Security Threat Modeling: Are Data Flow Diagrams Enough?

Laurens Sion
laurens.sion@cs.kuleuven.be
imec-DistriNet, KU Leuven

Koen Yskout
koen.yskout@cs.kuleuven.be
imec-DistriNet, KU Leuven

Dimitri Van Landuyt
dimitri.vanlanduyt@cs.kuleuven.be
imec-DistriNet, KU Leuven

Alexander van den Berghe
alexander.vandenberghe@cs.kuleuven.be
imec-DistriNet, KU Leuven

Wouter Joosen
wouter.joosen@cs.kuleuven.be
imec-DistriNet, KU Leuven

ABSTRACT

Traditional threat modeling approaches such as Microsoft's STRIDE rely on Data Flow Diagrams (DFDs) as the main input. As DFDs are constructed from only five distinct model element types, these system models are deliberately kept simple. While this lowers the bar for practical adoption, there are a number of significant drawbacks.

In this position paper, we identify and illustrate four key shortcomings of DFD models when used for security threat modeling, related to the inadequate representation of security concepts, data elements, abstraction levels, and deployment information. Based on these shortcomings, we posit the need for a dedicated, integrated language for threat modeling, and discuss the trade-offs that need to be made between the ease of adoption and the level of support for systematic and repeatable threat modeling.

KEYWORDS

security, security by design, data flow diagrams, threat modeling

ACM Reference Format:
Laurens Sion, Koen Yskout, Dimitri Van Landuyt, Alexander van den Berghe, and Wouter Joosen. 2020. Security Threat Modeling: Are Data Flow Diagrams Enough?. In *IEEE/ACM 42nd International Conference on Software Engineering Workshops (ICSEW'20), May 23–29, 2020, Seoul, Republic of Korea.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3387940.3392221

1 INTRODUCTION

The importance of early and extensive threat analysis to identify potential security issues before they become impactful and difficult to counteract cannot be understated. Especially in the context of mission- or safety-critical systems, a proper and in-depth analysis of cybersecurity threats and risks is an essential step of the development life cycle. Many threat modeling approaches and techniques have been proposed for performing design-level analyses of software systems, the most popular approach being STRIDE threat modeling [6, 10, 14]. STRIDE involves the following steps: (i) model the system under analysis using a Data Flow Diagram (DFD), (ii) elicit threats at the level of the modeled elements or interactions, and (iii) prioritize and document the identified threats.

DFDs find their origins in the area of structured programming and program analysis [5, 18, 20] and have found wide application in threat modeling due to their inherent simplicity. The DFD notation comprises four element types: *external entities*, *data flows*, *processes*, and *data stores*. In the context of security threat modeling, a fifth type is introduced: *trust boundaries* [6, 14, 19].

Figure 1 illustrates the DFD of a Hospital Information System (`HIS`) that continuously synchronizes data obtained from a diverse set of involved health practitioners (e.g. `Physicians` contributing to patient records) with a back-end cloud storage and backup service (the `Health Cloud`). DFDs such as this serve the clear purpose of structuring and naming the involved processes, data stores, and data flows, and can be obtained in a relatively light-weight and ad-hoc fashion. Drawing up such a DFD in a quick workshop-style session is a good way to start threat modeling exercises and find consensus among stakeholders and other workshop participants about the scope and high-level structure of the system under analysis.

Despite these eminent benefits, we argue that the use of DFDs forms a non-trivial roadblock towards more mature threat modeling approaches. More specifically, DFDs lack expressiveness in terms of (i) elements that pertain to the security architecture (e.g., explicit countermeasures), (ii) the involved data types, (iii) making explicit the dependencies on security properties provided by lower-level abstraction layer, and (iv) relations to other system aspects, such as deployment. Relying exclusively on DFDs, the success of a threat modeling exercise depends strongly on the involvement of stakeholders with security expertise and threat modeling experience. These are intangible influences that have a significant impact on the reproducibility and evolvability of the resulting threat model.

In Section 2, we further detail the above argumentation by systematically discussing the inherent strengths and weaknesses of the use of the DFD notation in support of threat modeling. In Section 3, we refine our vision towards more mature system modeling approaches in support of threat modeling, which does not necessarily implies adopting extremely formalized notations, but certainly entails a departure from the exclusive use of DFDs.

2 DFD STRENGTHS AND WEAKNESSES

The primary purpose of Data Flow Diagrams (DFDs) is to model how data flows through a system, by specifying which processes act upon what data, and where this data is stored. While data flows are an important aspect with security implications, which explains their appeal in threat modeling [6, 14, 19], the DFD representation is not specifically tailored towards security. Because of this, there is a mismatch between what can be modeled in a DFD and the information that is required during threat modeling. This mismatch leads to important assumptions being made outside of the models, which makes the models harder to interpret, reuse, and analyze.

ICSEW'20, May 23–29, 2020, Seoul, Republic of Korea
© 2020 Copyright held by the owner/author(s). Publication rights licensed to ACM. This is the author's version of the work. It is posted here for your personal use. Not for redistribution. The definitive Version of Record was published in *IEEE/ACM 42nd International Conference on Software Engineering Workshops (ICSEW'20), May 23–29, 2020, Seoul, Republic of Korea*, https://doi.org/10.1145/3387940.3392221.

The next subsections will go into detail on the different strengths and weaknesses of the DFD representation in the context of security threat modeling and illustrate these with a concrete example.

2.1 Strengths

We identify three strengths of DFDs that underlie their wide adoption: the simplicity of their notation, their ability to manage complexity, and the fact that they are technology-agnostic.

Notation simplicity The original DFD representation consists of only four different element types, each with distinct purposes and visual representations. Indeed, the DeMarco DFD representation [5] is even considered to be an example of graphical design excellence [11]. Consequently, stakeholders can get started with DFD without requiring extensive background knowledge. Furthermore, the simplicity of DFDs in terms of notation also simplifies the creation of tool support for creating and maintaining DFDs [10, 15].

In the context of threat modeling, the possible (types of) threats are typically associated with one or more modeling element types. For example, in STRIDE, the spoofing threat is associated with processes and external entities, whereas the tampering threat is related to processes, data stores and data flows [14]. The limited number of element types in a DFD allows a relatively straightforward mapping between possible threat types and model elements.

Complexity management A system is not immediately designed in the highest amount of detail. Consequently, the used modeling notation should allow to model a system at various levels of detail and add new information later on. DFDs support this by allowing the decomposition of processes into subprocesses, where the subprocesses and the data flows between them describe the containing process in a higher level of detail. For example, the coarse-grained `HIS` process in Figure 1 can be further refined or decomposed to represent the data flows between different subsystems installed in the hospital infrastructure.

This mechanism allows to *scope* the threat analysis efforts by conducting the analysis at this level of detail. A more fine-grained assessment can still be performed later on by further decomposing the processes into their subprocesses.

Technology-agnostic Due to its focus on how data flows through a system, DFDs are not tied to a specific technology (e.g., some middleware platform) or paradigm (e.g., object-oriented programming). It thus suffices for participants in a threat modeling exercise to know the four modeling elements provided by DFDs and have no further knowledge of software engineering. Consequently, this lowers the bar to participate in a threat modeling exercise, allowing a broad range of stakeholders to be involved.

2.2 Weaknesses

The weaknesses of the DFD notation in the context of security threat modeling stem from a lack of modeling support for security-relevant information. We identify four general classes of such information: security concepts, data model, abstraction levels, and deployment information. For each of these classes, we provide a description, a rationale for their importance, a concrete example, the research challenge associated with it, and references to existing work that has already (partially) addressed this weakness.

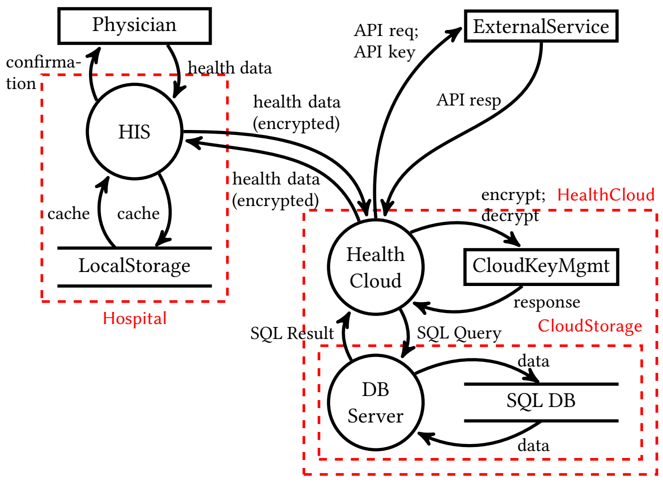

Figure 1: Example Data Flow Diagram of a Hospital Information System (HIS) that synchronizes health data with a Health Cloud to enable additional functionality via external third party services such as patient access via health portals.

Security concepts A model that supports threat modeling activities should provide an extensive set of security concepts that are precisely and unambiguously defined. Furthermore, this set of security concepts should support modeling a wide range of security countermeasures, incorporating information on the attacker assumptions, and trust relations in the system under consideration.

Importance: The lack of precisely defined security concepts can lead to differences in their interpretation, causing conflicting assumptions and security issues to be overlooked.

Examples: The DFD representation used in threat modeling has a trust boundary concept with a number of different interpretations: (i) denoting different levels of trust or privilege in the system; (ii) representing information or assumptions on the attacker model (e.g., parts of the system that are assumed to be inaccessible to an external attacker); (iii) expressing deployment information such as, for example, which resources are running on the same local network. Figure 1 shows such different usages of trust boundaries. The outer trust boundaries (`Hospital` and `HealthCloud`) convey deployment information, while the inner trust boundary (`CloudStorage`) represents a different privilege level in the system. Furthermore, the right outer trust boundary (`HealthCloud`) can denote the location of the attacker (outside of it), while this may not hold for the left trust boundary (`Hospital`).

Research challenge: Identify and precisely define the necessary security concepts and provide a manner to unambiguously encode these security concepts in the model.

Existing work: There are already several attempts to encode security solutions in DFDs [17], extend DFD elements with the effects of security solutions [4, 10], creating DFD element subtypes with security effects [10] and formalizing DFDs [8]. Explicitly documenting the provided security countermeasures is also a requirement for attaining higher levels in security maturity models [9, 12].

Data model Another semantic issue with DFDs is the lack of an explicit data model. While DFDs have a strong focus on how data flows through the system, the data is only modeled as labels on the data flows and thus remains described in very ad-hoc manner. This causes a number of problems for threat modeling: (i) data flows with a different label could implicitly refer to the same data, or vice versa; (ii) there is no distinction between data treated as data and data treated as code; and (iii) there is no support to specify important security properties such of data such as its sensitivity.

Importance: Data is an important factor in many security assessments, both in confidentiality or integrity, and in performing appropriate validation when processing external data.

Examples: DFDs only model data flows, not data types themselves. One of the most common security issues, despite its age, in web security is SQL injection [1]. Another similar and more recent issue is the EFAIL vulnerability [13]. Both these issues result from the treatment of untrusted data as code [2]. The DFD in Figure 1 shows that: (i) it is not possible to determine whether the `health data` from `Physician` to `HIS` is the same as from `HIS` to `Health Cloud`; (ii) there is no distinction between data treated as code such as `SQL Query` and other data; and (iii) it is not possible to see which data is sensitive without relying on background knowledge.

Research challenge: Enable modeling the involved data types, as well as their security properties, as first-class concepts.

Existing work: While the early DFD representations provide support for an explicit list of in and output parameters [18] or data dictionaries [5] specifying the data types, there mechanisms are not used in a threat modeling context.

Abstraction levels While DFDs do support hierarchical decomposition of specific processes for managing complexity (see before), they only offer support for a constructing the model of a system at a single level of abstraction. There is no support for making explicit the dependencies on lower-level layers that are required for providing specific security guarantees.

Importance: Many applications rely on security properties provided or guaranteed by lower-level layers such as a TLS connection that provided confidentiality, integrity, and authentication of the communication of a web applications.

Examples: Threats in security threat modeling frameworks, such as STRIDE, may be triggered by underlying layers that fail to provide the appropriate security guarantees that higher-level layers implicitly or explicitly rely on. For example, in Figure 1 the confidentiality and integrity of the `health data` communicated between the `HIS` and `Health Cloud` is provided by TLS at the communication layer, and not implemented by the application itself. The `(encrypted)` annotation on the flow does not suffice to clarify this. The abstraction levels also impact the elicitation of the different threat types. For example, *spoofing* the `Health Cloud` when communicating with the `ExternalService` in Figure 1 can be the result of *information disclosure* of the `API key` in the communication layer below. Another example in Figure 1 is the reliance on the hardware-backed cloud key management service which is provided by a lower-level API provided by the cloud service provider.

Research challenge: Introduce support for the explicit specification of security dependencies on lower-level layers, to prevent such dependencies as implicit security assumptions.

Existing work: While the importance of precisely modeling system properties and behavior is recognized [3], there is only limited work explicitizing dependencies between abstraction levels [16].

Deployment information DFDs do not provide support for capturing deployment information, which has important security implications. While the trust boundaries introduced in DFDs for security do allow to capture some of the information, they introduce ambiguity because of other conflicting interpretations (see *Security concepts* above) and are thus a suboptimal solution.

Importance: The deployment view provides information on the physical allocation of processes, which is directly related to the attacker model and has important security implications related to input sanitation, access control mechanisms, etc.

Examples: The DFD in Figure 1 lacks explicit deployment information and thus provides limited assistance in determining where, for example, appropriate access control measures should be instantiated. The trust boundaries could be used as guidance, but, due to the ambiguity in their interpretation, could also lead to wasted effort in misplacing these countermeasures in unnecessary locations.

Research challenge: Provide support for modeling deployment information consistent with the other views on the system to ensure that the relevant deployment information can be taken into account during threat modeling.

Existing work: The UML [7] provides deployment diagrams to model the relevant deployment information.

3 DISCUSSION AND OUTLOOK

While DFDs are commonly used as a basis for threat modeling exercises, their semantics are not sufficiently rich to capture and express all the information that is relevant for such a security analysis. One could argue that DFDs, despite this limitation, have gained traction for threat modeling because they only serve as a communication vehicle between stakeholders to bootstrap the activity, but serve no formal role beyond that. But even in that capacity, there is important security information missing from DFDs that must be captured in some form. In the best case, it is explicitly written down in the form as textual assumptions or through ad-hoc extended DFDs. In the worst case, this information only exists in the heads of the people who conducted the threat modeling, and is bound to be inconsistent, misinterpreted, implicit, or forgotten.

As illustrated with the issues outlined above, we claim that the DFD weaknesses in this context hinder the reproducibility and evolvability of the threat analysis exercises. A solution lies in an integrated approach, where all relevant information naturally fits within the modeling language used to conduct the threat modeling activity: you can only reason about what you can express.

By capturing this information in the modeling language, threat modeling would become less dependent on expertise and background knowledge, which are being replaced by explicitly recorded information, improving the reproducibility. Furthermore, it becomes easier to revisit, update, and evolve existing threat models.

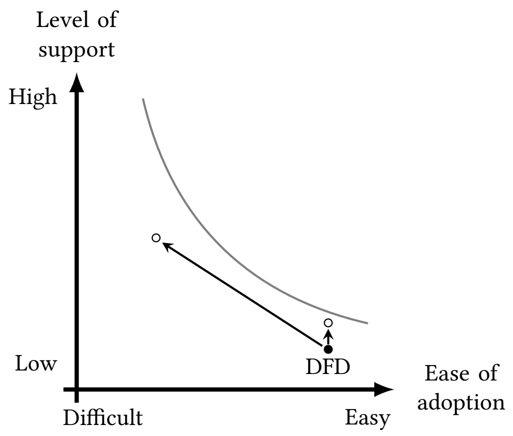

Figure 2: Illustration of the inherent trade-off involved between the ease of adoption of a modeling language such as DFDs (horizontal axis), and its level of support for systematic and reproducible threat modeling (vertical axis).

Finally, a modeling language that incorporates this information opens the door for more advanced tool support that can reduce the required effort and increase the level of automation (e.g., to integrate threat modeling in a DevOps context).

Such a modeling language is not readily available, though, as highlighted by the research challenges in the previous section. Furthermore, Figure 2 shows how a candidate language must balance two forces in order to become successful: complexity of the language which makes adoption more challenging and its support for systematic and repeatable threat modeling.

First, there is the need for a modeling language that is dedicated towards security threat modeling, and explicitly supports multiple, complementary yet consistent views that support performing a systematic, repeatable, thorough threat modeling exercise. That is, the language must provide a high '*level of support*' for threat modeling, a quality which we have argued, due to the aforementioned weaknesses outlined Section 2.2, DFDs do not sufficiently possess.

The second force is the '*ease of adoption*' of the modeling language. The strengths of DFDs (see Section 2.1), which underlie its popularity, are counteracted by tailoring and semantically enriching the modeling language. We believe that providing proper tool support is an important factor in fostering adoption by, for example, ensuring integration and consistency among the different views.

It is unlikely that there will be a single, one-size-fits-all modeling approach that makes the optimal trade-off for every context. Some systems or application domains may require a more rigid analysis, fully exploiting the semantic richness and formality of the language. For other systems, a more informal yet accessible language (akin to DFDs used today) would suffice. It is unclear at this point whether a single language can cover this entire spectrum, or whether multiple languages are needed. Whatever the outcome regarding this matter may be, this paper has argued that data flow diagrams are not enough for systematic and reproducible threat modeling.

ACKNOWLEDGMENTS

This research is partially funded by the Research Fund KU Leuven and the Flemish Research Programme Cybersecurity.

REFERENCES

[1] 2019. OWASP Top Ten Project. https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project
[2] Iván Arce, Neil Daswani, Jim Delgrosso, Danny Dhillon, Christoph Kern, Tadayoshi Kohno, Carl Landwehr, Gary Mcgraw, Brook Schoenfield, Margo Seltzer, Diomidis Spinellis, Izar Tarandach, and Jacob West. 2014. *Avoiding the Top 10 Software Security Design Flaws.* Technical Report. IEEE Center for Secure Design.
[3] Jason Bau and John C. Mitchell. 2011. Security modeling and analysis. *IEEE Security and Privacy* 9, 3 (2011), 18–25. https://doi.org/10.1109/MSP.2011.2
[4] Bernhard J. Berger, Karsten Sohr, and Rainer Koschke. 2016. Automatically Extracting Threats from Extended Data Flow Diagrams. *ESSoS 2016 (LNCS)* 9639 (2016), 56–71. https://doi.org/10.1007/978-3-319-30806-7
[5] Tom DeMarco. 1979. *Structured Analysis and System Specification.* Yourdon Press.
[6] Michael Howard and Steve Lipner. 2006. *The Security Development Lifecycle.* Microsoft Press.
[7] ISO/IEC. 2012. *ISO/IEC 19505-1:2012 Information technology - Object Management Group Unified Modeling Language (OMG UML), Infrastructure.* Standard 19505-1:2012(E). ISO/IEC. http://www.omg.org/cgi-bin/doc?formal/2012-05-06.pdf
[8] Atif Aftab Ahmed Jilani, Aamer Nadeem, Tai-Hoon Kim, and Eun-Suk Cho. 2008. Formal Representations of the Data Flow Diagram: A Survey. In *2008 Advanced Software Engineering and Its Applications.* 153–158. https://doi.org/10.1109/ASEA.2008.34
[9] Gary McGraw, Sammy Migues, and Jacob West. 2018. *BSIMM9.* Technical Report.
[10] Microsoft Corporation. 2016. Microsoft Threat Modeling Tool 2016. http://aka.ms/tmt2016.
[11] Daniel L Moody. 2009. The "Physics" of Notations: Toward a Scientific Basis for Constructing Visual Notations in Software Engineering. *IEEE Transactions on Software Engineering* 35, 6 (nov 2009), 756–779. https://doi.org/10.1109/TSE.2009.67
[12] OWASP. 2017. *Software Assurance Maturity Model Version 1.5.* Technical Report. OWASP. 72 pages.
[13] Damian Poddebniak, Christian Dresen, Jens Müller, Fabian Ising, Sebastian Schinzel, Simon Friedberger, Juraj Somorovsky, and Jörg Schwenk. 2018. Efail: Breaking S/MIME and OpenPGP Email Encryption using Exfiltration Channels. In *27th {USENIX} Security Symposium ({USENIX} Security 18).* {USENIX} Association, Baltimore, MD, 549–566. https://www.usenix.org/conference/usenixsecurity18/presentation/poddebniak
[14] Adam Shostack. 2014. *Threat Modeling: Designing for Security.* 590 pages.
[15] Laurens Sion, Dimitri Van Landuyt, Koen Yskout, and Wouter Joosen. 2018. Sparta: Security & privacy architecture through risk-driven threat assessment. In *2018 IEEE International Conference on Software Architecture Companion (ICSA-C).* IEEE, 89–92.
[16] Laurens Sion, Koen Yskout, Riccardo Scandariato, and Wouter Joosen. 2017. A Modular Meta-model for Security Solutions. In *Companion to the first International Conference on the Art, Science and Engineering of Programming.* ACM, 16.
[17] Laurens Sion, Koen Yskout, Dimitri Van Landuyt, and Wouter Joosen. 2018. Solution-aware data flow diagrams for security threat modeling. In *Proceedings of the 33rd Annual ACM Symposium on Applied Computing.* ACM, 1425–1432.
[18] W P Stevens, G J Myers, and L L Constantine. 1974. Structured design. *IBM Systems Journal* 13, 2 (1974), 115–139. https://doi.org/10.1147/sj.132.0115
[19] Frank Swiderski and Window Snyder. 2004. *Threat modeling.* Microsoft Press.
[20] Edward Yourdon and Larry Constantine. 1975. *Structured Design.*