Modelling and Analysing Cybersecurity Risks Using Attack Fault Trees and Game Theory

LEJLA SKENDERAGIĆ, University of Twente, The Netherlands

Cybersecurity risks can be investigated through Attack Fault Trees (AFTs), which integrate both safety and security aspects of a system in a novel way. Despite their potential, there is limited quantitative analysis available for these models. This research addresses this gap by utilising game theory to analyse AFTs quantitatively. By incorporating retry attacks, game theory allows attackers to repeatedly attempt to breach the system, provided they have a sufficient budget, introducing a new extent to the analysis. The PRISM model checker tool is utilised to perform this analysis and incorporate the retry function. The findings suggest that allowing hackers the option to retry attacks significantly increases the probability of system failure compared to scenarios without this option. This quantitative analysis emphasises the critical role of dynamic models in enhancing cybersecurity strategies. These insights can help develop more robust cybersecurity defences in increasingly interconnected digital environments.

Additional Key Words and Phrases: Cybersecurity, Attack Fault Trees, Game Theory, PRISM, Risk Assessment

1 INTRODUCTION

As high-tech systems advance, they face an increasing array of vulnerabilities from accidental failures (safety) and malicious attacks (security). In interconnected environments, the interplay between safety and security becomes increasingly important. Addressing these vulnerabilities requires analytical models that account for both accidental and deliberate threats. Effective risk analysis allows for the prioritisation of resources and strategies, ultimately safeguarding critical infrastructure from unexpected failures and attacks.

One widely used model for addressing these safety and security concerns is the Attack-Fault Tree (AFT). AFTs model how a top-level event, whether safety or security-relate, can be broken down into smaller sub-goals until no further refinement is possible. The leaves of the tree represent basic component failures (BCF) and basic attack steps (BAS). BCFs are failures that can occur within a system, such as human errors, power failures, or detection failures. Contrarily, BAS are attacks that a malicious intruder can attempt, including phishing attacks, insider attacks, or replay attacks. This integration allows for a comprehensive analysis of how various failures and attacks can compromise a system. Figure 1 illustrates a simple AFT, highlighting the structure of the model.

A valuable method for analysing AFTs is Pareto analysis, which identifies the most significant factors in a dataset. In the context of AFTs, Pareto analysis helps to understand the trade-offs between different objectives, such as minimising failure probability while managing costs [5]. Previous research has applied Pareto analysis to AFTs to highlight the most critical attack paths and their impacts [7]. However, these analyses often assume that attacks are single events, not accounting for scenarios where attackers can retry. This research seeks to address this gap by incorporating retry attacks into AFT models using game theory.

Traditional AFT analyses do not consider the dynamic nature of real-world attacks, where attackers can repeatedly attempt breaches if resources permit. This practical motivation drives the need for a more realistic model to enhance cybersecurity strategies. By introducing the concept of retry attacks, the analysis offers a new dimension to understanding cybersecurity risks.

AFTs can be viewed as games where the hacker aims to achieve a top-level event, such as system failure or breaching the system. Viewing AFTs through the lens of game theory allows for a robust framework for analysing strategic interactions. To model this PRISM, a comprehensive system used to model games and various scenarios incorporating probabilistic features, is used to obtain the data needed for the Pareto fronts.

Quantitative analysis of AFTs is particularly valuable for several reasons. Firstly, it allows for the calculation of the probability of various attack scenarios, which helps in understanding the likelihood of different attack paths and their potential impact on the system. Additionally, it enables the exploration of different "what-if" scenarios, providing insights into how changes in the system or threat environment can affect overall risk. Moreover, quantitative analysis reveals the impact of different budget allocations on the safety and security of the system, highlighting how financial constraints can influence the effectiveness of cybersecurity measures.

For complex or larger AFTs, obtaining the necessary data can be challenging and time-consuming. To address this, PRISM is introduced as a novel tool for analysing AFTs. PRISM automates the computation of probabilities, significantly reducing the time required to achieve these results. This thesis will focus on converting AFTs in PRISM to conduct an analysis, resulting in two Pareto fronts. The Pareto fronts will have the probability of failure and the budget as the two variables. To demonstrate the practical application of this approach, a case study involving GridShield will be used to conduct the experiments. This analysis follows the methodology of previous research according to (Lopuhaa-Zwakenberg, paper in preparation).

[9], however, this study incorporates a simpler model than the research mentioned. This is due to the retry option. For the experiments a budget is introduced and will be incremented, then the maximum probability of failure, breaching the system, will be calculated. This will be done for the same AFT, however, one PRISM model will have the retry option and the other will not. This allows for the analysis of the impact a budget has on reaching system failure and also the impact of being able to retry. Figure 2 outlines the process this paper used for the experiments.

This thesis introduces a novel framework for performing quantitative analysis on AFTs using PRISM and game theory. The addition

TScIT 41, July 5, 2024, Enschede, The Netherlands

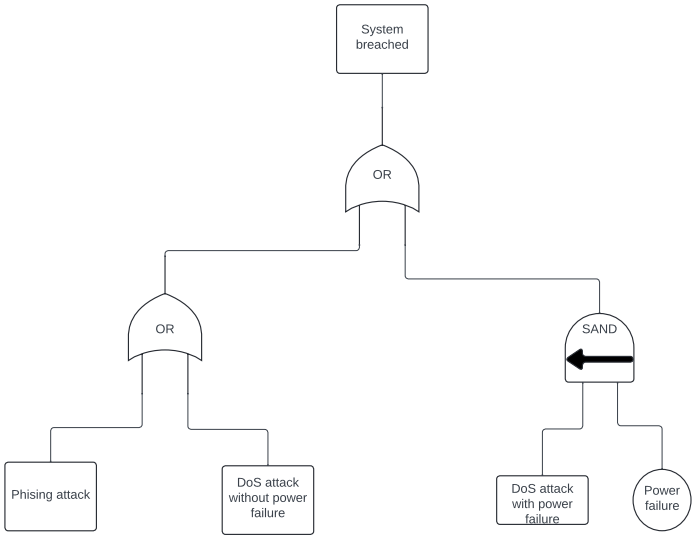


Fig. 1. An example of an AFT, where circles represent Basic Component Failures (BCFs) and rectangles represent Basic Attack Steps (BASs). The system can be breached if the hacker successfully launches a phishing attack or a Denial-of-Service (DoS) attack, each with their associated costs and probabilities of failure. Additionally, the SAND gate on the right indicates another option. If there is a power shortage and the hacker launches a DoS attack during this shortage, the probability of the attack's success increases, and the overall cost of the attack decreases.

Fig. 2. The paper will take an AFT and write it manually into PRISM then extract the probabilities and create a Pareto front.

of the retry mechanism enhances the AFT model, capturing the dynamic nature of system failures better than traditional methods. Furthermore, game theory considers costs and resource allocation, ensuring efficient defence strategies. The results show that allowing the hacker to retry significantly increases the probability of system failure, which continues to rise as the budget increases. In contrast, without the retry option, the probability of failure plateaus at a substantially lower level. The full code can be viewed in GitHub (LejlaSkenderagic, 2024/2024, [8]).

2 BACKGROUND

2.1 Attack-Fault Trees

As previously mentioned, safety and security are strongly correlated, measures that increase safety often decrease security and vice versa. For example, implementing access control measures, such as biometric locks or multi-factor authentication, enhances security by preventing unauthorised access. However, a downside is that these measures can hinder quick access to critical areas in case of an emergency, potentially delaying emergency response efforts and compromising safety. Thus, given this relationship between safety and security, they should be considered together allowing trade-offs to be made. [2].

AFTs provide a structured way to analyse and understand the interplay between safety and security. AFTs combine the concepts of fault trees (FT) and attack trees (AT), as a formalism AFTs can be used for the combined safety-security analysis [6] [4]. To capture both accidental failures and malicious attacks within a single model. This model is therefore particularly suited for the aforementioned problem.

In AFTs, the top-level goal typically represents a system failure or security breach. This goal is then decomposed into smaller sub-goals. These sub-goals are connected using logical operators like AND, OR, and SAND. These are depicted in Figure 3. The leaves of the tree represent BCF and BAS [7]. The gates operate as follows, for an AND gate, both nodes must be executed for the system to proceed. For an OR gate, only one of the nodes needs to be true. A SAND gate, however, depends on the direction indicated by the arrow, which specifies the order of execution. If the arrow points left, the action on the right must succeed first, followed by the action on the left. Both nodes need to be true, but they must also be executed in a specific sequence.

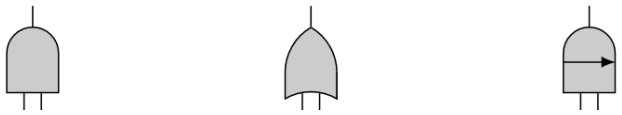


Fig. 3. Attack tree gates: AND, OR, and SAND gate respectively.

2.1.1 AFT Semantics. In the following section AFTs will be defined as used in this paper and the PRISM models. In this paper, a simple version of AFTs is used, meaning that the AFT only consists of AND/OR and SAND gates [9]. In the definition of the AFT, it is stated that the AFT is a directed acyclic graph (DAG) [7]. A DAG is a concept from mathematics and computer science, particularly in graph theory. A DAG is a finite directed graph that contains no cycles. Figure 1 is an example of a DAG. To be classified as a DAG the graph must:

- Be Directed: Every edge in the graph has a direction.
- Be Acyclic: The graph must not contain any cycles. This means that it is impossible to start at any node and follow a path of edges that eventually loops back to the starting node.
- Have a Root: There is a single root node from which every other node can be reached. A SAND node itself is a type of gate in an AFT that requires all its child events to occur in a specific sequence for it to be considered compromised. While the concept of a SAND gate is specific to AFTs, the entire structure of an AFT, including SAND gates, can still form a directed acyclic graph, if the overall structure maintains the properties of a DAG.

2.1.2 Cost and Reward Structure. An AFT has costs and rewards associated with the various nodes. This is due to the way an AFT is formalised. In AFTs, BCFs have a probability of failure, while BASs have a probability of success and an associated cost. These factors help the attacker generate a strategy, providing valuable insights into the system's safety and security. Since some attacks have higher costs than others. Due to factors such as time required, the skill level of the attacker, or other associated costs, the attacker must compare multiple scenarios and make careful decisions to maximise the probability of system failure.

To improve hacker efficiency. Penalties, such as a penalty per step, ensure that taking many steps becomes unattractive. Additionally, there is a retry penalty. This reflects real-world conditions where taking extra steps and retrying can introduce additional risks. These costs are estimated and assigned by the programmer taking reasonable estimations.

2.2 PRISM Modelling Language

PRISM is a probabilistic model checker, a tool for formal modelling and analysis of systems that exhibit random or probabilistic behaviour [3]. Models are described using the PRISM language, which is a simple, state-based language. The property specification language incorporates various logic statements, as well as extensions for quantitative specifications and costs/rewards. This means that PRISM allows us to create a 1, 1.5 and 2-player game to determine the best strategies for breaking into the system. The term "1.5-player game" is not a standard concept in game theory but is used in certain contexts to describe a scenario between a fully interactive two-player game and a single-player game against a static system [1]. In a 1.5-player game, there is one active player and a "half-player" who does not actively make strategic decisions but whose actions or state can change based on the primary player's actions. These half-player changes can be interpreted as a system with dynamic elements, influenced by the actions of the single player, yet not fully controlled by another strategic player. The random faults that can happen and change the system's state are one of the reasons for choosing a 1,5 player game for this thesis. A 1-Player Game is typically referred to as a solo game where a single player's decisions are made against a static environment or a predefined set of rules and probabilities, with no adaptive or responsive opponent. Conversely, creating a 2-player Game would involve the system reacting to the hacker, effectively turning the AFT into an Attack-Fault-Defence Tree, which is not suitable for this thesis. Therefore, the decision is made to create a 1.5-player Game in PRISM.

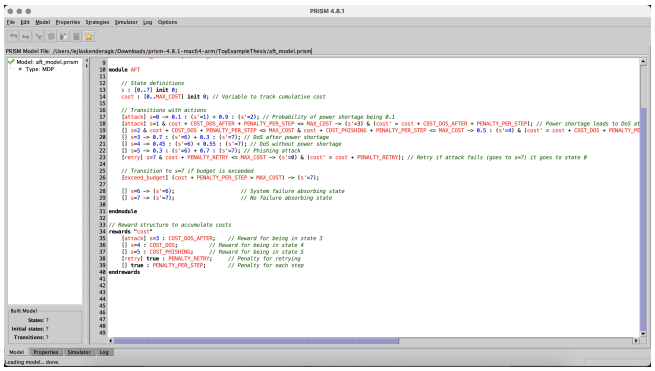


Fig. 4. The Graphical User Interface (GUI) of PRISM.

Game Theory The game theory is applied using MDPs. MDPs inherently support decision-making processes where actions are chosen based on the current state. This capability is crucial for representing the strategic interactions between attackers and defenders in cybersecurity scenarios [10].

3 CASE STUDY

3.1 Case Study: GridShield

GridShield serves as a protective mechanism that reduces the charging output at stations during instances of overload in smart grid assets like transformers. Additionally, GridShield is designed as a last-line defence strategy to avert disruptions in power grid services, supplementing existing digital and physical safety measures.

As electric vehicles (EVs) become more widespread within the low-voltage distribution grid, they significantly increase the electrical load. Heightening the risk of overloads and potential damage to grid infrastructure [12].

To reduce this, energy management systems designed for smart EV charging are being developed to prevent congestion. However, these systems are prone to failures, highlighting the need for robust

backup solutions. GridShield plays an essential role in this context as a local congestion management control system.

The operational principle of GridShield is straightforward. A sender module continuously monitors the load on these assets. Should this module detect an overload, it communicates with all nearby charging stations, prompting them to reduce their power output. These stations are fitted with receiver modules that adjust the charging power based on the instructions received. Should there be a change in the asset's load, a new message may be issued to either further reduce the power output or restore it to its original level.

The AFT obtained from the following paper [11] is quite complex, so for the sake of this thesis, it has been simplified to Figure 5. Furthermore, the AFT in the paper was not quantified meaning that, for the experiments costs/probabilities have been assumed and reasonable estimates have been made.

In addition to that, the decision has been made to also have a penalty for each step and retry to make these options less attractive. Forcing the strategy to be more efficient.

3.2 Assumptions

Several assumptions have been made in this study. It is assumed that the hacker has complete visibility of the model and possesses knowledge of the probabilities associated with different attack outcomes. This helps the analysis by showing the maximum probability of the maximum threat. Thus, protecting the system against the best hacker also protects the system against less capable hackers. The system also assumes that faults can only occur once. In addition, the probabilities, costs, and penalties are set based on assumptions rather than empirical data. Furthermore, the budget steps have been chosen by the author and do not have quantitative data to support the choices however, they are considered reasonable estimations.

4 METHODOLOGY

This section explains how the AFT from the case study is translated into a PRISM model.

4.1 Model Configuration and Simulation

This PRISM code models an attack scenario on a system with three receivers, considering various attack types and spontaneous faults. See Figure 5 for a visual representation of the system. The goal is to simulate the decision-making process of an attacker constrained by a budget, aiming to compromise the system by exploiting different attack vectors.

4.2 PRISM Setup

The way the AFTs are programmed in PRISM is by assigning costs and probabilities of success to the attacks and probabilities of failing to the faults. This is done by making them constants.

Subsequently, a module needs to be made. In PRISM, a module is a fundamental building block representing a specific part of a larger system. It is a self-contained component that describes and models the system's various aspects.

Then, state variables should be implemented when a module has been made. State variables are properties or characteristics that hold different values and represent the module's current state For example:

```
// states what probabilistic model will be used
mdp
// here all the constants are defined for example the budget
const int budget = 200
//here the module is initiated
module attack
//here all the state variables are stated for example the systems state
//System state: 0 = ongoing, 1 = system breached, 2 = system not breached
system_state : [0..2] init 0
// Boolean variables indicating if each fault has been checked
command_fail_checked_1 : bool init false;
```

Faults are evaluated only once to prevent the maximum probability of system failure from reaching 1. Allowing indefinite fault checks would result in a fault occurrence over time. Consequently, the player is restricted to a single verification of each fault per attack, with no possibility of subsequent checks for the same fault within that attack.

Furthermore, the system's state is used to transfer the system to an absorbing state. Which is a state in which no transitions are possible, meaning once the system enters this state, it remains there indefinitely. If there is still a budget left and not all receivers are compromised then the system state is 0, indicating that it keeps going. System state 1 is used if the boolean for all receivers breached is set to true indicating all receivers are breached. When the system state goes to 1, it enters the absorbing state for system breached. If the system enters state 2 this sends the system to absorbing state system not breached. This happens when not all receivers are breached and the budget is not sufficient to perform another action.

Then, the attack steps should be defined as a choice by the player. Meaning that the player should choose to attack and what attack. This is done by representing each type of attack as a command. For example, [replay-attack], [jam-command-1], and [insider-attack] are commands representing different types of attacks. Each command has conditions that must be met for the action to be executed. Example:

```
[jam_command-1] !receiver1_compromised & budget_left > = (cost_jam + step_penalty + retry_penalty) & system_state = 0
```

This line indicates that only if there is enough budget left for the cost of a jam attack, the step penalty, the penalty for retrying, and the system state is 0, then the attack can be executed. In Figure 6 a picture of the taken path step when the jam attack is executed is visible. However, the replay attack has to go through a sand gate. This means that first the attacker needs to send the attack, which goes to all three receivers in one go, then the fault that the replay attack went unnoticed has to occur. Thus, this attack could compromise one, two, three, or none of the receivers, this is visualised in the AFT of the case in Figure 5. This was a challenge to achieve, and the way this was done was to hard-code each scenario that was possible.

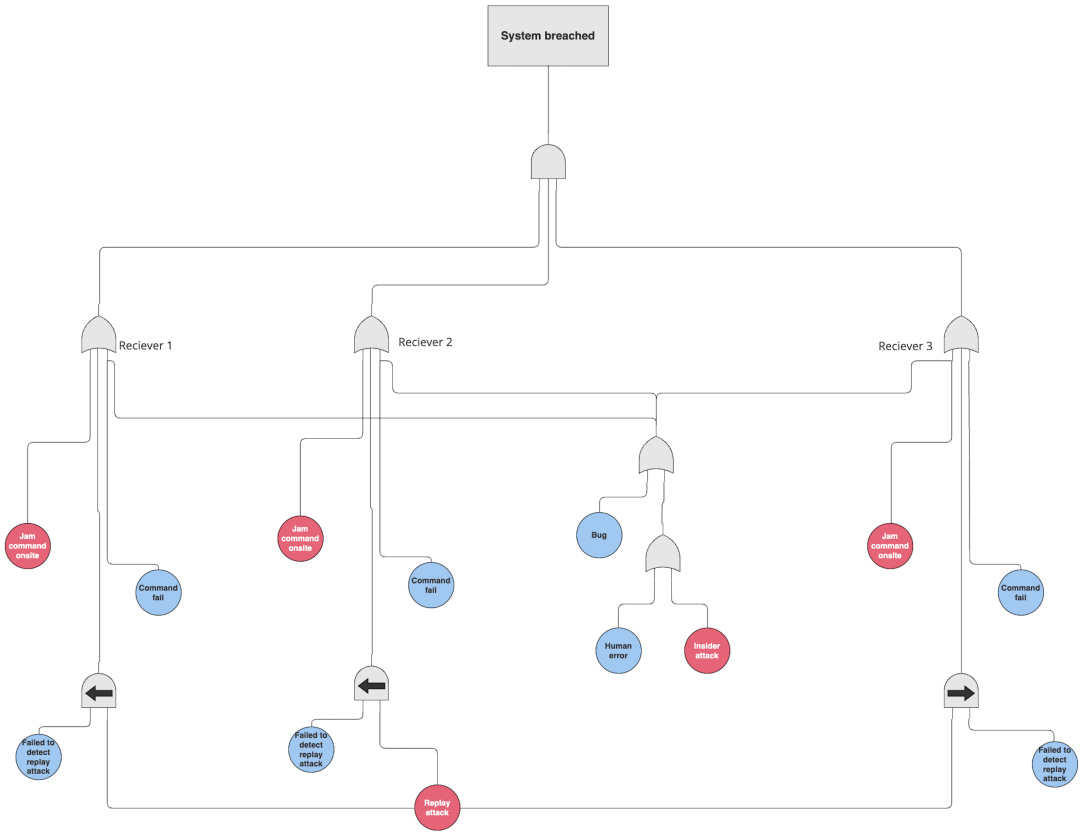


Fig. 5. Attack Fault tree of the GridShield that will be used in this thesis.

Step		attack										Rewards		
Module/[acti...	#	budget_left	receiver1_co...	receiver2_co...	receiver3_co...	system_state	command_fa...	command_fa...	command_fa...	human_error...	bug_checked	"budget_used"	"step_penalty"	["retry_pena...
	0	50	false	false	false	0	false	false	false	false	false	0	1	0
lt_command_fa	1	50	true	false	false	0	true	false	false	false	false	0	1	0
am_command_	2	41	true	false	false	0	true	false	false	false	false	9	1	?

Fig. 6. A snapshot of the simulation path shows that a command fault on receiver 1 was checked first, followed by a jam attack on receiver 2. The fault was successful, as indicated by the compromise of receiver 1. In contrast, the jam attack was unsuccessful, evidenced by the false boolean value for the breach of receiver 2.

To clarify one variation that could happen in the attack will be further elaborated on.

```
(p_failed_detect * p_failed_detect * (1 - p_failed_detect)) : (
  budget_left' = budget_left - cost_replay - step_penalty -
  retry_penalty) &
    (receiver1_compromised' = true) & (receiver2_compromised' =
  true) & (receiver3_compromised' = false)
```

The situation above describes the probability that the replay attack compromises receivers 1 and 2, but not 3. This can be seen with the boolean line where the receiverX' compromised gets set to either true or false. First, the probability must be set since it can affect 3 receivers at the same time. This is done by taking the probability of failure of the fault, failed to detect, for all compromised receivers and multiplying that with (1 - the probability of failure for failed to detect fault). As can be seen in the code in this line:

```
(p_failed_detect * p_failed_detect * (1 - p_failed_detect))
```

Afterwards, the budget gets adjusted by subtracting the correct costs and the right booleans are given to the right receivers.

Another special attack is the insider attack. This attack will, if successful, compromise all receivers in one attack. This is programmed by just setting all receivers breached Boolean's to true when the attack succeeds.

```
p_insider : (budget_left' = budget_left - cost_insider -
  step_penalty) &
            (receiver1_compromised' = true) & (
  receiver2_compromised' = true) &
            (receiver3_compromised' = true) & (system_state' = 1)
   +
// Failed insider attack, incurs retry penalty
(1 - p_insider) : (budget_left' = budget_left - cost_insider -
  step_penalty - retry_penalty);
```

As for the faults, they are programmed in the following manner:

```
[fault_command_fail_1] system_state = 0 & !receiver1_compromised
  & !command_fail_checked_1 ->
    p_command_fail : (receiver1_compromised' = true) & (
  command_fail_checked_1' = true) +
    (1 - p_command_fail) : (command_fail_checked_1' = true);
```

This code snippet within a PRISM model simulates the potential for a fault to occur at a specific point in the system, affecting a particular component (receiver1 in this case). The fault is considered only once, as indicated by the command fail check. The outcome can be either the compromise of the component or no effect, based on the defined probability. As aforementioned, the faults can only be checked once per attack, the code has the command fail checker. This checker checks to observe if the previously defined boolean is true, meaning it has already been checked. If that is the case, the fault cannot occur anymore.

Human error and Bug are special cases. As we can see in the Figure 5, if these faults occur, they affect all receivers in one go and breach them. This has been done in a similar fashion to the insider attack. When the probability of failure happens, all booleans of the receiver breached get set to true.

The following code makes sure the system transitions to the right states and absorbing states when necessary. The primary purpose of these transitions is to dictate how the system state changes in response to events such as a system breach, the exhaustion of resources, such as budget, or the incapability of further attacks.

```
[] (receiver1_compromised & receiver2_compromised &
  receiver3_compromised) -> (system_state' = 1);
without compromising all receivers
[system_not_breached] budget_left <= 0 & (!receiver1_compromised
  | !receiver2_compromised | !receiver3_compromised) -> (
  system_state' = 2);
[system_not_breached] (budget_left < (cost_replay + step_penalty)
   & budget_left < (cost_jam + step_penalty) & budget_left < (
  cost_insider + step_penalty)) &
   (!receiver1_compromised | !receiver2_compromised | !
  receiver3_compromised) -> (system_state' = 2);

[] (system_state = 1) -> (system_state' = 1); // Absorbing state
  for breached system
[] (system_state = 2) -> (system_state' = 2); // Absorbing state
  for not breached system
```

Lastly, the retry is implemented by having the probability that the attack failed to send the system to system state 0, which is the initial state. Meaning the attacker can try again. The model, without the retry, sends this probability immediately to system state 2.

To obtain the data in PRISM properties are used. Properties are formal specifications used to describe and analyse the behaviour of a model. They operate by enabling the verification of system characteristics like probabilities of events, expected rewards, and temporal properties. This is shown in the Figure 7.

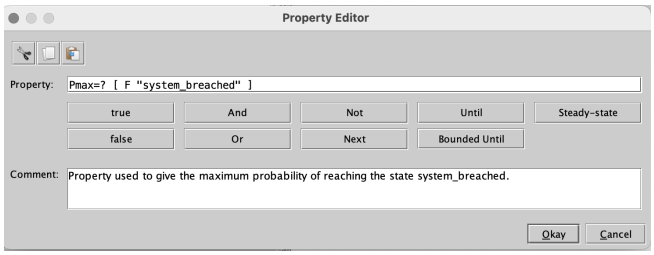


Fig. 7. A screenshot of the property page in PRISM with the property used in this experiment.

5 EXPERIMENT

5.1 Data Analysis

The data was analysed using a Markov Decision Process (MDP) statistical model in PRISM. Additionally, the data was manually collected and represented in graphs. Each data entry took approximately 0.049 seconds to compute in PRISM. The graphs were subsequently used to generate the Pareto fronts using Python.

As stated in the introduction, the full code can be viewed at GitHub (LejlaSkenderagic, 2024/2024, [8]).

5.2 Results

The simulation results demonstrate how varying budget constraints and the ability to retry attacks influence the probability of a system breach. Two scenarios were considered: GridShield with and without the retry option. In Tables 1 and 2 the results are depicted. Furthermore, in the Figures 8a and 8b these results are represented in two Pareto fronts. Here the plateau is visible for the GridShield model without the retries.

Without Retry: The results indicate a direct correlation between increased budget and the probability of system failure. With an initial budget of 5, the probability of a system breach was approximately 0.0125. This probability increased significantly as the budget was raised to 20, where it stabilised around 0.354. Notably, there is a significant jump in probability when the budget reaches 15. This is due to the cost of attacks. When the budget is 15 or higher, all attacks, including the previously too expensive insider attack, become possible. Significantly increasing the probability of failure.

Interestingly, further increases in the budget up to 200 did not change the failure probability, indicating a limit beyond which additional funds do not increase risk. This plateau suggests that the most effective attack vectors are exhausted, and additional funds do not provide the attacker with significant advantages. The absence of a retry option also limits the number of available attack paths, further restricting the possible attack vectors.

With Retry: Incorporating the retry option changes the dynamics significantly. Starting with the same low probability of failure at a budget of 5 (0.0045), compared to the no-retry scenario, the likelihood of a breach increases considerably as the budget increases. Notably, with a budget of 300, the probability of a system breach approaches near certainty (0.9999), underscoring the enhanced risk posed by allowing attackers multiple attempts to breach the system. This scenario highlights the critical impact of retries, where

attackers can leverage their budget more effectively over multiple attempts, substantially increasing the vulnerability of the system.

The comparison between the two scenarios underscores the significant role that strategic decisions, such as allowing retries, play in cybersecurity risk assessment. The ability to retry attacks amplifies the impact of the attacker's budget, significantly increasing the potential for a system breach. This demonstrates the importance of considering these capabilities in security models, as they can have a profound effect on the system's overall vulnerability.

These findings are significant for understanding the dimensions of risk associated with different security policies and system settings. They suggest that preventative measures, limiting the ability to retry attacks, could be an effective strategy for reducing cybersecurity risks. Furthermore, the results show that budget constraints significantly influence the likelihood of a system breach. More expensive attacks tend to be more dangerous, which is evident from the significant increase in the probability of failure when the budget reaches 15, making all attack options possible.

Table 1. Results GridShield without Retry.

Probability of Failure	Budget
0.012487640897203204	5
0.021123018176313603	10
0.05497917988965122	15
0.3540419416908001	20
0.3540419416908001	25
0.3540419416908001	30
0.3540419416908001	50
0.3540419416908001	100
0.3540419416908001	150
0.3540419416908001	200

5.3 Limitations

This research has several limitations that should be considered. One limitation is that the AFT must be manually programmed into PRISM code, which can be time-consuming and complex for larger models, reducing the method's scalability. Additionally, PRISM's execution time can be inconsistent, sometimes requiring more time to compute the maximum probability due to cache overload from multiple calculations. Restarting the program rid this issue, with calculations then completed in milliseconds. Lastly, PRISM does not support parameter sweeps, necessitating manual data extraction and further impacting scalability.

6 DISCUSSION

Implementing a retry mechanism significantly increases system vulnerability. Each retry affords attackers additional insights to refine their strategies, thereby increasing the likelihood of a system breach. This observation emphasises the importance of resilient security systems that adapt to ongoing threats, mirroring adaptive cybersecurity practices.

Unlike static Traditional Fault Trees, the Attack-Fault Tree (AFT) model includes dynamic interactions and retries, offering a more accurate simulation of real-life cyberthreats. This dynamic approach better captures the iterative and adaptive nature of attacks and defences, suggesting a shift towards more sophisticated analytical models in cybersecurity risk assessments.

Table 2. Results GridShield with Retry.

Probability of Failure	Budget
0.0045238315496	5
0.013335441018000	10
0.0549791798896512	15
0.3593280973718806	20
0.3597743838139057	25
0.38893925126857803	30
0.4371934552119903	35
0.5893606049873389	40
0.6408860359064776	50
0.8413682672986431	75
0.927800671618154	100
0.9748324651205272	125
0.9905943858155964	150
0.9991101544972914	200
0.9999181954666376	300
0.9999991819637514	400
0.9999994059434129	500
0.9999995342548149	600

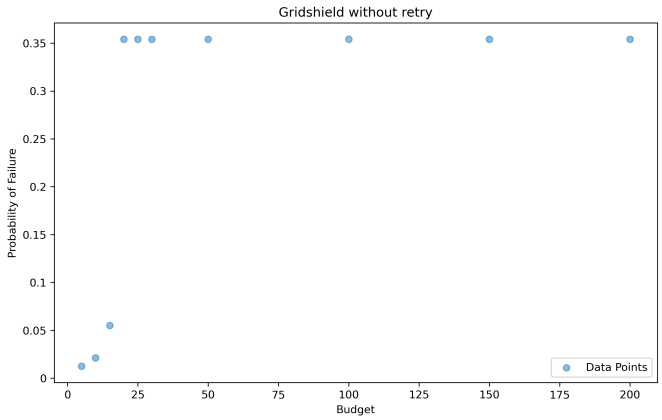
As for PRISM, PRISM is excellent for formal verification and quantitative analysis of AFTs, offering precise security insights and managing complex systems. However, it struggles with rapid growth in state complexity in larger AFTs, making analysis more demanding. Modelling complex dependencies can be challenging, and learning PRISM's language is difficult, posing a barrier for new users.

7 CONCLUSION

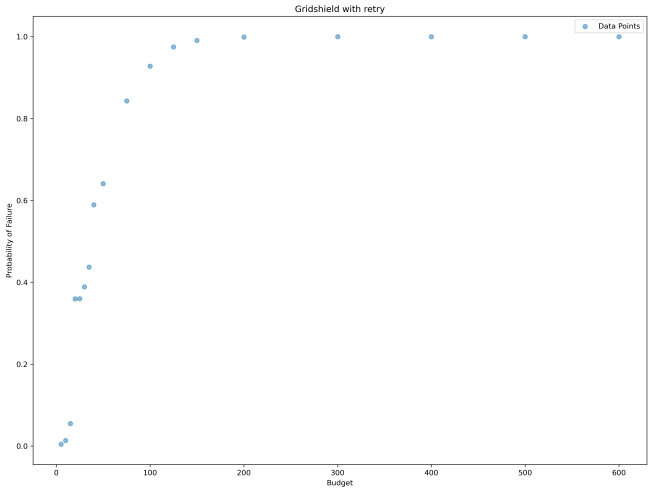
This thesis demonstrated the integration of Attack-Fault Trees with the PRISM modelling tool, incorporating game theory and introducing retry mechanisms to better understand attacker behaviours and system vulnerabilities. The findings indicate that allowing attackers to retry with increased budgets significantly heightens the risk of system breaches. By showing how repeated attempts can more effectively exploit vulnerabilities. This study emphasises the importance of advanced modelling techniques in discovering system weaknesses, adapting dynamic models for a realistic portrayal of the system and developing stronger cybersecurity strategies.

7.1 Future Work

Future research could explore how different defensive mechanisms, such as enhanced intrusion detection systems and rapid response strategies, mitigate the increased risks from retries and larger budgets. Additionally, automating the process by creating a framework to convert Galileo AT and FT models to PRISM code or enabling PRISM to perform parameter sweeps would be beneficial.



(a) The Pareto front for the Gridshield without the retry option. Meaning that the model could only execute one attack at a time.



(b) The Pareto front for the Gridshield with the retry option. Meaning that the model could continue attacks while there is a budget.

Fig. 8. Comparative Pareto fronts for the Gridshield with and without the retry option

8 USE OF GENERATIVE AI

During the preparation of this work, the author used ChatGPT to correct grammar, spelling and to generate Python code for plotting and visualisation and help debug the PRISM code. After using this tool/service, the author reviewed and edited the content as needed and took full responsibility for the content of the work.

REFERENCES

[1] Adaptive Artificial Intelligence in Games: Issues, Requirements, and a Solution through Behavlets-based General Player Modelling.
[2] Applying Game Theory and Computer Simulation to Fault Tree Analysis – DSIAC.
[3] PRISM - Probabilistic Symbolic Model Checker.
[4] Carlos E. Budde, Christina Kolb, and Mariëlle Stoelinga. Attack Trees vs. Fault Trees: Two Sides of the Same Coin from Different Currencies. In Alessandro Abate and Andrea Marin, editors, *Quantitative Evaluation of Systems*, volume 12846, pages 457–467. Springer International Publishing, Cham, 2021. Series Title: Lecture Notes in Computer Science.
[5] Ioannis Giagkiozis and Peter J. Fleming. Pareto Front Estimation for Decision Making. *Evolutionary Computation*, 22(4):651–678, December 2014.
[6] Raffaela Groner, Thomas Witte, Alexander Raschke, Sophie Hirn, Irdin Pekaric, Markus Frick, Matthias Tichy, and Michael Felderer. Model-Based Generation of Attack-Fault Trees. In Jérémie Guiochet, Stefano Tonetta, and Friedemann Bitsch, editors, *Computer Safety, Reliability, and Security*, pages 107–120, Cham, 2023. Springer Nature Switzerland.
[7] Rajesh Kumar and Mariëlle Stoelinga. Quantitative Security and Safety Analysis with Attack-Fault Trees. January 2017.
[8] LejlaSkenderagic. LejlaSkenderagic/PRISM_thesis, June 2024. original-date: 2024-06-21T11:24:33Z.
[9] Milan Lopuhaa-Zwakenberg. Quantitative analysis of attack-fault trees via Markov decision processes. note: In preparation.
[10] David Silver. Lecture 2: Markov Decision Processes. *Markov Processes*.
[11] Reza Soltani, Milan Lopuhaa-Zwakenberg, and Marielle Stoelinga. Safety-security analysis via Attack-Fault-Defense Trees: semantics and cut set metrics. note: In preparation.
[12] Hanna L. van Sambeek, Marisca Zweistra, Gerwin Hoogsteen, Ivo A. M. Varenhorst, and Stan Janssen. GridShield—Optimizing the Use of Grid Capacity during Increased EV Adoption. *World Electric Vehicle Journal*, 14(3), 2023.