Solution-aware Data Flow Diagrams for Security Threat Modeling

Laurens Sion, Koen Yskout, Dimitri Van Landuyt, Wouter Joosen
imec-DistriNet
KU Leuven
Heverlee, Belgium
firstname.lastname@cs.kuleuven.be

ABSTRACT

Threat modeling refers to a number of systematic approaches for eliciting security and privacy threats. Data Flow Diagrams (DFDs) are the main input for threat modeling techniques such as Microsoft STRIDE or LINDDUN. They represent system-level abstractions that lack any architectural knowledge on existing security solutions.

However, this is not how software is built in practice: there are often previously-made security- and privacy-relevant decisions that originate from the technological context or domain, reuse, or external dependencies. Not taking these into account leads to the enumeration of many non-applicable threats during threat modeling. While recording the effect of these decisions on individual elements can provide some relief, the lack of a proper first-class representation causes conflicts when modifying the architecture and inhibits traceability between effect and decision.

In this paper, we enrich Data Flow Diagrams with security solution elements, which are taken into account during threat elicitation. Our modeling approach is supported by a proof-of-concept implementation of a threat modeling framework and validated in the context of a STRIDE analysis of an industrial video conferencing solution that is based on WebRTC. The presented DFD enrichments are a key enabler for future efforts towards dynamic and continuous threat modeling.

CCS CONCEPTS

• **Security and privacy → Software security engineering**; • **Software and its engineering → Data flow architectures**; *Abstraction, modeling and modularity*;

KEYWORDS

Security, Design, Threats, Threat Modeling, DFD

ACM Reference Format:
Laurens Sion, Koen Yskout, Dimitri Van Landuyt, Wouter Joosen. 2018. Solution-aware Data Flow Diagrams for Security Threat Modeling. In *SAC 2018: Symposium on Applied Computing , April 9–13, 2018, Pau, France*. ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3167132.3167285

SAC 2018, April 9–13, 2018, Pau, France

1 INTRODUCTION

The principles of Security by Design (SbD) and Privacy by Design (PbD) are being increasingly recognized as essential to pro-actively and effectively deal with design flaws that may compromise security or privacy [1] early in the development life cycle. For example, the EU-wide General Data Protection Regulation (GDPR) [9] dictates privacy by design and data protection by design for any system or service that involves processing personal data. Systematic threat modeling approaches strongly contribute to the implementation of such principles, given their methodical and rigorous nature, and are therefore a cornerstone of architectural risk analysis [20], the secure development life cycle (SDLC) [15], and security maturity models such as OpenSAMM [22], for example.

Threat modeling methodologies, such as STRIDE [14, 15] or its privacy counterpart LINDDUN [7, 34], start from Data Flow Diagrams (DFDs) [6], which are system-level abstractions that represent the external entities interacting with the system, processes, data flows, and data stores. Based on a DFD, a systematic iteration over all model elements yields potential security or privacy threats that need to be assessed. In further steps, these threats are documented and prioritized, and subsequently guide the process of determining the appropriate security solutions to mitigate them.

Since DFDs represent the system using just four different model element types, they are in fact architectural views [4, 16] that are relatively easy to create and comprehend. Furthermore, a DFD-based view of the system under design is strongly suited for highlighting the security-relevant parts of the system (which often is centered around data items, and the locations where they are processed and stored). DFDs are used so frequently for this purpose that they are sometimes called '*threat model diagrams*' [26]. However, a key problem related to using them as the main input for security threat modeling is that they do not support expressing any security-related architectural decisions, namely the applied security measures, in a structured way.

This is problematic because in practice, some security decisions and constraints are already known early on, as a consequence of trends and practices such as agile software development, continuous integration, and the common practice of adopting and integrating third-party solutions (platforms, libraries, middleware, services, etc.). As already noted by Berger et al. [5], not being able to explicitly model these decisions (e.g., the use of SSL/TLS for confidentiality and integrity over a certain channel) prohibits reasoning about them during threat modeling, which eventually leads to duplicated or wasted effort, or worse, threats that are overlooked. To further underline the importance of making these decisions explicit, we bring to attention that OWASP's Software Assurance

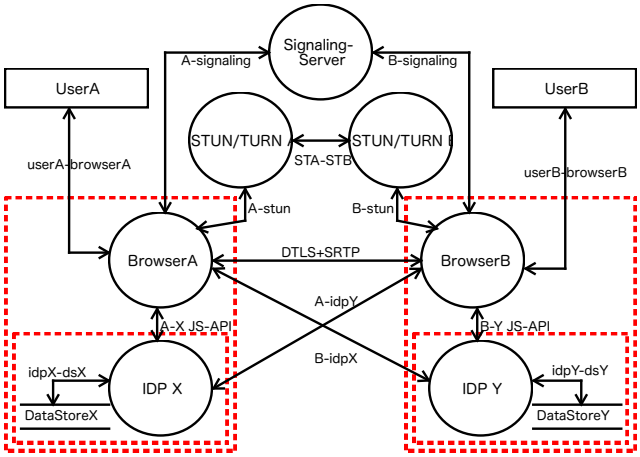

Figure 1: WebRTC Data Flow Diagram

Maturity Model (OpenSAMM) [22] requires annotating the threat model with 'compensating controls' (which include technical security measures), as well as keeping the controls up to date in every development cycle, as an explicit activity to attain the highest maturity (level 3) for the *Threat Assessment* practice.

Existing approaches [21] circumvent this problem in a pragmatic manner, by adding properties to the DFD elements that capture some of the *effects* of the chosen solutions, rather than the *solutions* themselves. In this paper, we argue why this is insufficient, and we present improved modeling support. More precisely, we (i) identify and discuss *four desired qualities* that emerge from the ability to express security solutions, related to semantics, traceability, separation of concerns, and dynamic and continuous threat assessment; (ii) present a meta-model that supports these qualities by enriching DFDs with security solutions, which are taken into account during the execution of threat modeling activities and also enables exploratory change impact analysis or *what-if* analysis; (iii) provide a *proof-of-concept* implementation of the proposed meta-model, which serves as an enabler for an improved threat modeling framework; and (iv) *validate* the resulting framework in the context of a STRIDE analysis of a WebRTC reference architecture.

This paper is structured as follows. Section 2 elaborates on the motivating case. Section 3 puts forward four qualities and analyses to what extent they are supported by existing solutions. Section 4 presents our meta-model which enables the incorporation of security solutions, followed by a functional validation and evaluation with the qualities in Section 5. Related work is discussed in Section 6, and the paper concludes in Section 7.

2 MOTIVATION

To illustrate the concepts in this paper, we use the example of a WebRTC-based multipoint collaboration system. Because the architecture is based on WebRTC, extensive documentation is available [13]. To evaluate the security of this reference architecture, a systematic threat analysis is performed using the STRIDE methodology [26].

Conducting such a threat analysis starts with the construction of a DFD model, such as the one shown in Figure 1. A DFD is a simple and accessible form of an architectural view, which focuses on the principal entities of the system and the data they communicate among each other. For threat analysis purposes, trust boundaries are typically added to the DFD as well. The example DFD contains 7 processes (circles), 2 external entities (rectangles), 2 data stores (parallel lines), 4 trust border boundaries (the dashed lines) and 28 data flows (bidirectional flows are actually two separate flows).

To perform a STRIDE analysis on such a model, the most well-known and readily available tool is the Microsoft Threat Modeling Tool 2016 (TMT) [21].[1] This tool comes with a catalog of 41 generic threat templates, specified as in Figure 3, which shows the template for tampering threats due to a lack of input validation. These threat templates can use the parameters *source*, *target*, and *flow*, which are instantiated for each individual (directed) data flow in the model to yield concrete threats that require inspection. In the WebRTC case, this would yield 236 threats, one of which would be "Potential Lack of Input Validation for *BrowserB*", generated by matching the "*DTLS+SRTP*" flow from *BrowserA* to *BrowserB*.

This tool already offers some basic support for augmenting DFDs with security-relevant information. It does this by attaching a customizable set of enum-like properties to the DFD elements, such as "*provides confidentiality*", which can take the value 'Yes' or 'No', for a data flow. Furthermore, it introduces some subtypes of existing elements, which constrain some of these properties. For example, the HTTPS data flow subtype constrains the "confidentiality", "integrity", and "destination authentication" properties to 'Yes'. These properties, in conjunction with the exclusion criteria of the threat template, lead to an elimination of 20% of the threats, thereby reducing the remaining effort.

Despite the simplicity of this approach, there are some incentives to include security solutions more explicitly into the model. First, the single decision of using HTTPS leads to, in total, six property constraints on two distinct data flows. Such a scattered representation of this solution can lead to errors being introduced in the model when the solution is instantiated and also makes it difficult to oversee the impact of applying a certain solution.

Furthermore, since security is a moving target, earlier solutions need to constantly be revisited when additional information on their effectiveness becomes available over time. Yet re-evaluating these solutions becomes more complicated when the only information about the applied solutions in a model is a scattered set of properties.

Depending on the context of the system under design, it may also be necessary to conduct auditing or compliance checking. The lack of an explicit representation of the security solutions that determine the security of the system severely hampers such auditing activities.

Finally, security expertise is a scarce resource and relies on an extensive body of security knowledge. Explicit security solutions are a prime candidate to become part of a reusable catalog which can make this body of knowledge easily accessible and reusable.

3 QUALITY ANALYSIS

This section elaborates on the desired qualities of security solution representations in support of threat modeling and discusses to what degree these qualities are supported in state-of-practice tools or state-of-the-art approaches.

[1] Note that, while we focus on the TMT as an example approach using properties, the analysis applies to these approaches in general, regardless of the specific tool.

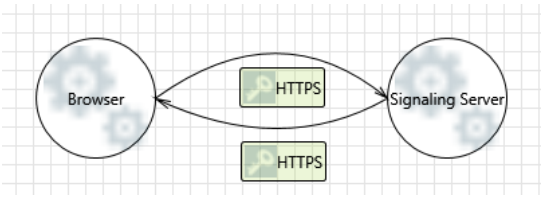

Figure 2: Data flow specialization for representing HTTP over TLS in the Microsoft Threat Modeling Tool [21]

Quality 1. – ***Semantics***
The inclusion of security information in a DFD model should support a first-class representation of security solutions, to enable verifying the correct application of these solutions and prevent interpretation ambiguity.

Security is a text-book example of a cross-cutting quality concern, which explains its impact on multiple elements in the architecture when security solutions are applied. This has several implications for representing them in a model: (i) because of the cross-cutting nature of security, security solution representations should capture which elements are affected by them, (ii) the representations should also capture in what capacity DFD elements take part in a solution, and finally (iii) the representations should facilitate the correct instantiation of these security solutions in a concrete design.

WebRTC Example. The quality is analyzed using a concrete example of the WebRTC Architecture. The communication between the *Browser* and the *SignalingServer* is protected by HTTP over TLS, a very common security solution for protecting web traffic.

Microsoft Threat Modeling Tool (TMT). The TMT offers a data flow specialization, HTTP over TLS (HTTPS), to represent this security solution, shown in Figure 2. This specialization is part of the built-in set of stencils, offered by the TMT. It has the following security properties pre-set: {*provides confidentiality*=true, *provides integrity*=true, *destination authentication*=true}.

Analysis. While *destination authentication* is a valid security property for the data flow from the browser to the server, this is not the case for the return flow. *Destination authentication* on the return flow would imply that the browser is authenticated (to the server), which is by default not the case for HTTPS. Indeed, it is more often not the case, as application-level authentication mechanisms (e.g., username/password) are used much more frequently. A careless instantiation of HTTPS for the return flows can in this case lead to the incorrect elimination of spoofing threats of the browser. This shows that a single specialization with fixed properties is insufficient to capture the semantics of this security solution.

Additionally, the representation of security solutions by adding properties to DFD elements raises more questions and semantical ambiguities, such as: (i) on which element the property should be set (source, flow, or target); (ii) whether a property on an element should be interpreted to hold for all flows connected to that element, only those with an additional property set, or only those that do not have a specific property set; (iii) keeping such interpretations consistent over time; (iv) whether a property on a flow is implemented by the sending or receiving element (or both); and (v) whether all the necessary elements participating in a security solution have the correct properties set (which is critical to ensure that the chosen solution is correctly realized).

Quality 2. – ***Traceability***
There should be traceability from the effects of a security solution to the security solution that caused them, and from a security solution to the threat type(s) it aims to prevent.

A software architecture can be considered as a set of architectural design decisions [17], which turns the management of these decisions into an architectural knowledge management [3] problem. This is no different for security decisions. On the contrary, keeping this information is essential to prevent the introduction of security flaws caused by misinterpretation of earlier security solutions. Capturing architectural decisions is thus essential for bridging the gap between the architecture and its rationale [33].

By offering traceability, this security decisional information can be captured in the model. The traceability that should be provided is two-fold. Firstly, there should be a link from the security effects to the security solutions providing these effects. By preserving this link, the actual security solution that was chosen to achieve these effects remains present in the model. This prevents later modifications from introducing conflicts because the knowledge about the original solution was lost. Secondly, the link between a security solution and the threat type(s) it aims to counter should also be preserved. Keeping this information preserves the rationale as to why a specific security solution was instantiated in the design.

By capturing both the decision information (in the form of instantiated security solutions) and the rationale behind this (in the form of the threat type(s) being countered by that security solution), previous security decisions can be revisited and the resulting documentation provides an excellent resource for compliance checking and auditing the security decisions.

WebRTC Example. The same example from Quality 1 can be revisited. What needs to be captured here is that HTTPS is used to achieve confidentiality and integrity of the dataflows between the *Browser* and *SignalingServer*, as well as ensuring the authenticity of the *SignalingServer* process, and that this security solution was instantiated to prevent (i) *information disclosure* and *tampering* of the data sent over these flows, and (ii) *spoofing* of the *SignalingServer*.

Microsoft Threat Modeling Tool. The property-based approach for adding security information does not support the concept of security solutions as a first-class entity. In the example from Figure 2, the decision could be reconstructed by combining the HTTPS flows, although there is no guarantee the correct flows are combined when reconstructing the decision later on. The only way this knowledge can be captured is in cases where the security solution can be represented with the specialization of a single element or flow.

Analysis. The lack of a first-class representation for security solutions makes it hard or impossible to link security effects to the actual security solution they stem from. The link between the security solution and why (for which threat type(s)) it was instantiated is even more implicit, requiring (in the worst case) a review of the complete threat template catalog to find out which threat type(s) are actually countered by the effects.

Title:
Potential Lack of Input Validation for {*target.*[Name]}
Include:

```
(source is [Generic Process] or
 source is [Generic External Interactor]) and
target is [Generic Process] and
(flow crosses [Generic Trust Line Boundary] or
 flow crosses [Generic Trust Border Boundary])
```

Exclude:

```
flow.[Provides Confidentiality] is 'Yes' and
flow.[Provides Integrity] is 'Yes'
```

Figure 3: Tampering threat type from the Microsoft Threat Modeling Tool [21]

```
MATCH //ommitted
WHERE //ommitted
  AND ANY (d IN flow.data WHERE d.IsConfidential)
  AND NOT flow.IsEncrypted
```

Figure 4: Threat pattern from Berger et al. [5]

Quality 3. – ***Separation of concerns***
The threat type and security solution catalogs should be structured to support their independent evolution, limiting the impact of adding, updating, or removing threat types or security solutions.

The threat type and security solution catalogs are separate concerns and should be structured in way to minimize the impact of changes to these artefacts, since complete isolation is not possible because of the dependency between threat types and security solutions preventing them. Given the dynamic nature of security, the objective of designing a secure system is a moving target, with flaws being discovered and existing (previously thought secure) solutions requiring replacement over time. The artefacts should support this dynamic nature of security, by allowing limited-impact changes to security solutions, ensuring the security solution catalog can remain up to date with changes and insights from the field.

Microsoft Threat Modeling Tool [21], Berger et al. [5]. In the Microsoft Threat Modeling Tool, but also the DFD extension presented by Berger et al. [5], security solution information is embedded in the threat types as exclusion conditions. Examples of these threat type expressions, in the Microsoft TMT and Berger et al. respectively, are displayed in Figures 3 and 4. The *'Exclude'* or *'AND NOT'* expressions in these types explicitly refer to the security properties set at the element.

Analysis. Including the information about which security solution effects prevent a threat type in the threat type catalog, creates a strong dependency from the threat type catalog to the security solution effects. This dependency makes it very difficult to introduce new security solutions, as every new solution requires a pass over all relevant threat type generation expressions to add the exclusion condition(s). This overhead becomes even greater when existing mechanisms are changed, because in those cases every threat type generation expression must be checked to make sure the exclusions correspond with the changes to the security solution.

Quality 4. – ***Dynamic and continuous threat assessment***
Threat modeling activities should support incomplete and lightweight architectural design efforts, continuous evolution, and frequent architectural refactoring of the system under design. Embedding threat modeling activities into agile development processes requires new approaches of dynamic and continuous threat modeling, and co-evolution with the system under design.

Traditional threat modeling approaches are designed to be single shot operations, to be conducted in the early stages of the development life cycle, the results of which to take into account in the early stages of architectural design. This is, however, in stark contrast with the reality of contemporary software development practice, which includes approaches such as continuous evolution/integration and agile software development.

To support this requirement, threat modeling tools should support and be embedded in these more lightweight software development approaches by providing security design assistance, such as (i) raising awareness of security threats that emerge when the architecture evolves, (ii) suggesting potential security solutions to the architect to counter these threats, (iii) providing an assessment of the architectural impact of these potential security solutions (change impact analysis), and (iv) integrating with knowledge bases on security issues that constantly evolve.

Tooling. To our knowledge, no tools exist in the state of the art nor practice that support this quality requirement.

The ameliorate the issues mentioned above, a richer representation for security is needed, so security solutions are first-class citizens in the model.

4 META-MODEL

To improve the state of the art with respect to the four qualities defined above, we propose a new meta-model as a foundation for threat modeling tools. The presented meta-model consists of two parts. The first part serves to capture plain DFDs. The second part (the security meta-model) is an extension of the first part, and introduces the following concepts: (i) threat types and their catalog, (ii) security solutions and their catalog, and (iii) instances of these security solutions to insert in a concrete model.

4.1 DFD Meta-model

Before being able to capture security information, there needs to be an underlying model to which the security information can refer in order to be able to express where and how security solutions are applied and which element(s) they affect.

The DFD notation is a visual and informal notation [30], with multiple variants [6, 12]. There is no standardized meta-model associated with DFDs to precisely specify the elements and their relations, as there is with more recent modeling languages such as the UML. Such a meta-model is nevertheless essential for expressing the effects and implications of applying security solutions.

The meta-model depicted in Figure 5 addresses this by precisely defining the DFD model elements (*Process*, *DataStore*, and *ExternalEntity*) and how they can be linked together using *DataFlow*s.

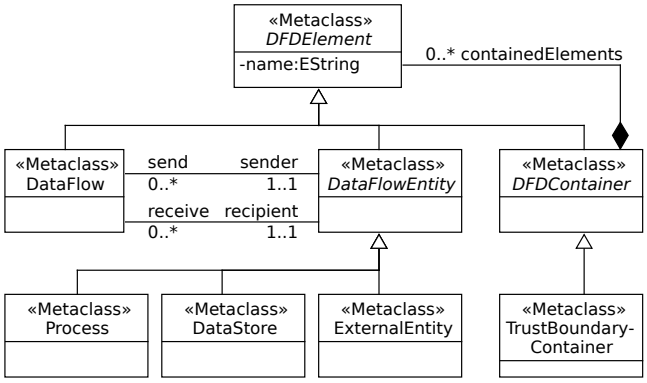

Figure 5: DFD meta-model

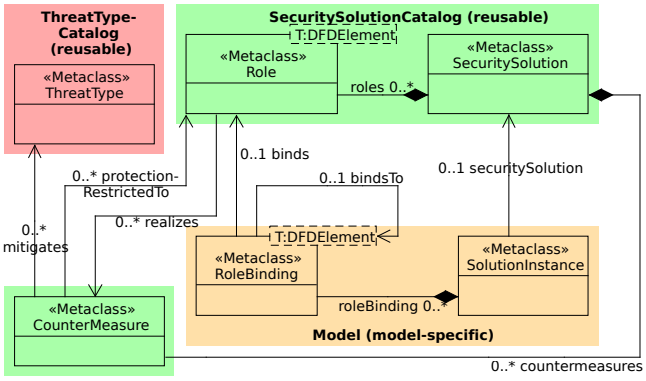

Figure 6: Security meta-model
The colors are used to indicate whether these elements are specified in the model or in separate reusable catalogs.

It also provides support for trust boundaries (with *TrustBoundaryContainer*s), which are used in the context of threat modeling to group elements together with the same 'trust level'. This allows the exclusion of some threats between elements with the same 'trust level' (e.g., machines on the internal network do not tamper with each other's traffic). This boils down to considering some threat types only if the data flow crosses a trust boundary.

Plain data flow diagrams do not support the representation of security solutions. While it is possible to extend the meta-model to include a list of model element properties (which is what threat modeling tools commonly do), this solution suffers from the issues discussed in the quality analysis in the previous section. Therefore, a more extensive security meta-model is presented in order to capture more details and intricacies of security solutions in the model.

4.2 Security Meta-model

Figure 6 depicts the proposed meta-model for expressing security solutions. A security solution is represented as an instance of *SecuritySolution*. These *SecuritySolution*s can be used to capture, for example, security patterns [10, 35] to enable their instantiation in DFD models. They are collected in a catalog (shown in color), which enables reuse of security solutions across multiple DFD models.

The details of a *SecuritySolution* are specified as follows. A *SecuritySolution* contains a list of *Role*s which are generic descriptions for the involved *DFDElement*s. They are parameterized according to the required type (e.g., *Process* or *DataFlow*). A *Role* can realize (i.e., implement) certain *CounterMeasure*(s), which determine which specific *ThreatType*(s) to that element are mitigated. Additionally, a *CounterMeasure* can specify to which *Role*s its protection applies. For example, the spoofing protection of the server in HTTPS only applies to those flows that are part of the HTTPS communication, so the server can still be spoofed in the context of other DFD elements communicating over different flows (e.g., if other clients communicate with the server process over plain HTTP).

Finally, to use the security solutions in a concrete DFD model, instances of these *SecuritySolution*s have to be inserted into the model. The meta-model allows the specification of a *SolutionInstance*, which represents such a concrete instantiation in a specific DFD model. The *SolutionInstance* contains a list of *RoleBinding*s which link the *Role*s from the *SecuritySolution* to concrete model elements of the DFD. The instantiation of a secure pipe security solution is illustrated in Figure 7, where the *SolutionInstance* contains the *RoleBinding*s that tie the model elements to the roles they fulfil in the solution.

During the threat elicitation, the threat elicitation engine (or analyst, in case of a manual elicitation) iterates over the *ThreatType*s in the threat type catalog, and looks for DFD elements that may be exposed to this *ThreatType*. This involves verifying whether the DFD element is not bound to a *Role* that provides a *CounterMeasure* against the *ThreatType* being considered (and whether that *CounterMeasure*'s protection is not limited to certain roles in the security solution). This means that the only check that happens is whether there already exists a *CounterMeasure* that prevents the considered *ThreatType*, independent from the actual *SecuritySolution* that provides that *CounterMeasure*. For example, the presence of an encryption countermeasure prevents an information disclosure threat, regardless of whether that countermeasure is part of a VPN or HTTPS security solution. This is a desired property, because it ensures that the threat type catalog remains maximally independent from the security solutions catalog.

5 EVALUATION

This section provides a functional validation of the presented meta-model and an evaluation using the four qualities from Section 3.

5.1 Functional validation

The presented meta-model is implemented in Ecore from the Eclipse Modeling Framework[2] (EMF) and is available as an Eclipse plug-in. A graphical editor for the creating and editing models in this meta-model is implemented using a Sirius Viewpoint Specification[3]. Included are a number model validation rules specified in Acceleo Query Language[4] (AQL), to prevent common flaws in DFD models. Since the focus is on the semantics of the meta-model, and not the visual syntax of DFD-models and their security solutions, there is not yet a visual syntax for representing security solutions.

[2]https://www.eclipse.org/modeling/emf/
[3]https://www.eclipse.org/sirius/
[4]https://www.eclipse.org/acceleo/documentation/aql.html

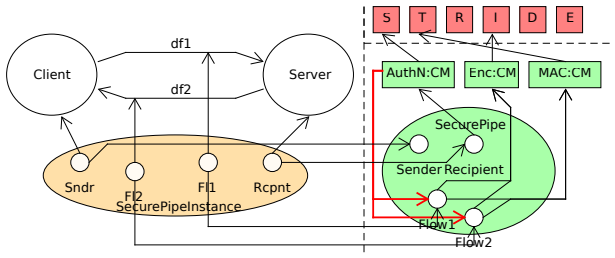

Figure 7: Example instantiation of the secure pipe solution. *The right-hand side shows (from top to bottom) the* ThreatTypes*,* CounterMeasures*, and* SecuritySolution *containing* 4 Roles*. The left-hand side shows the DFD model and (below that) the* SolutionInstance *containing the* 4 RoleBindings*, linking DFD model elements to the solution's roles. The same color-coding is used as in Figure 6. Note that the focus is on the semantics of the model, not on the syntax of the visual representation.*

For eliciting the threats, VIATRA[5] is used, which provides a graph-based pattern language. The specified patterns consist of 18 private patterns to be reused in other patterns to simplify commonly-occurring conditions (e.g., a flow crossing a trust boundary, checking for countermeasures against a certain threat type), and 21 DFD patterns (e.g., ExternalEntity-DataFlow-Process, Process-DataFlow-DataStore) used to match for concrete threats. VIATRA supports dynamic pattern matches, so the pattern results are updated on-the-fly while the model is being changed or enriched with security solutions. This allows the designer to conduct *what-if* analyses by making changes to the model and immediately evaluating the impact of those changes on the resulting threats.

More information on the prototype implementation is made available on the companion website for this paper [27].

5.2 Quality evaluation

In this section, we evaluate the presented approach to model security elements in terms of the four qualities discussed in Section 3.

Quality 1: Semantics

Security solutions are represented in the DFD-model by creating an instance of it and binding concrete DFD model elements to its roles. It is the binding to a role that realizes a countermeasure which determines whether a specific threat applies to an element. The generic representation of security solutions has the following advantages: (i) asymmetric security solutions (e.g., bi-directional HTTPS with server-side-only authentication) can be expressed, as the effects can be realized in separate roles, (ii) there is no confusion as to how a countermeasure needs to be interpreted (e.g., the restriction to certain flows can be expressed in the model), and (iii) there are no conflicts when multiple countermeasures are applied on a single element, as this just implies multiple role bindings (if specializations of elements are used, the combination of multiple countermeasures would require, for example, using two different data flows at the same time).

[5]https://www.eclipse.org/viatra/

The following reasoning provides an intuitive argument as to why the presented meta-model provides a strictly positive improvement of the semantic quality. It consists of two parts: (i) the presented meta-model is equally expressive as the property-based approach, and (ii) the presented meta-model can be used to express security solutions that cannot be properly expressed using the property-based approach.

(i) The meta-model is equally expressive as the property-based approach because it can simulate it. Every possible property on DFD elements can be represented as a very simplified security solution with a single role to specify the element with its property.

(ii) The example discussed in Section 3 for Quality 1 presents the application of SSL/TLS for the protection of communication between a client and a server. This mechanism is asymmetric because the destination authentication part of the security solution only for the communication from the client to the server, while instead, source authentication holds for the communication in the other direction. Consider the situation in which S/MIME is used on top of the SSL/TLS connection. S/MIME is a common email security solution that protects the confidentiality and integrity of email messages, but also provides non-repudiation of the sender. In the property-based approach, this cannot be added as a specialization of a dataflow, because another specialization (for TLS/SSL) would already be present. The only solutions to this problem would be to either create a new SSL/TLS+S/MIME dataflow type, or manually set the properties of the dataflow element to the combined effect of both countermeasures. Neither of these solutions is satisfactory, nor corresponds with a correct representation of the actual security solution. Such mismatches between the actual security solutions and their suboptimal representations can cause certain properties to be missed (or carelessly changed afterwards), leading to the incorrect application of security solutions.

Quality 2: Traceability

Traceability is a critical quality for being able to revisit, evaluate, and check previously made security decisions. The closest that the property-based approach can come to realizing traceability is by extensive use of specializations of the DFD elements. However, this quickly reaches its limits when multiple specializations of an element are needed at the same time, as illustrated before with SSL/TLS and S/MIME. By providing an explicit, first-class representation for security solutions, every decision to apply a certain security solution manifests itself explicitly, namely by including an instance of that security solution in the DFD model. This makes every effect traceable to its underlying security solution.

Besides the traceability from effect to solution (and vice versa), the security solutions are also explicitly linked (via the solution catalog) to the security threat types that they counter, thereby also enabling traceability from security solution to threat type. In case of the Microsoft TMT, on the other hand, the internal threat generation expressions of all threat types would have to be checked to find out which threat types are countered by some property.

Keeping the link intact between the security solutions and the threat types they counter has several additional benefits: (i) threats are not eliminated; instead, there is a 'mitigated by' relation between a concrete threat and the security solution instance; (ii) the explicit link enables additional analysis on threat impact and likelihood,

taking the precise DFD elements, threat type, and security solution–context into account; and (iii) changes to the protections provided by a solution can be evaluated simultaneously over all instances of that solution (for example, assessing the impact of the failure of some security solution's information disclosure countermeasure), thereby also enabling Quality 4.

Quality 3: Separation of Concerns

An evaluation of the separation of concerns quality is provided based on the impact of the following change scenarios: (i) changing security solutions (add/modify/remove), and (ii) changing threat types (add/modify/remove) in a property-based and our approach.

Changing security solutions – property-based. A new security solution requires the definition of new properties for the effects of the solution, possibly new specializations of existing DFD elements, and modifying the exclusion conditions for each threat type which is countered by the solution.
Modifying or removing a solution, requires the following changes: all the threat type expressions must be checked and updated to reflect the changes in the properties, any specializations using those properties may need to be updated, and any existing models using those security solutions may require updates. There is thus a large ripple effect when existing measures are updated, especially in the threat type expressions of which all the exclusion criteria have to be rechecked.

Changing security solutions – our approach. A new security solution only requires a change in the security solution catalog to introduce the (model representation of the) new solution, its roles, and the information about which threat types it counters.
Modifying or removing a solution only requires a change to existing models if the solution's roles change. If the change is limited to the threat types being prevented, existing models can use the new solution definition as-is, and the threat analysis results are automatically updated.

Changing threat types – property-based. Adding, modifying, or removing threat types only requires changes to the threat type catalog. Even the removal does not require changes to the properties or stencils, although some properties may no longer be used. Keeping these properties can lead to a bloated list of properties, causing confusion or flaws in a model when they are still used later on.

Changing threat types – our approach. Adding or modifying threat types are changes limited to the threat type catalog. Only the removal of threat types will require updating the security solution catalog as the threat types, the security solutions refer to, no longer exist.

The dynamic nature of security makes changes to existing security solutions very likely as flaws may be found in these solutions and new versions are made available; such changes can be problematic for the property-based approach as they require checking and updating the threat type generation expressions for each change.

Quality 4: Dynamism and continuous threat assessment

Because this quality strongly depends on appropriate tool support being available and the presented approach is limited to a prototype implementation, the evaluation of this quality is limited to a discussion on how, and to what extent, the presented model provides the foundation for enabling this quality in future tool support implementations.

Four elements are considered: (i) threat evolution, (ii) suggesting solutions, (iii) solution impact analysis, and (iv) knowledge base integration. We consider to what degree these elements are already realized in the prototype implementation, or could be realized in future implementations.

(i) The basic mechanism for threat evolution is already realized in the prototype. Threat type pattern matching is implemented using VIATRA IncQuery patterns, which enables dynamic pattern matching so any changes to the model are immediately reflected in the list of concrete threats. Further refinements are possible however, such as, for example, highlighting recently changed threats.

(ii) Suggesting security solutions is currently not implemented. However, making the security solutions and their associated roles and countermeasures explicit provides an essential building block to develop such a feature in the future.

(iii) The prototype implementation already has basic support for evaluating the architectural impact security solutions. By updating the concrete threat list, immediate feedback is provided on the impact of instantiating a particular solution.

(iv) Finally, the consolidation of security solutions in a catalog forms a good starting point for a reusable design-level security knowledge base. This catalog can evolve into a reusable and centrally shared resource, embedding current knowledge on security solutions and the threat types they counter.

6 RELATED WORK

Threat modeling has been introduced by Microsoft as a part of its security efforts [14, 15, 25, 29, 31] and is part of its security development life cycle [15]. It has been applied several times since, in real-world industrial contexts [8, 25, 31].

Data Flow Diagrams (DFDs) are the core artefacts used in these threat modeling approaches, but mainly as graphical models. There are some efforts towards more formal underpinnings for functional correctness [11, 30], or extensions to focus security threat analysis on the most relevant threats [32]. Berger et al. [5] also recently proposed a DFD extension for threat extraction. Their work is centered around an extensive catalog of threats such as CAPEC and CWE, while our approach is focused on the representation of security solutions.

Following the growing importance of privacy over the last few years, threat modeling has been extended to also elicit privacy threats [1, 7, 34], thereby enabling privacy by design. To support this in DFD models, Antignac et al. [2] presented a DFD extension to support privacy concepts. While our implementation does not yet support such privacy annotations (e.g., data, purpose), this would be a relevant future extension, especially if the solution catalog is extended with privacy solutions.

Other approaches of for the elicitation of security requirements include risk analysis [28], which can actually be used in a complementary fashion to threat modeling by using it for prioritization [23]. Such a complementary approach is closely related to the realization of Quality 4, by moving from a binary (threat is applicable or not) analysis result to a more quantitative threat analysis result based on risk (e.g., likelihood and impact).

Besides STRIDE, there are also more extensive threat classification systems [18]. The presented meta-model does not enforce a certain classification scheme, so other threat types can also be used.

Finally, there are some other approaches to security such as attack trees [24], which start from the perspective of an attacker that wants to achieve a goal, and from there explore the attacker's possibilities to achieve that goal and possible countermeasures to prevent it. Such an approach is more perpendicular, as it enables an in-depth analysis of a particular threat and provides guidance in choosing how to protect against that threat. Related to that, there are systematic approaches, such as Li et al. [19], that leverage attack pattern repositories such as CAPEC to identify attacks.

7 CONCLUSION

The integration of threat modeling in the software development life cycle is a highly promising way to implement Security by Design and Privacy by Design. Unfortunately, existing threat modeling approaches that rely on plain DFDs commonly lead to combinatorial explosions of threats. In practice, they therefore rely extensively on extensions that limit the amount of raised threats and the effort required for processing them.

However, when used carelessly, such extensions can lead to the omission of security or privacy threats that are not actually mitigated. In this paper, we have presented an approach to limit and scope the threat elicitation space by leveraging knowledge about existing security solutions in the system under design.

For our approach, we have shown positive improvements in terms of semantic quality, traceability, separation of concerns, and dynamism, respectively due to (i) the proper instantiation of security solutions in the system under design, (ii) traceability of security effects to the security solutions causing these effects, and thereby ensuring that the decisions to apply these security solutions remain documented, (iii) independent evolution of the security solution catalog to stay on par with evolutions in the field, and (iv) dynamic and continuous threat assessment by providing impact analysis and architecture-level security decision making support.

We believe that the additional complexity introduced by our approach is manageable in practice given appropriate tool support, which will be evaluated more thoroughly in the future. Furthermore, we will look at more dynamic approaches, in which binary threat classifications (e.g., *threat mitigated* or *threat not applicable*) are replaced with probabilistic values that better characterize the degree of certainty with which a specific security solution prohibits certain security threats. This will allow for constant re-assessment of threats, for example when new vulnerabilities arise or when specific architectural assumptions are invalidated over time. In addition, we will research the degree to which the selection of appropriate security solutions can be approached as a search problem (in search-based software engineering), using the coverage of threats as an optimization goal.

ACKNOWLEDGMENTS

This research is partially funded by the Research Fund KU Leuven, the Secure Design project of the imec HI2 Distributed Trust program, and the imec PRO-FLOW research project.

REFERENCES

[1] M. Alshammari and A. Simpson. Towards a Principled Approach for Engineering Privacy by Design. 2016.
[2] T. Antignac, R. Scandariato, and G. Schneider. *A Privacy-Aware Conceptual Model for Handling Personal Data*, pages 942–957. Springer, 2016.
[3] M. A. Babar, T. Dingsøyr, P. Lago, and H. van Vliet, editors. *Software Architecture Knowledge Management*. Springer Berlin Heidelberg, 2009.
[4] L. Bass, P. Clements, and R. Kazman. *Software Architecture in Practice*. Addison-Wesley Professional, 3rd edition, 2012.
[5] B. J. Berger, K. Sohr, and R. Koschke. Automatically Extracting Threats from Extended Data Flow Diagrams. *ESSoS 2016 (LNCS)*, 9639:56–71, 2016.
[6] T. DeMarco. *Structured Analysis and System Specification*. Yourdon Press, 1979.
[7] M. Deng, K. Wuyts, R. Scandariato, B. Preneel, and W. Joosen. A privacy threat analysis framework: supporting the elicitation and fulfillment of privacy requirements. *Requirements Engineering*, 16(1):3–32, 2011.
[8] D. Dhillon. Developer-Driven Threat Modeling: Lessons Learned in the Trenches. *IEEE Security Privacy*, 9(4):41–47, jul 2011.
[9] European Union. Regulation (EU) 2016/679. *Official Journal of the European Union*, 59(L 119):1–88, may 2016.
[10] E. Fernandez-Buglioni. *Security patterns in practice: designing secure architectures using software patterns*. John Wiley & Sons, 2013.
[11] R. B. France. Semantically Extended Data Flow Diagrams: A Formal Specification Tool. *IEEE Transactions on Software Engineering*, 18(4):329–346, 1992.
[12] C. Gane and T. Sarson. *Structured Systems Analysis: Tools and Techniques*. Prentice Hall Ptr, 1979.
[13] Google. WebRTC Project. https://webrtc.org/, September 2017.
[14] S. Hernan, S. Lambert, T. Ostwald, and A. Shostack. Threat Modeling: Uncover Security Design Flaws Using The STRIDE Approach. *MSDN Magazine*, 6, nov 2006.
[15] M. Howard and S. Lipner. *The Security Development Lifecycle*. Number May. Microsoft Press, 2006.
[16] ISO/IEC/IEEE. ISO/IEC/IEEE Systems and software engineering – Architecture description. *ISO/IEC/IEEE 42010:2011(E)*, 1:1–46, 2011.
[17] A. Jansen and J. Bosch. Software Architecture as a Set of Architectural Design Decisions. In *5th Working IEEE/IFIP Conference on Software Architecture (WICSA'05)*, pages 109–120. IEEE, 2005.
[18] M. Jouini, L. B. A. Rabai, and A. B. Aissa. Classification of security threats in information systems. *Procedia Computer Science*, 32:489–496, 2014.
[19] T. Li, E. Paja, J. Mylopoulos, J. Horkoff, and K. Beckers. Security attack analysis using attack patterns. *Proceedings - International Conference on Research Challenges in Information Science*, 2016-August, 2016.
[20] G. McGraw. *Software security: building security in*. Addison-Wesley Professional, 2006.
[21] Microsoft Corporation. Microsoft Threat Modeling Tool 2016. http://aka.ms/tmt2016, 2016.
[22] OWASP. Software Assurance Maturity Model Version 1.5. Technical report, OWASP, 2017.
[23] T. Rauter, N. Kajtazovic, and C. Kreiner. Asset-Centric Security Risk Assessment of Software Components. *2nd International Workshop on MILS: Architecture and Assurance for Secure Systems*, 2016.
[24] B. Schneier. Attack trees. 1999.
[25] A. Shostack. Experiences threat modeling at microsoft. In *Modeling Security Workshop. Dept. of Computing, Lancaster University, UK*, 2008.
[26] A. Shostack. *Threat Modeling: Designing for Security*. John Wiley & Sons, Indianapolis, Indiana, 2014.
[27] L. Sion, K. Yskout, D. Van Landuyt, and W. Joosen. Companion site. https://people.cs.kuleuven.be/ laurens.sion/sac2018/, 2017.
[28] J. L. Spears. A Holistic Risk Analysis Method for Identifying Information Security Risks. pages 185–202, 2005.
[29] F. Swiderski and W. Snyder. *Threat modeling*. Microsoft Press, 2004.
[30] Y. Tao and C. Kung. Formal definition and verification of data flow diagrams. *The Journal of Systems and Software*, 16(1):29–36, 1991.
[31] P. Torr. Demystifying the threat modeling process. *IEEE Security & Privacy Magazine*, 3:66–70, 2005.
[32] K. Tuma, R. Scandariato, M. Widman, and C. Sandberg. Towards security threats that matter. In *3rd Workshop On The Security Of Industrial Control Systems & Of Cyber-Physical Systems (CyberICPS 2017)*, pages 1–16. Springer, 2017.
[33] J. S. Ven, A. G. J. Jansen, J. A. G. Nijhuis, and J. Bosch. Design Decisions: The Bridge between Rationale and Architecture. In A. Dutoit, R. McCall, I. Mistrik, and B. Paech, editors, *Rationale Management in Software Engineering*, pages 329–348. Springer Berlin Heidelberg, 2006.
[34] K. Wuyts. *Privacy Threats in Software Architectures*. PhD thesis, January 2015.
[35] K. Yskout, T. Heyman, R. Scandariato, and W. Joosen. A system of security patterns. 2006.