# A Method for Quality Assessment of Threat Modeling Languages: The Case of enterpriseLang

Wenjun Xiong[a], Simon Hacks[a,b] and Robert Lagerström[a]

[a]School of Electrical Engineering and Computer Science, KTH Royal Institute of Technology, Stockholm, Sweden
[b]Southern University of Denmark, Denmark

**Abstract**
Enterprise systems are growing in complexity, and the adoption of cloud and mobile services has greatly increased the attack surface. To proactively address these security issues in enterprise systems, a threat modeling language for enterprise systems called enterpriseLang was proposed. It is a domain-specific language (DSL) designed using the Meta Attack Language (MAL) framework and focuses on describing system assets, attack steps, defenses, and asset associations. The threat models can serve as input for attack simulations to analyze the behavior of attackers within the system. However, whether and to what extent the functionality of these threat modeling languages is achieved has not been addressed. To ensure the correct functionality of threat modeling languages, this paper proposes a method to assess the quality of such languages and illustrates its application using enterpriseLang.

**Keywords**
Threat modeling, Attack simulations, Domain-specific language, Design guidelines, Test coverage

## 1. Introduction

Enterprise systems are growing in complexity and are becoming more and more connected. Such connected systems can increase flexibility and productivity while also introducing security threats. Recent years saw some of the largest, most sophisticated, and most severe cyber attacks, such as the SolarWinds attack[1] and the Florida water supply attack[2], and the Facebook information leak[3], which affected millions of consumers and thousands of businesses.

To proactively deal with these security issues, threat modeling [1, 2] is one approach that includes identifying the main assets within a system and threats to these assets. The approach can be used to both assess the current state of a system and as a security-by-design tool for developing secure systems and software. These threat models can serve as input for attack simulations, which are used to analyze the behavior of attackers within the system [3], and provide probabilistic security evaluations [4]. Based on such objective evaluations, security controls can be prioritized and implemented to counter anticipated threats.

*PoEM'21 Forum: 14th IFIP WG 8.1 Working Conference on the Practice of Enterprise Modelling, November 24-26, 2021*
wenjx@kth.se (W. Xiong); shacks@kth.se (S. Hacks); robertl@kth.se (R. Lagerström)
0000-0003-0434-4436 (W. Xiong); 0000-0003-0478-9347 (S. Hacks); 0000-0003-3089-3885 (R. Lagerström)

[1]https://www.cnet.com/news/solarwinds-hack-officially-blamed-on-russia-what-you-need-to-know/
[2]https://www.industrialdefender.com/florida-water-treatment-plant-cyber-attack/
[3]https://www.cbsnews.com/news/millions-facebook-user-records-exposed-amazon-cloud-server/

Previously, a domain-specific language (DSL) called enterpriseLang [5] was designed. It is based on the Meta Attack Language (MAL) framework [4], which allows for analyzing weaknesses related to known attack techniques and providing mitigation suggestions for these attacks. Therefore, stakeholders of an enterprise can assess threats to their enterprise IT environment and analyze what security settings could be implemented to secure the system more effectively.

However, designing a MAL-based DSL is like any other complex task that is sometimes error-prone and usually time-consuming, especially if the language shall be of high-quality and comfortably usable [6]. To ensure the quality and usability of threat modeling languages, established DSL design guidelines can be applied to assist in language development [6], and automated tests based on threat models can be designed to check if the designed language behaves as expected [3]. In this paper, we propose a method to assess the quality of MAL-based DSLs exemplified with enterpriseLang. First, we compare a set of DSL design guidelines to the development of enterpriseLang as a qualitative assessment. Then, we calculate the test coverage of enterpriseLang to estimate how much of the assets, attack steps, edges, and defenses are covered by existing test cases.

The remainder of this paper is structured as follows. In Section 2 and Section 3, we address the related work and the background. Our proposed method is presented in Section 4. In Section 5, we address the qualitative assessment of enterpriseLang, and in Section 6, we address the quantitative assessment. Our work is discussed in Section 7 and concluded in Section 8.

## 2. Related Work

Our work relates to the evaluation of DSLs, which has been addressed previously in qualitative and quantitative manner. Rodrigues et al. [7] conducted a systematic literature review (SLR) on the usability evaluation of DSLs, where the techniques identified include software engineering evaluation methods e.g., case studies [8] and experiential studies [9], and usability evaluation methods such as usability tests [8] and heuristic evaluations [10]. Venable et al. [11] proposed an extended design science research (DSR) evaluation framework for assessment of design artifacts, such as DSLs, and pointed out some methods that can be used for evaluation, including observational, analytical, experimental, and testing methods [12]. Haugen et al. [13] presented a structured questionnaire based on three dimensions of a DSL, including expressiveness, transparency, and formalization.

There are also a number of empirical evaluations of model-based threat analysis approaches. For example, to empirically validate the STRIDE framework, Scandariato et al. [14] performed a descriptive study to determine the completeness of the security analysis results. Similarly, Wuyts et al. [15] presented empirical studies to evaluate a privacy threat analysis methodology called LINDDUN.

Some modeling and simulation languages and related approaches have addressed the evaluation and validation explicitly. For example, coreLang [16] was developed as a simulation language for general IT domains. It was evaluated through tests, a series of brainstorming sessions with domain modelers, and with known cyber attack scenarios. powerLang [17] was proposed as a probabilistic attack simulation language for the power domain and was validated using a known cyber attack. vehicleLang [18] was designed to model vehicle IT infrastructure, and it was evaluated by performing an SLR to identify possible attacks against vehicles, which then served as a blueprint for test cases. pwnPr3d [19] was proposed as a threat modeling approach for automatic attack graph generation based on network modeling and addressed its validation by thorough experimentation of real-life systems (e.g., in-depth modeling of the UNIX operating system). CySeMoL [20] was proposed as a cyber security modeling language for enterprise-level system architectures and was validated through a test comparing the CySeMoL assessments with the assessments of security professionals.

## 3. Background

### 3.1. Introduction to MAL

MAL is a language framework that serves as a basis to develop DSLs for modeling systems, threats, and attacks in different domains, and generates attack graphs from the models [4]. Such a language defines what information is required and specifies the generic attack logic about the domain studied.

To create a MAL-based language, the first thing is to identify all relevant assets and their associations within a particular domain. Each asset contains multiple attack steps, representing real threats. A compromised attack step can lead to (represented by −>) a next attack step, where each attack step is either of type OR (represented by |) or AND (represented by &). OR indicates that an attacker can work on this attack step as soon as one of its parent attack steps is compromised, while AND indicates that all its parent attack steps must be compromised for an attacker to reach this step. An asset may also feature defenses (represented by #). The sum of attack paths is the attack/defense graph used for the attack simulation. Also, assets can inherit from each other, which means that an inherited asset inherits all attack steps and defenses of its parent asset (unless explicitly stated otherwise).

### 3.2. A MAL-based Language for Enterprise Systems

Based on the MAL framework, enterpriseLang [5] was designed for modeling attacks on enterprise IT systems. The enterpriseLang metamodel visualizing its assets and associations can be found in Figure 1. In total, enterpriseLang contains 22 assets, including 12 main assets and 10 inherited assets, which represent the fundamental components in enterprise systems.

The attack steps and defenses for each asset were extracted from the MITRE Enterprise ATT&CK Matrix[4]. The links between attack steps and defenses reflect possible transitions of an attacker among different assets. By using available tools (i.e., securiCAD [22]), enterpriseLang enables attack simulations on its system model instances, and the simulation results can be used to assess the current state of a system (e.g., the most vulnerable asset within the system) and foster a higher degree of resilience against cyber attacks by simulating the impact of new defenses and architectural changes.

[4]https://attack.mitre.org/

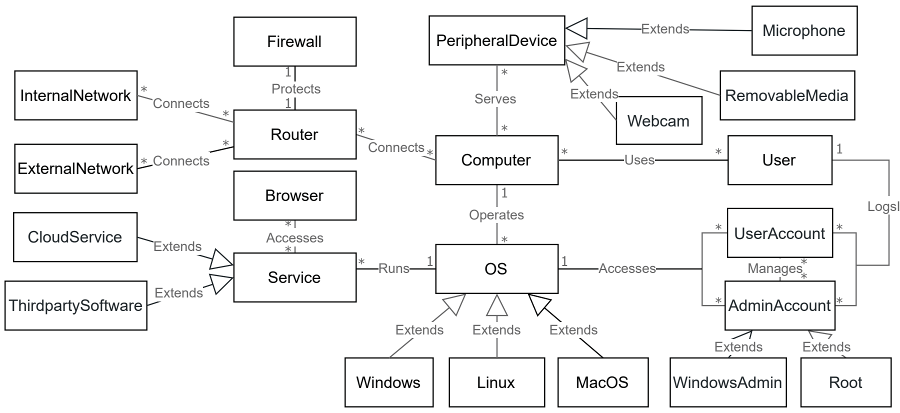

**Figure 1:** The enterpriseLang metamodel containing assets and associations [5, 21].

To evaluate enterpriseLang, test cases are developed to check if the attack simulations executed by enterpriseLang behave as expected, and attacks and potential defenses are modeled accurately.

## 4. Research Method

enterpriseLang was developed according to the DSR guidelines of Peffers et al. [23]. To ensure the correct functionality of enterpriseLang and foster better quality, we investigate if the language development process follows the established DSL design guidelines as a qualitative assessment. The evaluation process of enterpriseLang corresponds to the fifth step of the overall design process [23]. According to Peffers et al. [23], the evaluation step involves observing and measuring how successfully an artifact is created as a solution for the identified problem. Five methods can be used to evaluate the designed artifacts, namely, observations, analysis, experiments, tests, and descriptions [12].

To evaluate if a design artifact can achieve DSR goals and objectives, the evaluation process can be divided into two dimensions: 1) the functional purpose of the evaluation, and 2) the paradigm of the evaluation study [24]. The first dimension refers to *why to evaluate*, where the functional purpose of formative evaluations is to help improve the outcomes of the process, and the functional purpose of summative evaluations is to judge to what degree the outcomes match expectations. The second dimension refers to *how to evaluate*, where the artificial evaluation includes laboratory experiments, simulations, mathematical proofs, and naturalistic evaluation methods including case studies, field studies, and surveys.

To address these dimensions, we apply guidelines collected for developing DSLs [25] addressing the different stages of action design research (ADR) [26], as shown in Figure 2. The guidelines were systematically collected and structured based on an ongoing work [25], which are established DSL design guidelines and can describe the best practices for designing DSLs of high quality. For analytical evaluation of enterpriseLang and focusing on static analysis, we propose to compare its development process to the aforementioned set of DSL guidelines as a qualitative assessment. To ensure the correct functionality of the attack simulations produced

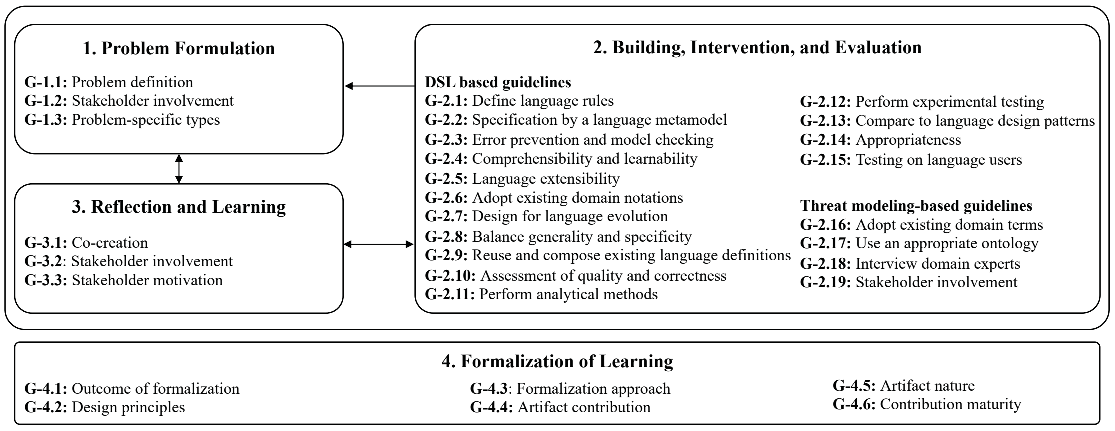
1. Problem Formulation
G-1.1: Problem definition
G-1.2: Stakeholder involvement
G-1.3: Problem-specific types
3. Reflection and Learning
G-3.1: Co-creation
G-3.2: Stakeholder involvement
G-3.3: Stakeholder motivation
2. Building, Intervention, and Evaluation
DSL based guidelines
G-2.1: Define language rules
G-2.2: Specification by a language metamodel
G-2.3: Error prevention and model checking
G-2.4: Comprehensibility and learnability
G-2.5: Language extensibility
G-2.6: Adopt existing domain notations
G-2.7: Design for language evolution
G-2.8: Balance generality and specificity
G-2.9: Reuse and compose existing language definitions
G-2.10: Assessment of quality and correctness
G-2.11: Perform analytical methods
G-2.12: Perform experimental testing
G-2.13: Compare to language design patterns
G-2.14: Appropriateness
G-2.15: Testing on language users
Threat modeling-based guidelines
G-2.16: Adopt existing domain terms
G-2.17: Use an appropriate ontology
G-2.18: Interview domain experts
G-2.19: Stakeholder involvement
4. Formalization of Learning
G-4.1: Outcome of formalization
G-4.2: Design principles
G-4.3: Formalization approach
G-4.4: Artifact contribution
G-4.5: Artifact nature
G-4.6: Contribution maturity

**Figure 2:** DSL guidelines adapted to evaluate MAL-based DSLs [25].

**Table 1**
Addressed Guidelines in enterpriseLang

| ADR stage | Addressed in enterpriseLang |
|---|---|
| Stage 1 | G-1.1; G-1.3 |
| Stage 2 | G-2.1; G-2.15; G-2.16 |
| Stage 3 | - |
| Stage 4 | G-4.2 |

by enterpriseLang, we propose to use a structured testing method as a quantitative assessment (developing enterpriseLang is similar to developing source code). Specifically, we calculate the test coverage of enterpriseLang using the method proposed by Hersén et al. [3]. The calculated test coverage percentage reflects what parts of the language that have not been fully tested yet and thus may function wrongly, and the results would indicate if more test cases are needed.

## 5. Qualitative Assessment

In this section, we related the DSL guidelines presented in Figure 2 to the development of enterpriseLang as a qualitative assessment, and the result is shown in Table 1.

**Stage 1 – Problem Formulation** Regarding Stage 1, enterpriseLang was developed to assess the security level of enterprise systems and support analysis of the security settings that can be implemented to secure the system effectively (G-1.1). The problem that enterpriseLang tries to solve can be categorized as a problem-specific type (G-1.3).

**Stage 2 – Building, Intervention, and Evaluation** Regarding Stage 2, enterpriseLang was designed based on the MITRE ATT&CK Matrix and tried to cover all the adversary techniques and mitigations contained within the matrix (G-2.1), while it was not further detailed for a specific enterprise system. Therefore, it adopts existing domain terms from the MITRE ATT&CK Matrix for enterprise systems (G-2.16), and there was no direct stakeholder involvement. Finally, regarding the evaluation of enterpriseLang, both unit tests and integration tests (G-2.15) were used to ensure the correct functionality of the language and demonstrated by two real-world attack cyber attacks.

**Stage 3 – Reflection and Learning** Because there were no activities conducted between the language developers and stakeholders and no learning activities, Stage 3 has not been addressed in the development of enterpriseLang.

**Stage 4 – Formalization of Learning** Regarding Stage 4, enterpriseLang was designed following the DSR guidelines (G-4.2) with the following steps: 1) Identify Problem & Motivate, 2) Define Objectives, 3) Design & Development, 4) Demonstration 5) Evaluation, and 6) Communication.

To reflect on the development process of enterpriseLang, there is an opportunity for improvement, where the guidelines that are not fulfilled by enterpriseLang should be further investigated. Especially, stakeholders should be involved during the development process of enterpriseLang, which could help to increase the usability of enterpriseLang and the actual application of security measures for stakeholders.

## 6. Quantitative Assessment

To validate enterpriseLang, test cases were designed to check its desired behavior. The tests confirm that the attack simulations run by enterpriseLang behave as expected and that attacks and potential mitigations are modeled accurately. However, to what degree these test cases cover the completeness of the language has not been evaluated.

Hence, we apply the test coverage method proposed by Hersén et al. [3], where common structural software testing methods were transferred to work with threat models and the generated simulation data. The statement coverage, branch coverage, and interface-based input domain modeling in source code testing [27] were transferred to attack step coverage, edge coverage, and defense coverage. A metric for asset coverage was also proposed for computing the percentage of compromised assets during an attack simulation [3].

The attack step coverage refers to the percentage of the compromised attack steps covered in an attack simulation. The asset coverage refers to the percentage of compromised assets during an attack simulation, where full asset coverage represents that all the attack steps related to the asset are compromised, and partial asset coverage considers that at least one attack step related to the asset is compromised. Moreover, the edge coverage refers to the ratio between the compromised edges (linking either attack steps to each other or a defense to an attack step) and the number of edges in the modeled system. To calculate the defense coverage, enabling and disabling defenses may result in different outcomes in multiple executions of an attack simulation, where the preconditions of the attacker do not change, and also influence the result whether certain attack steps or edges are compromised.

For the attack scenario presented in Figure 3, given that the attacker has initial access to *phishing* and all defenses (i.e., *exploitationForClientExecution* and *userTraining*) are disabled, four attack steps (*phishing, spearphishingLink, userExecution, exploitationForClientExecution*) and the three edges between them are compromised; when all the defenses are enabled, only the attack steps *phishing* and *spearphishingLink* can be reached, and only the edge between them

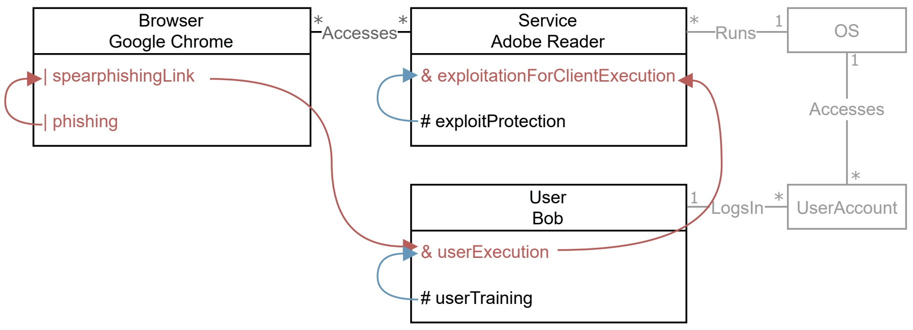

**Figure 3:** Graphical representation of the threat model used for the example coverage calculations.

**Table 2**
Attack Step Coverage

| | Defense enabled | Defense disabled | ∪ |
|---|---|---|---|
| Compromised attack steps | 2 | 4 | 4 |
| Number of attack steps | 4 | 4 | 4 |
| Attack step coverage | 50% | 100% | 100% |

**Table 3**
Edge Coverage

| | Defense enabled | Defense disabled | ∪ |
|---|---|---|---|
| Compromised edges | 1 | 3 | 3 |
| Number of edges | 5 | 5 | 5 |
| Edge coverage | 20% | 60% | 60% |

**Table 4**
Asset Coverage

| | Defense enabled | Defense disabled | ∪ |
|---|---|---|---|
| Compromised asset (partial) | 1 | 3 | 3 |
| Compromised asset (full) | 1 | 1 | 1 |
| Number of assets | 3 | 3 | 3 |
| Partial asset coverage | 33.3% | 100% | 100% |
| Full asset coverage | 33.3% | 33.3% | 33.3% |

is compromised. In this scenario, the attack step coverage is 100% (see Table 2) and the edge coverage is 60% (see Table 3). Moreover, the overall partial asset coverage is 100%, and the full asset coverage is 33.3% (see Table 4). Moreover, the defense coverage is 50% because only the cases that both of the two defenses are enabled and disabled at the same time are considered.

So far, the test coverage result of enterpriseLang is shown in Table 5. The defense coverage is approximately zero, because the created test cases only considered the case that all the defense

**Table 5**
Test Coverage of enterpriseLang

| | |
|---|---|
| Partial asset coverage | 94.7% |
| Full asset coverage | 10.5% |
| Attack step coverage | 44.4% |
| Edge coverage | 64.2% |
| Defense coverage | $\approx$ 0% |

values are false and tried to form longer attack paths. However, simply running simulations by enabling and disabling each defense may produce misleading results [3]. Therefore, different defenses should be enabled to investigate how the opportunities of an attack to be successful change according to the presence of certain defenses.

## 7. Discussion

Based on the qualitative assessment result, the guidelines not fulfilled by enterpriseLang can guide us with its future improvements. For example, G-2.9 is not fulfilled because enterpriseLang was not designed based on an existing DSL. A possible way to follow this guideline would be to redesign enterpriseLang so that it embeds an existing language, for example, coreLang [16], which contains the most common IT entities and attack steps. We would thus solely implement specific capabilities for enterprise systems that are not covered by coreLang. In addition, based on the quantitative assessment result, more test cases should be created for the missing parts reflected by the test coverage value. For example, the `Router` asset was covered by any test case, because in enterpriseLang there is no attack step contained in this asset. Therefore, future work includes exploring enterprise system-related attacks to enrich enterpriseLang.

In addition, our method should not be used in isolation, as we solely cover structural aspects similar to white box testing. Consequently, we are only able to identify design flaws from the developer's point of view. This leads to the fact that the users' perspective is not reflected and, thus, missing functionality cannot be discovered. To overcome this, we previously facilitated real-world attacks [5] to ensure that those can be modeled and simulated with enterpriseLang. Moreover, consulting experts and conducting surveys (e.g., [25]) could also be done in the future to further validate the applicability.

The proposed method could also be generalized and used for evaluating other MAL-based DSLs because they have the same need for evaluating the effectiveness of their languages [28], and further research is needed to generalize our proposal or to confirm that it works as is.

## 8. Conclusion

In this paper, we propose an approach to assess the quality of threat modeling languages and illustrate its application on enterpriseLang, to see if its development process follows the principles of good language design, and whether to what extent enterpriseLang functions as expected. The qualitative assessment result could foster better quality of enterpriseLang by fulfilling the missing DSL design guidelines, such as stakeholder involvement. Moreover, through creating more test cases and achieving higher test coverage, the correct functionality of enterpriseLang can be ensured.

## Acknowledgments

This project has received funding from the European Union's H2020 research and innovation programme under the Grant Agreement No. 832907.

## References

[1] W. Xiong, R. Lagerström, Threat modeling - a systematic literature review, Computers & Security 84 (2019) 53–69.

[2] K. Tuma, G. Calikli, R. Scandariato, Threat analysis of software systems: A systematic literature review, Journal of Systems and Software 144 (2018) 27–294. URL: https://www.sciencedirect.com/science/article/pii/S0164121218301304.

[3] N. Hersén, S. Hacks, K. Fögen, Towards measuring test coverage of attack simulations, in: Enterprise, Business-Process and Information Systems Modeling, Springer International Publishing, 2021, pp. 303–317.

[4] P. Johnson, R. Lagerström, M. Ekstedt, A meta language for threat modeling and attack simulations, in: 13th International Conference on Availability, Reliability and Security, ACM, 2018, p. 38.

[5] W. Xiong, E. Legrand, O. Åberg, R. Lagerström, Cyber security threat modeling based on the MITRE Enterprise ATT&CK Matrix, Software and Systems Modeling (2021). URL: https://doi.org/10.1007/s10270-021-00898-7.

[6] G. Karsai, H. Krahn, C. Pinkernell, B. Rumpe, M. Schindler, S. Völkel, Design guidelines for domain specific languages, in: 9th OOPSLA Workshop on Domain-Specific Modeling (DSM' 09), 2009, pp. 1–7.

[7] I. Poltronieri Rodrigues, M. de Borba Campos, A. F. Zorzo, Usability evaluation of domain-specific languages: A systematic literature review, in: Human-Computer Interaction. User Interface Design, Development and Multimodality, Springer, 2017.

[8] I. Gibbs, S. Dascalu, F. C. Harris, Jr., A separation-based ui architecture with a dsl for role specialization, Journal of Systems and Software 101 (2015) 69–85.

[9] D. Albuquerque, B. Cafeo, A. Garcia, S. Barbosa, S. Abrahão, A. Ribeiro, Quantifying usability of domain-specific languages: An empirical study on software maintenance, Journal of Systems and Software 101 (2015) 245–259.

[10] A. Barisic, V. Amaral, M. Goulão, Usability evaluation of domain-specific languages, in: Quality of Information and Communications Technology, 2012, pp. 342–347.

[11] J. Venable, J. Pries-Heje, R. Baskerville, A comprehensive framework for evaluation in design science research, in: Design Science Research in Information Systems. Advances in Theory and Practice, Springer, 2012, pp. 423–438.

[12] A. R. Hevner, S. T. March, J. Park, S. Ram, Design science in information systems research, MIS Quarterly 28 (2004) 75–105.

[13] Ø. Haugen, P. Mohagheghi, A multi-dimensional framework for characterizing domain specific languages, in: Proc. of the 7th OOPSLA Workshop on Domain Specific Modeling, 2007, pp. 1–10.

[14] R. Scandariato, K. Wuyts, W. Joosen, A descriptive study of microsoft's threat modeling technique (2013).

[15] K. Wuyts, R. Scandariato, W. Joosen, Empirical evaluation of a privacy-focused threat modeling methodology, Journal of Systems and Software 96 (2014) 122–138.

[16] S. Katsikeas, S. Hacks, P. Johnson, M. Ekstedt, R. Lagerström, J. Jacobsson, M. Wällstedt, P. Eliasson, An attack simulation language for the it domain, in: Graphical Models for Security, Springer International Publishing, 2020, pp. 67–86.

[17] S. Hacks, S. Katsikeas, E. Ling, R. Lagerström, M. Ekstedt, powerlang: a probabilistic attack simulation language for the power domain, Energy Informatics 3 (2020) 1–17.

[18] S. Katsikeas, P. Johnson, S. Hacks, R. Lagerström, Probabilistic modeling and simulation of vehicular cyber attacks: An application of the meta attack language, in: ICISSP 2019, 2019.

[19] A. Vernotte, P. Johnson, M. Ekstedt, R. Lagerström, In-depth modeling of the unix operating system for architectural cyber security analysis, in: 2017 IEEE 21st International Enterprise Distributed Object Computing Workshop (EDOCW), IEEE, 2017, pp. 127–136.

[20] T. Sommestad, M. Ekstedt, H. Holm, The cyber security modeling language: A tool for assessing the vulnerability of enterprise system architectures, IEEE Systems Journal 7 (2013) 363–373.

[21] W. Xiong, S. Hacks, R. Lagerström, A method for assigning probability distributions in attack simulation languages, Complex Systems Informatics and Modeling Quarterly (2021) 55–77. URL: https://doi.org/10.7250/csimq.2021-26.04.

[22] M. Ekstedt, P. Johnson, R. Lagerström, D. Gorton, J. Nydrén, K. Shahzad, Securi cad by foreseeti: A cad tool for enterprise cyber security management, in: 2015 IEEE 19th International Enterprise Distributed Object Computing Workshop (EDOCW), IEEE, 2015, pp. 152–155.

[23] K. Peffers, T. Tuunanen, M. A. Rothenberger, S. Chatterjee, A design science research methodology for information systems research, Journal of Management Information Systems 24 (2007) 45–77.

[24] J. Venable, J. Pries-Heje, R. Baskerville, Feds: a framework for evaluation in design science research 25 (2016) 77–89.

[25] A. Author(s), Towards a systematic method for developing meta attack language instances, in: submitted, 2021.

[26] M. K. Sein, O. Henfridsson, S. Purao, M. Rossi, R. Lindgren, Action design research, MIS quarterly 35 (2011) 37–56.

[27] P. Ammann, J. Offutt, Introduction to Software Testing: 2nd Edition, Cambridge University Press, 2016.

[28] J. Horkoff, F. B. Aydemir, F.-L. Li, T. Li, J. Mylopoulos, Evaluating modeling languages: An example from the requirements domain, in: Conceptual Modeling, Springer, 2014, pp. 260–274.