



UNICA

UNIVERSITÀ
DEGLI STUDI
DI CAGLIARI



Università di Cagliari

UNICA IRIS Institutional Research Information System

This is the Author's *accepted* manuscript version of the following contribution:

Cataldo Basile, Daniele Canavese, Leonardo Regano, Ignazio Pedone and Antonio Lioy, "A model of capabilities of Network Security Functions" in *Proceedings of the 2022 IEEE Conference on Network Softwarization NetSoft 2022 June 27- July 1 2022 - Milan, Italy Network Softwarization Coming of Age: New Challenges and Opportunities*, pp. 474-479.

© 2022 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.

The publisher's version is available at:

<http://dx.doi.org/10.1109/NetSoft54395.2022.9844057>

When citing, please refer to the published version.

A model of capabilities of Network Security Functions

Cataldo Basile, Daniele Canavese, Leonardo Regano, Ignazio Pedone, and Antonio Lioy
Dip. Automatica e Informatica, Politecnico di Torino, Torino, Italy
name.surname@polito.it

Abstract—This paper presents a formal model of the features, named security capabilities, offered by the controls used for enforcing security policies in computer networks. It has been designed to support policy refinement and policy translation and address useful, practical tasks in a vendor-independent manner. The model adopts state-of-the-art design patterns and has been designed to be extensible. The model describes the actions that the controls can perform (e.g. deny packets or encrypt flows), the conditions to select on what to apply the actions, how to compose valid configuration rules from them, and how to build configurations from rules. It proved effective to model filtering controls and iptables.

Index Terms—software networks, security controls, network security functions

I. INTRODUCTION

The complexity of the security management of computer networks demands new solutions. The networking field's progress has lead to software networks, which are now the most convenient way to manage large and sophisticated information systems. Nonetheless, parallel simplification of the management of the security requirements' enforcement did not follow this progress.

The policy-based network and security management, proposed almost twenty years ago [1], promised a level of innovation and automation that has not been yet reached in practice because several open issues remain unsolved. One of these issues is the *policy refinement*. Several works have proposed refinement models, but they are either very limited in scope [2], [3] or did not smoothly extend to all the types of security controls [4], [5]. Further progress is needed to allow leaving the research prototypes and enter the real world.

Finding a solution for the policy refinement strongly depends on the possibility to formally model what security controls can do in terms of security policy enforcement, that is, their *security capabilities*. Indeed, security administrators may evaluate the features owned by security controls and learn how to configure them to implement corporate security policies. This intelligence is not yet available to machines. Coupled with precise models of security policies (and security intents) and models of the network landscape, a formal security capabilities model will enable machines to perform several important decision processes. For instance, capabilities can be used to select among devices that may look equivalent at “marketing level” or choose the best security controls to enforce a given policy for a specific portion of the network.

This paper's contribution is a formal model of the security capabilities that can be used to formally represent the features offered by the controls used for enforcing security in corporate networks. These features include the actions that a security control can enforce (e.g. deny forwarding or encrypt the traffic) and how it can be configured to select the entities on which to enforce actions (e.g. the conditions exposed by its configuration language that allow selecting the traffic of interest). The model has been designed following state-of-the-art design patterns that have been selected to allow further operations: generating abstract languages for configuring a control from the description of its capabilities and automatically translating abstract configurations into the control-specific configuration settings using a model-driven approach. The model is also a market enabler, as it allows the performing of typical security management tasks, like selecting the best devices to buy and use in a vendor-independent manner. It builds on existing literature works and has been conceived as an extension of the initiatives in the I2NSF (Interface to Network Security Functions) IETF Working Group. Since the scope of the work is about software networks, in the rest of the paper, security controls will also be named *Network Security Functions* (NSFs).

This paper is structured as follows. Section II provides an overview of the application domains of our research, deriving a set of requisites for our model. Section III provides an overview of some recent works related to our research. Section IV describes in detail our model, while Section V validates it providing an overview of a concrete model instance describing the iptables packet filtering firewall. Finally, Section VI draws conclusions, providing also some future directions for our research.

II. MOTIVATING EXAMPLE

This research started from the analysis of practical tasks that need to be addressed by security administrators, and demanding a solution that minimizes efforts and reduces the human errors that plaguing the security [6]. The rapidity of management operation possible with software networks exacerbates the scenario.

Cybersecurity market. When protecting a network, security administrators and network designers, who know the security requirements they want to enforce, have to deal with the specifications of security controls that are often difficult to

compare. Indeed, vendors tend to re-brand old ideas, sell incremental updates as huge advancements, and use buzzwords and trending topics to attract customers, the usual marketing noise. However, in cybersecurity, we need precision. Therefore a model of capabilities to formally (or at least very precisely) identify security features has been considered very welcome also in the I2NSF WG¹.

Vendors' dependency. If one of the corporate security controls breaks or a device performance is not deemed adequate to cover the increased needs, it is usually a safer bet to buy the same device or a more powerful one from the same vendor, even if a more secure, more reliable, or cheaper product is available on the market. Indeed, translating the security policy in a different language or integrating different management environments is an additional risk that overwhelmed people cannot afford. A capability model may allow comparing controls and identify all the alternatives, regardless of the vendor. It may also allow building a tool that receives a configuration file and determines the controls that can enforce it.

Refinement. Refining policies is a complex task that requires precise modelling of the semantics of the high-level policies and the security controls' capabilities. Automated refinement is the task that mimics security administrators when interpreting high-level directives written in natural language or legalese from requirements' documents, select the controls to use and, in the end, write their configuration settings. A capability model describes the entities (e.g. messages or flows) a security control can select and the actions it can enforce on them (e.g. forward or encrypt payload).

Translation. Once it is understood, through manual or automatic refinement, which features must be enabled to implement a security policy, it would be desirable to generate the configuration settings to configure the devices automatically. The capability model allows mapping the abstract concepts into low-level settings written in the control-specific language.

III. RELATED WORKS

The I2NSF WG [7] has presented an information model, namely *CapIM*, to express the security capabilities of NSFs as a part of an architecture to provide standard interfaces for the efficient control and management of NSFs [8]. This model is composed of two classes (*NSF* and *SecurityCapability*), which are abstractions of a Network Security Function and a generic security capability, and an association class (*HasSecurityCapabilityDetail*), to link them according to the *decorator pattern* [9]. The models presented in this paper extends the decorator pattern from the I2NSF model to support refinement and translation, and provide a model for describing filtering devices.

Apart from the *CapIM*, no dedicated modelling effort can be found in literature. Some past research leveraged a representation of NSF capabilities to implement policy-based security functions and frameworks that could fit real use case scenarios. Hyun et al. aim at designing and developing an

I2NSF-compliant architecture [10]. Moreover, they propose an enhancement of this architecture using the Software-defined Networking (SDN) paradigm and discuss the open research challenges for I2NSF and its evolution. Basile et al. focus instead on automatic enforcement of security policies in a Network Functions Virtualisation (NFV) scenario [5], where security policy enforcement dynamically adapts to network changes. The approach is based on a refinement model which translates high-level security requirements to low-level security configurations for the NSFs, and on optimization models that allow the proper selection of the NSFs. Those models rely on a formal description of the NSF capabilities. Molina et al. [11] present a policy-based security framework for dynamic management of Authentication, Authorization, Accounting (AAA), and channel protection, leveraging virtual security functions in IoT networks and a policy refinement process based on a formal model of NSF security capabilities provided by the EC-funded SECURED [12] project.

IV. THE MODEL

When designing the capability model, security controls have not been organised according to the market categories, namely firewalls, VPNs, proxies. Our experience in past projects proved that this approach, which looks very promising and straightforward, is, in the end, inefficient, does not scale well, and does not allow specifying fine details about the security controls for decision making. For instance, it is not possible answering questions like “how squid compares against ModSecurity?” or “Is a device of a vendor X able to implement the configuration I have deployed in the device of vendor Y?” [12]. Administrators can start from the high-level categorisation of the devices and compensate for the lack of details by reading the data sheets and the configuration manual where all the available conditions are illustrated. An automatic system cannot do so, and manually writing ad-hoc code in refinement systems to individually support each device and its operations is not an acceptable solution.

Therefore, the capabilities have been structured in six areas, according to the generic features they offer to specify configurations, as already proposed in the I2NSF WG draft [7]. The first four areas concern the creation of a rule. The *conditions*, describe the features that are available at the NSF to identify the element (e.g. the traffic) to which to enforce (e.g., conditions on the IP addresses or regex on the HTTP MIME type). The *actions* are the operations that a NSF can perform on individual elements (e.g. deny packets or encrypt flows). Finally, the model describes the *events*, that, for specific classes of controls, allow triggering the rules' evaluation. The last field is related to the condition clause. It may express that a rule is activated when all the conditions evaluate to true (DNF, Disjunctive Normal Form), or just one condition is satisfied (CNF, Conjunctive Normal Form). The next two areas are related with the building of the desired security policy from individual rules. The *resolution strategies* describe how the control behaves when more than one rule applies to the same entity. For instance, the first matching rule applies the rule's

¹See <https://datatracker.ietf.org/wg/i2nsf/about/>

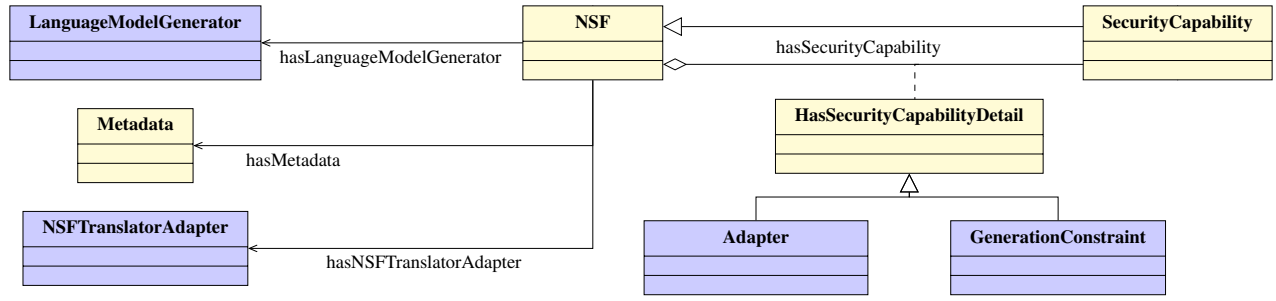


Fig. 1. Proposed Information Model.

action at the highest priority when more rules apply. Support for *default actions* allows determining how the control behaves when no rule matches the element. For instance, it is important to know if the control allows specifying the “deny all” default action, as implemented by most firewalls, or not.

A. Model requirements and high-level design principles

Together with the practical aspects presented in Section II, the following requirements have been considered for the design of the capability model presented here. Starting from the description of the capabilities owned by an NSF, the model must allow generating an abstract language that can be used to specify valid configurations for the described NSF. This translation must not rely on the NSF specific code.

Given the security policy (i.e. configuration) of an NSF written in its abstract language, the model must allow translating that configuration into the NSF-specific configuration settings. This translation must not rely on the NSF specific code.

The model must ease the description of the capabilities by supporting templates, allowing merging them and individually adding and subtracting features from a description. For instance, the description of the capabilities of iptables can be determined by extending the description of a generic packet filter and a generic filter on TCP states, i.e. by adding more features to these generic templates. Furthermore, the v2.0 of an NSF can be obtained by the description of the v1.0 of the NSF by adding or removing individual capabilities.

To this purpose, this research proposes a meta-model describing the high-level concepts and how they relate to meet the requirements presented above, which is then extended in domain-specific models. Section V describes the filtering model created to support the iptables filtering features.

The following design principles have been adopted to meet the requirements. The *decorator pattern* [9], inherited from the original work of Strassner in the I2NSF draft, has been used to associate NSF with their security features. This pattern is also useful to provide additional information (as an association class) when capabilities are “attached” to the decorated objects and are important to define how capabilities map to device-specific settings. Moreover, it may be used to add capabilities both statically, when an NSF is described, and dynamically, during network operations. The adapter pattern has been employed to allow NSFs [9], which may have

different capabilities and certainly have different languages, thus exposing different interfaces, to be uniformly accessed to perform the translation into the low-level settings. This pattern allows employing a unique implementation of the translator without the need for NSF-specific code.

Finally, this work aims to minimise the semantics that a tool (e.g. refinement, translation) has to know to work with NSFs capabilities. The only hardcoded semantics is the one associated with the previously mentioned 6-tuple. A refinement tool must only be aware of the six categories of capabilities that have been inherited from the I2NSF WG approach.

B. The meta-model

Fig. 1 presents the UML classes of the meta-model, which is composed of the following entities:

- NSF is the class used to describe an NSF; it can be seen as a container of decorations;
- SecurityCapability extends NSF and represents security features that NSFs may own;
- HasSecurityCapabilityDetail allows adding further details when a security capability is used to decorate an NSF instance;
- Adapter inherits from HasSecurityCapabilityDetail and provides information on how to translate a specific feature to be used by an NSF. For instance, this class is used to explain that the source port condition capability can be specified by the `--sport` in iptables;
- GenerationConstraint also inherits from HasSecurityCapabilityDetail and adds details for the generation of the NSF specific abstract configuration (see Section IV-D);
- Metadata abstracts all the possible data that could be associated to an NSF.

Moreover, LanguageModelGenerator is the singleton that implements the features for the generation of the abstract configuration language from an NSF instance and its decorations. Finally, NSFTranslatorAdapter is the singleton that implements the translation of an abstract configuration into an instantiated NSF’s low-level language, based on the capabilities and translation constraints.

These entities are bound by the following relationships:

- `hasMetadata` is a many-to-many relationship that links an NSF to all its Metadata;
- `hasLanguageModelGenerator` is the generic one-to-many relationship binding the NSF to the abstract language generation tool;
- `hasNSFTranslatorAdapter` is the generic one-to-many relationship that binds the NSF to the tool responsible for the generation of the low-level configuration settings, written in the NSF-specific language, from a policy written using the NSF abstract language;
- the `hasSecurityCapability` many-to-many aggregation allows the decoration of NSF with `SecurityCapability`.

Any NSF can be described and decorated with capabilities. Moreover, an NSF can be interpreted as a set of security capabilities, which can be included in another NSF, allowing a template-based approach for NSF description.

The meta-model can be extended with models. These models can be considered like catalogues of security capabilities. As anticipated, following the I2NSF standard, capabilities are categorised according to the 6-tuple, sketched in Fig. 2.

C. The filtering model

This paper also introduces a model created to describe filtering NSFs with a focus on iptables. It mainly describes `ConditionCapability` and `ActionCapability` classes, obtained by inheritance to hierarchies of subclasses of the 6-tuple (see the GitHub repository). For example, the model subclasses the `ActionCapability` to obtain the `DenyAction` and `AllowAction`. To model the feature that allows packet filters to recognise and make decisions on the protocol type IP field, the filtering model uses the `IpProtocolTypeCapability` class. It inherits from a broader `Ipv4Capability` class. The latter in turn inherits from the top-level `ConditionCapability` class.

The model also contains the possible attributes for each `SecurityCapability`. For example, the `IpProtocolTypeCapability` class provides the `protocolType` attribute of type `protocolType`, an union of a `protocolTypeNumber` attribute of type `integer` and of a `protocolTypeName` of type `ProtocolTypeEnumeration` (an enumeration of IANA registered protocol names). Thus, we can refer to a protocol using either its name or number.

D. Language constraint

The problem of generating an NSF's abstract language from its capabilities has been dealt with using a model-driven approach. By defining all the attributes for each `SecurityCapability`, the names and types involved in the generation of `SecurityCapability` instances are standardised, i.e. there is a default data type. Hence, different NSF instances, having common security capabilities, will employ the same abstract language constructs for such common capabilities. However, the model provides the class

`LanguageConstraint`, a subclass of the I2NSF association class `HasSecurityCapabilityDetail`, which enables editing the default semantic of the NSF, by customising the values assignable to model parameters (e.g. integers or user-defined enumerations). By interpreting these directives, the `LanguageModelGenerator` can generate a valid abstract language with a model-driven approach, i.e. without resorting to code. That is, `GenerationConstraint` instances are used to modify the data type associated with these decorations and the operations allowed on them. For instance, constraints can be used for specifying that an NSF can only use a subset of the protocol values in the `protocolType` data type or that only protocol numbers are allowed. When generating the abstract language, these constraints allow modelling all the data types correctly. They may also indicate that ranges or set of protocols can be specified instead of individual protocol values.

E. Adapter and translations

The model uses the `Adapter` class in the `HasSecurityCapabilityDetail` to enable the model-driven translation of policies written in the abstract NSF language into the ones written in the NSF-specific language. The NSF Translator, starting from the target policy instance and the translation rules defined in the `Adapter` class instance, can generate the policy in the low-level format of the target NSF. The `Adapter` class is modeled as a subclass of the I2NSF `HasSecurityCapabilityDetail`. As for the aforementioned `LanguageConstraint`, in this way, we bind the `Adapter` instance to its security capability and NSF. This class's main objective is to enable the description of syntax and semantics of the low-level language when the capability is used for decorating an NSF.

When a security capability is used to decorate a NSF, `Adapter` is used to provide indications on how to represent this capability in the NSF-specific language. It is also possible to define the command names, the attribute separators, and the dependencies by means of the following attributes:

- 1) `commandName`: name of the command in the low-level NSF language;
- 2) `bodyConcatenator`: specifies the value separator (e.g. the comma in lists) and uses the additional `preVariable` and `postVariable` attributes to define the need for surrounding characters (e.g. the '[');
- 3) `dependency`: used to define the dependencies of a security capability from other ones; the attribute `presenceOfCapability` imposes the addition of another command from a specific security capability that were not explicitly present in the policy (e.g. in iptables `-p tcp,udp` is needed if a source port condition needs to be specified with the `--sport 80` is used), `absenceOfCapability` indicates that the option is not possible when another capability is specified, while `presenceOfValue` and `absenceOfValue` impose the presence or absence of specific values;

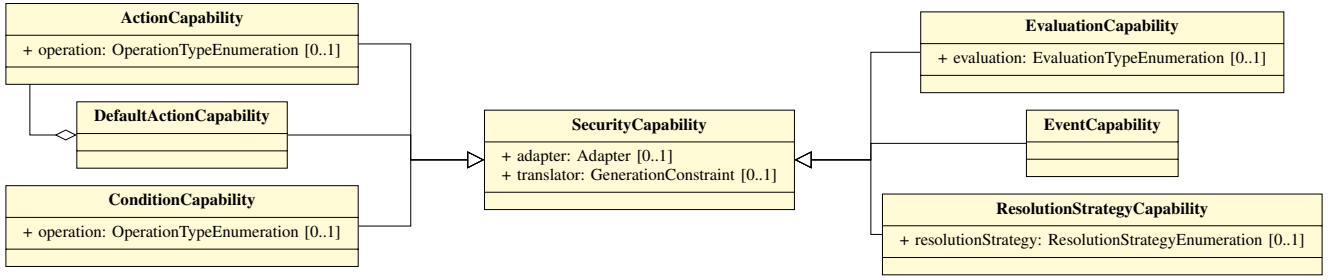


Fig. 2. 6-tuple of capabilities for an NSF.

- 4) `internalClauseConcatenator`: specifies the value used to separate the command name from the values (e.g. some tool want ':' or '=');
- 5) `clauseConcatenator`: specifies the value used to separate commands (e.g. ';').

V. VALIDATION: IPTABLES

`iptables` is an interesting validation of our capability model that proves the flexibility of our approach. The validation has been performed by representing the UML schema via an XMLSchema to use eXtensible Markup Language (XML) manipulation and validation tools. Since XML does not support the concept of association class, the classes `Language` and `Adapter` are expressed with the XML tags `language` and `adapter` contained in the tag `securityCapability` related to the `SecurityCapability`.

A. High-level policies

The `Language` class is used to define the high-level policy language related to a capability and it is implemented via the `language` tag in the `securityCapability` tag. As an example, Listing 1 reports the representation in XML of the automatically generated `IpProtocolTypeCondition` class obtained from the `IpProtocolTypeCapability` class (subclass of `ConditionCapability`, which is used to define conditions on the protocol type in a filtering rule. The generated high-level policy language defines a list of supported protocols (e.g. HOPOPT, ICMP, ...) and their equivalent integer numbers, even if it is only reported a very small snippet. Listing 2 shows a rule written in the generated abstract `iptables` language. Six capabilities form a new filtering rule to the `iptables`' FORWARD chain in the `filter` table to accept all the TCP traffic from the ports in the range 80–90 to the IP addresses in the subnet 192.168.1.0/255.255.255.0.

B. NSF language

The `Adapter` classes have been specified to describe how to map capabilities into valid `iptables` commands with the `adapter` tag, as shown in Listing 3. The snippet used when decorating the `iptables` instance of the NSF class with the `AppendRuleCapability` capability shows that it translates into the `-A` command and that it also needs to be inserted into the `TableCapability`. Finally, using the `iptables`' adapters, the policy in Listing 2 can be translated as the

Listing 1. The `Language` class implementation in XML for `iptables`.

```
<securityCapability
  xsi:type="p:IpProtocolTypeCondition">
  ...
  <language>
    <enumerationName>
      ProtocolTypeEnumeration
    </enumerationName>
    <modifyDefault>
    <addNewValue>
      <newValue>hopopt</newValue>
      <integerMatching>0</integerMatching>
    </addNewValue>
    <addNewValue>
      <newValue>icmp</newValue>
      <integerMatching>1</integerMatching>
    </addNewValue>
    ...
  </modifyDefault>
</language>
</securityCapability
  xsi:type="p:IpProtocolTypeCondition">
```

command shown in Listing 4. The details of the translation are customized by the implementation of the translator that receives an NSF filtering rule.

C. Validating requirements

The proposed model fulfils the requirements laid out in Section II. First, the model has enough expressiveness to formally model the capabilities of complex NSFs, as reported in Section V-B. Moreover, additional NSFs can be described by providing models that include the missing capabilities, as done with the `iptables` case.

The model can be useful in a manual refinement process. Instead of writing policies directly in the NSF's target language, it is more convenient to have them represented in the abstract language. In case the NSF is substituted with an equivalent one, the new settings can be easily obtained using the translation tool. Indeed, the latter can produce a low-level configuration of the `iptables` firewall, starting from high-level policies expressed in the abstract `iptables`-specific language. Moreover, the abstract `iptables` language can represent manually written `iptables` configurations.

VI. CONCLUSIONS

This paper presented a formal security capability model that can describe the features that can be configured in security

Listing 2. A filtering rule in XML in the abstract iptables language.

```
<securityCapability
  xsi:type="TableCapability">
  <table>filter</table>
</securityCapability>
<securityCapability
  xsi:type="AppendRuleCapability">
  <chain>FORWARD</chain>
</securityCapability>
<securityCapability
  xsi:type="IpDestinationAddressCondition">
  <ipAddress>
    <address>192.168.1.0</address>
    <mask>255.255.255.0</mask>
  </ipAddress>
</securityCapability>
<securityCapability
  xsi:type="IpProtocolTypeCondition">
  <protocolType>
    <protocolTypeName>TCP</protocolTypeName>
  </protocolType>
</securityCapability>
<securityCapability
  xsi:type="SourcePortCondition">
  <port>
    <startValue>80</startValue>
    <endValue>90</endValue>
  </port>
</securityCapability>
<securityCapability
  xsi:type="FilteringAction">
  <value>ACCEPT</value>
</securityCapability>
```

Listing 3. The Adapter class used for the AppendRuleCapability.

```
<securityCapability
  xsi:type="p:AppendRuleCapability">
  ...
  <adapter>
    <commandName>
      <realCommandName>-A</realCommandName>
    </commandName>
    <dependency>
      <presenceOfCapability>
        TableCapability
      </presenceOfCapability>
    </dependency>
  </adapter>
</securityCapability>
```

controls. The model also presents the foundation for the automatic generation of abstract policy languages for the described controls and support for the automatic translation of abstract policies into low-level configuration settings written in the NSF-specific language.

Having at disposal a well-constructed capability model is the first critical step to perform a series of (hopefully automatic) processes such as policy refinement, translation and enforcement. An implementation based on XML and XML Schema has been developed and validated on a real-world tool, ipsec/netfilter, to prove the model's flexibility and effectiveness. Indeed, it has been validated as fully representing all the intricacies of iptables' parameters and modules.

The model presented in this paper represents only the beginning of a powerful tool that can be used in the real world.

Listing 4. Translation of the policy reported in Listing 2.

```
iptables -t filter -A FORWARD
-d 192.168.1.0/255.255.255.0 -p TCP
--sport 80:90 -j ACCEPT
```

Plans for extending its functionalities include adding new capabilities paired with their adapters and generation constraints for new security controls as VPN terminators, layer7 filters, and WAFs (Web Application Firewalls). We are aware that several vendors produce incomplete documentation, and they may not be interested in producing very precise documentation and descriptions of the features of their components. To this purpose, we are continuing the effort on the standardization of this model in the IETF WGs. Moreover, to also support the needs from the I2NSF WG, the capability model will describe features that explain how to allocate/deallocate NSFs in a virtual environment and how to reach them actually to push the generated configurations.

ACKNOWLEDGMENTS

This work has received funding from the European Union's Horizon 2020 research and innovation programme, under projects PALANTIR (Grant Agreement No. 883335) and FISHY (Grant Agreement no. 952644).

REFERENCES

- [1] D. C. Verma, "Simplifying network administration using policy-based management," *IEEE Network*, vol. 16, no. 2, pp. 20–26, 2002.
- [2] A. K. Bandara, E. C. Lupu, J. Moffett, and A. Russo, "A goal-based approach to policy refinement," in *Proceedings. Fifth IEEE International Workshop on Policies for Distributed Systems and Networks, 2004. POLICY 2004.*, 2004, pp. 229–239.
- [3] A. K. Bandara, E. C. Lupu, A. Russo, N. Dulay, M. Sloman, P. Flegkas, M. Charalambides, and G. Pavlou, "Policy refinement for ip differentiated services quality of service management," *IEEE Transactions on Network and Service Management*, vol. 3, no. 2, pp. 2–13, 2006.
- [4] C. Basile, A. Lioy, C. Pitscheider, F. Valenza, and M. Vallini, "A novel approach for integrating security policy enforcement with dynamic network virtualization," in *Proceedings of the 2015 1st IEEE Conference on Network Softwarization (NetSoft)*. IEEE, 2015, pp. 1–5.
- [5] C. Basile, F. Valenza, A. Lioy, D. R. Lopez, and A. P. Perales, "Adding support for automatic enforcement of security policies in nfv networks," *IEEE/ACM Transactions on Networking*, vol. 27, pp. 707–720, 2019.
- [6] M. Evans, Y. He, L. Maglaras, I. Yevseyeva, and H. Janicke, "Evaluating information security core human error causes (is-heck) technique in public sector and comparison with the private sector," *International Journal of Medical Informatics*, vol. 127, pp. 109–119, 2019. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S1386505618310955>
- [7] L. Xia, J. Strassner, C. Basile, and D. Lopez, "Information model of nsfs capabilities," *draft-xibassnez-i2nsf-capability-05*, 2019.
- [8] D. Lopez, E. Lopez, L. Dunbar, J. Strassner, and R. Kumar, "Framework for Interface to Network Security Functions," RFC 8329, Feb. 2018. [Online]. Available: <https://rfc-editor.org/rfc/rfc8329.txt>
- [9] E. Gamma, Ed., *Design patterns*, 38th ed., ser. Addison-Wesley professional computing series. Boston [u.a.]: Addison-Wesley, 2010, literaturverz. S. 375 - 381.
- [10] S. Hyun, J. Kim, H. Kim, J. Jeong, S. Hares, L. Dunbar, and A. Farrel, "Interface to network security functions for cloud-based security services," *IEEE Communications Magazine*, vol. 56, no. 1, pp. 171–178, 2018.
- [11] A. Molina Zarca, D. Garcia-Carrillo, J. Bernal Bernabe, J. Ortiz, R. Marin-Perez, and A. Skarmeta, "Enabling virtual aaa management in sdn-based iot networks," *Sensors*, vol. 19, no. 2, p. 295, 2019.
- [12] "The secured project." [Online]. Available: <https://www.secured-fp7.eu>