Universidade Estadual de Londrina

GABRIEL ESTEVES MESSAS

SAIFE: TOWARDS A LIGHTWEIGHT THREAT MODELING APPROACH TO SUPPORT MACHINE LEARNING APPLICATION DEVELOPMENT

LONDRINA
2024

GABRIEL ESTEVES MESSAS

SAIFE: TOWARDS A LIGHTWEIGHT THREAT MODELING APPROACH TO SUPPORT MACHINE LEARNING APPLICATION DEVELOPMENT

Dissertação apresentada ao Programa de Mestrado em Ciência da Computação da Universidade Estadual de Londrina para obtenção do título de Mestre em Ciência da Computação.

Supervisor: Prof. Dr. Bruno Bogaz Zarpelão

LONDRINA

2024

Ficha de identificação da obra elaborada pelo autor, através do Programa de Geração Automática do Sistema de Bibliotecas da UEL

M583s Messas, Gabriel Esteves.
sAIfe: Towards a Lightweight Threat Modeling Approach to Support Machine Learning Application Development / Gabriel Esteves Messas. - Londrina, 2024.
75 f. : il.

Orientador: Bruno Bogaz Zarpelão.
Dissertação (Mestrado em Ciência da Computação) - Universidade Estadual de Londrina, Centro de Ciências Exatas, Programa de Pós-Graduação em Ciência da Computação, 2024.
Inclui bibliografia.

1. inteligência artificial - Tese. 2. segurança - Tese. 3. machine learning - Tese. 4. threat modeling - Tese. I. Zarpelão, Bruno Bogaz. II. Universidade Estadual de Londrina. Centro de Ciências Exatas. Programa de Pós-Graduação em Ciência da Computação. III. Título.

CDU 519

GABRIEL ESTEVES MESSAS

SAIFE: TOWARDS A LIGHTWEIGHT THREAT MODELING APPROACH TO SUPPORT MACHINE LEARNING APPLICATION DEVELOPMENT

Dissertação apresentada ao Programa de Mestrado em Ciência da Computação da Universidade Estadual de Londrina para obtenção do título de Mestre em Ciência da Computação.

BANCA EXAMINADORA

Orientador: Prof. Dr. Bruno Bogaz Zarpelão
Universidade Estadual de Londrina - UEL

Prof. Dr. André Luís Andrade Menolli
Universidade Estadual do Norte do Paraná - UENP

Prof. Dr. Rodolfo Ipolito Meneguette
Universidade de São Paulo - USP

Londrina, 13 de dezembro de 2024.

This work is dedicated to everybody who believes science should be democratic, uncomplicated and used to make the world a better place to live in.

ACKNOWLEDGEMENTS

The path to completing this dissertation has been fraught with both professional challenges and personal adversities. I am profoundly grateful for the love and enduring support of my family throughout the process.

I must also express my gratitude to my advisor, Prof. Dr. Bruno Zarpelão, for his continued guidance and our combined resilience and effective teamwork.

Lastly, special thanks are directed to the Center of Artificial Intelligence in Agro (CIA-Agro)[1] of the State University of Londrina (UEL).

[1] http://ciaagro.uel.br

MESSAS, G. E.. **sAIfe: rumo a uma abordagem leve de modelagem de ameaças para apoiar o desenvolvimento de aplicativos de Aprendizado de Máquina**. 2024. 74f. Dissertação (Mestrado em Ciência da Computação) – Universidade Estadual de Londrina, Londrina, 2024.

RESUMO

Com a crescente popularização do campo da Inteligência Artificial (IA), o desenvolvimento de sistemas que empregam, pelo menos, uma de suas subáreas também tem experimentado um grande aumento. A recente adoção de técnicas de IA em sistemas comuns - como aplicativos para celular e equipamentos domésticos - requer um maior nível de atenção, a fim de garantir sua segurança e funcionamento adequado. Neste cenário, garantir o funcionamento adequado destas soluções culmina, na maioria dos casos, em garantir a segurança da aplicação e dos seus dados durante todo o ciclo de vida de desenvolvimento do software. Desenvolvedores de software, no entanto, muitas vezes consideram as tarefas relacionadas à segurança difíceis de aprender e executar, e frequentemente as deixam de lado. Além disso, os frameworks de modelagem de ameaças atualmente disponíveis são difíceis de integrar nos ciclos de vida de desenvolvimento de software, que priorizam a agilidade e a automação em detrimento de análises e documentação extensas. Este trabalho, portanto, propõe o sAIfe, um novo método de modelagem de ameaças para análise de segurança de aplicações de Machine Learning (ML) em desenvolvimento. O sAIfe fornece etapas prescritivas, com elementos gráficos e resultados que incluem listas com ameaças e sugestões de remediação já prontas para o sistema analizado. Esta abordagem visa simplificar e agilizar o processo de avaliação de risco para o programador, revelando possíveis fragilidades e sugerindo respectivas soluções de forma prática. Ainda neste trabalho, o sAIfe é testado numa aplicação de IA do mundo real, revelando resultados positivos, com muitos problemas potenciais e opções de mitigação detectados pelo método, que são registados na forma de um estudo de caso. Adicionalmente, este estudo é comparado a outro, realizado com um método alternativo da literatura, evidenciando as vantagens do sAIfe. Por último, são realizadas duas validações: uma com pesquisadores na academia e outra com desenvolvedores na indústria, retornando ótimos feedbacks sobre a facilidade de uso e a velocidade de aplicação do sAIfe.

Palavras-chave: Inteligência Artificial. Aprendizado de Máquina. Segurança. Modelagem de Ameaças.

MESSAS, G. E.. **sAIfe: Towards a Lightweight Threat Modeling Approach to Support Machine Learning Application Development**. 2024. 74p. Master's Thesis (Master in Science in Computer Science) – State University of Londrina, Londrina, 2024.

ABSTRACT

With the growing popularization of the Artificial Intelligence (AI) field, the development of systems that rely on, at least, one of its subareas has also experienced a great increase. The recent adoption of AI techniques in common systems - such as mobile apps and household appliances - requires a higher level of attention, in order to ensure their safety and proper operation. In this scenario, assuring the adequate functioning of these solutions culminates, in most cases, in ensuring the security of the application and its data throughout the software development life cycle. Software developers, however, often find security-related tasks challenging to learn and execute, and frequently put them aside. Additionally, currently available threat modeling frameworks are difficult to integrate into software development life cycles, which prioritize agility and automation over extensive analysis and documentation. This work, therefore, proposes sAIfe, a new threat modeling method for security analysis of Machine Learning (ML) applications under development. sAIfe provides prescriptive steps, with graphical elements and results that include lists with threats and ready-made remediation suggestions for the analyzed system. This approach aims at simplifying the risk assessment process for the programmer, unveiling possible weaknesses and suggesting respective solutions in a practical way. Still in this work, sAIfe is tested on a real-world ML application, revealing positive results, with many potential issues and mitigation options detected by the method, which are registered in the form of a case study. Additionally, this study is compared to another one, carried out with an alternative method from the literature, highlighting sAIfe's advantages. Finally, two validations are carried out: one with researchers in academia and another with developers in industry, returning great feedback on sAIfe's ease of use and speed of application.

Keywords: Artificial Intelligence. Machine Learning. Security. Threat Modeling.

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS AND ACRONYMS

AI	Artificial Intelligence
ML	Machine Learning
SDLC	Software Development Life Cycle
DSR	Design Science Research
MSA	Meta-modeling Security Analysis
ARA	Architectural Risk Analysis
DFD	Data Flow Diagramming

CONTENTS

1 INTRODUCTION

The beginning of the 21st century marked a solid acceleration in the development of sectors related to electronic technologies, especially in the area of communication and information technology. Among those, one that has most gained appreciation and space in the industry was Artificial Intelligence (AI), propelled by key concepts, such as Machine Learning (ML) [2] and Deep Learning (DL) [3]. This sudden popularization [4], however, combined with the lack of good practices, may bring with itself the growth of threats targeting AI systems and contribute, consequently, to the increase in failure rates. These systems, nevertheless, now increasingly present in common applications - such as mobile apps and household appliances - raise a special security concern, as their use in these environments can expand the attack surface and allow for the occurrence of new undocumented security events [5, 6]. In this scenario, therefore, efforts to ensure the security of the systems in question become even more necessary.

This matter might look, at first, like a standard case of software security. AI-based systems share some components and characteristics with regular systems, making much of the existing cybersecurity knowledge applicable to them, as noted by [7] and [8]. However, these systems also introduce new security challenges that the traditional body of knowledge does not fully address. Besides presenting a different set of possible security threats, the particularities of an AI/ML-focused application - intrinsic to its purpose -, such as the possible communication with sensors and the high dependence on data, shape it in a specific way. For this reason, not only code but even infrastructure acquires singular formats, justifying the study of this area individually. Therefore, expanding the current cybersecurity body of knowledge by integrating new insights that address the specific security needs of AI-based systems is essential.

Academic studies have sought this goal in recent years. They, however, do not have the software developer as their target audience. These studies tend, in their vast majority, to focus on very particular topics [9, 10, 11, 12, 13], addressing highly specific cases, directed to a single segment of the area - such as exploring a single kind of attack/defense in a singular scenario. Although these matters are highly relevant to the respective specialized academic community, there still remains a gap between the theory being discussed and the practical implementation in a real software development cycle.

Furthermore, the knowledge needed to assess the security of real-world software of this kind is generally spread throughout various sources. The development team would, then, have to search through lots of academic articles, books and documents available on the Internet - such as the ones maintained by OWASP [14, 15], NIST [16] and Microsoft [17], for instance - for information on vulnerabilities and possible solutions. However, with

the heavy workload programmers tend to have nowadays - immersed in a development model that typically features tight schedules and prioritizes agility -, compiling all this knowledge from different sources would be a very challenging activity. A hindrance like this, in turn, can even cause teams to put security - and other non-functional requirements - aside, especially if they do not aggregate instantaneous visible value to the product [18, 19, 20].

This scenario would actually benefit more from a comprehensive security assessment method, that could gather all of this information and compile it in a seamless way, becoming much more readable and easier for the end-programmer to use. In this line, a protocol that helps the developer understand their system's characteristics and already presents them with possible threats and ways of mitigating them becomes relevant, as it saves them from having to scour the Internet for security recommendations. These attributes would bring much more practicality and make the process of increasing security in an application both easier and faster, enough for this activity to generate an acceptable time-value trade-off under the eyes of modern software development teams [18]. Moreover, according to [19], a process that can be quick and not require much documentation from the involved users can be a good alternative for these squads.

Increasing security, in these cases, often includes applying a threat modeling (TM) approach to the application, i.e., a process for identifying potential security risks. However, performing this process is often highly time-consuming for the programming teams involved, which end up skipping it. This drawback is just one of the obstacles modern development teams typically find, along with the requirement of high levels of expertise and the elevated complexity of the methods currently available [21]. Moreover, the TM methods currently available either do not address the AI part as the focus of the software [22] or are overly intricate and do not provide practical actionable suggestions in the end [23] [17].

To address this problem, the present work proposes a new threat modeling process - named sAIfe - to be used on ML-related applications in development. Its goal is to guide software developers during development, in order to achieve more secure ML-enabled systems. The process itself guides the developer through the creation of a conceptual model of the system under analysis, in the form of a graphical diagram. This generated artifact will, then, help the developer identify the critical areas in their application, under the security aspect. For each of these, the method offers a group of possible threats and their respective solutions - a comprehensive but light knowledge base, gathered and compiled from various sources, including security rankings - published by entities such as OWASP [14, 15] and NIST [16] - and academic articles, to which the developer can resort in order to model the threats in their software. By using visual elements and prescriptive steps, the method aims at being a lightweight approach, quick and easy to apply, which,

therefore, turns it into a highly practical solution, compatible with fast-paced development teams. The result, therefore, aims to be a valuable contribution to help developers increase the security of ML programs under construction.

This work was developed under the scope of the CIA-Agro research group - at the State University of Londrina (UEL) -, a joint effort focused on studying and applying high-tech AI solutions to the agribusiness field.

The document is organized as follows. In Section 2, the theoretical background is detailed and existing work in the field is reviewed. Section 3 presents the method and its way of functioning in detail. In Section 4, evaluation results are described, including two real-life case studies - one with sAIfe and the other with a comparative method - and two validations: one with researchers in academia and another with developers in industry, along with comments and discussions on the analyses. Finally, Section 5 concludes the work and summarizes the contributions generated.

2 THEORETICAL BACKGROUND

This chapter presents the theoretical foundations for the thesis, citing and explaining the main concepts involved, along with the state-of-the-art situation of these topics in recent researches.

2.1 Threat Modeling

Essentially, threat modeling (TM) is a process for finding security problems in an application and its environment. It involves creating a representation of the system with all its components, then identifying possible weak spots [24]. Ideally, developers and security engineers use threat modeling throughout the software development process — before the application goes live, not just after. Performing this activity can help produce a prioritized list of security improvements and make an application much more secure than it would have been otherwise, but it should not be expected to make it invulnerable.

In 2020, a group of threat modeling practitioners, researchers and authors got together to write the Threat Modeling Manifesto [25] in order to share a distilled version of their collective threat modeling knowledge in a way that should inform, educate, and inspire other practitioners to adopt threat modeling as well as improve security and privacy during development. The Manifesto contains values and principles connected to the practice and adoption of Threat Modeling, as well as identified patterns and anti-patterns to facilitate it.

2.1.1 Existing Threat Modeling Methods

Many independent authors and technology-related organizations have already engineered and published their own threat modeling methodologies. New methods are regularly published aiming at being better than the previous ones, either by identifying a wider range of problems or by doing so in a quicker or more effective way.

One of the most popular contributions in this area is the STRIDE security threat model. It was created by Loren Kohnfelder and Praerit Gargat at Microsoft in 1999 [26] and was supposed to be used by all of the company's products to identify various types of threats a product is susceptible to during the design phase. According to them, identifying the threats - based on the design of the product - is the first step in a proactive security analysis process. The next steps in the process are identifying the vulnerabilities in the implementation and then taking measures to close security gaps. STRIDE is an acronym that stands for: Spoofing of user identity; Tampering with data; Repudiability; Information disclosure (privacy breach); Denial of Service (D.o.S.); Elevation of privilege.

Since then, plenty of works have been using the STRIDE threat model as a basis of correlation to categorize the threats and vulnerabilities found in their methods.

Conklin et al., for instance, published an article [22] containing a structured approach for threat modeling that would enable a developer to identify, quantify, and address the security risks associated with their application. Their approach is based on a three-step process that involves: decomposing the application; determining and ranking threats; and determining countermeasures and mitigation. According to the authors, the first step allows the involved team to better understand the system at issue, by generating diagrams and the threat model (document) for it. Moving on, the second phase has the goal of identifying and categorizing threats - both from the attacker (using STRIDE) and the defensive perspective (using an Application Security Framework (ASF)) - besides determining the security risk for each threat, using a value-based risk model such as DREAD [27]. Finally, the third step conducts to the decision on what to do with the threats: accept - decide that the business impact is acceptable; eliminate - remove components that make the vulnerability possible; or mitigate - add checks or controls that reduce the risk impact.

The OWASP Foundation, in turn, issued an official document called OWASP Threat Modeling Project [28] providing information on threat modeling techniques for applications of all types. An important information mentioned by the authors is that the TM activity is best applied continuously throughout a software development project; the process is essentially the same, but at different levels of abstraction, as more details are added to the system and new attack vectors are created and exposed. Moreover, they claim the TM process should be based on four steps, represented by the questions: “What are we working on?”, “What can go wrong?”, “What are we going to do about it?” and “Did we do a good job?”, being agnostic when it comes to the tools used during this reasoning. Furthermore, another key point stated in this project is that “[..] threat modeling integrates into modern software development methodologies by asking "what are we working on, now, in this sprint/spike/feature?"”. As claimed by the authors, trying to answer this question can be an important aspect of managing security debt, despite possibly becoming overwhelming if asked in every single sprint meeting. If nothing significant has changed in the system’s architecture, then it is unlikely that the answers will change. When any part of the application changes, then it’s useful to examine what can go wrong as part of the current work package, and to understand what the team is going to address in this sprint and in the next one. Finally, during a sprint review/delivery, the question “Did we address these threats” is an important measure to mark the end - or not - of the current cycle of the system’s TM.

Specifically for the AI scenario, Mauri and Damiani [23] presented their methodology — STRIDE-AI — for assessing the security of AI/ML-based systems. It is, essentially,

an adaptation of the well-known Microsoft's STRIDE approach to the AI/ML domain, tailoring the threat modeling process to take into account the specificities of this sector. In this scenario, the authors discuss how to identify how assets generated and used at different stages of the ML life-cycle may fail, besides mapping these potential failure modes to threats and security properties these threats may endanger.

2.1.2 Threat Modeling and Modern Software Development Methodologies

Developing software is, most of the time, making it work according to the requirements, i.e., ensuring it presents the expected output behavior for a given input. Testing the application beyond its normal functioning conditions is an activity frequently neglected by developers, especially as the pressure for deliveries and productivity increases. This is one of the hardest points to handle when working under nowadays' software development methodologies: the time-value trade-off. In this scenario, knowing how to strike a balance between covering all aspects of the ideal software development process and being quick in finishing tasks becomes a very difficult job for the programmer [19].

Security stands out as one of the aspects most affected by the challenge programmers face in balancing productivity with meeting all the software system's needs. As a non-functional requirement with not so explicitly visible benefits, it is usually one of the first items to be put aside by software teams. According to Afaneh et al. [29], "putting in place strong security measures is a challenge that agile development teams frequently face". The cited work elaborates on this matter, stating that this can result from a series of possible issues. One of these, that mostly relates to the current discussion, is, according to the authors, the fact that agile development teams frequently place a higher priority on the speedy delivery of new features and functionality over security. This may result in a disregard for security best practices and the introduction of vulnerabilities in the software. Furthermore, they say it may be difficult for teams that are already overworked or for businesses that lack the resources to handle the added burden and complexity brought on by agile security. This is justified by the fact that adopting agile security frequently requires the purchase of new tools and technologies, as well as staff training in their usage, which can be even worse for enterprises with limited financial resources. Finally, the authors state that the fact that security testing takes up a large amount of time and money is one of the key obstacles to incorporating it into the development process. This practice is, therefore, sometimes viewed as an extra expense that could not immediately pay for itself, being put aside.

To try and solve this neither simple nor straightforward issue, the study published by Dave et al. [30] presents an "[..] outline of an agile design methodology to generate efficient, reliable and secure ML systems based on user-defined constraints and objectives". According to the authors, the integration with the fast-paced aspect happens because of

the design methodology proposed: "[..] the key idea is to define design space description, which can enable comprehensive exploration of arbitrary architectures for target ML workloads". In this scenario, they state that the dynamic development characteristic was not even a choice for the method, but a necessity, because sustaining acceleration becomes challenging as ML workloads evolve, requiring a special way to be dealt with. In the end, however, the development methodology part does not receive much focus and/or details in the text other than simple mentions throughout the method, which highlights the aforementioned difficulties of conciliating these concepts.

2.2 Artificial Intelligence

At its simplest form, artificial intelligence (AI) is a wide-ranging branch of computer science concerned with building smart systems capable of performing tasks that typically require human intelligence [31]. It, essentially, combines computing processes and robust datasets to enable problem-solving through statistical methods and trend prediction [32]. While AI is an interdisciplinary science with multiple approaches, advancements in the subfields of machine learning and deep learning, in particular, are frequently mentioned, gradually creating a paradigm shift in many sectors of the tech market.

AI works by combining large amounts of data with fast, iterative processing and intelligent algorithms, allowing the software to learn automatically from patterns or features in the data. This characteristic, therefore, enables a system to predict and infer the behavior of a hypothetical situation not exactly present in the real dataset.

It allows machines to model, or even improve upon, the capabilities of the human mind. And from the development of self-driving cars to the proliferation of generative AI tools like ChatGPT and Google's Bard, AI is increasingly becoming part of people's everyday life — and an area companies across every industry are investing in [33].

2.2.1 Types of Artificial Intelligence

Artificial intelligence can be organized in several ways, depending on stages of development or actions being performed.

For instance, according to Google's classification [34], four stages of AI development are commonly recognized:

- Reactive Machines: a type of limited AI that only reacts to different kinds of stimuli based on preprogrammed rules. It does not use memory and thus cannot learn with new data. IBM's Deep Blue which beat chess champion Garry Kasparov in 1997 [35] was an example of a reactive machine.

- Limited Memory: the most common type of modern AI; it can use memory to improve over time by being trained with new data, typically through an artificial neural network or other training model. Deep learning, a subset of machine learning, is considered limited memory artificial intelligence.
- Theory of Mind: this kind of AI does not currently exist, but research is ongoing into its possibilities. It goes beyond solving problems and describes AI that can emulate the human mind and has decision-making capabilities equal to that of a human, including recognizing and remembering emotions and reacting in social situations as a human would.
- Self-aware: a step above the previous, self-aware AI describes a mythical machine that is aware of its own existence and has the intellectual and emotional capabilities of a human. Like theory-of-mind AI, self-aware AI does not currently exist.

There are also three other categories - mentioned by this same source -, that classify AI into less specific, more general groups, according to its level of intelligence:

- Narrow AI: also called weak AI, focuses on one specific task and cannot operate beyond its limitations. It targets a single problem-solving operation and advances in that spectrum. Narrow AI applications are becoming increasingly common in people's day-to-day lives, having machine learning and deep learning as their most representative subfields.
- Artificial General Intelligence (AGI): would be the ability for a machine to "sense, think, and act" just like a human. It is conventioned that AGI does not currently exist.
- Artificial Superintelligence (ASI): The next and final level, in which the machine would be able to function - in every way - not just similar, but superior to a human being.

AI is a highly dynamic and always-evolving field. Making sure the recent developments in this area have enough quality and are safe for the final user is each time more critical.

2.2.2 Machine Learning

As a subfield of Artificial Intelligence, Machine Learning and its disciplines are, generally, comprised of AI algorithms that seek to create expert systems that make predictions or classifications based on input data. It is a technique that enables a system to

autonomously learn and improve using, for example, neural networks and deep learning, without being explicitly programmed, by feeding it large amounts of data.

ML allows computer software to continuously adjust and enhance themselves as they accumulate more experience. Thus, the performance of these programs can be improved by providing larger and more varied datasets to be processed [36]. The most common classification distinguishes four kinds of models used in Machine Learning [36, 37]:

- Supervised Learning: involves an ML model that uses labeled training data in order to match a specific set of input features - characteristics - to an associated output value. In this type of model, during the training phase, the ideal output of a determined input is known and provided along with it. The algorithm, then, learns the trend and, during the inference phase, is capable of receiving an input - with no label - and predicting its expected outcome. For instance, if the algorithm is trained with various images of apples, inputting - during the prediction phase - even a different unused image of an apple should result in the expected label. Common algorithms in this category include:
  - Linear Regression
  - Polynomial Regression
  - K-nearest Neighbors
  - Naive Bayes
  - Decision Trees
- Unsupervised Learning: involves an ML model that uses unlabeled training data to learn patterns. Unlike supervised learning, the expected output is not known during the training phase. Instead, the algorithm learns from the data being input and categorizes it into groups based on its attributes. For instance, if the algorithm is trained with images of apples and bananas, it will work by itself - during the prediction phase - to categorize which image has an apple and which has a banana. Common algorithms in this category include:
  - Fuzzy Means
  - K-means Clustering
  - Hierarchical Clustering
  - Partial Least Squares
- Semi-supervised Learning: a mixed approach, in which only part of the training data is labeled. In this type of learning, the algorithm has to figure out how to organize and structure the rest of the data to achieve the expected result during the prediction phase.

- Reinforcement Learning: involves an ML model that learns through a series of trial and error experiments. In this type of model, the algorithm learns to perform a defined task through a feedback loop until its performance is within a desirable range. More specifically, it receives positive reinforcement when it performs the task well and negative reinforcement when it performs it poorly. For instance, if the algorithm is learning to play a board game, it will start by simply moving pieces randomly and, as it receives feedback, will gradually perfect its moves.

Because ML is highly dependent on large amounts of data and - if not correctly programmed - might not always output the expected result, ensuring the accuracy and security of these systems is a very important task, that remains difficult and with no silver bullet, up to the present days.

2.2.3 AI/ML Security

As Artificial Intelligence applications become increasingly popular, and their interaction, closer to the user - even physically - a possible security breach could be catastrophic. It is, however, not so trivial to secure this kind of software, as it generally involves many different components - some not so well known by the general software worker. Malicious hackers, then, exploit this plurality of failure points and abuse the field's recent surge, combined with the lack of specific experience of its developers, to achieve successful attacks.

This complexity in increasing AI/ML applications' security happens mostly because these systems combine traditional software elements - such as front-end interfaces (Web, Mobile or Desktop), back-end servers, databases, etc. - with very specific AI parts. In this scenario, therefore, every feasible vulnerability existent in the domain of traditional cybersecurity can be a threat. As if that were not enough, the Machine Learning specific parts also bring together their own very distinct possible security failures.

When it comes to the specific parts of these systems - beyond traditional software parts -, the wide variety of different ML models and algorithms that can exist allows for a large set of possible vulnerabilities. The following list gathers - from various sources [38] [39] [40] [41] - the most common and well-known attacks to Machine Learning models:

- Training-time-focused attacks
  - Poisoning: tries to manipulate the model by altering the training data being fed to it:
    - Label manipulation: switches or alters input labels (in a supervised classifier) in order to degrade the model's future performance;
    - Input manipulation:

            - Direct poisoning of learning inputs: alters features of the data being input in order to change the future trained model for a specific behavior;
            - Indirect poisoning of the learning inputs (before pre-processing): poisons the data before pre-processing only to disturb model training unspecifically.

- Inference-time-focused attacks
    - Exploratory: tries to induce specific outputs by varying the input provided;
    - Oracle/model extraction: tries to extract the model itself by providing inputs, analyzing and combining their results;
    - Evasion/input manipulation:
        - Direct manipulation of model inputs: alters the feature values processed by the model to get different predictions:
            - Source-target misclassification/targeted: does so with the intention of getting a specific classification;
            - Simple misclassification/untargeted: does so with the intention of getting any different than optimal classification.
        - Indirect manipulation of model inputs (before pre-processing): alters the data processed by the model before pre-processing to get abnormal predictions.
    - Membership inference: tries to figure out whether a determined set of inputs was part of the model's training data;
    - Model inversion/training data extraction: attempts to reverse engineer the inferred results to recover training data.

- General-time attacks
    - Logic corruption: alters the ML logic/algorithm itself and the way it learns and infers outputs.

2.2.4 MLOps

It can be highly challenging to automate and operationalize AI/ML products, developing and rapidly bringing them into production. The paradigm of Machine Learning Operations (MLOps), therefore, addresses this issue. This concept includes several aspects, such as best practices, sets of concepts, and development culture [42]. In this scenario, various authors cite different MLOps principles, i.e., a value or a guide to how things should be performed. The work by Kreuzberger et al. [42] searched and gathered a set

of these concepts mentioned by diverse sources throughout the current literature. The resulting nine of these principles were:

1. CI/CD automation: provides continuous integration, continuous delivery, and continuous deployment. It carries out the build, test, delivery, and deploy steps. It provides fast feedback to developers regarding the success or failure of certain steps, thus increasing overall productivity.

2. Workflow orchestration: coordinates the tasks of an ML workflow pipeline according to directed acyclic graphs (DAGs). DAGs define the task execution order by considering relationships and dependencies between each step of the pipeline.

3. Reproducibility: is the ability to reproduce an ML experiment and obtain the exact same results.

4. Versioning: ensures the versioning of data, model, and code to enable not only reproducibility but also traceability (for compliance and auditing reasons).

5. Collaboration: ensures the possibility to work collaboratively on data, model, and code. Besides the technical aspect, this principle emphasizes a collaborative and communicative work culture aiming to reduce domain silos between different roles.

6. Continuous ML training & evaluation: means periodic retraining of the ML model based on new feature data. Continuous training is enabled through the support of a monitoring component, a feedback loop, and an automated ML workflow pipeline. Continuous training always includes an evaluation run to assess the change in model quality.

7. ML metadata tracking/logging: is required for each training job iteration (e.g., training date and time, duration, etc.), including the model-specific metadata — e.g., used parameters and the resulting performance metrics, model lineage: data and code used — to ensure the full traceability of experiment runs.

8. Continuous monitoring: implies the periodic assessment of data, model, code, infrastructure resources, and model serving performance (e.g., prediction accuracy) to detect potential errors or changes that influence product quality.

9. Feedback loops: are required to integrate insights from the quality assessment step into the development or engineering process (e.g., a feedback loop from the experimental model engineering stage to the previous feature engineering stage). Another feedback loop is required from the monitoring component (e.g., observing the model serving performance) to the scheduler to enable the retraining.

These authors also mention the importance of a few tools and components besides the ones involved in the principles aforementioned. Amongst them, are:

1. Feature Store System: ensures central storage of commonly used features. Generally, made out of databases, this is where most of the data for training ML models will come from. Moreover, data can also come directly from any kind of data store.

2. Model Registry: stores centrally the trained ML models together with their metadata. It has two main functionalities: storing the ML artifact and storing the ML metadata.

3. Model Serving Component: can be configured for different purposes. Examples are online inference for real-time predictions or batch inference for predictions using large volumes of input data. The serving can be provided, e.g., via a REST API. As a foundational infrastructure layer, a scalable and distributed model serving infrastructure is recommended.

4. Monitoring Component: takes care of the continuous monitoring of the model serving performance (e.g., prediction accuracy). Additionally, monitoring of the ML infrastructure, CI/CD, and orchestration are required.

2.2.5 AI/ML Architecture

The ultimate goal of an AI/ML application is, generally, not simply engineering a model, but making it available to receive inputs and make predictions. This is usually done by embedding it into an application that can handle this kind of interaction, such as a Web/Mobile front-end backed by an API and the whole adjacent architecture this type of system generally needs.

The characteristics of this architecture and the way it should be standardized are often mentioned by academic articles, such as [43] and [44]. Wöstmann et al. [44], for instance, engineered a conception of a reference architecture for Machine Learning that includes not only the model but the IT Systems underlying it, such as databases and supporting applications, citing even edge devices, such as sensors. The authors present each of these involved components in detail through visual diagrams and, in the end, merge them all into a single architecture diagram. This representation, then, highlights each of the individual elements, their scope and interconnections, serving as relevant guidance on the current use of these technologies.

In the same line, Salvucci [43], for example, mentions in his work an example end-to-end architecture for ML applications. It is actually a combination of different well-known software components put together by industry company Logical Clocks and named Hopsworks [1]. In this architecture, shown in Figure 1, the diverse technologies

used segment and simplify the end-to-end pipeline, providing a higher level of control for developers to work with when designing new ML software.

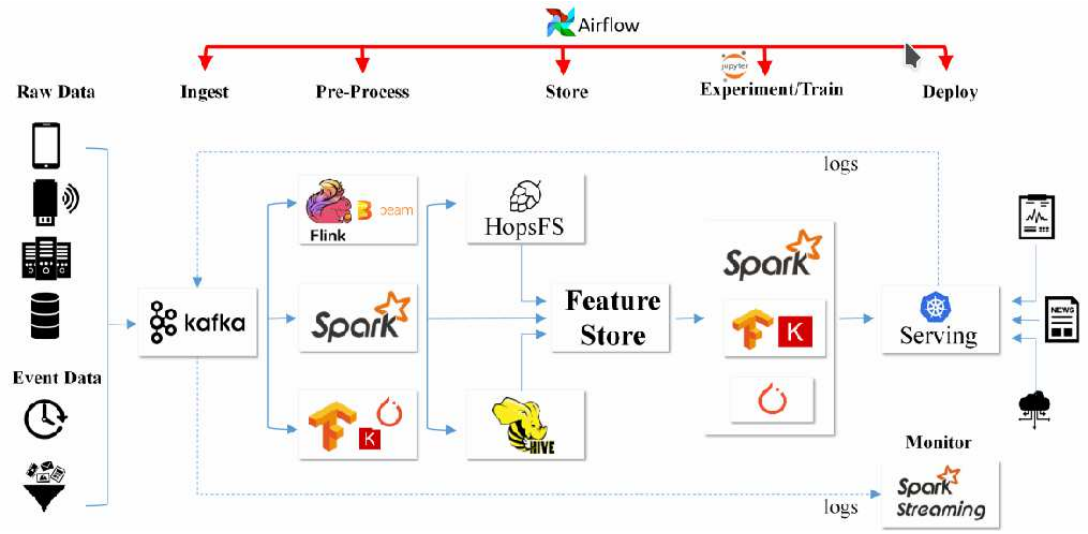

Figure 1 – Example of an ML Architecture. Reprinted from "Horizontally scalable ML pipelines with a feature store", by Ormenisan et al. [1].

sAIfe is going to base itself on this concept and rely on a similar architecture, with a couple of key differences: sAIfe will aim at generalizing the pipeline, as it has a premise of being technology-agnostic and not restricting to specific components; sAIfe will target a lower level of complexity, depicting the architecture in a slightly higher level of abstraction.

2.3 Related Work

The sudden popularization of the Artificial Intelligence field [4] - led mostly by advances in Machine Learning - culminated in a rush of applications being developed and algorithms experimented without much concern for security - or enough previous knowledge to apply it - as it happens in most novel technologies. As the bubble started to burst - and ML applications got associated with real-time critical scenarios -, researchers began to realize that these pieces of software could be a major access door to impact sensitive systems. Since then, an ever-increasing number of studies have been released every year analyzing security breaches, exactly as confirmed by [45], who mentions this growing quantity corroborated by a market report [46].

This line of work is one of the two core subdivisions into which research in the field of AI/ML security can be divided. However, as stated previously by this document, it focuses mainly on possible attacks and/or defenses. These are generally highly specialized, individually crafted against specific scenarios [47, 48, 49, 50, 51], usually under the denomination of Adversarial Machine Learning (AML) [52]. It is, therefore, known for typically addressing the topics with a narrow focus on the specific type of compromise

being targeted, neglecting a wider analysis of the overall security conditions. For this reason, they are not so useful for an average industry programmer seeking instructions.

The second research branch, on the other hand, broadens the scope, performing studies that either catalog threats - now, unspecific -, organized in a taxonomy (first approach), or present new threat-modeling processes for AI/ML systems, with varied approaches and diverse methods, sometimes under the denomination of framework (second approach).

The first approach is used, for instance, by Fazelnia et al. [53], who got together countless types of AI/ML attack strategies, mitigation techniques and tools on a common database, all organized under their created taxonomy, available through an interactive tool. It does, however, require the framework user to have significant specialized knowledge on the area, to be able to discern and identify which of the provided scenarios - among the big number of names and classifications - is going to be applicable and useful to their specific situation, which is not so easy or quick, especially for a general developer who is not specialized in AI.

Moving forward to the second approach, Grotto and Dempsey [54] perform a thorough analysis on the current situation of AI/ML security. They present a new classification of the possible kinds of threats - under another taxonomy - and provide a series of recommendations for dealing with vulnerability disclosure and management in these systems. The authors base themselves on existing traditional cybersecurity literature, which, according to them, is still totally valid content for these cases, that should not be siloed from AI-specific topics. The resulting document, however, only cites suggested approaches and does not include direct practical instructions, still being on a slightly higher level than the developers would need it to be, to be useful *per se*, with no further efforts.

Additionally, the American National Institute of Standards and Technology (NIST) has published a series of documents focused on Artificial Intelligence security. Among them, the most aligned to this context, titled AI Risk Management Framework (AI RMF) [16], presents a very extensive collection of definitions and explanations for a series of concepts related to AI, especially the ones that intersect with the security aspect - such as safety and privacy. The paper also provides very extensive guidelines on how to structure AI software development regarding the involved personnel and the team organization, under a risk governance view. In the end, however, the deep non-technical topics addressed, alongside the lack of practical technical prescriptions, might make it lose direct value for the average AI/ML developer - in most cases, searching for a more explicit set of instructions.

Furthermore, Marshall et al. [17] present their insights on how to threat model an application that involves AI/ML capabilities. The authors list a series of questions that should be made by the development team during the process and also provide a list of

AI/ML-specific threats and their respective mitigation forms. However, besides focusing only on the AI/ML-specific parts and neglecting the rest of a possible software system, the chosen classification, combined with the large number of suggested questions, makes the method open to excessive subjectivity and turns its application time-consuming.

Still in the same line, Kumar et al. [55] introduce a new threat modeling process for AI-based applications. They develop a model of the software development process for these systems and separate them into three main phases: Data Processing, Model Development and Deployment, to which they assign a set of possible threats, respectively. The generated method, then, requires the user to represent the analyzed application according to the reference model provided, identify the matching phases and proceed to check the list of threats offered by the process. Once again, however, this work focuses on AI/ML-specific parts and addresses very few aspects of traditional cybersecurity, besides not presenting actionable remediation suggestions.

Finally, Mauri and Damiani [23] develop an entire threat modeling process, applying key concepts of the well-known STRIDE framework [56] adapted to the new AI/ML scenario. The authors introduce a relation table between STRIDE - threats - and the CIA^3 - R hexagon [57] - security properties -, in which the former elements act as attackers of the latter ones, both tailored to the ML-specific scenario, to guide security practitioners during security assessment tasks. In the end, however, the authors make clear their awareness of the fact that a framework that only identifies threats and does not provide actionable mitigation suggestions is ineffective. They even state that an optimal method like that is not available in the literature yet.

Compilation articles with characteristics close to those mentioned above [5, 54] can be used as a basis for understanding the state-of-the-art of the area in question from a security perspective. They, nonetheless, still fail partially when it comes to providing direct and practical value to industry professionals, given that they make higher-level analyses and do not always suggest objective, ready-to-use resolutions, which is, for instance, not so attractive to modern-day industry development methodologies, that benefit more from easy and quick processes [18, 19, 20].

Therefore, when compared to the cited studies, the present work has the potential to fill a very important gap, already identified by the scientific community and detailed throughout the current section. In order to do so, the proposed method must be capable of being useful to the industry developer, putting the non-academic public first, as the target audience. Focusing on this segment means developing a solution that can be easy to understand and quick to be applied, besides showing explicit potential of gain with the results provided, to be perceived as a worthwhile tool.

Consequently, creating a method that can both help to identify security flaws and already provide a way of fixing them acquires a high level of importance under

the discussed circumstances. The ability to present itself as a complete security tool, that, unlike others, is not complicated nor time-consuming to use, would grant it all the characteristics needed to solve the problems with existing approaches. Thus, this project proposes a complete ML application security assessment method that can both help to identify security flaws and already provide a way of fixing them. It helps the developer identify critical areas in their application and provides a compilation of possible threats and technical solutions ready to be used, for both traditional and ML-related components.

Table 1 shows a comparison between the aforementioned studies and sAIfe, when it comes to three main flaws: overfocus on the AI part; excessive subjective questions; and lack of practical suggestions. In this representation, the fewer marks a work has, the better.

Work	Overfocus on AI	Subjective questions	Lack of practical suggestions
Grotto and Dempsey [54]			X
AI Risk Management Framework (AI RMF) [16]			X
Marshall et al. [17]	X	X	
Kumar et al. [55]	X		X
Mauri and Damiani [23]		X	X
Hu et al. [5]	X		
sAIfe (this work)			

Table 1 – Related work comparison

3 SAIFE: A LIGHTWEIGHT THREAT MODELING METHOD FOR ML-ENABLED APPLICATIONS

This section introduces sAIfe[1], a new method for threat modeling applications that include ML capabilities. It consists, mostly, of graphical conceptual models that help the developer describe the architecture of the system under development and its components, besides identifying which of these demand greater attention when it comes to security aspects. Additionally, for each of these detected components, the proposed process provides lists of possible threats and their respective solutions - obtained from the analysis and curation of sources such as OWASP [15] and NIST [16] - presented in a very clear and concise way. From then on, the developer can more easily detect possible weaknesses in their application and quickly obtain access to mitigation forms. In this scenario, offering the protocol and the lists in a prescriptive format spares the programmer from reading various separate documents and analyzing complex taxonomies while trying to improve the security of their software. Finally, the goal of sAIfe is not covering every single problem an ML-enabled system might have, but providing a practical assessment method that includes the most common security risks, which, if corrected, can significantly decrease the chances of an attack being successful.

3.1 Research Methodology

sAIfe was engineered based on the Design Science Research (DSR) paradigm, which states that researches should aim at developing unspecific generalized design knowledge in a specific field to help practitioners create specialized solutions to their problems [58]. Moreover, according to [59], the DSR approach typically involves the creation of a novel artifact as a means to improve the current state of practice of a science field. sAIfe is, therefore, aligned with both statements, given that, respectively, it provides knowledge eligible to be applied to any ML-enabled application and, as a method, is the artifact itself. The DSR process generally includes six activities, as cited in [60]. Table 1 presents the research steps and explains them in sAIfe's specific context.

sAIfe has been designed with a set of key characteristics in sight. They served as guidelines during the creation process and generated a resulting method that is, unlike other existing approaches:

a) focused on being programmer/developer-friendly, as its process runs around the development team and is meant to be carried out by a technical person, whose

[1] The chosen name involves a wordplay on how software users should be during utilization - safe - and the main area the project is concerned with - Artificial Intelligence (AI)

DSR steps	sAIfe
(1) identification of the problem	How to help developers increase the security of their ML-enabled applications under development using one single threat modeling method that contains a compiled knowledge base, identifies possible problems and already suggests solutions for them
(2) definition of objectives for the solution	Construction of a method that is: developer-friendly; quick and easy to apply; intuitive and visual; with explicit and ready-to-be-implemented suggestions; compatible with modern software development methodologies
(3) design and development of artifacts (constructs, models, methods, etc.)	Engineering of sAIfe's method itself, with its graphical and textual elements, including: 1) identification of the most common ML system's architectures and components, through a thorough study of ML infrastructure documents; 2) a component problem/solution knowledge base, generated after a careful curation process on various sources, including security rankings and academic articles; 3) creation of a visual artifact a developer can use to represent their system
(4) demonstration by using the artifact to solve the problem	Application of the method on a real-world application, in the form of a case study, to serve as a practical example and test its technical results (described in Section 4.2), besides a validation with researchers in academia (in Section 4.5) and another one with developers in industry ((in Section 4.6), to assess its efficiency and ease of use
(5) evaluation of the solution, comparing the objectives and the actual observed results from the use of the artifact	Case Study analysis and method results discussion (detailed along Section 4)
(6) communication of the research to other researchers and practicing professionals	The present paper

Table 2 – DSR steps explanation for sAIfe

interests were put first during the method ideation - from a programmer to another;

b) capable of combining traditional cybersecurity and ML-specific topics;

c) pragmatic, direct, easy and quick to apply [21], as its steps and guidelines are intuitive and visual, take little time to be applied and lead to results that are more explicit and ready to be implemented;

d) suited for modern software development methodologies, as the combination of its characteristics generates a reliable method that can prevent problems and the need for alterations in future phases of development. This enables important cost-saving opportunities, that, when matched with the method's short application time, generate a very attractive cost-time trade-off: one of the pillars of modern software project management.

3.2 sAIfe's Threat Modeling Process

This section introduces the sAIfe threat modeling process, presenting and explaining in detail the steps and, in the subsections, each of the involved elements. sAIfe was designed to be applied from the early stages of the software development life cycle. Therefore, as soon as the software application scope has been defined and the initial architectural choices have been made, the process can be carried out with the following steps:

- Step 1: The developer builds the Architecture Diagram: a visual conceptual model that represents their system's architecture. sAIfe provides a set of guidelines to assist the developer through this construction, which are presented in Section 3.2.1. This step helps them perform an abstraction process on the architecture in question, identifying points that may require attention from the security perspective. Within sAIfe, these points are referred to as Application Parts;
- Step 2: After that, for each of the Application Parts used - now, in the diagram, referred to as Critical Areas - the developer checks the respective list of possible threats - also provided by sAIfe, under the name Critical Area Detailed Document [Section 3.2.2] - and selects the items applicable to their software application;
- Step 3: Finally, for each of the threats selected, they pick, from the respective list of possible mitigation forms, all the entries applicable to the system under analysis. This mitigation list is part of sAIfe and, like the list of possible threats, is compiled from multiple information sources about cybersecurity and AI.

This process has been inspired by and originated, partially, from a mixture of a couple of other known abstractions: from architectural risk analysis (ARA) [61], for instance, sAIfe's process shares the focus on the system infrastructure as a whole; moreover,

similarly to data flow diagramming (DFD) [62], it values the information cycle throughout the application pipeline and the diagram representation. These ideas have been selected because they match sAIfe's requirements when it comes to ease of visualization and speed of understanding, due to their intrinsic graphical aspect. The process can be better understood by looking at the scheme in Figure 2. Sections 3.2.1 and 3.2.2 explain each of the involved method's elements in detail.

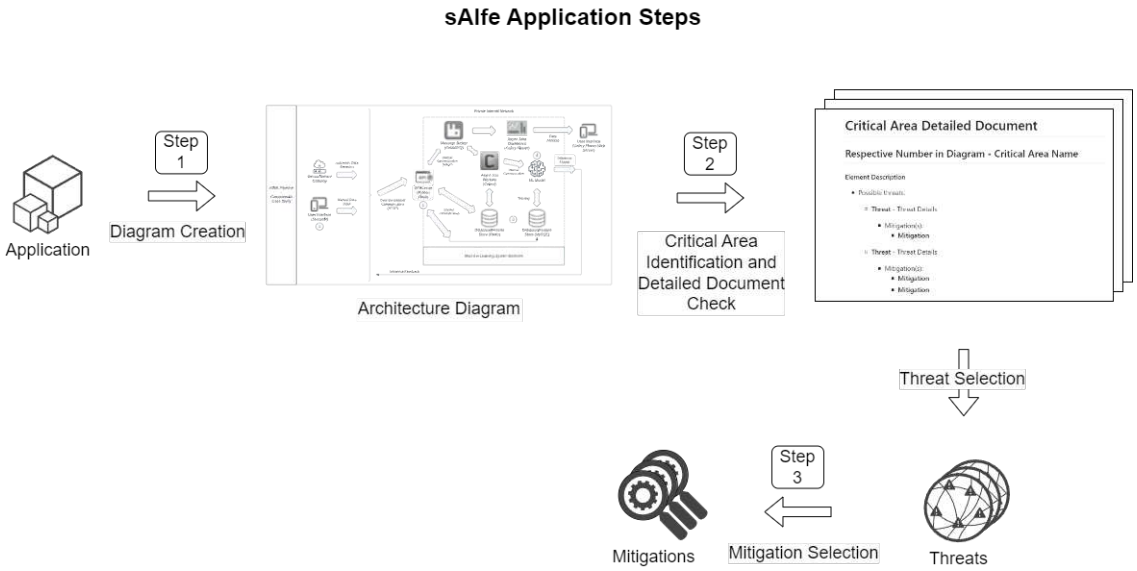

Figure 2 – sAIfe's Process

3.2.1 Architecture Diagram Guidelines

sAIfe's method is initially based on the Architecture Diagram created by the developer. In this scenario, depicting the application to be analyzed in an adequate way becomes important, as to achieve a proper representation and a good level of coverage. sAIfe provides suggested guidelines for the construction of this artifact, as more prescriptive and systematic steps should be easier for the programmer to follow, rather than starting from scratch with zero instructions.

Initially, the developer should virtually segment the layers of the system to be analyzed according to the following level of abstraction: front-end - encompassing components a user can directly interact with and/or that are the initial source of data - and back-end - regarding elements that are not directly accessible by the user and/or only receive data from other parts of the application itself. This segmentation will more easily highlight the involved components, that should feature, for instance, amongst others: user interfaces and sensors, under the front-end scope, and server applications, databases and ML assets, under the back-end section. These elements do not need to be specialized (such as identifying a specific language or vendor);

The identified components are called “Application Parts” by sAIfe, and should be represented graphically in the diagram. These elements, in turn, can all be highly interconnected and each of their data exchange flows - referred to as “Connections between Application Parts” by sAIfe - should also be depicted in the developer’s representation. Finally, the union of a single Application Part and its adjacent Connections is deemed as the most relevant entity by sAIfe and receives a special denomination: “Critical Area” - a section of an application in which security flaws are likely to happen.

For a general ML-enabled system, sAIfe considers there could be typically four principal Critical Areas (zero or more of each): User Interface applications (for data entry and output visualization); APIs/Servers (to interconnect different parts of the application and exchange data); Databases/Feature Stores (to hold all the non-volatile data); and Machine Learning models (to actually calculate inferences). To help the developer with a more concrete example, sAIfe provides the Reference Architecture Diagram: a ready-made graphical diagram of an example ML-enabled software application, created according to the method’s guidelines. It is a reference made available to the method applicant to be used as an example while creating their own Architecture Diagram and structuring the important parts. It, essentially, contains the architecture of an entire example application, using all four Critical Areas mentioned, portraying possible computational elements participating in the process, along with the connections established between each of them. There is no restriction on the number of Critical Areas used (either of the same kind or not), it is up to the programmer and depends on their system.

Figure 3 shows the Reference Architecture Diagram, with front-end elements on the leftmost part and back-end ones on the right. It portrays Application Parts represented as icons; Connections depicted as arrows; and Critical Areas marked with numbers, while Figure 4 zooms in and illustrates one of the Critical Areas found in it. Additionally, Figure 5 goes deeper and highlights the Application Part contained in this section, whereas Figure 6 portrays the Connections between Application Parts.

By employing a higher level of abstraction and being technology-agnostic, the Architecture Diagram artifact aims at being able to represent - conceptually - the technical organization of a wide range of diverse systems in this field. In the end, therefore, this segmentation should highlight the most common software components present in nowadays ML-enabled applications - as corroborated by other authors [43, 44, 63].

3.2.2 Critical Area Detailed Document

A Critical Area Detailed Document is a security document repository that contains a compilation of information about a single specific Critical Area, including possible security threats and proposed mitigation forms. It is, therefore, composed of the sections described as follows:

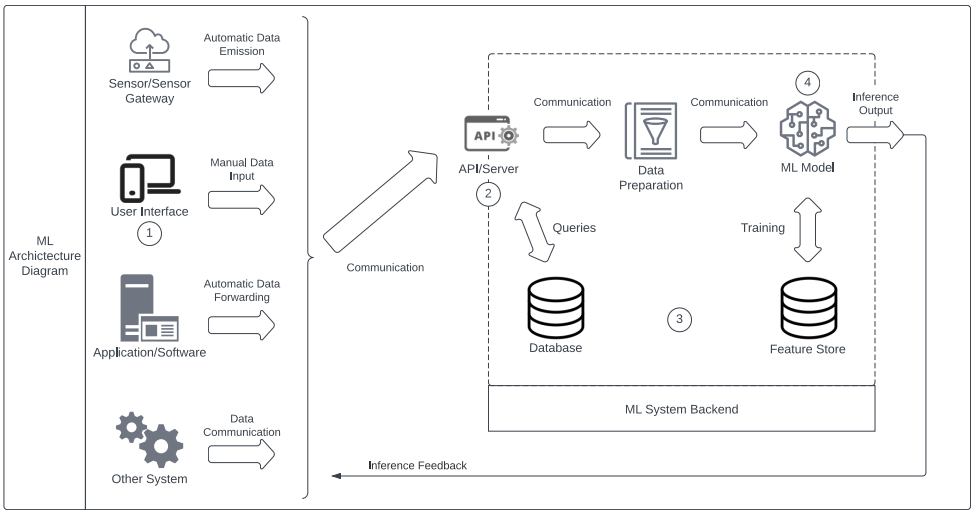

Figure 3 – Reference Architecture Diagram

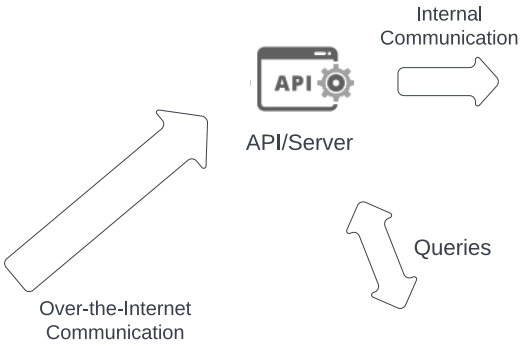

Figure 4 – Example of Critical Area

Figure 5 – Example of Application Part

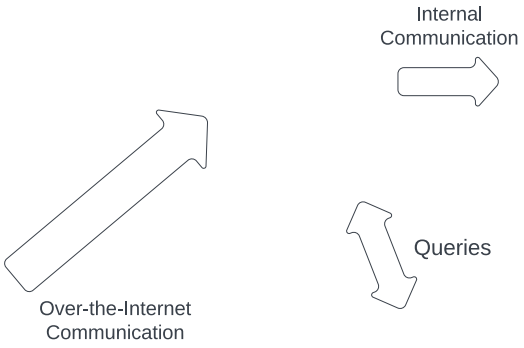

Figure 6 – Example of Connections between Application Parts

- Element Description: short contextual description about the Application Part in question;
- Threat[2]: possible weakness/failure point the Critical Area at issue might have, followed by one or more concrete examples;
- Mitigation[3]: suggested prevention action for the respective threat.

sAIfe provides documents for the four principal Critical Areas mentioned earlier (User Interface, API/Server, Database/Feature Store and Machine Learning model), in line with the segmentation mentioned by the Architecture Diagram Guidelines (Section 3.2.1). The contents, presented in full in sAIfe's GitHub repository[4], aim at being practical summaries of their respective areas and are to be used as reference for both the threat selection in Step 2 and the mitigation picking in Step 3 of the method application process.

These lists themselves are a result of an extensive search throughout available scientific publications and similar projects, whose content has been got together, treated and filtered, as to encompass the biggest and most diverse number of risk possibilities and their respective mitigation alternatives. Some of the main sources of data included OWASP [14, 15] - with their published application security rankings and vulnerability lists - and NIST [16] - with their AI risk management framework -, for instance. Furthermore, specific ML-related threats and solutions were also curated from scientific articles - such as [53] and [17] -, which culminated in a scenario that encompasses seamlessly both traditional cybersecurity elements and ML-specific assets, treating the ML part as an extension of regular software [54], which, in turn, allows all the well-known conventional security recommendations to be applied. Finally, this knowledge base allows for easy scalability, and, as cybersecurity is a dynamic and ever-evolving field, the base has the ability to be easily changed and stay up-to-date on new data - as new documents are published -, renewing its value in future versions. To illustrate, Figure 7 shows the typical structure of a Critical Area Detailed Document, with hierarchical topics.

[2] Each Critical Area Detailed Document contains many threat entries
[3] Each Threat item can contain one or more mitigation forms
[4] <https://github.com/gabriel-messas/sAIfe/tree/main/Critical%20Area%20Detailed%20Documents>

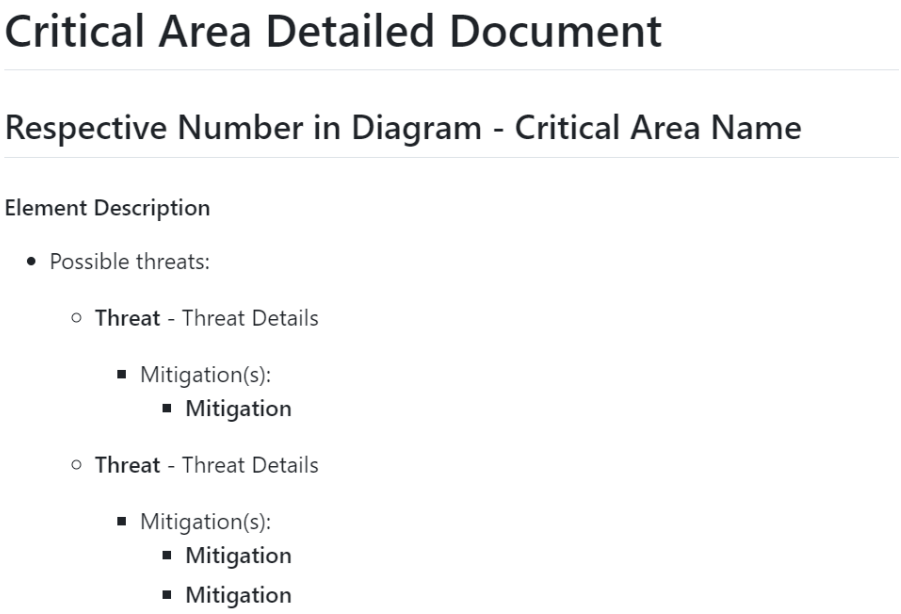

Figure 7 – Structure of a Critical Area Detailed Document

4 RESULTS

This section presents the main results obtained when evaluating sAIfe. First, it analyzes the method application by using a real case scenario system as the study object, and then, a second case study is described, now using STRIDE-AI, a different method present in the literature, followed by a comparison between both. In addition, a separate validation with researchers in academia is also carried out and its results described. Finally, another evaluation of the method is run, this time with the help of software developers.

4.1 ConsistencIA

The application to be analyzed both in Sections 4.2 and 4.3, named ConsistencIA, is one of the practical results being developed under the scope of a research group at the State University of Londrina (UEL). The latter, labeled CIA-Agro (an acronym, from Portuguese, that means Center for Artificial Intelligence in Agro) is a joint effort focused on studying and applying high-tech AI solutions to the agribusiness field.

ConsistencIA itself is a project mainly concentrated on accumulating climate data and detecting anomalous variations across planting regions using a Machine Learning approach. It was thought of as an online system that could be trained on a data batch, initially, and then, periodically receive a set of statistical weather data as input, returning, as inference, a prediction on possible anomalous weather characteristics for the periods of time in question.

Due to the need of receiving data from sensors in remote locations, the system has to be reachable publicly over the Internet. For that, a REST HTTP API was the selected component to handle incoming data. Moreover, to be easily accessible by the interested audience – possibly non-technical –, a graphical interface was planned, and the platform chosen for it was the Web.

4.2 Case Study: ConsistencIA Analysis Using sAIfe

As soon as the initial requirements and architecture were defined for ConsistencIA, the sAIfe method steps could be started.

4.2.1 Architecture Diagram Generation

The first step was building the Architecture Diagram of the entire system, based on the Reference Architecture Diagram provided. For ConsistencIA, this element – graph-

ically generated – is the one represented by Figure 8.

The image shows each Application Part – as icons – and the Connections between them – as arrows. Both are depicted separately by the functions they carry and are labeled generally; when a specific technology is worth being mentioned, it is shown written between parentheses. Moreover, Critical Areas are identified with little circles, numbered according to the method specification: 1 represents a User Interface; 2 highlights an API/Server; 3 shows Databases/Feature Stores; and 4 marks an ML Model. In short, this architecture consists of user interfaces and connected sensors – as "front-end" elements, portrayed on the leftmost edge –, and the core processing services, such as the ML model itself – as "back-end" elements, highlighted on the inside of the dashed box. These ends are, then, interconnected by an API – positioned in the middle of the pipeline.

To increase accuracy and to make the diagramming process easier, even Application Parts that do not have a direct correlation with those specifically mentioned by the method were included in the graphical representation. This is why elements such as message queues and async job workers appear in this system diagram, being up to the method user's discretion.

Hypothetically, to speed up the process, at the cost of visualization and verisimilitude, a more experienced method applicator could already depict the application with no additional elements other than the core ones covered by sAIfe. This action would save a certain amount of time during the comparison in the next step.

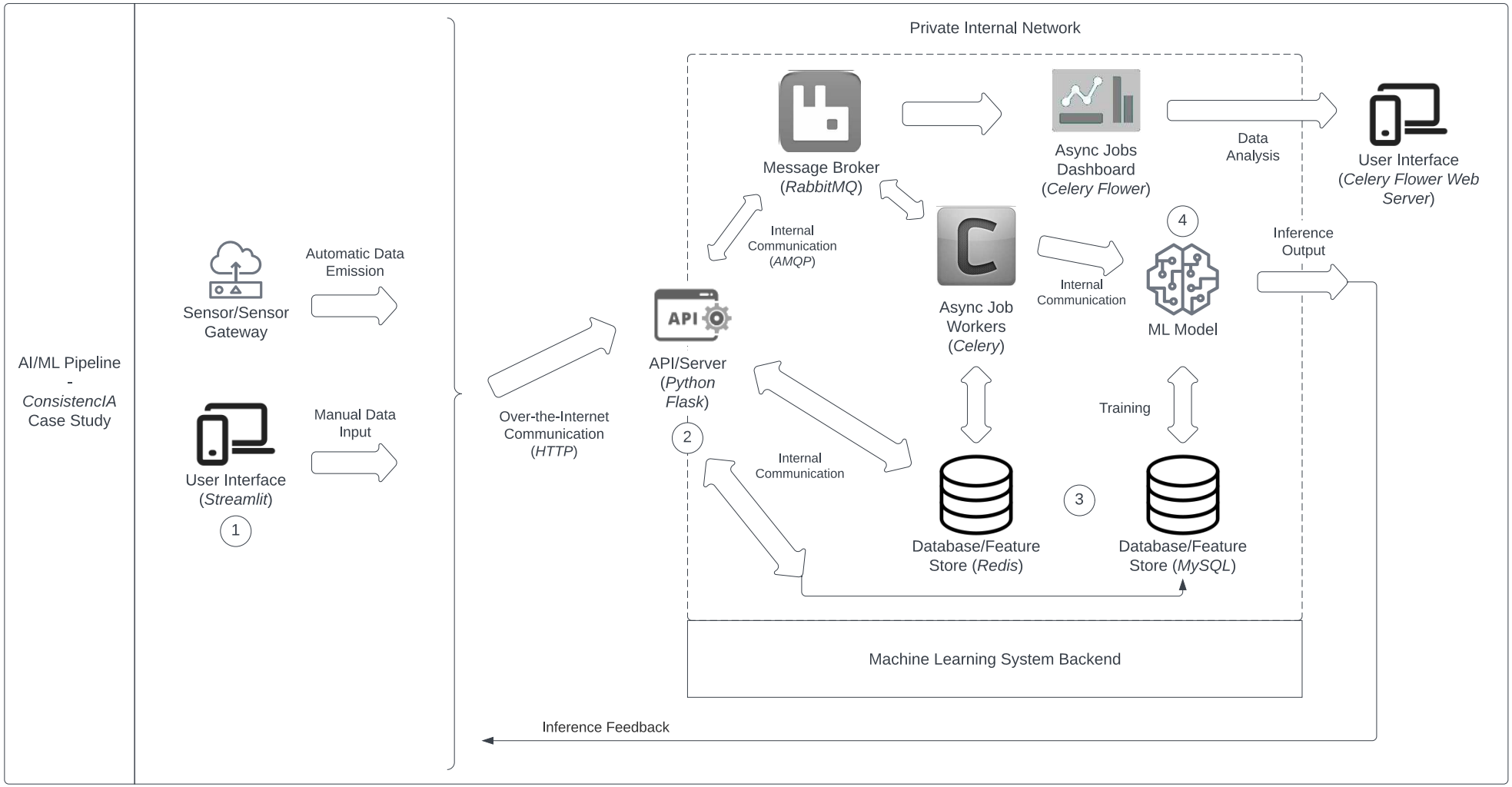

Figure 8 – ConsistencIA Architecture Diagram

4.2.2 Critical Area Analysis

The second step consisted of analyzing each of the Critical Areas - covered by sAIfe - present in the diagram and resorting to their respective detailed document (provided by the method), with the goal of picking – from the list – weaknesses that are capable of affecting the system under analysis.

For the application in this case study, this step was repeated four times: once for each Critical Area found previously. The complete analysis report, presented in full as an appendix section [A], shows every single item classified as a risk for ConsistencIA and its explanation. The following paragraphs summarize the results and highlight relevant points found in the process.

4.2.2.1 User Interface

In this category, the application showed possible vulnerabilities of all the five subtypes covered by sAIfe for this area: Injection Flaws, Authentication Flaws, Authorization Flaws, Security Misconfiguration and Sensitive Data Exposure. For this system, the User Interface Critical Area is a very important and delicate section, as it can be the beginning of a whole chain of vulnerability exploitation and security compromise, due to its data input capabilities and initial position on the system pipeline.

Even though Streamlit - the front-end visualization framework chosen for the Website - already has a certain level of protection against unsanitized data in its components, a raw input element used by the developer could generate an entrance door for a malicious user. Restricting the content that can be inserted and performing some form of validation, already at this first stage, can be highly beneficial, as it can reduce the burden left for the three following Critical Areas ahead in the pipeline (API/Server, Database/Feature Store and Machine Learning Model) and address a problem before it travels deeper in the pipeline.

Overall, the risks found here are, at first glance, all related to traditional cybersecurity, focused on well-known concepts. A not-so-obvious consideration is that every oracle or evasion ML attack mechanism - i.e. that relies on testing inputs to obtain outputs - starts in a data entrance functionality: a solid proof that simple actions on a User Interface can save complicated algorithm implementations on an ML Model.

4.2.2.2 API/Server

In this category, the application showed possible vulnerabilities of all the seven subtypes covered by sAIfe for this area: Injection Flaws, Authentication Flaws, Authorization Flaws, Security Misconfiguration, Sensitive Data Exposure, Resource Exhaustion and Mass Assignment. For the analyzed application, the API/Server Critical Area is also

a relevant section, as it is, most times, the system component with the greatest amount of computational resources available, allowing for more and heavier defense tools against unwanted malicious behavior. The software parts ahead of it in the pipeline generally have fewer protection mechanisms capable of blocking an active attack.

Moreover, in the case of sensors and other devices communicating directly with the API, the latter is now the one and only security barrier left - as the User Interface is being bypassed - before getting to the most sensitive sectors.

This Critical Area generally has a strong connection with the previous one (User Interface) when it comes to limiting access and treating exchanged information, as they tend to share these risks and the responsibility for their mitigation. Therefore, in the end, the risks found here are also prone to be related to traditional cybersecurity topics, very similar to those of the previous section. Even though it might seem repetitive, each software component has to make sure it addresses its vulnerabilities as if no other is doing so; security is a joint effort.

4.2.2.3 Database/Feature Store

In this category, the application showed possible vulnerabilities of all the six subtypes covered by sAIfe for this area: Authentication Flaws, Authorization Flaws, Security Misconfiguration, Sensitive Data Exposure, Data Loss and Resource Exhaustion. For this system, the Database/Feature Store Critical Area is a very sensible, defenseless section, as it is the center of the data and, generally, simply performs the operation requested, indistinctly, assuming the authenticated user is rightfully entitled to it.

The risks cataloged here go along the same lines as the ones of the previous Critical Areas, as they mention primarily the authentication and authorization pair, a recurrent vulnerability unit. Therefore, interestingly, the risks assessed so far are, in their vast majority, the result of a very similar set of vulnerabilities, perpetrated as a serial chain along the application flow. Furthermore, counterintuitively, none of them seem to be exclusive to an ML application, showing that traditional cybersecurity topics are still highly valid in this scenario.

4.2.2.4 Machine Learning Model

In this category, the application showed possible vulnerabilities of all the five subtypes covered by sAIfe for this area: Training Data Integrity Flaws, Inference Output Integrity Flaws, Model Logic Confidentiality Flaws, Training Data Confidentiality Flaws and Model Logic Integrity Flaws. For this system, the Machine Learning Critical Area is, theoretically, the most important section, just like in any other ML application, as it represents the deepest stage of the data flow.

Here, in turn, the results have come out quite different than the common ones from previous areas. As a Machine Learning Model generally does not have any kind of security interface *per se*, vulnerabilities such as authentication and authorization simply do not exist, showing this component relies on preceding software parts to perform such filter. Therefore, if it all fails and a malicious input gets to the model, the latter has to be capable, alone, of defending itself and responding accordingly; when it is not, that is when risks arise.

Four out of the five risk subtypes found are closely related to the oracle or evasion ML attack mechanism - the most common malicious exploitation form -, now bringing up these well-known offensive concepts exactly where they take place. The last subtype (Model Logic Integrity Flaws), however, stands out, as it relates to an attacker having direct access to the programming code that guides the model and literally altering it.

80% of the risks (the first four) are ML-exclusive, while only 20% (the last one) relate to traditional cybersecurity. The ML Model Critical Area is the first - and last - software section in which this inversion of dominance happens. Even here, however, traditional cybersecurity is not absent.

4.2.3 Solution Gathering

The third and last step was, essentially, checking the method's documentation, for each of the vulnerabilities found in the previous step, and implementing the appropriate solutions to the system under design. For ConsistencIA, this step was also repeated four times: once for each Critical Area found previously.

The complete solutions report, presented in full as an appendix section [A], shows every single mitigation item classified as eligible for ConsistencIA and its explanation. The following paragraphs summarize the results and highlight the most interesting points found in this step.

4.2.3.1 User Interface

For this category, all 13 available solutions were selected from sAIfe's repository for ConsistencIA. The majority of mitigation approaches for the User Interface Critical Area revolve around passive solutions, such as appropriate software configuration - with adequate application versions, proper installation and parameterization - and data encryption - with HTTPS (TLS).

One of the only items that gets closer to an active defense type is Input Sanitization. This is also the solution that is most directly connected with the ML aspect itself, as the inference query will start with this data input. Performing type and format validation alongside sanitizing known dangerous expressions is a simple yet effective way to reduce

risks that could culminate in a Database or ML Model attack.

4.2.3.2 API/Server

For this category, all 18 available solutions were selected from sAIfe's repository for ConsistencIA. Similarly to the previous category, the primary focus for mitigation measures within this API/Server Critical Area also leaned towards passive solutions. They were mostly related to proper server configuration and data encryption, keeping the trend seen immediately before, as these topics are the standard security recommendations for Web-related server applications.

Here, however, the active defense - Input Sanitization - gets significantly more powerful, as there are no more environment restrictions and the code does not have to run on a client-side browser. In this scenario, with the help of database connection libraries, sanitizing a SQL query, for instance, becomes easier. Moreover, the ability to add auxiliary server-side libraries can make the process of validating a request and its content easier.

The actual difference in this section came from the Resource Exhaustion subtype. Defining proper computational resources configuration and well-established limits for the server prevents a malicious user from easily overloading the system and causing general outage, which could impact directly the Machine Learning Model availability.

4.2.3.3 Database/Feature Store

For this category, all 12 available solutions were selected from sAIfe's repository for ConsistencIA.

Defense suggestions for this Database/Feature Store Critical Area followed the trend and kept orbiting around more passive solutions. These involved similar topics, such as adequate software configuration and communication encryption, proving these concepts are important in more than one Application Part.

As database servers are not common programming interfaces, executing code here tends to be unusual and a bit more complicated. More importantly, as the query content itself can be maliciously crafted by an attacker, allowing it to be processed - as is - is already dangerous in the first place. Because of that, most of the filtering should be performed, ideally, in a preceding element on the pipeline, leaving the queries sent to the DBMS as pretty much final and already secure.

The biggest difference in this section came from the Data Loss subtype. Defining proper data backup plans - with periodicity aligned with the Machine Learning training schedule - and an appropriate secure structure to hold the already backed-up content can prevent a malicious user from compromising the whole information repository in the case

of a security breach. As this kind of AI software is highly based on data, keeping this asset secure is a majorly important mitigation measure.

4.2.3.4 Machine Learning Model

For this category, all 15 available solutions were selected from sAIfe's repository for ConsistencIA. It is fair to point out that 7 of these solutions repeat themselves across the list, under different vulnerability categories, therefore, there are only 8 different kinds of mitigation.

Mitigation measures for this Machine Learning Model Critical Area are somewhat different than the trend that had been seen in earlier sections. There are still mentions to the so-called passive solutions, such as access control and rate limiting, but they are far from being the majority now.

The biggest difference in this section comes from the ML-specific aspect. The suggestion to perform input filtering, now using the ML Model itself, along with developing manipulation detection techniques and performing adversarial training are all only possible in the ML scenario. This very point is the farthest it gets from traditional cybersecurity, as no other application part can take on this task.

Even with proper filtering on all the previous software parts, it is possible that a malicious input is well formatted and, therefore, still bypasses all the verification levels, getting to the inference step itself. Having a robust ML model is, consequently, indispensable to creating a solid system, along with all the previous recommendations, in order to strengthen every aspect of the application.

4.2.4 Results Analysis and Discussions

The first thing that can be noticed while following sAIfe steps for ConsistencIA is the capacity a relatively simple ML system has of presenting feasible security flaws and fitting into multiple error conditions. This fact could be perceived by the number of possible risk categories – in this case, Critical Areas – and sub-categories – vulnerabilities contained within them – the project under analysis encompassed and was actively related to.

Another relevant fact that came up during the method application – especially through the last couple of steps – is the relation different aspects and/or elements of an ML system can have with one another. In this scenario, the strong data-driven trait of Machine Learning applications causes data to be at the center of all components, following a well-defined path from the source to the destination. This, consequently, generates a sequential chaining characteristic in the pipeline, which, in every step, receives, transforms and dispatches information to the next phase. In the end, such tight connection ends up

causing adjacent Application Parts to share related vulnerabilities.

This is also what causes connected elements to – apparently – fit into very similar (if not identical) flaw categories, such as the User Interface-API/Server combination case, when it comes to authentication issues. In this setting, for instance, either of them could compensate for the technical debts left by the other and perform the necessary actions to achieve optimal security. It is, however, wise to note that the sooner – in the pipeline flow – the corrections are made, the less room for errors there will be.

Additionally, the interdependence that arises from these interactions only strengthens the theory that ML security should not be siloed from traditional cybersecurity; instead, it ought to be considered just another subdivision of it, encompassed by the broad umbrella of software security.

It is possible to notice that, as threats are treated early in the pipeline, the burden left for elements further ahead gets drastically reduced – if not extinguished. In other words, addressing possible issues on traditional cybersecurity-related Application Parts, such as the ones closer to data entry components, has proved to be an efficient strategy to reduce risks arriving at more ML-related parts. This is exactly what happens with the recurrent "Input Filtering" solution sections both in the "User Interface" and in the "API/Server" Critical Areas: the more effective they are, the less complicated hardening measures should ML Models further ahead apply to defend, for example, against poisoning. This is, therefore, a major advantage these kinds of systems can leverage to make the security process easier: relying on traditional cybersecurity techniques – which are, generally, more well-known and less effortful to apply.

Finally, when it comes to the effective value of the sAIfe method application on this system, it is clear that the straightforward technical recommendations may provide a major gain in time and productivity, as opposed to lengthy and non-unified security guides. Moreover, the element-focused approach allowed the method user to more easily situate and understand their needs: a process that is made even simpler because of the documentations' focus on real-world ML applications, wisely generalized to cover a wider and more universal area.

4.3 Case Study: ConsistencIA Analysis Using STRIDE-AI

This section shows an example of the application of another published framework and its results, to serve as a form of comparison with sAIfe. The method chosen was STRIDE-AI - engineered by Mauri and Damiani [23] - as it employs a well-known security assessment technique tailored to the AI/ML scenario. The system analyzed was the same as the one used in the previous section - ConsistencIA - to allow for easier correlation.

4.3.1 STRIDE-AI Contextualization

STRIDE-AI's authors claim their "extension to the original STRIDE provides an ML-specific, security property-driven approach to threat detection which can also provide guidance in selecting the security controls needed to alleviate the identified threats". This characteristic, theoretically, allows it to be compared with sAIfe, as both methods' objectives are aligned in detecting vulnerabilities and providing the applicant with risk mitigation suggestions.

The framework is based on the possible assets an ML application can feature, which are grouped into six different macro-categories: "Data", "Models", "Actors", "Processes", "Tools", and "Artefacts". Each asset of the analyzed system is, then, identified, labeled and goes through an extensive threat modeling process, aiming at extracting its weak points.

4.3.2 STRIDE-AI Application

To reduce the extension of this study, the scope of the analysis was limited to a single ConsistencIA component: the Machine Learning Model. This restriction, nevertheless, does not impact the outcome, as the method itself is meant to be executed equally over each system component and does not depend on other results.

Starting the process, the first step was identifying the elements and labeling them according to the six macro-categories mentioned above. Here, ideally, a diagram of the entire application would be generated and every single element tagged. In this scenario, the ML Model was fitted, intuitively, into the category "Models".

Moving on, the second step consisted of choosing a set of properties of interest within the CIA^3-R hexagon (defined by the authors as the union of the concepts "Authenticity", "Integrity", "Non-repudiation", "Confidentiality", "Availability" and "Authorization") for the asset. With the help of a correlation table provided by the authors, it was possible to classify the ML Model in ConsistencIA as belonging to the "Deployed models" asset category. For this reason - and according to another provided correlation table - the chosen properties of interest here should be all the six available ones: "Authenticity", "Integrity", "Non-repudiation", "Confidentiality", "Availability" and "Authorization".

The third step was threat identification. According to the same table used last, assets in this category should have, as threats, all the following: "Spoofing", "Tampering", "Repudiation", "Information Disclosure", "Denial of Service" and "Elevation of Privilege".

The next step of the threat modeling process was vulnerability identification, to try and describe under which conditions the threats associated to the assets could materialize. This was the hardest step of the process so far, as it varied from system to system and the provided explanation table contained a generic single example per threat, requiring

actual thinking.

For this case study, therefore, when it comes to the ML Model:

- the “Spoofing” threat would likely happen when a user takes on the identity of another, posing to the ML Model as an authorized user and being able to access predictions not available to them in a normal scenario;
- the “Tampering” threat would happen when an attacker maliciously modifies data contained in the application through an ML Model inference request, such as what happens in a poisoning attack;
- the “Repudiation” threat would likely happen if an attacker managed to delete foundational information in database about a model parameter or training data, preventing the system from justifying and explaining the result of a prediction;
- the “Information Disclosure” threat would happen if sensitive information was somehow returned in a model prediction, such as what happens in an oracle attack;
- the “Denial of Service” threat would likely happen if an attacker managed to exhaust computational resources required for the ML Model to work, e.g. by requesting too many predictions at a time;
- the “Elevation of Privilege” threat would possibly happen in a situation similar to the one in “Spoofing”, when an attacker poses to the ML Model as an authorized user and is able to access predictions not available to them in a normal scenario;

As the last part of the process, these six items are the final result of the STRIDE-AI framework application to the concerned system.

4.4 Comparison Between sAIfe and STRIDE-AI

This section highlights the most important points in both methods used in the case studies above - sAIfe and STRIDE-AI - and compares their results from an efficiency point of view.

Initially, it is fair to point out sAIfe’s advantage when it comes to user-friendliness and instruction-clearness. STRIDE-AI’s documentation can get quite complex due to the increased number of sections and concepts, even though things get a bit clearer in the end, with the use case example.

Roughly speaking, at the beginning, both methods have a relatively similar way of functioning, with the first step requiring the identification of the elements involved in the architecture and further actions demanding analyses of these components. Moving

on, however, sAIfe differs from the compared method, as the former already provides the user with a list of concrete possible vulnerabilities, instead of just plausible at-risk security concepts. This can be seen, for instance, when STRIDE-AI mentions the "Tampering" concept and requires the user themselves to develop possible failure scenarios for it. sAIfe, instead, already provides these scenarios, under the section "Training Data Integrity Flaws", requiring fewer steps and, thus, making the process quicker.

What really sparks attention, however, is the fact that STRIDE-AI does not actually provide its user with suggested mitigation forms and/or solutions for the threats discovered. For this analysis, for example, sAIfe provided 8 different solutions feasible to be applied to the system at issue, only for the ML Model section. This can be a great asset while helping the development team save time in a longer threat modeling process.

Table 3 shows a comparison between the aforementioned method and sAIfe, when it comes to three main aspects: involving subjective questions; providing concrete threats; being restricted to a few concepts; and providing threat mitigation forms.

	sAIfe	STRIDE-AI
Involves subjective questions	No	Yes
Provides concrete threats	Yes	No
Security concepts addressed	Unrestricted, broader	Restricted to STRIDE
Provides threat mitigation forms	Yes	No

Table 3 – STRIDE-AI case study comparison

4.5 Evaluation with Researchers

sAIfe is the result of a series of self-analysis and refactorings. One of the most important sources of data for these changes was the validation carried out within the academic environment. It was a study conducted in such a way that participants had to use the proposed method introduced in Section 3 to assess the security of an ML system of their choice. This led to valuable lessons and discoveries that were applied to sAIfe's protocol.

This validation happened with the help of two Computer Science researchers who also have experience in software development. The experiment aimed at collecting feedback from this team of academic developers on the implementation of a previous version of sAIfe's threat modeling method (with one more step, slightly more complex), with the specific goal of analyzing their perception regarding the method's efficiency, speed and ease of application. This feedback was, then, used to improve the method to the current version - a process described in this section.

To achieve this, the participants were taught how to use sAIfe's method and instructed to apply it to an application of their choice. Since the focus of this experiment was not exactly the application analyzed and the technical results found, the chosen application is not going to be explained in detail, although it is important to mention that, just like ConsistencIA, it includes, at least, all four Critical Areas covered by sAIfe.

Upon performing the method application, the researchers were requested to fill out a questionnaire, whose items were specially crafted to obtain strategic information on sAIfe's characteristics. Overall, it included topics with predetermined selectable answers, in the form of:

- Level of difficulty (Very hard, Hard, Neutral, Easy, Very easy)
- Amount of time spent per step (5 minutes or less, 15 minutes, 30 minutes, 1 hour, 2 hours, 3 hours or more)
- Amount of time spent in total[1]
- Quality of contributions generated to the software analyzed (Poor, Fair, Satisfactory, Very good, Excellent)

Moreover, it also contained open written questions, such as:

- What aspects of this method were most useful or valuable?
- Do you think this method could be inserted into a modern software development methodology's product pipeline?
- How would you improve this method?

Under these circumstances, the results came back quite positive, with answers declaring, on average: sAIfe is an Easy/Neutral method to be used; does not have any step that takes longer than 30 minutes to be executed, with an average method execution time of 18.3 minutes; takes around one hour to be fully applied; and provides Very good/Excellent contributions to the software it applied to.

Moving on to the written questions, according to the developers: the highlighted aspects of sAIfe were the facilitation of understanding what threats could be present in the application and the promotion of an easier search for mitigation implementations; and sAIfe was capable of being inserted into a modern development pipeline, as its application from the beginning of the Software Development Life Cycle (SDLC) helps planning ahead tasks and demonstrating value to stakeholders. As for the last question, one of the

[1] This item did not have predetermined answers and could receive any numeric value

statements the team made was that the method could be improved by becoming even quicker to apply.

At this point in time, sAIfe was still a four-step method, requiring - as step number two - a comparison between the created Architecture Diagram and the Reference one, with the goal of spotting similarities and identifying Critical Areas. When analyzing this step again, however, it seemed to be inefficient, as Critical Areas were already easy to find in the diagram and as the Reference Architecture Diagram was not always a suitable comparison artifact for every ML system being depicted. Therefore, step two was completely removed and the surrounding ones were slightly altered to adapt to this change. In this scenario, sAIfe has, then, turned into a three-step method, exactly the way it is at the present moment, described in Section 3.

Furthermore, the Reference Architecture Diagram has also experienced changes: at first, in this previous version of the method, it was a mandatory comparison artifact, to be used in - the outdated - step two. Consequently, once step two was removed - exactly as described immediately above - the Reference Architecture Diagram had to be relocated. It, then, moved from being a required artifact in the extinguished step to a recommended supporting example for the diagram creation activity in step one. With these alterations, sAIfe has effectively reached its current, up-to-date version, explained in Section 3. This set of changes, then, triggered by the received feedback, turned the method application into a much lighter process, with significant speed improvement (up to 33% faster, according to the questionnaire statistics).

In summary, it is possible to conclude that this validation in academia proved to be a very helpful tool, both for assessing sAIfe's current state and receptivity from the developer's perspective and for suggesting ways to improve it and make it even more valuable for its target audience.

4.6 Evaluation with Developers

sAIfe was also evaluated with the help of software developers working on the industry, to collect the opinions of users who match very closely the method's intended target audience. This section describes the use of an adapted version of the Technology Acceptance Model (TAM) and the Goal Question Metric (GQM) with 12 participants, to judge the usability-related characteristics of sAIfe. These characteristics must be considered since the complexity of a threat modeling tool has been shown to be a prevalent and significant hindrance in previous approaches.

Amongst the 12 respondents, the majority were Software Developers working in a large multinational private company (with more than 100,000 employees worldwide), with a Junior or Mid-Senior classification.

TAM is a widely used model for evaluating user acceptance of new technologies. It was first introduced in the 1980s by Davis [64], and consists of two main factors: perceived usefulness and perceived ease of use. These two factors have a significant impact on the user's intention to use new technology. The evaluation was planned using the GQM system [65], which consists of four steps: Planning, Definition, Data Collection, and Interpretation.

4.6.1 Context

This evaluation was planned and executed in a similar way to how the previous one was (Section 4.5), via an online questionnaire, individually answered. This time, however, the experiment had two distinct moments: one before the actual method application by the user and another one after it.

Initially, the participants were taught about sAIfe and how to use it through textual content and a video explanation (10 minutes long). After this initial step, a first round of questions was presented to the questionnaire respondent. Moving on, the participants were also briefed on the application to be analyzed - ConsitencIA - through textual content and a video explanation (5 minutes long). In possession of all this data, they were asked to apply sAIfe to this application (with Steps 2 and 3 limited to one single Critical Area, in order to keep the experiment shorter) and submit the generated artifacts. Finally, a second round of questions was presented to the questionnaire respondent. All the information was, then, collected and analyzed.

4.6.2 Goal and Method

The goal of this validation is detailed in Table 2. It followed the Goal Question Metric (GQM) goal definition template [66], which is a structured approach commonly used in Software Engineering and other disciplines, to help establish a clear connection between the overall goal, the specific questions that need to be answered and the metrics used to measure progress.

Goal Step	Specific Complement
Analyze	the proposed solution's efficiency and results
for the purpose of	justifying the statements made in this work and possibly improving the method's protocol
with respect to	ease of use, method application speed and quality of results generated
from the viewpoint of	software developers
in the context of	the industry environment

Table 4 – Study goal definition for the validation with developers

4.6.3 Planning and Definition

The experiment questions are presented in Table 3, along with the indication of whether they are metric-based (with predefined possible answers) or open-ended, whereas the 5 used metrics are shown in Table 4.

Question	Content	Metric-based	Yes/No
Q1	How difficult does applying the method seem to you?	M1	
Q2	How efficient does the method seem to you?	M2	
Q3	In case you needed a solution of this kind, would you be inclined to try sAIfe?		X
Q4	How difficult did applying the method actually feel to you? (Whole method)	M1	
Q5	How difficult did applying the method actually feel to you? (Step 1)	M1	
Q6	How difficult did applying the method actually feel to you? (Step 2)	M1	
Q7	How difficult did applying the method actually feel to you? (Step 3)	M1	
Q8	How much time did it take you to actually apply the method? (Whole method)	M3	
Q9	How much time did it take you to actually apply the method? (Step 1)	M3	
Q10	How much time did it take you to actually apply the method? (Step 2)	M3	
Q11	How much time did it take you to actually apply the method? (Step 3)	M3	
Q12	How were the contributions to the system analyzed? (Suggested threats)	M4	
Q13	How were the contributions to the system analyzed? (Suggested solutions)	M4	

Table 5 – Questions for the validation with developers

Metric	Definition
M1	Level of difficulty
M2	Effectiveness
M3	Time taken
M4	Quality

Table 6 – Metric definition for the validation with developers

These questions were applied at two different moments in the questionnaire. Questions 1-3 were applied first, after the method explanation and before any experiment information, in order to acquire the participant's perception on the method even before a possible use. On the other hand, the rest of the questions (4-15) were applied only

after this step, following the experiment instruction and its actual realization by the respondents, to collect the participant's true perception after having effectively applied the method.

4.6.4 Results and Discussion

The following contents present the results and their distributions for each of the 13 questionnaire questions described in the previous section.

Starting with the 3 pre-experiment questions, Question 1 ("How difficult does applying the method seem to you?") has its results described by Figure 9. They show participants had a relatively good first impression on the method's level of difficulty after being introduced to it. According to the principles of TAM, it is possible to conclude that, by not perceiving sAIfe as a possibly hard-to-use method, a developer would be more willing to try it: a positive feedback.

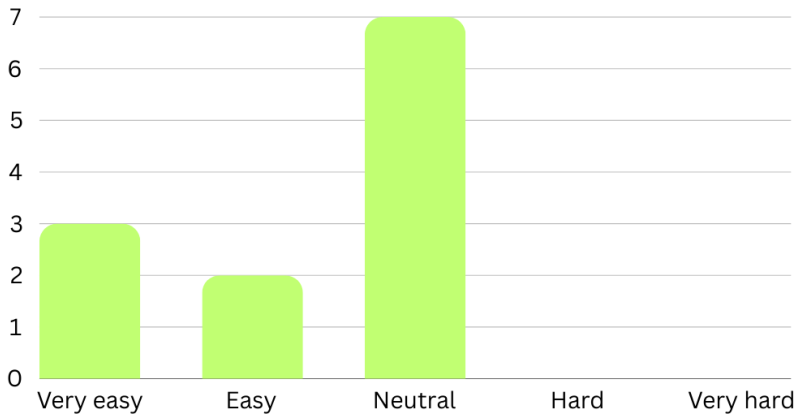

Figure 9 – Q1 Results

Moving on, Figure 10 represents the results for Question 2 ("How efficient does the method seem to you?"). The data shows an outcome in line with the one from the previous question, suggesting participants saw sAIfe as a promising tool even before using it.

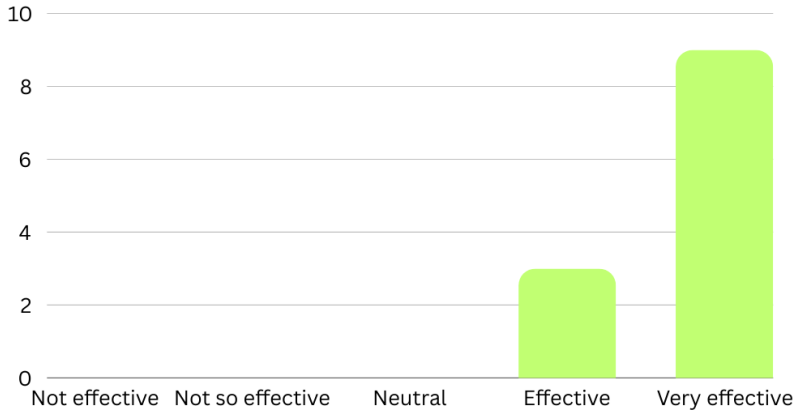

Figure 10 – Q2 Results

Finally, Question 3 (“In case you needed a solution of this kind, would you be inclined to try sAIfe?”) extends the perception analyses made by the first two questions and concludes the topic. As a Yes or No question, it collected 100% positive answers, concluding, once and for all, the main point of the TAM goal: developers hearing of sAIfe for the first time have a first perception good enough to go ahead and actually try the method.

Shifting to the second part of the questionnaire - after the actual implementation of sAIfe by the participant -, Question 4 (“How difficult did applying the method actually feel to you? (Whole method)”) starts to collect the respondent’s real perceptions, represented by Figure 11. The data shows the majority of participants considered sAIfe to be an overall easy method to apply, with little discrepancy.

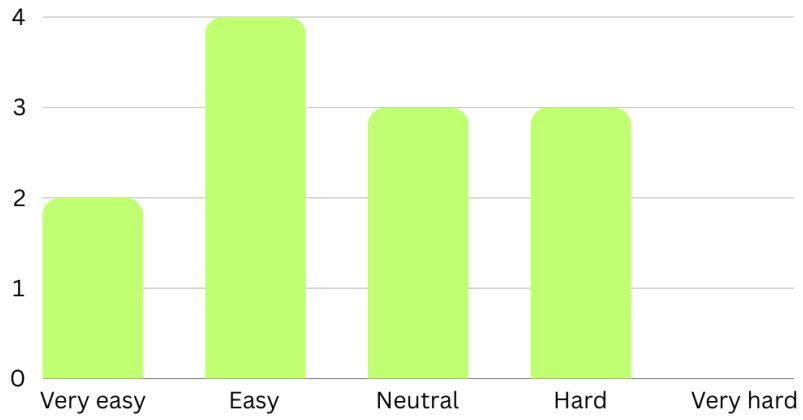

Figure 11 – Q4 Results

Analogously, Questions 5, 6 and 7 also address the difficulty of using sAIfe, but now in a specific way for each of the three steps of the method. Figures 12, 13 and 14 show this information. The data for these three questions came out in a very similar pattern, with a significant spike on the “Easy” alternative: a good indicative for sAIfe.

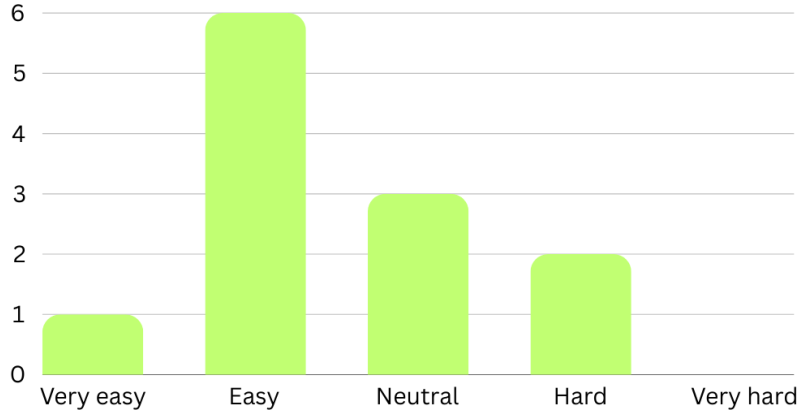

Figure 12 – Q5 Results

Continuing, Question 8 (“How much time did it take you to actually apply the method? (Whole method)”) addresses the amount of time taken by the user to implement

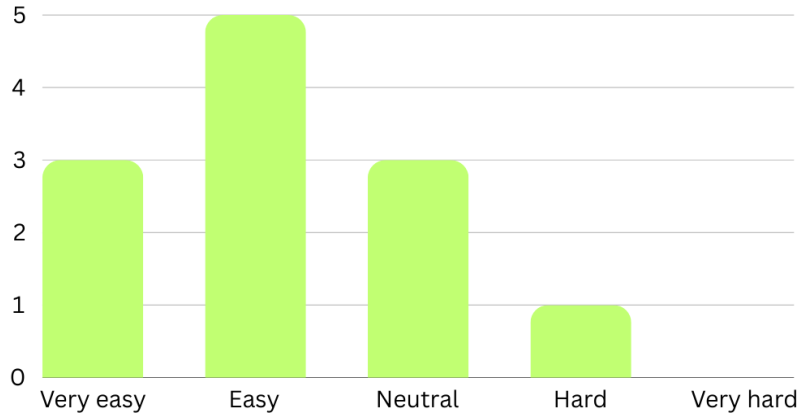

Figure 13 – Q6 Results

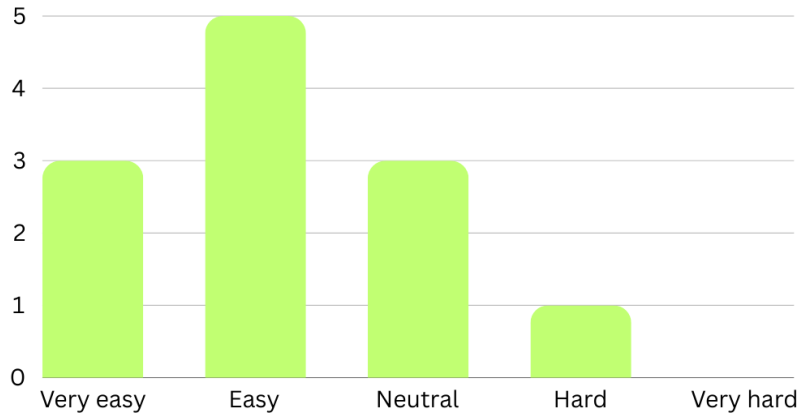

Figure 14 – Q7 Results

the method, with results expressed by Figure 15. The collected data shows the majority of participants took around 15 minutes to finish applying the whole protocol: a great mark, that supports the affirmation that sAIfe is a quick method.

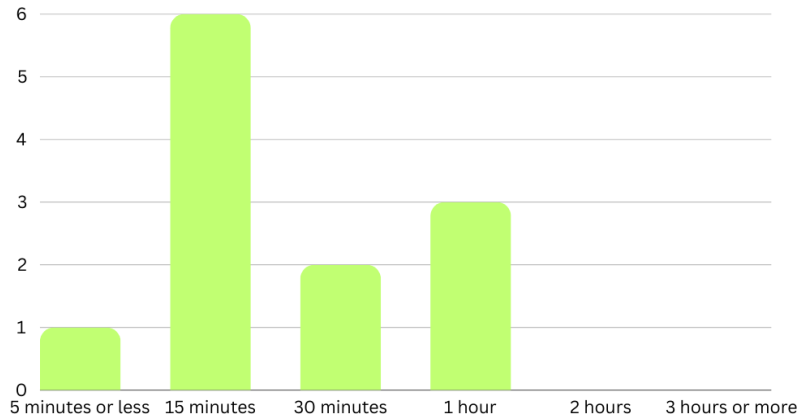

Figure 15 – Q8 Results

In a similar way, Questions 9, 10 and 11 also analyze application time, but now in a specific way for each of the three steps of sAIfe. Figures 16, 17 and 18 show these data respectively. The information for these three questions also came out in a very similar pattern, with the most common option being the “5 minutes or less” one, the quickest

alternative available.

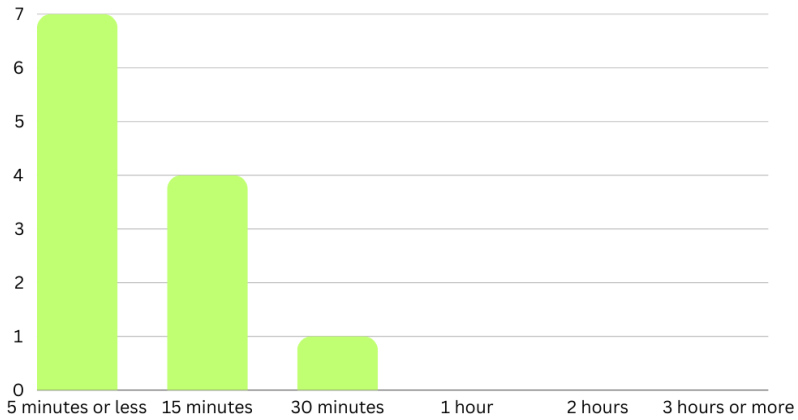

Figure 16 – Q9 Results

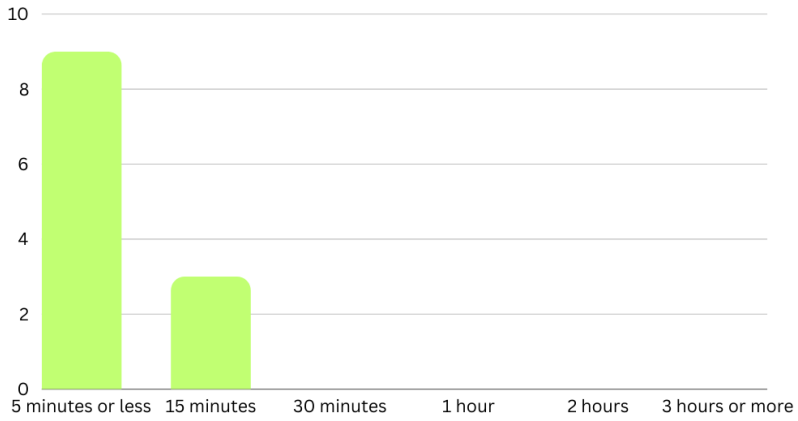

Figure 17 – Q10 Results

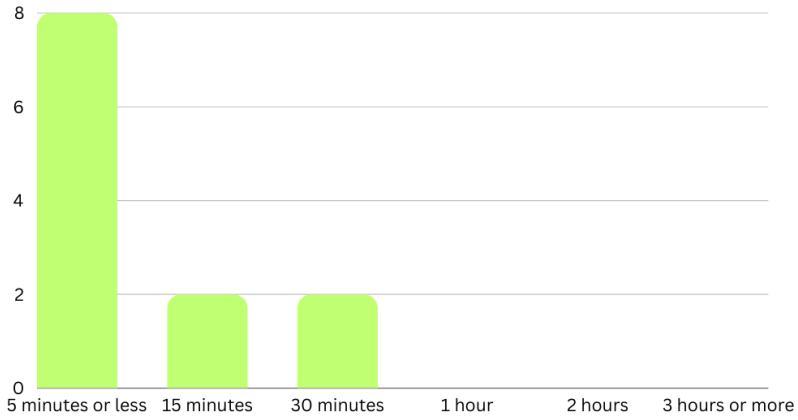

Figure 18 – Q11 Results

In the end, Question 12 (“How were the contributions to the system analyzed? (Suggested threats)”) and 13 (“How were the contributions to the system analyzed? (Suggested solutions)”) analyze the quality of sAIfe’s outputs from the perspective of the participants. Figures 19 and 20 represent the distribution of results, which reveals a significant predominance of the “Very good” and “Good” categories: a great feedback for the ultimate phase of sAIfe.

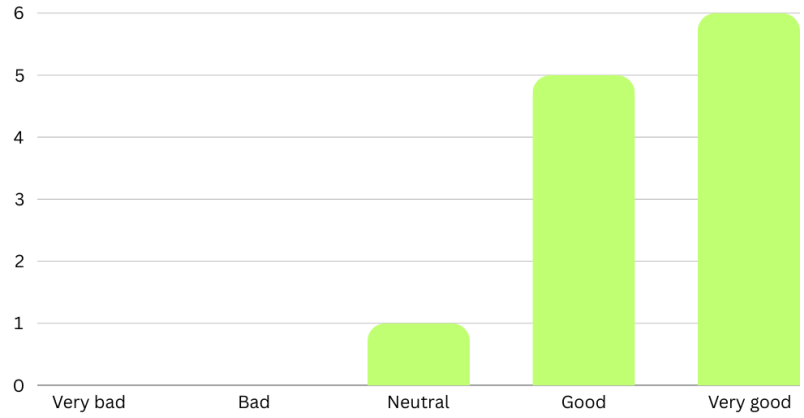

Figure 19 – Q12 Results

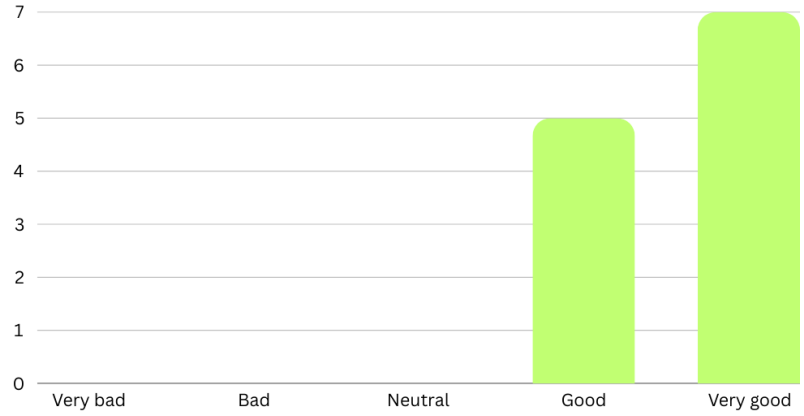

Figure 20 – Q13 Results

Summing up, it is possible to say that the results obtained in this experiment were, in their majority, positive for sAIfe. The method received an overall positive feedback, with a significant amount of the participants agreeing that sAIfe can be, indeed, fast to apply and easy to implement. This conclusion, in turn, supports the respective affirmations made throughout this work and highlights the method's affinity with the intended target audience.

5 CONCLUSION

sAIfe aims at filling the currently existing gap between threat identification processes and remediation techniques by providing a method that performs and links these two procedures in a seamless and effective way. With all that has been exposed throughout this document, it is fair to say that the proposed method is capable of acting as a solid security assurance process for ML applications.

Furthermore, the experiments shown in Section 4 support the claims that sAIfe is an easy-to-use method and has a fast application time. In the case study with ConsistencIA (Section 4.2), for instance, applying sAIfe to this ML system revealed a great number of possible security flaws and an even greater number of ready mitigation forms. Even though the analyzed system seemed to follow a traditional architecture, with well established patterns, there still were many possible threats revealed by sAIfe that did not seem explicit at first glance and were shown in a seamless way to the developer. On the other hand, the test with the alternative framework from the literature (Section 4.3) seemed more complex and did not achieve the same practical results.

Moreover, the validations carried out both with researchers in academia (Section 4.5) and with developers in industry (Section 4.6) brought concrete confirmation to these affirmations. These experiments reflected a very positive opinion coming from the respondents, with the majority of them stating that sAIfe seemed easy to use (both before and after actually applying it), effective in the results and quick to apply.

Finally, due to the way Critical Area Detailed Documents were planned and structured, sAIfe allows for easy update and alteration of the weaknesses and solutions lists. This activity is exactly what is intended to be done as a main future effort on the method. As computation is an ever-evolving area of technology, new threats might be discovered as time passes and, thus, sAIfe is expected to keep up with them. Having future work to take on this task regularly is a great opportunity to maintain the method viable and accurate for years to come, an efficient way to increase the level of security of Artificial Intelligence systems launched, the utmost goal sought by sAIfe.

BIBLIOGRAPHY

[1] ORMENISAN, A. A. et al. *Horizontally scalable ML pipelines with a feature store*. 2019. <https://mlsys.org/Conferences/2019/doc/2019/demo_7.pdf>. [Accessed 28-11-2024].

[2] ALPAYDIN, E. *Machine Learning*. [S.l.]: MIT Press, 2021.

[3] HINTON, G. E.; OSINDERO, S.; TEH, Y. W. A fast learning algorithm for deep belief nets. *Neural Computation*, v. 18, p. 1527–1554, 2006.

[4] PANDYA, Y. *Why have AI suddenly become a big thing?* 2023. Disponível em: <https://www.linkedin.com/pulse/why-have-ai-suddenly-become-big-thing-yagnesh-pandya/>.

[5] HU, Y. et al. Artificial intelligence security: Threats and countermeasures. *ACM Computing Surveys (CSUR)*, v. 55, p. 1 – 36, 2021.

[6] COMITER, M. Z. Attacking artificial intelligence: Ai's security vulnerability and what policymakers can do about it. 2019.

[7] DEMPSEY, J. *Managing the Cybersecurity vulnerabilities of Artificial Intelligence*. 2021. Disponível em: <https://www.lawfareblog.com/managing-cybersecurity-vulnerabilities-artificial-intelligence>.

[8] GROTTO, A.; FALCO, G.; MAIFELD-CARUCCI, I. Response to 'request for information: Artificial intelligence risk management framework'. Sep 2021. Disponível em: <https://fsi-live.s3.us-west-1.amazonaws.com/s3fs-public/2021-09-15_-_nist_ai_risk_rfi_grotto_falco_maifeld-carucci_stanford_hopkins.pdf>.

[9] KIM, B. et al. Decamouflage: A framework to detect image-scaling attacks on convolutional neural networks. *ArXiv*, abs/2010.03735, 2020.

[10] XIAO, H. et al. Is feature selection secure against training data poisoning? In: *International Conference on Machine Learning*. [S.l.: s.n.], 2015.

[11] BIGGIO, B.; NELSON, B.; LASKOV, P. Poisoning attacks against support vector machines. In: *International Conference on Machine Learning*. [S.l.: s.n.], 2012.

[12] YANG, C. et al. Generative poisoning attack method against neural networks. *ArXiv*, abs/1703.01340, 2017.

[13] BIGGIO, B. et al. Bagging classifiers for fighting poisoning attacks in adversarial classification tasks. In: *International Workshop on Multiple Classifier Systems*. [S.l.: s.n.], 2011.

[14] FOUNDATION, O. *OWASP Top 10*. OWASP Foundation, 2021. Disponível em: <https://owasp.org/Top10/>.

[15] FOUNDATION, O. *OWASP Project API Security*. OWASP Foundation, 2023. Disponível em: <https://owasp.org/www-project-api-security/>.

[16] ARTIFICIAL Intelligence Risk Management Framework (AI RMF 1.0). NIST, 2023. Disponível em: <https://nvlpubs.nist.gov/nistpubs/ai/NIST.AI.100-1.pdf>.

[17] MARSHALL, A. et al. *Threat modeling AI/ML Systems and Dependencies*. 2019. Disponível em: <https://learn.microsoft.com/en-us/security/engineering/threat-modeling-aiml>.

[18] NGUYEN, T. Integrating security into agile software development methods. 2015. Disponível em: <https://www.umsl.edu/~sauterv/analysis/F2015/Integrating%20Security%20into%20Agile%20methodologies.html.htm>.

[19] THEURICH, P.; WITT, J.; RICHTER, S. Practices and challenges of threat modelling in agile environments. *Informatik Spektrum*, v. 46, 09 2023.

[20] RENATUS, S.; TEICHMANN, C.; EICHLER, J. Method selection and tailoring for agile threat assessment and mitigation. In: . [S.l.: s.n.], 2015.

[21] SHOSTACK, A. Nothing is good enough: Fast and cheap are undervalued as influencers of security tool adoption. *IEEE Security & Privacy*, v. 21, p. 78–83, 2023.

[22] CONKLIN VICTORIA DRAKE, S. S. L.; BRAITERMAN, Z. *Threat Modeling Process*. 2022. Disponível em: <https://owasp.org/www-community/Threat_Modeling_Process>.

[23] MAURI, L.; DAMIANI, E. Modeling threats to ai-ml systems using stride. *Sensors (Basel, Switzerland)*, v. 22, 2022.

[24] XIONG, W.; LAGERSTRöM, R. Threat modeling – a systematic literature review. *Computers & Security*, v. 84, p. 53–69, 2019. ISSN 0167-4048. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0167404818307478>.

[25] BRAITERMAN ADAM SHOSTACK, J. M. Z. *Threat Modeling Manifesto — threatmodelingmanifesto.org*. 2020. <https://www.threatmodelingmanifesto.org/>. [Accessed 11-03-2024].

[26] KOHNFELDER, L.; GARG, P. *The threats to our products*. 1999. <https://shostack.org/files/microsoft/The-Threats-To-Our-Products.docx>. [Accessed 12-03-2024].

[27] LEBLANC, D.; HOWARD, M. *Writing Secure Code*. 2. ed. Redmond, WA: Microsoft Press, 2002. ISBN 9780735617223.

[28] DOUGLEN, A.; WIERCKX, S. *OWASP Threat Modeling Project | OWASP Foundation — owasp.org*. 2021. <https://owasp.org/www-project-threat-model>. [Accessed 13-03-2024].

[29] AFANEH, S. et al. Security challenges review in agile and devops practices. In: . [S.l.: s.n.], 2024.

[30] DAVE, S. et al. Special session: Towards an agile design methodology for efficient, reliable, and secure ml systems. In: *2022 IEEE 40th VLSI Test Symposium (VTS)*. IEEE, 2022. Disponível em: <http://dx.doi.org/10.1109/VTS52500.2021.9794253>.

[31] BASU, K. et al. Artificial intelligence: How is it changing medical sciences and its future? *Indian Journal of Dermatology*, v. 65, p. 365, 05 2020.

[32] WHAT is Artificial Intelligence (AI)? | IBM — ibm.com. 2024. <https://www.ibm.com/topics/artificial-intelligence>. [Accessed 25-04-2024].

[33] SCHROER, A. *Artificial Intelligence.* 2023. Disponível em: <https://builtin.com/artificial-intelligence>.

[34] GOOGLE. *What is Artificial Intelligence?* Google, 2023. Disponível em: <https://cloud.google.com/learn/what-is-artificial-intelligence#section-3>.

[35] GOODRICH, J. *How IBM's Deep Blue Beat World Champion chess player Garry Kasparov.* IEEE Spectrum, 2022. Disponível em: <https://spectrum.ieee.org/how-ibms-deep-blue-beat-world-champion-chess-player-garry-kasparov>.

[36] WHAT is Machine Learning? Types & Uses. Google, 2023. Disponível em: <https://cloud.google.com/learn/what-is-machine-learning>.

[37] SARKER, I. Machine learning: Algorithms, real-world applications and research directions. *SN Computer Science*, v. 2, 03 2021.

[38] NICK, E. *Top 7 artificial intelligence data security threats to AI and ML.* 2024. Disponível em: <https://www.datasciencecentral.com/top-7-data-security-threats-to-ai-and-ml/>.

[39] ILMOI. *Poisoning attacks on machine learning.* Towards Data Science, 2019. Disponível em: <https://towardsdatascience.com/poisoning-attacks-on-machine-learning-1ff247c254db>.

[40] ILMOI. *Evasion attacks on machine learning (or "adversarial examples").* Towards Data Science, 2019. Disponível em: <https://towardsdatascience.com/evasion-attacks-on-machine-learning-or-adversarial-examples-12f2283e06a1>.

[41] GU, T. et al. Badnets: Evaluating backdooring attacks on deep neural networks. *IEEE Access*, v. 7, p. 47230–47244, 01 2019.

[42] KREUZBERGER, D.; KüHL, N.; HIRSCHL, S. *Machine Learning Operations (MLOps): Overview, Definition, and Architecture.* 2022.

[43] SALVUCCI, E. Mlops-standardizing the machine learning workflow. In: . [s.n.], 2021. Disponível em: <https://api.semanticscholar.org/CorpusID:237462653>.

[44] WöSTMANN, R. et al. Conception of a reference architecture for machine learning in the process industry. In: *2020 IEEE International Conference on Big Data (Big Data).* [S.l.: s.n.], 2020. p. 1726–1735.

[45] LEMOS, R. *Expect an increase in attacks on AI systems.* Dark Reading, 2021. Disponível em: <https://www.darkreading.com/vulnerabilities---threats/advanced-threats/expect-an-increase-in-attacks-on-ai-systems/d/d-id/1340833>.

[46] ADVERSA. *Adversa releases Secure and Trusted AI report with exclusive retrospective, trends, predictions.* Adversa AI, 2021. Disponível em: <https://adversa.ai/blog/adversa-releases-secure-and-trusted-ai-report-with-exclusive-retrospective-trends-predictions/>.

[47] BIGGIO, B. et al. Bagging classifiers for fighting poisoning attacks in adversarial classification tasks. In: *International Workshop on Multiple Classifier Systems*. [S.l.: s.n.], 2011.

[48] GU, T.; DOLAN-GAVITT, B.; GARG, S. Badnets: Identifying vulnerabilities in the machine learning model supply chain. *CoRR*, abs/1708.06733, 2017.

[49] XIAO, H. et al. Is feature selection secure against training data poisoning? In: *International Conference on Machine Learning*. [S.l.: s.n.], 2015.

[50] KIM, B. et al. Decamouflage: A framework to detect image-scaling attacks on convolutional neural networks. *ArXiv*, abs/2010.03735, 2020.

[51] AYUB, M. A. et al. Model evasion attack on intrusion detection systems using adversarial machine learning. *2020 54th Annual Conference on Information Sciences and Systems (CISS)*, p. 1–6, 2020.

[52] HERNANDEZ-CASTRO, C. J. et al. Adversarial machine learning. In: *Security and Artificial Intelligence*. [S.l.: s.n.], 2022.

[53] FAZELNIA, M.; OKUTAN, A. E.; MIRAKHORLI, M. Supporting artificial intelligence/machine learning security workers through an adversarial techniques, tools, and common knowledge framework. *IEEE Security & Privacy*, v. 21, p. 37–48, 2023.

[54] GROTTO, A.; DEMPSEY, J. J. Vulnerability disclosure and management for ai/ml systems: A working paper with policy recommendations. *SSRN Electronic Journal*, 2021.

[55] KUMAR, V.; MAYO, J.; BAHISS, K. *ADMIn: Attacks on Dataset, Model and Input. A Threat Model for AI Based Software*. 2024.

[56] HERNAN, S. et al. *Uncover Security Design Flaws Using The STRIDE Approach*. 2006. Disponível em: <https://learn.microsoft.com/en-us/archive/msdn-magazine/2006/november/uncover-security-design-flaws-using-the-stride-approach>.

[57] HASAN, R. et al. Toward a threat model for storage systems. In: *ACM International Workshop on Storage Security And Survivability*. [S.l.: s.n.], 2005.

[58] ENGSTRöM, E. et al. How software engineering research aligns with design science: A review. *Empirical Software Engineering*, v. 25, n. 4, p. 2630–2660, 2020.

[59] BASKERVILLE, R. et al. Design science research contributions: Finding a balance between artifact and theory. *Journal of the Association for Information Systems*, v. 19, p. 358–376, 05 2018.

[60] PEFFERS TUURE TUUNANEN, M. A. R. K.; CHATTERJEE, S. A design science research methodology for information systems research. *Journal of Management Information Systems*, Routledge, v. 24, n. 3, p. 45–77, 2007. Disponível em: <https://doi.org/10.2753/MIS0742-1222240302>.

[61] MCGRAW, G. Berryville Institute of Machine Learning (BIML), 2020. Disponível em: <https://www.garymcgraw.com/wp-content/uploads/2020/02/BIML-ARA.pdf>.

[62] BEYNON-DAVIES, P. Data flow diagramming. In: . [s.n.], 1998. Disponível em: <https://api.semanticscholar.org/CorpusID:64169441>.

[63] RAFFIN, T. et al. A reference architecture for the operationalization of machine learning models in manufacturing. *Procedia CIRP*, v. 115, p. 130–135, 11 2022.

[64] DAVIS, F. D. Perceived usefulness, perceived ease of use, and user acceptance of information technology. *MIS Quarterly*, Management Information Systems Research Center, University of Minnesota, v. 13, n. 3, p. 319–340, 1989. ISSN 02767783, 21629730. Disponível em: <http://www.jstor.org/stable/249008>.

[65] SOLINGEN, R.; BERGHOUT, E. The goal/question/metric method: A practical guide for quality improvement of software development. 01 1999.

[66] BASILI, V. R.; CALDIERA, G.; ROMBACH, H. D. The goal question metric approach. In: . [s.n.], 1994. Disponível em: <https://api.semanticscholar.org/CorpusID:13884048>.

PAPERS PUBLISHED BY THE AUTHOR

1. Gabriel Esteves Messas, Bruno Bogaz Zarpelão, Rodrigo Sanches Miani, **sAIfe: Towards a Lightweight Threat Modeling Approach to Support Machine Learning Application Development**, SBQS 2024: XXIII Brazilian Symposium on Software Quality, Salvador, Brazil, November 2024, DOI: 10.1145/3701625.3701640 (Qualis CC 2020, B1)

A CASE STUDY - CRITICAL AREA ANALYSIS REPORT

A.1 User Interface

For this category, the following vulnerabilities can be exploited and pose a threat to ConsistencIA:

- Injection Flaws – as the Streamlit-hosted Website is going to have input sections;
- Authentication Flaws – as the Website users – for selected functions – are going to have to authenticate to prove their legitimacy;
- Authorization Flaws – as some functions must only be accessible by users with specific roles;
- Security Misconfiguration – as the Streamlit Webserver will have to be set up properly;
- Sensitive Data Exposure – as the data shown and the communication via HTTP requests must not leak critical information to users or third-parties.

A.2 API/Server

For this category, the following vulnerabilities can be exploited and pose a threat to ConsistencIA:

- Injection Flaws – as the API is going to receive inputs coming both from the Website and from the sensors;
- Authentication Flaws – as the users – for determined endpoints – are going to have to authenticate to prove their legitimacy;
- Authorization Flaws – as some server functions must only be accessible by users with specific roles;
- Security Misconfiguration – as the Python Flask server will have to be set up properly, following industry standards.
- Sensitive Data Exposure – as the data returned by the API and the communication via HTTP requests must not leak critical information to users or third-parties;
- Resource Exhaustion – as multiple simultaneous requests to the server may cause computational problems;

- Mass Assignment – as write-operation endpoints might compromise other data that should not be in scope;

A.3 Database/Feature Store

For this category, the following vulnerabilities can be exploited and pose a threat to ConsistencIA:

- Authentication Flaws – as the connection requests between the API and the databases might expose credential information;
- Authorization Flaws – as an unnecessarily highly-privileged database user account can generate risks;
- Security Misconfiguration – as both database servers will have to be set up properly, following industry standards;
- Sensitive Data Exposure – as both data in rest and in transit cannot be leaked to unallowed third-parties;
- Resource Exhaustion – as unlimited database connection channels might cause computational problems;
- Data Loss – as the data stored cannot be permanently lost due to unforeseen circumstances.

A.4 Machine Learning Model

For this category, the following vulnerabilities can be exploited and pose a threat to ConsistencIA:

- Training Data Integrity Flaws – as malicious inputs can cause unwanted model alterations;
- Inference Output Integrity Flaws – as maliciously crafted inputs can generate wrongful inference results;
- Model Logic Confidentiality Flaws – as critical information about the model can be exposed if it was not set up properly;
- Training Data Confidentiality Flaws – as private information contained in the training data can get exposed through inference results;
- Model Logic Integrity Flaws – as the model algorithm itself can get altered by malicious attackers.

B CASE STUDY - SOLUTION GATHERING

B.1 User Interface

For this category, the following detected vulnerabilities have the respective associated solutions for ConsistencIA:

- Injection Flaws
  - Input Sanitization: the majority of Streamlit-based output elements already has data sanitization against the most well-known types of injection flaws (just like most modern Web frameworks), such as HTML and JavaScript DOM injection. The important part is actually being careful when using pure HTML elements and/or when transferring raw untreated input data to another part of the application (e.g. to the API, for database query or ML model inference), where mitigating actions should be taken.
- Authentication Flaws
  - Password Encryption: authentication-related sensitive data must be encrypted while in transit to other parts of the application (e.g. to the API, for validation), for instance, by using a secure communication channel, such as HTTPS(TLS);
  - Session Data Safe Handling: Streamlit (just like most modern Web frameworks) already has its safe way of dealing with sessions. Nevertheless, guidelines are: session ID names must not be common/default; the ID itself should be long and random enough to prevent brute-force attacks; the content encoded must not be meaningful enough to disclose information in case of an attack; session data must only travel through encrypted connections (such as HTTPS);
  - Authentication Timeouts: expiring login sessions and invalidating temporary authentication tokens regularly is important, as to avoid compromising safety in the case of a breach;
  - Setting the Appropriate Security Flags on Cookies: if using cookies to carry session data, the 'Secure', 'HttpOnly' and 'SameSite' flags must be present on the 'Set-Cookie' headers;
- Authorization Flaws
  - Role-Based Access Control: implementing restrictions based on the level of access of each user (as directory/path protection) guarantees they are not going to be shown restricted content for their role.

- Security Misconfiguration
  - Up-to-date Software Versions: making sure the Streamlit library and all the accessory software installed are in their last versions helps avoiding possible problems;
  - Disabling Debug Mode: not revealing debug and/or error handling information can make a possible intruder's life harder as they get less insights into the system's way of functioning;
  - Disabling Directory Listing: not revealing the server's internal folder/file structure is important to reduce the amount of information available for a possible malicious person;
  - Changing Default Values: not using default/standard keys and passwords for application configuration is critical as this is generally the most trivial attack tried by a hacker.
- Sensitive Data Exposure
  - Encrypted Communication: using a cryptographed channel to transmit information (such as a TLS-backed HTTPS connection) is highly recommended, as a possible network intruder cannot access the actual content of the requests;
  - URL Protection: not using URL (path or query) parameters to convey sensitive unencrypted data is a good practice – even over encrypted connections – as they get easily shown on the client browser and can be accessed in many ways;
  - Storage Protection: avoiding storing sensitive unencrypted data in browser storage mechanisms (Local Storage, Session Storage and Indexed DB) is important, as to avoid a possible data leak in case of a XSS (Cross-Site Scripting) attack.

B.2 API/Server

For this category, the following detected vulnerabilities have the respective associated solutions for ConsistencIA:

- Injection Flaws
  - Input Sanitization: when receiving input data coming from another part of the application (e.g. from the User Interface, for database query or ML model inference), mitigating actions should be taken to clean the incoming content before actually using it, in order to avoid SQL/NoSQL injection attacks or ML-specific attacks. For the former, for instance, making use of the appropriate database connector functions is a good solution, as they have built-in statement

preparation and argument sanitization. For the latter, a more complex cleaning algorithm should be used to prepare the data;

  - Performing Active Data Validation Against Data Types and Patterns: incoming data must also be validated against pre-determined types and patterns right as the request is received.

- Authentication Flaws

  - Password Encryption/Hashing: authentication-related sensitive data must be encrypted while in transit from other parts of the application (e.g. from the User Interface, for validation), for instance, by using a secure communication channel, such as HTTPS (TLS). User credentials should also be hashed when received, and not used as plain text;
  - Authentication Timeouts: expiring login sessions and invalidating temporary authentication tokens regularly is important, as to avoid compromising safety in the case of a breach;
  - Authentication Validation: enforcing authentication validation for every single request/endpoint guarantees the user is still properly legitimate;
  - Setting the Appropriate Security Flags on Cookies: if using cookies to carry session data, the 'Secure', 'HttpOnly' and 'SameSite' flags must be present on the 'Set-Cookie' headers.

- Authorization Flaws

  - Role-Based Access Control: implementing restrictions based on the level of access of each user (as endpoint protection) guarantees they are not going to be able to perform a restricted request for their role – such as a model training operation.

- Security Misconfiguration

  - Up-to-date Software Versions: making sure the Flask library and all the accessory software installed are in their last versions helps avoiding possible problems;
  - Disabling Debug Mode: not revealing debug and/or error handling information can make a possible intruder's life harder as they get less insights into the system's way of functioning;
  - Changing Default Values: not using default/standard keys and passwords for application configuration is critical as this is generally the most trivial attack tried by a hacker;

  - Implementing Cross-Origin Resource Sharing (CORS) Policy: restricting access to the API from unallowed origins.

- Sensitive Data Exposure
  - Encrypted Communication: using a cryptographed channel to transmit information (such as a TLS-backed HTTPS connection) is highly recommended, as a possible network intruder cannot access the actual content of the requests;
  - Return Minimal Data: returning the smallest number of properties for an object is a good practice, as it can avoid exposing unnecessary sensitive information.

- Resource Exhaustion
  - Request Throttling: limiting the amount of requests (including authentication ones) a user can make in a given period of time to the the Flask API helps preventing possible credential stuffing/brute-force attacks and possible Denial of Service (DoS) attacks by exhausting computational resources, which would cause inference unavailability;
  - Limited Payload Sizes: defining a maximum size for incoming/outgoing data per request (and implementing pagination, when possible) can help improving performance, preventing bottlenecks and unavailability;
  - Maximum Pagination Parameters: defining a maximum size for returned paginated content can avoid performance drops and resource exhaustion;
  - Limiting Hardware Resource Levels for Server Applications: specifying maximum computational resource limits can avoid RAM memory or CPU overload under heavy traffic.

- Mass Assignment
  - Properties Whitelist: setting up a schema-based validation mechanism (restricting specific attributes to be received/returned in requests) prevents a malicious person from accessing out-of-scope data.

B.3 Database/Feature Store

For this category, the following detected vulnerabilities have the respective associated solutions for ConsistencIA:

- Authentication Flaws
  - Password Encryption: authentication-related sensitive data must be encrypted while in transit from/to other parts of the application (e.g. from the API, to

establish a connection), for instance, by using a secure communication channel, such as one with the TLS protocol.

- Authorization Flaws
  - Least Privilege Account: when being accessed by a normal service that is only going to run queries, an unnecessarily highly-privileged database user account can generate risks, in the case of a breach, exposing the training data and models parameters to risks.
- Security Misconfiguration
  - Up-to-date Software Versions: making sure both the Redis and the MySQL server instances are in their last versions helps avoiding possible problems;
  - Disabling Debug Mode: not revealing debug and/or error handling information can make a possible intruder's life harder as they get less insights into the system's way of functioning;
  - Changing Default Values: not using default/standard keys and passwords for server configuration is critical as this is generally the most trivial attack tried by a hacker;
  - Private Network: not leaving the databases exposed to public/external network is critical, as the API – on the internal private network – is the only element with real necessity of exchanging data with it.
- Sensitive Data Exposure
  - Encrypted Communication: using a cryptographed channel to transmit information (such as a TLS-backed HTTPS connection) is highly recommended, as a possible network intruder cannot access the actual content of the requests;
  - Encrypted Data at Rest: encrypting the data stored in the databases is a good practice, as, even in the case of a security breach, an attacker in possession of this data cannot access the actual content – mainly, the training data – without decrypting it.
- Resource Exhaustion
  - Limiting Hardware Resource Levels for Server Applications: specifying maximum computational resource limits can avoid RAM memory or CPU overload under heavy traffic;
  - Limited Connections: defining a maximum amount of database client connections simultaneously open can help improving performance, preventing bottlenecks and unavailability.

- Data Loss
  - Backup Plan: having periodic backup routines scheduled is a good practice, as to avoid permanently losing information, in the case of an unforeseen event;
  - Backup Protection: backed-up information must be protected and encrypted, as, even in the case of a security breach, an attacker in possession of this data cannot access the actual content without decrypting it.

B.4 Machine Learning Model

For this category, the following detected vulnerabilities have the respective associated solutions for ConsistencIA:

- Training Data Integrity Flaws
  - Protecting Training Data: limiting direct access to training data stored in the MySQL database – from any unnecessary portions of the application or from manual procedures – is a good practice, as it restricts points of flaw;
  - Input Filtering: running a pre-training analysis (before actually applying the ML model) on the incoming data – against pre-determined parameters/criteria – can detect highly significant outliers and less sophisticated manipulations;
  - Manipulation Detection: developing a preventive algorithm, with an input manipulation detection technique tailored to the specific ML model in use, although very complex – as it has to encompass very diverse kinds of data variation –, can be effective against adversarial attacks;
  - Adversarial Training: using the adversarial attack inputs themselves to retrain the ML model – but, then, with the right associated output – is an effective way of robustifying the algorithm and preparing it to deal with further poisoning attempts;
  - Periodic Data Checks: running passive regular analyses on the training data stored in the MySQL database – to check the values against pre-determined parameters/criteria – can detect highly significant outliers and less sophisticated manipulations, in order to trigger a reactive measure.
- Inference Output Integrity Flaws
  - Inference Request Rate Limiting: limiting the amount of inference requests a user can make – in a determined period of time – is a safe restriction technique not to release exceeding information on the model internals. It prevents a possible attacker from running countless queries with various input values,

combining them to get insights and assembling their own "truth table" on the targeted model (similar to a model evasion attack). With this artifact, the abuser can "clone the model" and gradually generate a specific input that would return their expected inference output;

  - Minimal Inference Data Output: returning the lowest possible amount of information (explanations, percentages, additional model logic values and parameters) – besides the direct inference output itself – is also a good practice, as not to release exceeding insights into the model way of functioning;
  - Input Filtering: similar to the solution presented for the section "Training Data Integrity Flaws" above;
  - Manipulation Detection: similar to the solution presented for the section "Training Data Integrity Flaws" above;
  - Adversarial Training: similar to the solution presented for the section "Training Data Integrity Flaws" above.

- Model Logic Confidentiality Flaws
  - Inference Request Rate Limiting: similar to the solution presented for the section "Inference Output Integrity Flaws" above;
  - Minimal Inference Data Output: similar to the solution presented for the section "Inference Output Integrity Flaws" above.

- Training Data Confidentiality Flaws
  - Inference Request Rate Limiting: similar to the solution presented for the section "Inference Output Integrity Flaws" above;
  - Minimal Inference Data Output: similar to the solution presented for the section "Inference Output Integrity Flaws" above.

- Model Logic Integrity Flaws
  - Model Code Protection: limiting direct access to the model algorithm and to its parameters stored in the MySQL database – from any unnecessary portions of the application or from manual procedures – is a good practice, as it restricts points of flaw.