCE, STATIC_USERMODEHELPER, STATIC_USERMODEHELPER_PATH	**Valid**	-
2	Linux, SECURITY, SECURITY_NETWORK, SECURITY_NETWORK_XFRM, FORTIFY_SOURCE, HAVE_HARDENED_USERCOPY_ALLOCATOR, HARDENED_USERCOPY, HARDENED_USERCOPY_PAGESPAN, HARDENED_USERCOPY_FALLBACK	**Valid**	-
3	Linux, SECURITY, SECURITY_NETWORK_XFRM, INTEL_TXT, FORTIFY_SOURCE, STATIC_USERMODEHELPER, STATIC_USERMODEHELPER_PATH	**No valid**	**Select**: SECURITY_NETWORK **Deselect**: SECURITY_NETWORK_XFRM
4	Linux, SECURITY, SECURITY_NETWORK, SECURITY_NETWORK_XFRM, FORTIFY_SOURCE, HAVE_HARDENED_USERCOPY_ALLOCATOR, HARDENED_USERCOPY_FALLBACK, HARDENED_USERCOPY_PAGESPAN	**No valid**	**Select**: HARDENED_USERCOPY **Deselect**: HAVE_HARDENED_USERCOPY_ALLOCATOR, HARDENED_USERCOPY_FALLBACK, HARDENED_USERCOPY_PAGESPAN
5	Linux, SECURITY_DMESG_RESTRICT, INTEL_TXT, LSM, STATIC_USERMODEHELPER_PATH	**No valid**	**Select**: STATIC_USERMODEHELPER **Deselect**: STATIC_USERMODEHELPER_PATH
6	Linux, SECURITY_PATH, SECURITYFS, INTEL_TXT, FORTIFY_SOURCE	**No valid**	**Select**: SECURITY **Deselect**: SECURITY_PATH

Regarding the invalid configurations of the *Android* model, the first configuration has the feature *LOCATION_MODE_OFF* selected, when it requires the selection of the *LOCATION* feature. Therefore, the diagnosis consists of the selection of *LOCATION* feature and deselection of *LOCATION_MODE_OFF*. Regarding the second invalid configuration, the *INPUT_METHOD* feature is selected but this feature requires that any input method will be chosen. Therefore, the diagnosis proposes the selection of any input method (i.e., *DEFAULT_INPUT_METHOD*, *INPUT_METHOD_SELECTOR_VISIBILITY*, *SELECTED_INPUT_METHOD_SUBTYPE*, *ENABLED_INPUT_METHODS*). For the third invalid configuration, parameter *ACCESSIBILITY_SPEAK_PASSWORD* has been chosen. This configuration is invalid because, as indicated in the documentation of the Android security options (API 29), the dictation of passwords has become controlled by individual accessibility services and the apps ignore this parameter. Thus, in the configuration diagnosis there are two options to correct it: deselect *ACCESSIBILITY_SPEAK_PASSWORD* and choose some of the available accessibility options or deselect *ACCESIBILITY* to not select accessibility parameters. In the case of the fourth invalid configuration, parameter *TTS_DEFAULT_COUNTRY* has been chosen. This configuration is invalid because, as indicated in the documentation of the Android security options (API 29), the apps no longer use this parameter to know the country used by default by the text-voice converter, but consult the classes provided by the *TTS* framework. This way, in the configuration diagnosis there are two options to correct it: deselect *TTS_DEFAULT_COUNTRY* and choose some of the available *TTS* options or deselect TTS so as not to select parameters related to the text-voice converter.

Table 7. Diagnosis of *Android* model configurations.

#	Configuration	Result	Diagnosis
1	Android, ACCESSIBILITY, ACCESSIBILITY_ENABLED, ENABLED_ACCESSIBILITY_SERVICES, LOCATION, LOCATION_MODE_OFF, TTS, TTS_DEFAULT_PITCH, TTS_DEFAULT_RATE, TTS_ENABLED_PLUGINS, TTS_DEFAULT_SYNTH	**Valid**	-
2	Android, INPUT_METHOD, ENABLED_INPUT_METHODS, SELECTED_INPUT_METHOD_SUBTYPE, INPUT_METHOD_SELECTOR_VISIBILITY, MISCELLANEOUS, RTT_CALLING_MODE, ANDROID_ID, SKIP_FIRST_USE_HINTS, TOUCH_EXPLORATION_ENABLED	**Valid**	-
3	Android, ACCESSIBILITY, ACCESSIBILITY_ENABLED, ENABLED_ACCESSIBILITY_SERVICES, LOCATION_MODE_OFF, TTS, TTS_DEFAULT_PITCH, TTS_DEFAULT_RATE	**No valid**	**Select**: LOCATION **Deselect**: LOCATION_MODE_OFF
4	Android, INPUT_METHOD, MISCELLANEOUS, RTT_CALLING_MODE, ANDROID_ID, SKIP_FIRST_USE_HINTS, TOUCH_EXPLORATION_ENABLED	**No valid**	**Select**: DEFAULT_INPUT_METHOD, INPUT_METHOD_SELECTOR_VISIBILITY, SELECTED_INPUT_METHOD_SUBTYPE, ENABLED_INPUT_METHODS **Deselect**: INPUT_METHOD
5	Android, ACCESSIBILITY, ACCESSIBILITY_SPEAK_PASSWORD, LOCATION, LOCATION_MODE_OFF, INPUT_METHOD, INPUT_METHOD_SELECTOR_VISIBILITY	**No valid**	**Select**: ENABLED_ACCESSIBILITY_SERVICES, ACCESSIBILITY_DISPLAY_VERSION_ENABLED, ACCESSIBILITY_ENABLED **Deselect**: ACCESSIBILITY_SPEAK_PASSWORD, ACCESSIBILITY
6	Android, INPUT_METHOD, DEFAULT_INPUT_METHOD, ENABLED_INPUT_METHODS, TTS, TTS_DEFAULT_COUNTRY	**No valid**	**Select**: TTS_ENABLED_PLUGINS, TTS_DEFAULT_RATE, TTS_DEFAULT_SYNTH, TTS_DEFAULT_PITCH **Deselect**: TTS_DEFAULT_COUNTRY, TTS

5.3. Discussion of Results

In the previous section, a realistic scenario has been presented where several feature models represent a security context in which three product configurations have been analysed. The analysis has been carried out in twofold: (1) a first analysis in which the number of features, relations, validation of the model and the number of products were determined; (2) a second analysis in which particular configurations were validated and diagnosed. The aim of the analysis is to provide a perspective in the domain of the configuration of products in the security context and to help the involved practitioners in the task of the verification of policies and configurations according to the context.

The usefulness of the results obtained with CyberSPL can be observed from different points of views:

- For Business and security managers, CyberSPL enables improving the task of verification and analysis of cybersecurity policies due to the automatic analysis. Moreover, the use of catalogues and historical records of models enables doing a more flexible, effective, adaptable, and easy-to-maintain security contexts for organisations. The models can be shared and reused in multiple scenarios according to the necessity of the organisations. For instance, the models can be customised and enriched with attributes to indicate which risk level certain features have according to the organisation expectancy.
- For Business and security provisioners, CyberSPL enables automatically verifying and diagnosing whether the configurations comply with established policies and contexts. Moreover, CyberSPL

can be useful to obtain valid blueprints to secure and configure products or systems according to the organisation policies.

The advantage of CyberSPL is to provide a framework where users all over the world can develop, update, and share their models. Moreover, CyberSPL provides automatic analysis tools to verify and diagnose configurations and mechanisms to export and track the versions of the models. The main drawback of CyberSPL is that it requires a high initial effort in defining feature models as complete as possible according to the context. Nevertheless, CyberSPL provides a catalogue of public models that can be reused for that task; furthermore, once feature models are defined, it only requires an update cycle.

On the other hand, the problem of security configuration and misconfiguration is not transferred to a feature-domain problem, thus, feature models and the reasoning techniques help to solve the problem of the product and system configurations since the feature models gather all the information (features and constraints) to correctly (or not) configure specific products or systems according to a policy. Therefore, the feature model will not substitute the configuration problem, CyberSPL can be seen as a tool or complement which aids businesses, security, and operational practitioners to ensure the compliance of the configurations according to policies and contexts.

6. Conclusions and Future Work

In this article, the problem of cybersecurity configuration of products and systems has been tackled. In particular, the paper is focused on the compliance of those configurations with regard to cybersecurity policies. In order to confront this problem, the CyberSPL framework and its implementation is presented aligned with the five objectives presented. CyberSPL provides users with a set of tools for the definition and the maintenance of a catalogue of cybersecurity policies based on feature models. These policies are associated with different contexts of software products and systems related to cybersecurity to maintain a catalogue (OBJ1). CyberSPL enables the obtaining of the properties of the feature models (OBJ2). On the other hand, CyberSPL enables the automatic detection of configuration failures to validate with cybersecurity policies through the description of features of the different cybersecurity contexts (OBJ3). Moreover, it provides the diagnosis of configurations to isolate and identify the mistakes that cause the non-conformance of the cybersecurity policies (OBJ4). CyberSPL has been evaluated by presenting a catalogue of four real feature models related to a complex scenario encompassed of four different cybersecurity contexts, that is, the configuration of security options in the *Linux* kernel; the security settings of applications provided by *Android* SDK API level, the configuration of security channels in the *Apache* application server, and; the configuration of a Cipher Suite for the *SSL/TLS* protocol (OBJ5). Those models have been validated and automatic diagnosis capabilities have been also demonstrated fulfilling the defined objectives as described in the methodology Design Science.

In summary, the CyberSPL framework enables the verification of compliance with cybersecurity policies, and the analysis of product configurations, applications and services that participate. The use of CyberSPL facilitates an important advance in the automated management of cybersecurity policy compliance by cybersecurity managers taken into consideration the results obtained with the evaluation. Moreover, for operational development (DevOps) perspective, where ensuring and checking that a configuration is following the policy, the use of CyberSPL increases the alignment significantly between the development layer and the operational layer.

Furthermore, our work can be extended in two main directions: (1) to automatically update the feature models in accordance with technological advances or vulnerabilities that occur over time; (2) to maintain a register of the evolution of the feature models for each context, in order to allow more precise diagnosis when changes in cybersecurity policies occur.

Author Contributions: All the authors are responsible for the concept of the paper, the results presented and the writing and contributed equally to this work.

Funding: This work has been partially funded by the Ministry of Science and Technology of Spain by ECLIPSE project (RTI2018-094283-B-C33) and the European Regional Development Fund (ERDF/FEDER) via METAMORFOSIS project.

Acknowledgments: The authors would like to thank the Cátedra of Telefónica "Intelligia en la Red" of the University of Sevilla for their support in the development of this work, and José A. Galindo and David Benavides for the support of the FAMA tool.

Conflicts of Interest: All the authors have approved the final content of the manuscript. No potential conflict of interest was reported by the authors.

Appendix A

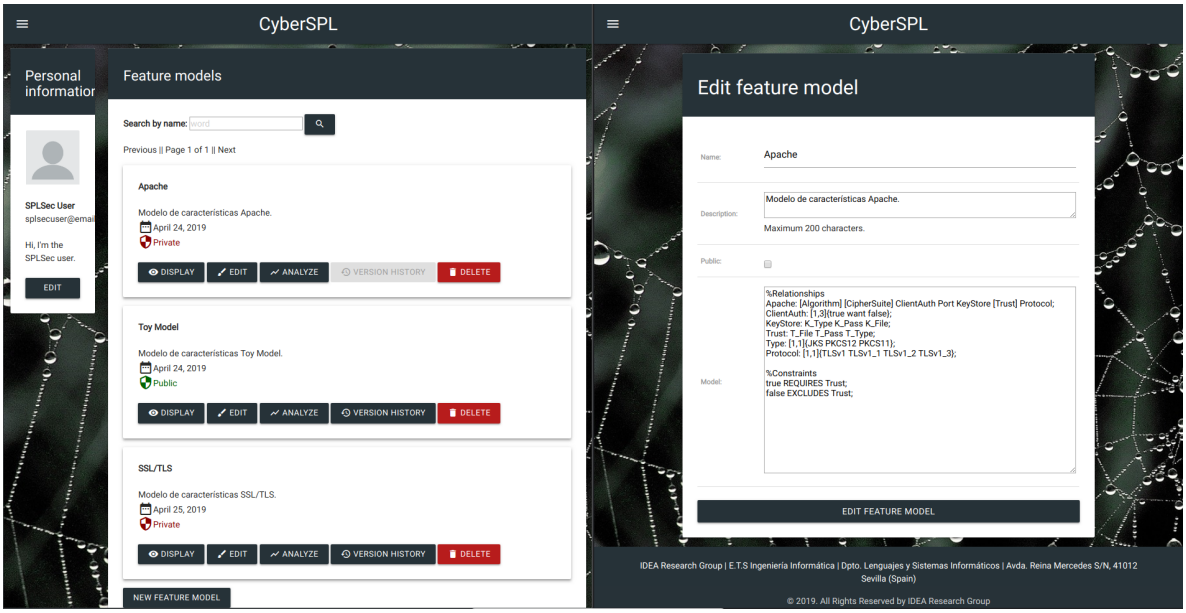

Figure A1. Catalogue of user models.

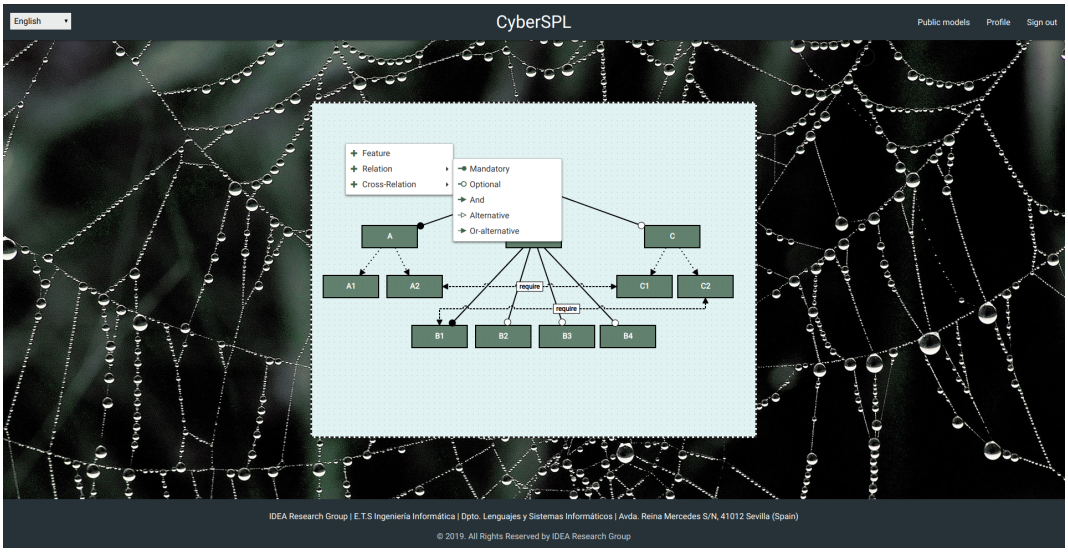

Figure A2. Graphical edition of feature models.

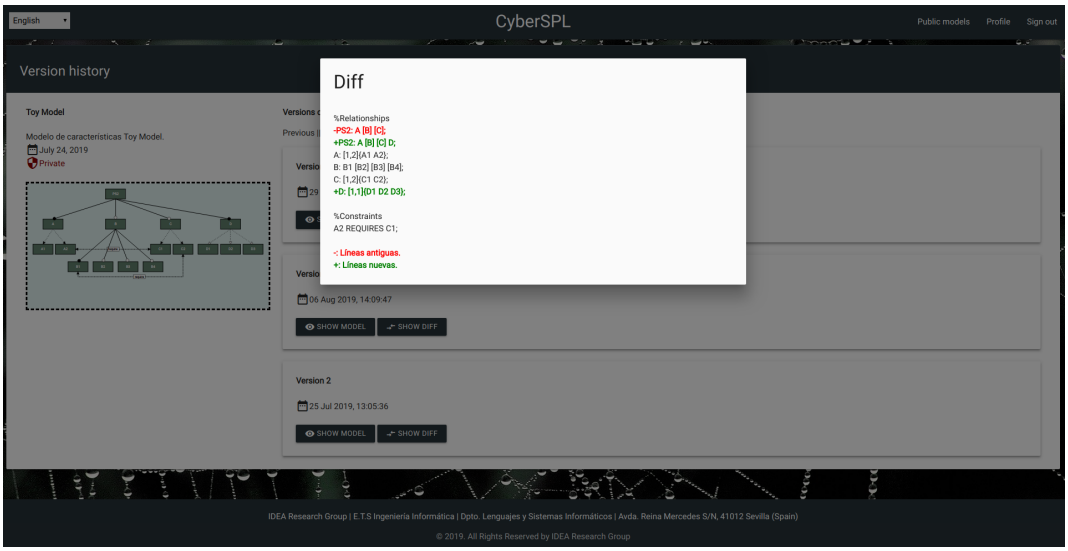

Figure A3. History version and diff between models.

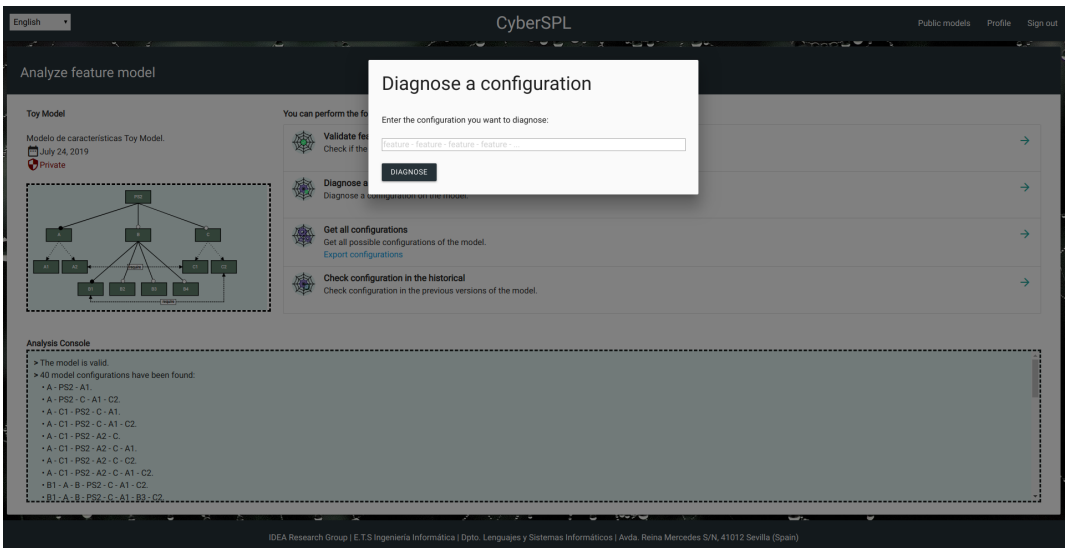

Figure A4. Model analysis.

References

1. Knapp, E. Chapter 11—Common Pitfalls and Mistakes. In *Industrial Network Security*; Syngress: Amsterdam, The Netherlands, 2011; pp. 303–312, ISBN 9781597496452.
2. Martínez, S.; Cosentino, V.; Cabot, J. Model-based analysis of Java EE web security misconfigurations. *Comput. Lang. Syst. Struct.* **2017**, *49*, 36–61. [CrossRef]
3. Fernández-Cerero, D.; Varela-Vaca, Á.J.; Fernández-Montes, A.; Gómez-López, M.T.; Alvárez-Bermejo, J.A. Measuring data-centre workflows complexity through process mining: The Google cluster case. *J. Supercomput.* **2019**. [CrossRef]
4. Bai, X.; Xing, L.; Zhang, N.; Wang, X.; Liao, X.; Li, T.; Hu, S. Apple ZeroConf Holes: How Hackers Can Steal iPhone Photos. *IEEE Secur. Priv.* **2017**, *15*, 42–49. [CrossRef]
5. Alfaro, J.G.; Boulahia-Cuppens, N.; Cuppens, F. Complete analysis of configuration rules to guarantee reliable network security policies. *Int. J. Inf. Secur.* **2008**, *7*, 103–122. [CrossRef]
6. Lallie, H.S.; Debattista, K.; Bal, J. Evaluating practitioner cyber-security attack graph configuration preferences. *Comput. Secur.* **2018**, *79*, 117–131. [CrossRef]
7. Li, X.; Xue, Y. A survey on server-side approaches to securing web applications. *ACM Comput. Surv.* **2014**, *46*, 29. [CrossRef]

8. OWASP Top Ten Project. OWASP. Available online: https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project#Users_and_Adopters (accessed on 25 September 2019).
9. Lotufo, R.; She, S.; Berger, T.; Czarnecki, K.; Wąsowski, A. Evolution of the Linux Kernel Variability Model. In *Software Product Lines: Going Beyond*; Springer: Berlin/Heidelberg, Germany, 2010, pp. 136–150.
10. Kyo, K.; Sholom, C.; James, H.; William, N.; Peterson. A. *Feature-Oriented Domain Analysis (FODA) Feasibility Study*; Technical Report CMU/SEI-90-TR-021; Carnegie Mellon University: Pittsburgh, PA, USA, 1990.
11. Batory, D. Feature Models, Grammars, and Propositional Formulas. In *Software Product Lines*; Springer: Berlin/Heidelberg, Germany, 2005; pp. 7–20.
12. *Software Product Line*; Carnegie Mellon Software Engineering Institute: Pittsburgh, PA, USA. Available online: https://resources.sei.cmu.edu/library/index.cfm?fp=sei_topic:Software+Product+Lines&global=true (accessed on 6 December 2019).
13. Sisiaridis, D.; Markowitch, O. Automating Feature Extraction and Feature Selection in Big Data Security Analytics. In *Artificial Intelligence and Soft Computing*; Springer International Publishing: Berlin/Heidelberg, Germany, 2018; pp. 423–432.
14. Costa, G.; Merlo, A.; Verderame, L.; Armando, A. Automatic security verification of mobile app configurations. *Future Gener. Comput. Syst.* **2018**, *80*, 519–536. [CrossRef]
15. Behringer, B.; Lehser, M.; Rothkugel, S. Towards Feature-Oriented Fault Tree Analysis. In Proceedings of the 38th International Computer Software and Applications Conference Workshops, Vasteras, Sweden, 21–25 July 2014.
16. Varela-Vaca, A.J.; Gasca, R.M. Towards the automatic and optimal selection of risk treatments for business processes using a constraint programming approach. *Inf. Softw. Technol.* **2013**, *55*, 1948–1973. [CrossRef]
17. Benavides, D.; Segura, S.; Ruiz-Cortés, A. Automated analysis of feature models 20 years later: A literature review. *Inf. Syst.* **2010**, *35*, 615–636. [CrossRef]
18. Schumacher, M. *Security Engineering with Patterns*; Springer: Berlin/Heidelberg, Germany, 2003.
19. Varela-Vaca, A.J.; Gasca, R.M. Formalization of security patterns as a means to infer security controls in business processes. *Log. J. IGPL* **2014**, *23*, 57–72. [CrossRef]
20. Peffers, K.; Rothenberger, M.A.; Kuechler, W.L. Design Science Research in Information Systems. In Proceedings of the 7th International Conference, (DESRIST), Las Vegas, NV, USA, 14–15 May 2012.
21. vom Brocke, J.; Braccini, A.M.; Sonnenberg, C.; Spagnoletti, P. Living IT infrastructures—An ontology-based approach to aligning IT infrastructure capacity and business needs. *Int. J. Account. Inf. Syst.* **2013**, *15*, 246–274. [CrossRef]
22. Varela-Vaca, A.J.; Gasca, R.M.; Ceballos, R.; Bernáldez-Torres, P. CyberSPL: Plataforma para la verificación del cumplimiento de políticas de ciberseguridad en configuraciones de sistemas usando modelos de características. In Proceedings of the Actas de las V Jornadas Nacionales de Investigación en Ciberseguridad (JNIC 2019), Extremadura, Spain, 5–7 June 2019.
23. Varela-Vaca, A.J.; Galindo, J.A.; Ramos-Gutiérrez, B.; Gómez-López, M.T.; Benavides, D. Process Mining to Unleash Variability Management: Discovering Configuration Workflows Using Logs. In Proceedings of the 23rd International Systems and Software Product Line Conference—Volume A (SPLC '19), Paris, France, 9–13 September 2019.
24. Galindo, J.A.; Benavides, D.; Trinidad, P.; Gutiérrez-Fernández, A.-M.; Ruiz-Cortés, A. Automated analysis of feature models: Quo vadis? *Computing* **2018**, 101, 387–433. [CrossRef]
25. Benavides, D.; Galindo, J.A. Automated analysis of feature models. In Proceedings of the 22nd International Conference on Systems and Software Product Line-SPLC '18, Gothenburg, Sweden, 10–14 September 2018.
26. Trinidad, P.; Benavides, D.; Durán, A.; Ruiz-Cortés, A.; Toro, M. Automated error analysis for the agilization of feature modeling. *J. Syst. Softw.* **2008**, *81*, 883–896. [CrossRef]
27. Felfernig, A.; David, R.W.A.G.; Seda, B.; Erdeniz, P.; Atas, M.; Reiterer, S. Anytime diagnosis for reconfiguration. *J. Intell. Inf. Syst.* **2018**, *51*, 161–182. [CrossRef]
28. Semmak, F.; Gnaho, C.; Laleau, R. Extended KAOS Method to Model Variability in Requirements. In *Communications in Computer and Information Science*; Springer: Berlin/Heidelberg, Germany, 2010; pp. 193–205.
29. Pérez, J.; Laguna, M.A.; González-Carvajal, Y.C.; González-Baixauli, B. Requirements Variability Support Through MDA^{TM} and Graph Transformation. *Electron. Notes Theor. Comput. Sci.* **2006**, *152*, 161–173. [CrossRef]
30. Sawyer, P.; Mazo, R.; Diaz, D.; Salinesi, C.; Hughes, D. Using Constraint Programming to Manage Configurations in Self-Adaptive Systems. *Computer* **2012**, *45*, 56–63. [CrossRef]

31. Mellado, D.; Fernández-Medina, E.; Piattini, M. Towards Security Requirements Management for Software Product Lines: A Security Domain Requirements Engineering Process. *Comput. Stand. Interfaces* **2008**, *30*, 361–371. [CrossRef]
32. Mellado, D.; Fernández-Medina, E.; Piattini, M. Security Requirements Management in Software Product Line Engineering. In Proceedings of the International Conference, ICETE 2008, Porto, Portugal, 26–29 July 2008.
33. Mellado, D.; Mouratidis, H.; Fernández-Medina, E. Secure Tropos Framework for Software Product Lines Requirements Engineering. *Comput. Stand. Interfaces* **2014**, *36*, 711–722. [CrossRef]
34. Sion, L.; Landuyt, D.; Yskout, K.; Joosen, W. Towards systematically addressing security variability in software product lines. In Proceedings of the 20th International Systems and Software Product Line Conference, Beijing, China, 16–23 September 2016; pp. 342–343.
35. Fagri, T.; Hallsteinsen, S. A Software Product Line Reference Architecture for Security. In *Software Product Lines: Research Issues in Engineering and Management*; Springer: Berlin/Heidelberg, Germany, 2006; pp. 275–326.
36. Arciniegas, J.; Dueñas, J.; Ruiz, J.; Cerón, R.; Bermejo, J.; Oltra, M. Architecture Reasoning for Supporting Product Line Evolution: An Example on Security. In *Software Product Lines: Research Issues in Engineering and Management*; Springer: Berlin/Heidelberg, Germany, 2006; pp. 327–372.
37. Peldszus, S.; Strüber, D.; Jürjens, J. Model-Based Security Analysis of Feature-Oriented Software Product Lines. In Proceedings of the 17th International Conference on Generative Programming: Concepts and Experiences (GPCE '18) ACM SIGPLAN, Boston, MA, USA, 5–6 November 2018.
38. Mauro, M.D.; Sarno, C.D. Improving SIEM capabilities through an enhanced probe for encrypted Skype traffic detection. *J. Inf. Secur. Appl.* **2018**, *38*, 85–95. [CrossRef]
39. Zolanvari, M.; Teixeira, M.A.; Gupta, L.; Khan, K.M.; Jain, R. Machine Learning-Based Network Vulnerability Analysis of Industrial Internet of Things. *IEEE Internet Things J.* **2019**, *6*, 6822–6834. [CrossRef]
40. Mellado, D.; Fernández-Medina, E.; Piattini, M. Security requirements engineering framework for software product lines. *Inf. Softw. Technol.* **2010**, *52*, 1094–1117. [CrossRef]
41. Mohsin, M.; Anwar, Z.; Zaman, F.; Al-Shaer, E. IoTChecker: A data-driven framework for security analytics of Internet of Things configurations. *Comput. Secur.* **2017**, *70*, 199–223. [CrossRef]
42. Acher, M.; Collet, P.; Lahire, P.; France, R.B. FAMILIAR: A domain-specific language for large scale management of feature models. *Sci. Comput. Program.* **2013**, *6*, 657–681. [CrossRef]
43. Gears. Available online: https://biglever.com/solution/gears/ (accessed on 6 December 2019).
44. Apel, S.; Leich, T.; Rosenmüller, M.; Saake, G. FeatureC++: On the Symbiosis of Feature-Oriented and Aspect-Oriented Programming. In *Generative Programming and Component Engineering*; Glück, R., Lowry, M., Eds.; Springer: Berlin.Heidelberg, Germany, 2005; pp. 125–140.
45. pure::variants. Available online: https://www.pure-systems.com/ (accessed on 6 December 2019).
46. Mendonca, M.; Branco, M.; Cowan, D. S.P.L.O.T.: SoftwareProduct Lines Online Tools. In Proceedings of the 24th ACM SIGPLAN ConferenceCompanion on Object Oriented Programming Systems Languages and Applications(OOPSLA '09), Orlando, FL, USA, 25–29 October 2009; pp. 761–762.
47. Mazo, R.; Muñoz-Fernández, J.C.; Rincón, L.; Salinesi, C.; Tamura, G. VariaMos: An extensible tool for engineering (dynamic) product lines. In Proceedings of the 19th International Conference on Software Product Line, SPLC 2015, Nashville, TN, USA, 20–24 July 2015; pp. 374–379.
48. Anna, S.; Christian, B.; Georg, R. Glencoe: A Toolfor Specification, Visualization and Formal Analysis of Product Lines. In *Transdisciplinary Engineering Methods for Social Innovation of Industry 4.0*; IOS Press: Amsterdam, The Netherlands, 2018; pp. 665–673.
49. Benavides, D.; Trinidad, P.; Cortés, A.R.; Segura, S. *FaMa*; Springer: Berlin/Heidelberg, Germany, 2013.
50. Constraint Processing. 2003. Available online: https://doi.org/10.1016/b978-1-55860-890-0.x5000-2 (accessed on) 6 Decembre 2019.
51. Cook, S.A. The complexity of theorem-proving procedures. In Proceedings of the Third Annual ACM Symposium on Theory of Computing-STOC '71, Shaker Heights, OH, USA, 3–5 May 1971.
52. Prud'homme, C.; Fages, J.-G.; Lorca, X. Choco Documentation. 2017. Available online: http://www.choco-solver.org (accessed on 6 December 2019).
53. Hickman, K. *The SSL Protocol*; Netscape Communications Corp.: Dulles, VA, USA, 1995.
54. Dierks, T.; Rescorla, E. The TLS Protocol Version 1.2—RFC 5246. 2008.
55. Rescorla, E. The TLS Protocol Version 1.3—RFC 8446. 2018.

56. Open Security Architecture. Available online: http://www.opensecurityarchitecture.org/cms/ (accessed on 6 December 2019).
57. Varela-Vaca, A.J.; Parody, L.; Gasca, R.M.; Gómez-López, M.T. Automatic Verification and Diagnosis of Security Risk Assessments in Business Process Models. *IEEE Access* **2019**, *7*, 26448–26465. [CrossRef]