Enhancing Threat Model Validation: A White-Box Approach based on Statistical Model Checking and Process Mining

Roberto Casaluce[1,2], Andrea Burratin[3], Francesca Chiaromonte[1,4], Alberto Lluch Lafuente[3] and Andrea Vandin[1,3,*]

[1]Institute of Economics and L'EMbeDS, Sant'Anna School of Advanced Studies, Pisa, Italy
[2]Department of Computer Science, University of Pisa, Italy
[3]DTU Technical University of Denmark, Lyngby, Denmark
[4]Dept. of Statistics and Huck Institutes of the Life Sciences, Penn State University, USA

Abstract

Our method addresses the challenge of validating threat models by comparing actual behavior with expected behavior. Statistical Model Checking (SMC) is frequently the more appropriate technique for validating models, as it relies on statistically relevant samples to analyze systems with potentially infinite state spaces. In the case of black-box systems, where it is not possible to make complete assumptions about the transition structure, black-box SMC becomes necessary. However, the numeric results of the SMC analysis lack insights on the model's dynamics, prompting our proposal to enhance SMC analysis by incorporating visual information on the behavior that led to a given estimation. Our method improves traditional model validation using SMC by enriching its analyses with Process Mining (PM) techniques. Our approach takes simulated event logs as inputs, and uses PM techniques to reconstruct an *observed model* to be compared with the graphical representation of the *original model*, obtaining a *diff model* highlighting discrepancies among expected and actual behavior. This allows the modeler to address unexpected or missing behaviors. In this paper we further customize the diff model for aspects specific to threat model analysis, incorporating features such as new colored edges to symbolize an attacker's initial assets and a automatic fix for simple classes of modeling errors which generate unexpected deadlocks in the simulated model. Our approach offers an effective and scalable solution for threat model validation, contributing to the evolving landscape of risk modeling and analysis.

Keywords

Threat models, Probabilistic modeling, Statistical model checking, Process mining, Attack-defense trees

1. Introduction

Our work introduces an approach to validate threat models by providing accessible tools for observing and comparing the actual behavior of a model with the expected one. When dealing with quantitative aspects in model behavior validation, exact or statistical analysis techniques are commonly employed. Exact techniques, while providing precise quantitative properties, may face challenges in handling complex models due to the state-space explosion problem [1]. On the contrary, statistical analysis techniques, like Statistical Model Checking (SMC) [2], rely on statistically relevant samples, offering a viable solution for analyzing complex systems with potentially infinite state spaces, albeit providing statistically reliable estimations rather than exact results. In the case of black-box systems, where it is not possible to assume the complete transition structure, employing a black-box SMC [3] becomes necessary for analyzing dynamics without prior system knowledge, yielding numerical estimates and plots. However, these come without an explanation for the obtained numerical results, i.e., we do not get an explanation on the observed behavior and on why it led to such results.

First International Workshop on Detection And Mitigation Of Cyber attacks that exploit human vuLnerabilitiES (DAMOCLES) - Workshop at AVI '24, Arenzano (Genoa), Italy, June 4th 2024

*Corresponding author.
roberto.casaluce@santannapisa.it (R. Casaluce); andbur@dtu.dk (A. Burratin); francesca.chiaromonte@santannapisa.it (F. Chiaromonte); albl@dtu.dk (A. L. Lafuente); andrea.vandin@santannapisa.it (A. Vandin)
0000-0002-5786-9167 (R. Casaluce); 0000-0002-0837-0183 (A. Burratin); 0000-0002-9421-8566 (F. Chiaromonte); 0000-0001-7405-0818 (A. L. Lafuente); 0000-0002-2606-7241 (A. Vandin)

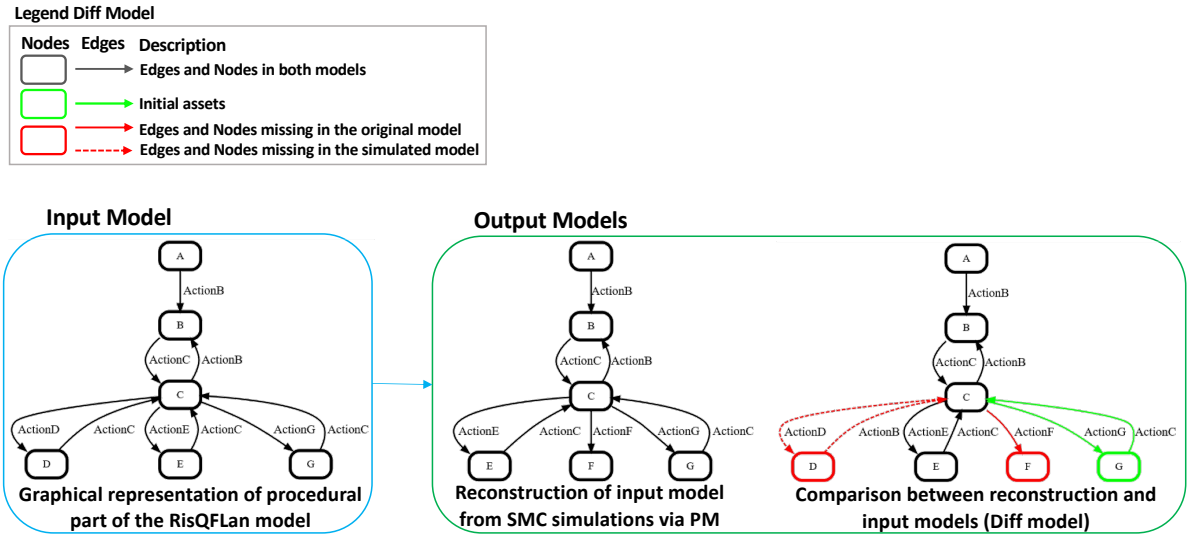

Figure 1: Illustration of the input and output generated through the methodology introduced in this paper. The input model undergoes simulation using SMC techniques, and the resulting traces are employed to generate a model. Subsequently, this new model is compared to the original, and a comprehensible output is provided to the modeler, highlighting the discernible differences.

Consider a classic threat model depicting a thief's strategies for robbing a bank [1], analyzed using SMC to quantify the probability of a successful robbery, resulting in an unexpected low value. This prompts questions about the reasons behind the extreme value and whether it aligns with the model's intended dynamic or whether it signifies a bug. The numeric results do not provide any insights into why the analysis yielded those results or if there are issues in the model. On this basis, we formulated our proposal [4, 5]. Our approach enhances SMC analysis results by automatically incorporating explicit visual information highlighting discrepancies between the model and specified criteria. This not only aids in model refinement but also serves as a comprehensive testing method, allowing experimentation with various settings within the same model structure to assess simulated model correctness. To be more specific, our method involves enriching SMC analyses by post-processing and analyzing SMC byproducts, such as log files stored while performing the simulations. We use data-driven techniques like Process Mining (PM). PM helps to identify process patterns, bottlenecks, and other model issues through visualizing activity flow [6].

Figure 1 depicts an abstract example of a model validated using our method. The input model represents an abstract model with different states, and actions used to move between those states. The simulator starts from the node A by choosing the corresponding action to move to other states of the model. To validate the output of an SMC, without our methodology, a modeler can only use the obtained numerical results, such as the probability of reaching a certain node in the model. Conversely, our approach enables the modeler to scrutinize simulation results in a model-to-model language, as opposed to model-to-numbers. Our method takes as inputs the simulated event logs, stored while performing the simulations necessary for statistical model checking. These logs are mined with Process Mining techniques and, then, used to reconstruct the first "Output Model". The rightmost model depicts the graph obtained by comparing the "Input Model" and the reconstructed one, highlighting the differences between the original model and the simulated one. The differences highlighted in the rightmost "Output Model," i.e., the diff model, enable the modeler to quickly identify issues, such as unexpected or missing behaviors.

Several studies on risk modeling and analysis utilize SMC to evaluate the properties of a threat model [7, 8]. Additionally, there are other works that perform threat modeling analysis using PM - refer to [9] for an overview of works on cybersecurity and PM. Nevertheless, to the best of our knowledge, the first preliminary work in which SMC results were enhanced with PM techniques to aid the modeler in visually identifying unwanted behaviors was presented in our previous works [4]. We further extended

such work in [5] by automatically computing a diff model based on the original model specification language. The latter study demonstrated the effectiveness and scalability of our method, showcasing its applicability in both Product Line Engineering (PLE), and threat analysis domains. In the current work, we expand upon the diff model introduced in [5] and further customize it for threat model analysis. The updated diff model incorporates a new type of edge colored in green, symbolizing the initial assets possessed by the attacker before initiating the attack. For instance, an initial asset could be one of the two combinations required to open the bank vault, held by a bank clerk who decides to rob the bank. Our enhanced diff model takes into account the transitions and states in the model, depicting the attacker's initial assets by singling them out from other types of nodes and edges. This distinction helps to separate them from other transitions that are missing due to an error in the formal model. Additionally, we have incorporated a component in our method that is able to automatically fix specific classes of issues discovered in the modeler, in particular when unexpected deadlocks are detected in the simulated model. In Section 4, we conducted experiments on the RobBank model [1] to illustrate the effectiveness of our method, and we exemplify the automatic fixes proposed by our framework for the deadlocks identified in the original model.

The remainder of the paper is structured as follows: Section 2 introduces necessary background material, along with a brief introduction to the RobBank model as a running example. Following this, Section 3 presents our methodology for the diff model and the approach to fixing deadlocks. Section 4 validates our method on the presented case study. Finally, Section 5 concludes the paper and outlines future work.

We make available the use of our updated tool with all the models, the full-size figures and the replication material for this paper are available at https://t.ly/FVs6Z.

2. Background

2.1. Modeling in RisQFLan

In the present work, we use RisQFLan [1], a domain-specific instantiation of QFLan [10, 11, 12] applied to the risk modeling and analysis domain. RisQFLan provides more capabilities than other risk analysis tools since it allows for quantitative analysis of the probability of the attack strategies in security threat models. RisQFLan allows for SMC analysis and the Probability Model Checking (PMC) technique. In RisQFLan, the probabilistic simulator and SMC (MultiVeStA) [13] interact to estimate the properties of the model statistically.

For the experiments, we adopt the threat model from [1], outlining potential attack strategies for a bank robbery. The authors emphasized that the model has infinite state space which necessitates using SMC for analysis due to the impracticality of exhaustive exploration. RisQFLan models consist of a declarative part (Attack-defence tree) and a procedural part (probabilistic attacker). Refer to [1] for a detailed presentation of RisQFLan. The Attack-defence tree (ADT) illustrates the potential attack strategies of an attacker to exploit a system. Attack trees are widely used in several domains like, e.g., defense (e.g., their use is recommended by NATO [14]), aerospace [15], or safety-critical cyber-physical systems [16].

Figure 2 illustrates the ADT depicting the root attack goal of robbing the bank and sub-attacks like `OpenVault` and `BlowUp`. `OpenVault` refines into `LearnCombo` and `GetToVault`, connected by an AND-relation. `LearnCombo` further breaks down into three nodes, `FindCode`, governed by a 2-out-of-3-relation. An attack node succeeds only if its refinements permit it. *Defense* nodes, such as `Memo` and `LockDown`, decrease attack success probabilities. *Countermeasures*, like `LockDown`, are dynamic defenses activated by monitored attacks. The ADT suggests two robbery strategies: opening the vault or blowing it up, both requiring access to the vault. Opening the vault necessitates learning the combination, itself dependent on discovering at least two codes.

In addition, RisQFLan enables users to specify probabilistic attacker behavior, thereby facilitating specific analyses on different types of attackers. Figure 3 illustrates the probabilistic attacker behavior in the original RobBank model. The capacity to define various attacker types is beneficial for evaluating

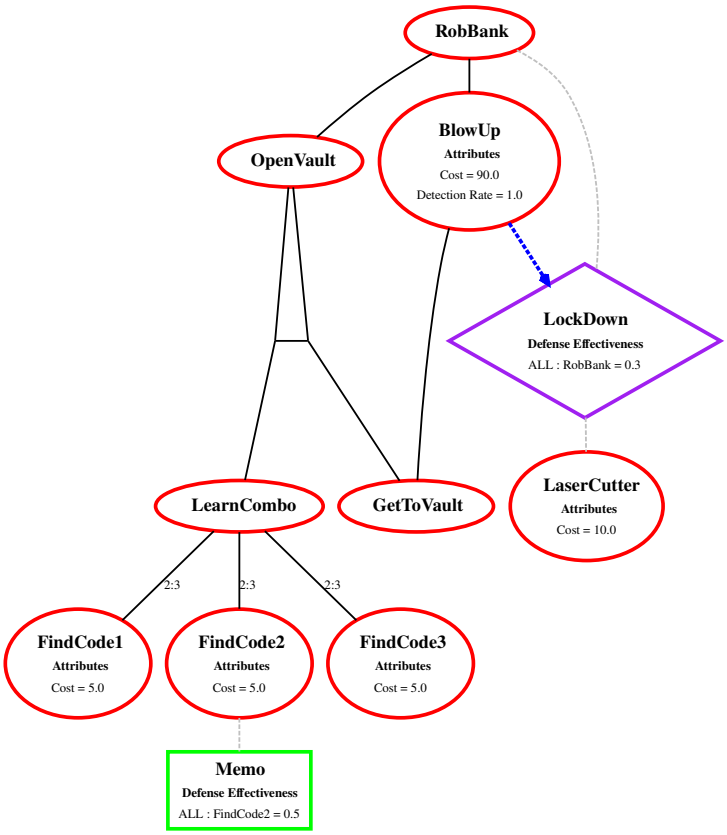

Figure 2: Attack-Defence tree RobBank model (Source)

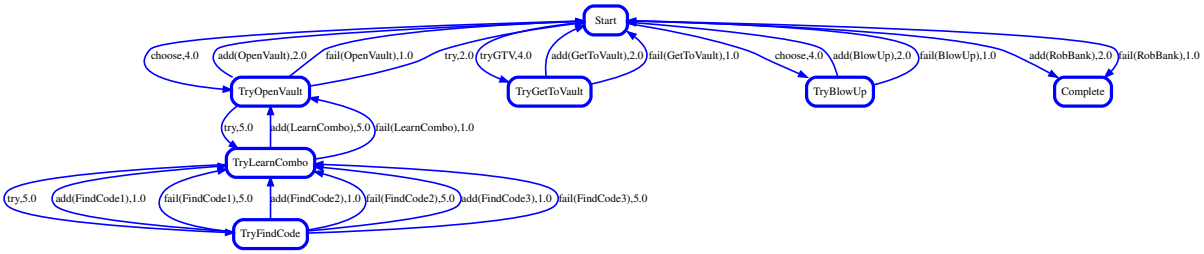

Figure 3: Graphical representation of the procedural part of the RisQFLan model of the original RobBank model.

system vulnerabilities and allocating resources for system protection. Furthermore, this probabilistic attacker behavior in RisQFLan may enable the incorporation of human factors into the model. For instance, this can be achieved by modifying probabilities associated with specific vulnerabilities in the system due to user interaction, such as clicking on infected links or falling for phishing emails. Alternatively, users may design attacker behaviors that account for these human factors. In RisQFLan, the attacker behavior is modeled as a rated transition system. The transitions are labeled with the actions executed, and the rates used to compute the probability of completing one of the actions. These transitions also include the so-called guards that represent the conditions in which an action can be executed and the possible effects of executing a specific action. Besides the transition guards on a specific action, RisQFLan supports quantitative constraints that allow the imposition of limitations on the attacker's behavior by adding a total cost that an attacker can spend during the attack on a system. Any attempts to perform an attack will cost the attacker even if the attack fails, and they can keep trying to complete the attack until the limit imposed by the total cost is not reached. Once the attacker has run out of resources, they cannot try any other attacks on the system. These quantitative constraints allow us to test the system against different types of attackers with different resources.

```
1
2   begin attributes
3   Cost = {LaserCutter = 10, BlowUp = 90, FindCode1 = 5, FindCode2 = 5, FindCode3 = 5}
4   end attributes
5   begin quantitative constraints
6       { value(Cost) <= 100 }
7   end quantitative constraints
8   begin actions
9       tryAction tryGTV choose
10  end actions
11
12  begin attacker behaviour
13      begin attack
14          attacker = Thief
15          states = Start, TryOpenVault, TryLearnCombo, TryFindCode, TryGetToVault, TryBlowUp, Complete
16          transitions =
17          Start - (succ(RobBank), 2, allowed(RobBank)) -> Complete,
18          Start - (fail(RobBank), 1, allowed(RobBank)) -> Complete,
19          //Attempt to Get to the vault attempt
20          Start -(tryGTV, 4, !has(GetToVault)) -> TryGetToVault,
21          TryGetToVault -(succ(GetToVault) , 2, {AttackAttempts = AttackAttempts + 1}) -> Start,
22          TryGetToVault -(fail(GetToVault), 1, {AttackAttempts = AttackAttempts + 1}) -> Start,
23          //Attempt to Open the vault attempt
24          Start -(choose, 4) -> TryOpenVault,
25          TryOpenVault -(succ(OpenVault) , 2, {AttackAttempts = AttackAttempts + 1},has(LearnCombo) and
                has(GetToVault)) -> Start,
26          TryOpenVault -(fail(OpenVault), 1, {AttackAttempts = AttackAttempts + 1},has(LearnCombo) and
                has(GetToVault)) -> Start,
27          TryOpenVault -(tryAction , 2, has(LearnCombo) and !has(GetToVault)) -> Start,
28          TryOpenVault -(tryAction, 5, !has(LearnCombo)) -> TryLearnCombo,
29          //Attempt to Learn the combinations of he vault attempt
30          ...
31          //Attempt to Blow up the vault attempt
32          Start -(choose, 4) -> TryBlowUp,
33          ...
34      end attack
35  end attacker behaviour
```

Figure 4: Probabilistic attacker in RisQFLan

Figure 4 displays and exerts of textual representation of of the RobBank model in RisQFLan; the full model is discussed in [1]. We focus on relevant parts for our experiments. For instance, in Figure 4 line 3 are listed the cost for each attack node, and in line 6 quantitative constraints allow the attacker to complete the robbery, ensuring that the total cost will never exceed 100 EUR.

2.2. SMC

Black-box statistical model checking is a method employed in analyzing RisQFLan models using the tool MultiVeStA. This approach is applied to quantitative properties, such as the likelihood of robbing the bank. MultiVeStA conducts numerous probabilistic simulations, ensuring statistically reliable estimations of the specified property. It operates as a simulation-based technique, making no assumptions about the overarching model behavior. MultiVeStA, a black-box SMC tool, facilitates simultaneous statistical analyses of multiple properties, offering scalability and the generation of confidence intervals for user-specified parameters. For instance, when estimating the expected value E[X] for robbing the bank, MultiVeStA computes the mean x from n independent simulations within a confidence interval. It has been successful applied in diverse domains, including economic agent-based models [13], highly-configurable systems [11, 10], public transportation systems [17, 18], security risk modeling [1], lending pools in decentralized finance [19], business process modeling [20], crowd steering scenarios [21], and robotic scenarios with planning capabilities [22].

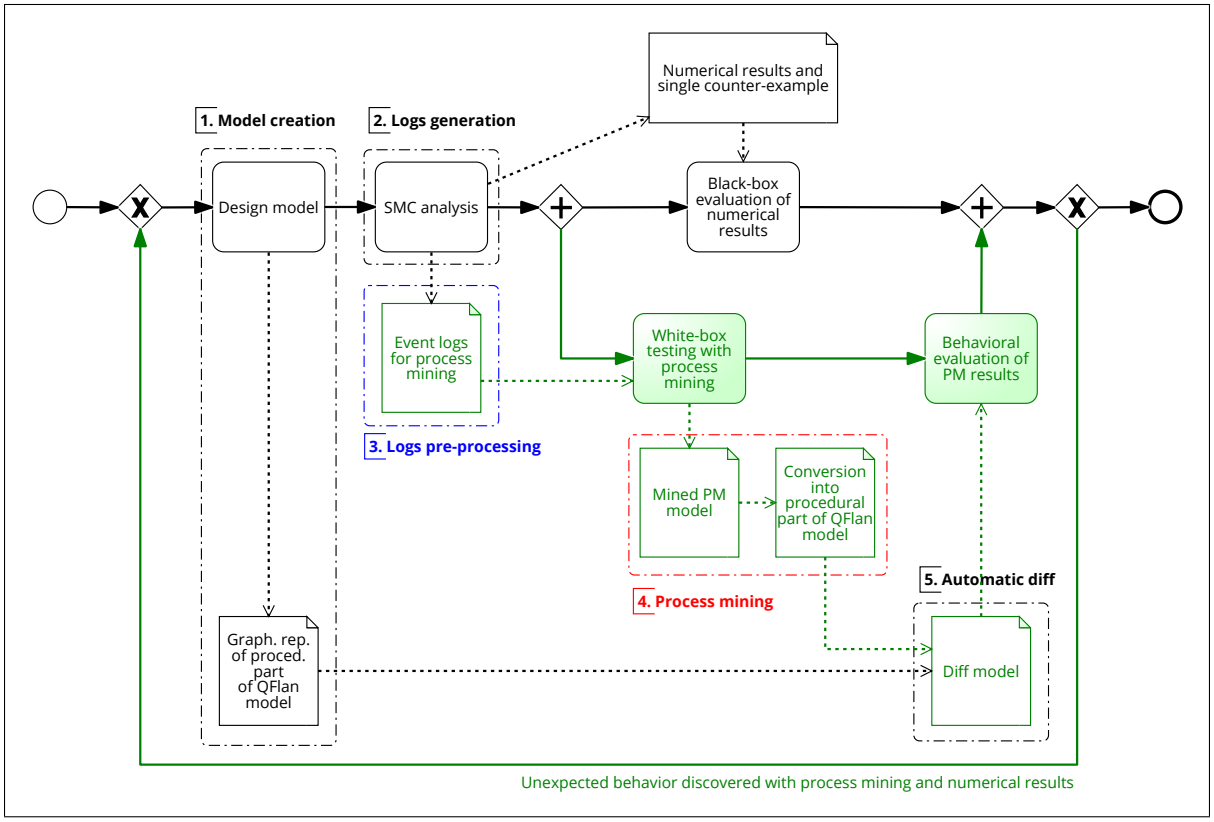

Figure 5: Our methodology combines effectively SMC and PM for automatic white-box behavioral validation - Source [5].

2.3. Process mining

This section provides a brief overview of the primary tasks involved in process-oriented data-driven techniques, commonly referred to as Process Mining (PM). PM serves as the interdisciplinary field that connects data-centric and model-based analysis techniques [6]. It utilizes real process executions to derive pertinent insights into how the observed behavior deviates from the intended behavior. PM encompasses three key activities: (control-flow) discovery, enhancement, and conformance checking. Discovery seeks to unveil an abstract representation of the executed process by consolidating all observed instances into a unified model. Once the process model is discovered, it can be enriched with additional information, such as the frequency of activity/path executions—an activity known as enhancement. Finally, conformance checking aims to identify cases where the actual executions deviate from the expected normative model. For the current work, we utilize discovery techniques to extract information from the execution traces obtained from SMC analyses. We then apply the mined model to evaluate how well the generated behavior aligns with the initial expectations.

In the following Section 3, we detail our method for constructing the updated diff model, which graphically highlights disparities between the simulated model's behavior and the model originally intended by the modeler. Additionally, we outline the steps employed to automatically resolve deadlocks identified in the simulated model.

3. Method

Our method enriches SMC with PM techniques to overcome the limitations of the classic SMC black-box validation which relies only on numerical results and counterexamples. PM allows us to incorporate a white-box analysis of the behavior of the model by leveraging mining techniques on simulated model executions. However, we go beyond a simple discovery algorithm applied to event logs and, instead, we

integrate these two techniques, i.e., SMC and PM, through a graphical component given in the model specification language to facilitate the automatic discovery of missing or undesirable behaviors in the model. Figure 5 illustrates our methodology presented in [5]. The steps employed to create the diff model are listed below:

- *1. Model creation* - Our methodology starts with creating the model where the modeler defines all the system components, e.g., attack nodes, a list of constraints, and its procedural part. RisQFLan will then automatically generate the corresponding Attack-defence Tree and a graphical representation of the procedural part of the RisQFLan model describing the attacker's behavior.
- *2. Log creation* - The second step consists of the simulation of the model in which the modeler chooses the properties of the modeler that will be evaluated using the SMC tool MultiVeStA. MultiVeStA has been extended with log capabilities in which two new functionalities enable the creation of an empty log file, invoked once per SMC analysis, and to add the rows to the log, invoked whenever an event (of interest) is to be recorded.
- *3. Log pre-processing* - In this step, we preprocess the event logs before applying PM techniques. To maintain the correct order and prevent the loss of information about transitions in the logs, we rename actions by adding names of origin and target states. This step helps avoid losing crucial information about the executed process.
- *4. Process mining* - After pre-processing the event logs, we mine them using the Heuristic Miner (HM) [23, 6] algorithm. Once the Mined PM model is discovered, we convert it from a PM model into a mined RisQFLan model (representing the procedural part of a RisQFLan model). In Figure 1, this corresponds to the first "Output Model". In this step, we return the names of the actions to their original ones which helps to compare the original RisQFLan model with the mined one – the "Input model" with the first "Output Model" in Figure 1.
- *5. Automatic diff* - The final step involves parsing graphical representations of the original RisQFLan model and the mined one to create a diff model, highlighting differences. The diff model includes common edges and nodes in both models colored in black, while edges and nodes unique to one model are highlighted in red. Dashed red edges indicate transitions present in the original model but missing in the mined one, suggesting constraints preventing those transitions. Continuous red edges represent transitions only in the mined model, potentially indicating simulator deadlock states or errors. Our methodology identifies and automatically fixes such bugs, providing opportunities to enhance the model beyond classic SMC black-box validation. Additionally, the diff model includes green edges and nodes, as depicted in Figure 1 in the rightmost "Output model," representing the initial nodes acquired by the attacker before initiating the system breach.

Figure 6 illustrates the steps employed in our method to automatically fix deadlocks in the simulated model. First, we reuse the mined PM model created by mining the simulated event logs and identify the transitions that end in a deadlock. We then add new transitions to resolve these issues. Next, we extract all the system components from the RisQFLan file of the original model, which are used in an empty template of the RisQFLan file. We fill in the template with the new transition system that addresses the deadlocks.

In the following section, we demonstrate how our method can identify unwanted behaviors in the model and provide an automatic fix for those issues identified in the model.

4. Experiments

4.1. Experiments on the original RobBank model

To showcase the capability of our approach in identifying undesired behaviors within this domain, we employ MultiVeStA for the analysis of the query presented in Figure 7. This directs MultiVeStA to assess the likelihood of success for eight different attacks (line 7 in Figure 7) at each simulation step

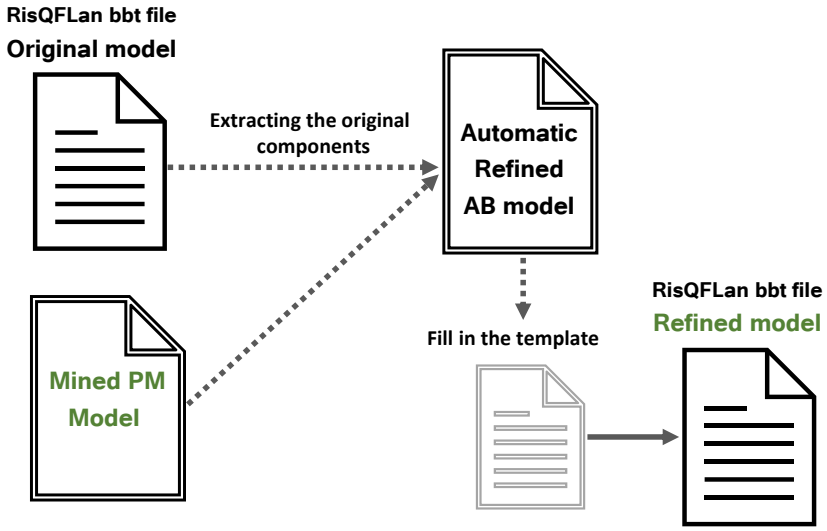

Figure 6: Automatic Refinement

```
1   begin init
2       Thief = {FindCode1, LaserCutter } // LockDown cannot be activated if we have LaserCutter
3   end init
4
5   begin analysis
6       query = eval from 1 to 100 by 1 :
7         {RobBank, OpenVault, BlowUp, LearnCombo, GetToVault, FindCode1, FindCode2, FindCode3, LockDown}
8       default delta = 0.1   alpha = 0.1   parallelism = 1
9       logs = "log_RobBank.csv"
10  end analysis
```

Figure 7: Initial setup and the query to invoke MultiVeStA RobBank model

Table 1
Numerical results experiments on the original RobBank model

Property	RobBank	OpenVault	BlowUp	LearnCombo	GetToVault	FindCode1	FindCode2	FindCode3	LockDown
Probability	0.19	0.14	0.11	0.44	0.47	1	0.16	0.37	0.0

ranging from 1 to 100. Similar to the approach outlined in [1], we presume that the attacker possesses a `LaserCutter` capable of bypassing the `Lockdown` defense and has already acquired the initial code for the vault (line 2 `FindoCode1`, Figure 7). Our goal is to demonstrate that our methodology effectively identifies issues in the model and prompts a way to fix them.

4.1.1. Results analysis on the original RobBank model.

MultiVeStA instructed the probabilistic simulator to execute 320 simulations. The analysis results for step 100 are summarized in Table 1. As expected, the probability of a complete lockdown is zero, attributed to the presence of the `LaserCutter`. Consequently, the likelihood of `FindCode1` is 1, as both already belong to the attacker's initial assets. The SMC results show that the probability of a successful root attack is 0.19 even with `LockDown` disabled. This represents a low percentage (approximately 19%) of simulations concluding with a successful bank robbery. Understanding why this occurs proves to be challenging through a mere black-box examination of numerical outcomes.

Illustrated in Figure 8 is the diff model generated by our methodology. It exhibits two red edges and a red node, two green edges while the remaining edges and nodes are black. The green edges represent transitions that the simulator never traversed because they were already included in the initial setup of the attacker (in Figure 7 line 2 `FindCode1` is already gained by the attacker). The red

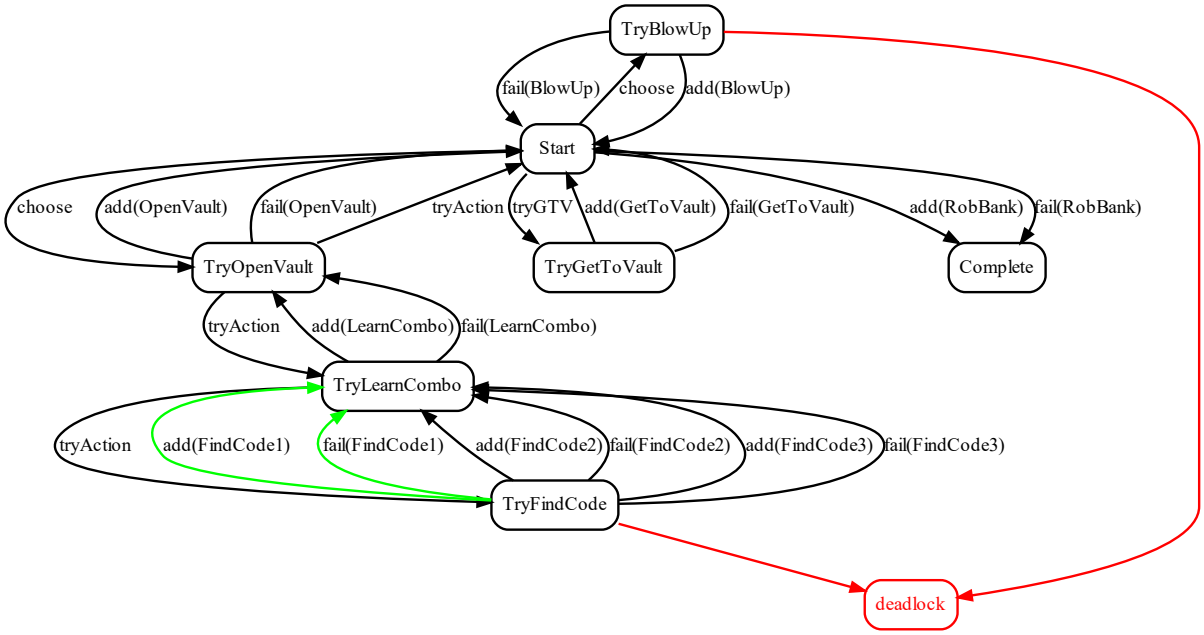

Figure 8: Diff model on the original RobBank model

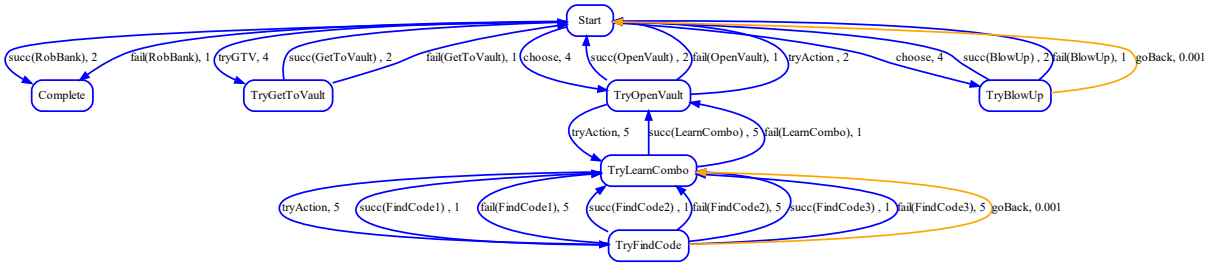

Figure 9: Refined Attacker behavior of the RobBank model

node signifies a unique addition to our methodology: a deadlock node, indicating simulations that ended unexpectedly due to no enabled transitions. Examining the two red edges reveals that some simulations ended unexpectedly in the states `TryFindCode` and `TryBlowUp`, diminishing the overall success probability.

Issue in the model, and an automatic fix. The model has an issue with states `TryFindCode` and `TryBlowUp`, as highlighted in Figure 8. These states can only transition to attempt attacks, and if the attacker lacks sufficient funds for the corresponding attack, they get stuck due to a maximum cost constraint of 100 EUR (line 6 in Figure 4). To address this, the modeler should add escape transitions in these nodes, leading back to their parent nodes with no cost, resolving the deadlock problem and ensuring a more accurate evaluation of properties. Using the mined model, we have implemented an automatic refinement of the model that adds an escape action in the presence of a deadlock due to violating the quantitative constraints.

4.2. Refiend model

In Figure 9, we enhance our model by introducing a `goBack` action, creating transitions from `TryBlowUp` to `Start` and from `TryFindCode` to `TryLearnCombo` in Figure 9. The refined attacker behavior, shown in orange, assigns a low weight of 0.001 to these transitions, encouraging the simulator to select them

Table 2
Numerical results experiments on the refined RobBank model

Property	RobBank	OpenVault	BlowUp	LearnCombo	GetToVault	FindCode1	FindCode2	FindCode3	LockDown
Probability	0.45	0.58	0.07	0.58	0.80	1	0.2	0.44	0.0

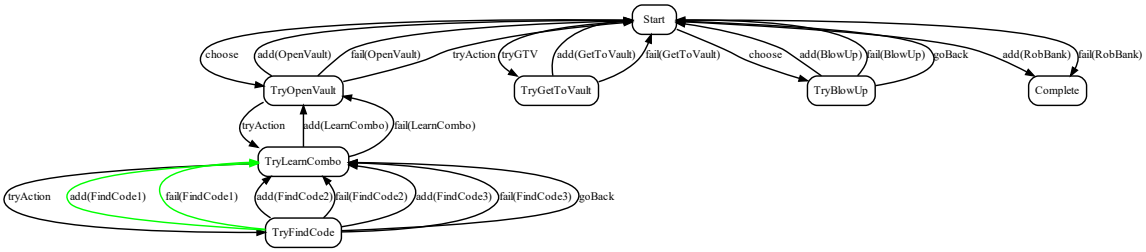

Figure 10: Diff model on the refined model

only when the other transitions are not allowed.[1]

4.2.1. Results on the refined model

Thanks to the introduction of these new transitions, the observed dynamics no longer manifest the previously discussed deadlocks. To ensure this outcome, we utilize the identical query as depicted in Figure 7. In this particular case, MultiVeStA necessitated the execution of 320 simulations. The numerical results are listed in Table 2. Except `LockDown`, which remains static at zero and `FindoCode1` which is again equal to 1, all other properties have exhibited an increment Now, the probability of successfully robbing the bank is approximately 45%. The resultant diff model is shown in Figure 10, where the absence of red edges or nodes means that the identified issues in the original model are fixed.

5. Conclusion and future research

In the present work, we have expanded our approach for validating simulation models, with a specific focus on threat model analysis, and we are now making our updated tool available for practitioners. The methodology involves a combination of simulation-based analysis techniques derived from Statistical Model Checking (SMC)[2] and process-oriented data-driven techniques from Process Mining (PM)[6]. More specifically, we employ PM to elucidate SMC analyses, generating a graphical representation of the system behavior as observed in the SMC simulations. This graphical representation has been enhanced from the original implementation by incorporating additional information beneficial for the modeler when examining the results of our method. In our experimental evaluation, we demonstrate that: (1) our methodology aids in identifying issues in the model and distinguishes them from other information that are not errors, i.e., transitions belonging to the initial assets of the attacker, and (2) it generates a refined model when errors are detected in the simulated model.

In future investigations, we will focus on developing an automated tool designed to construct a comprehensive probabilistic attacker behavior based on an attack tree. This tool will assist a modeler capable of describing a system using an attack tree but unfamiliar with creating a probabilistic attacker behavior in RisQFLan. The tool will derive the system rules governing the actions permitted or restricted for the attacker from the attack tree. For example, consider an attack on a parent node, feasible only when all its child nodes are acquired. In this scenario, the sequence of attacks initiated by the simulator should be restricted from gaining the parent node before obtaining all the child nodes. This restriction is crucial, particularly when considering, for instance, an attacker attempting a lateral movement step

[1]Figure 9 is the preview of the new attacker behavior model created with our automatic tool.

before gaining access to the system. Allowing such a sequence of attacks would make the model unrealistic. The new automated tool will generate a complete RisQFLan file with a realistic attacker behavior that understands the rules. The modeler can easily refine it based on the type of attacker they intend to simulate. Furthermore, as part of future research, we will aim to extend the use of PM techniques beyond discovering techniques to help the modeler validate a model. Indeed, our diff model assists the modeler in checking if the behavior of the simulated model corresponds to the behavior intended by the modeler in the original model. However, the diff model can only highlight mismatches between the two models without checking if the order of sequences of attacks is correct, which, as explained above, is crucial. To ensure that the model's behavior also follows the correct order in simulating the attack on the system, we need to extract all the possible paths allowed from the attack tree. We use these paths to mine a process model, and thanks to the use of conformance checking techniques, we can ensure that the simulated model does not include cases in which the actual behavior differs from that imposed by the attack tree.

References

[1] M. H. ter Beek, A. Legay, A. L. Lafuente, A. Vandin, Quantitative security risk modeling and analysis with RisQFLan, Computers & Security 109 (2021) 102381.

[2] G. Agha, K. Palmskog, A survey of statistical model checking, ACM Trans. Model. Comp. Simul. 28 (2018) 6:1–6:39.

[3] H. L. Younes, Probabilistic verification for "black-box" systems, in: CAV 2015, Springer, 2005, pp. 253–265.

[4] R. Casaluce, A. Burattin, F. Chiaromonte, A. Vandin, Process mining meets statistical model checking: Towards a novel approach to model validation and enhancement, in: C. Cabanillas, N. F. Garmann-Johnsen, A. Koschmider (Eds.), Business Process Management Workshops, Springer International Publishing, Cham, 2023, pp. 243–256.

[5] R. Casaluce, A. Burattin, F. Chiaromonte, A. L. Lafuente, A. Vandin, White-box validation of quantitative product lines by statistical model checking and process mining, Journal of Systems and Software 210 (2024) 111983. URL: https://www.sciencedirect.com/science/article/pii/S0164121224000268. doi:https://doi.org/10.1016/j.jss.2024.111983.

[6] W. M. van der Aalst, Process Mining, 2nd ed., Springer, 2016.

[7] H. Vallant, B. Stojanović, J. Božić, K. Hofer-Schmitz, Threat modelling and beyond-novel approaches to cyber secure the smart energy system, Applied Sciences 11 (2021) 5149.

[8] A. Jawad, J. Jaskolka, Analyzing the impact of cyberattacks on industrial control systems using timed automata, in: 2021 IEEE 21st international conference on software quality, reliability and security (QRS), IEEE, 2021, pp. 966–977.

[9] M. Macak, L. Daubner, M. F. Sani, B. Buhnova, Cybersecurity analysis via process mining: A systematic literature review, in: International Conference on Advanced Data Mining and Applications, Springer, 2022, pp. 393–407.

[10] A. Vandin, M. H. ter Beek, A. Legay, A. Lluch-Lafuente, QFLan: A Tool for the Quantitative Analysis of Highly Reconfigurable Systems, in: FM, 2018.

[11] M. H. ter Beek, A. Legay, A. Lluch-Lafuente, A. Vandin, A Framework for Quantitative Modeling and Analysis of Highly (Re)configurable Systems, IEEE Trans. Software Eng. 46 (2020) 321–345.

[12] M. H. ter Beek, A. Legay, A. Lluch-Lafuente, A. Vandin, Statistical analysis of probabilistic models of software product lines with quantitative constraints, in: D. C. Schmidt (Ed.), Proceedings of the 19th International Conference on Software Product Line, SPLC 2015, Nashville, TN, USA, July 20-24, 2015, ACM, 2015, pp. 11–15. URL: https://doi.org/10.1145/2791060.2791087. doi:10.1145/2791060.2791087.

[13] A. Vandin, D. Giachini, F. Lamperti, F. Chiaromonte, Automated and distributed statistical analysis of economic agent-based models, Journal of Economic Dynamics and Control (2022) 104458.

[14] Research and Technology Organisation of NATO, Improving Common Security Risk Analysis report, RTO Technical Report TR-IST-049, 2008.
[15] U.S. Department of Defense, Defense Acquisition Guidebook, Section 8.5.3.3, 2009. URL: https://bit.ly/3NJjs07.
[16] J. Hu, H. Niu, J. Carrasco, B. Lennox, F. Arvin, Fault-tolerant cooperative navigation of networked uav swarms for forest fire monitoring, Aerospace Science and Technology 123 (2022) 107494.
[17] S. Gilmore, M. Tribastone, A. Vandin, An analysis pathway for the quantitative evaluation of public transport systems, in: IFM, 2014.
[18] V. Ciancia, D. Latella, M. Massink, R. Paškauskas, A. Vandin, A tool-chain for statistical spatio-temporal model checking of bike sharing systems, in: ISOLA'17, 2017.
[19] M. Bartoletti, J. H. Chiang, T. Junttila, A. Lluch-Lafuente, M. Mirelli, A. Vandin, Formal analysis of lending pools in decentralized finance, in: T. Margaria, B. Steffen (Eds.), Leveraging Applications of Formal Methods, Verification and Validation. Adaptation and Learning - 11th International Symposium, ISoLA 2022, Rhodes, Greece, October 22-30, 2022, Proceedings, Part III, volume 13703 of *Lecture Notes in Computer Science*, Springer, 2022, pp. 335–355. URL: https://doi.org/10.1007/978-3-031-19759-8_21. doi:10.1007/978-3-031-19759-8_21.
[20] F. Corradini, F. Fornari, A. Polini, B. Re, F. Tiezzi, A. Vandin, A formal approach for the analysis of BPMN collaboration models, JSS 180 (2021) 111007.
[21] D. Pianini, S. Sebastio, A. Vandin, Distributed statistical analysis of complex systems modeled through a chemical metaphor, in: HPCS, 2014, pp. 416–423.
[22] L. Belzner, R. De Nicola, A. Vandin, M. Wirsing, Reasoning (on) service component ensembles in rewriting logic, in: Spec., Alg., and Soft., 2014, pp. 188–211.
[23] A. Weijters, W. M. van Der Aalst, A. A. De Medeiros, Process mining with the heuristics miner-algorithm, Technische Universiteit Eindhoven, Tech. Rep. WP 166 (2006) 1–34.