Developing Domain-Specific Threat Models for Greater Software Security

Aspen Olmsted
School of Computer Science and Data Science
Wentworth Institute of Technology
Boston, MA 02115
olmsteda@wit.edu

Abstract— **Developing secure software applications in modern, complex environments presents significant challenges, as traditional threat modeling approaches often fall short in addressing domain-specific vulnerabilities. This paper introduces and discusses three novel, domain-specific threat models designed to enhance secure software development: BIRFS, CRIRTA, and PERTD. BIRFS (Bias, Input, Reasonable, Forensics, Sensitive) is a specialized threat model tailored for software systems that leverage Artificial Intelligence and Machine Learning algorithms, focusing on unique risks arising from data inputs, model behavior, and algorithmic biases. CRIRTA (Column, Row, Inference, Relationship, Table, Availability) provides a comprehensive framework for identifying and mitigating security threats in database applications, moving beyond generic data flow analysis to address specific database vulnerabilities. PERTD (Partition, Execution, Requisite, Timing, Data) is designed for Cloud Application Threat Modeling, emphasizing the distinct security challenges inherent in cloud environments, including distributed architecture, shared tenancy, and dynamic resource allocation. Collectively, these models aim to enable proactive risk identification during the design phase, enabling the implementation of targeted mitigation strategies earlier in the software development lifecycle. By moving beyond a sole focus on malicious user threats, these models address a broader spectrum of vulnerabilities stemming from poor design, misunderstood use cases, and environmental changes, thereby contributing to more robust and resilient software systems across diverse domains.**

Keywords- cyber-security; software engineering; software development lifecycle

I. Introduction

The landscape of software development has undergone a profound transformation in recent decades, driven by the proliferation of cloud computing, the pervasive integration of Artificial Intelligence (AI) and Machine Learning (ML) algorithms, and the ever-increasing reliance on complex database systems. While these advancements have unlocked unprecedented capabilities and efficiencies, they have simultaneously introduced a new generation of security challenges. Traditional approaches to secure software development, often centered on identifying and mitigating threats from malicious actors, are proving increasingly inadequate. Many contemporary vulnerabilities stem not from external attacks, but from inherent design flaws, a lack of understanding of system use cases, and insufficient planning for dynamic environmental changes.

In this context, the need for robust and adaptable threat modeling has become paramount. Threat modeling is a proactive security practice that enables developers and security professionals to identify potential threats and vulnerabilities early in the software development lifecycle, thereby allowing for the implementation of effective mitigation strategies before code is even written. However, the one-size-fits-all approach to threat modeling often fails to capture the nuanced risks specific to particular domains or technological paradigms. For instance, the security considerations for a cloud-native application differ significantly from those of an AI-driven system or a highly sensitive database.

This paper addresses this critical gap by introducing and discussing three novel, domain-specific threat models designed to enhance the security posture of modern software applications. We propose:

BIRFS (Bias, Input, Reasonable, Forensics, Sensitive): A specialized threat model meticulously crafted for software systems that integrate Artificial Intelligence and Machine Learning algorithms. BIRFS extends traditional security concerns to encompass unique risks such as data poisoning, model manipulation, algorithmic bias, and fairness issues, which generic threat models often overlook.

CRIRTA (Column, Row, Inference, Relationship, Table, Availability): A comprehensive framework developed to identify and mitigate security threats specifically within database applications. CRIRTA moves beyond conventional data flow analysis to address the unique vulnerabilities inherent in data storage, retrieval, and management, ensuring robust data protection and system resilience.

PERTD (Partition, Execution, Requisite, Timing, Data): A dedicated threat model for Cloud Application Threat Modeling. PERTD focuses on the distinct security challenges posed by cloud environments, including shared tenancy, complex distributed architectures, API security, and data privacy concerns across multi-tenant infrastructures.

By adopting these domain-specific models, organizations can achieve a more granular and practical approach to identifying and mitigating risks, leading to the development of inherently more secure, resilient, and trustworthy software systems. The subsequent sections of this paper will delve into the details of each of these models, outlining their principles, methodologies, and practical applications, followed by a discussion of their collective impact on the future of secure software development. This work is an extension of a previous published conference paper [1].

The organization of the paper is as follows. Section II describes the related work and the limitations of current methods. Section III describes workflow engines used in our motivating example of a distributed cloud application. Section IV discusses a current Threat Modeling technique called STRIDE. Section V discusses an alternative Threat modeling technique called DREAD. In Section VI, we give a motivating example from our distributed system modeling. Section VII describes our distributed modeling methodology. In Section VIII, we provide a motivating example from our database system modeling. Section IX describes our database modeling methodology. In Section X, we give a motivating example for our AI/ML system modeling. Section XI describes our AI/ML modeling methodology. We conclude and discuss future work in Section XII.

II. RELATED WORK

Functional requirements can be defined and represented in various ways. While these requirements serve as the foundation for software development, non-functional requirements (NFRs) provide the essential guidelines for coding implementation. Many authors have examined NFRs and the challenges of incorporating them into the design process. Pavlovski and Zou [1] NFRs are defined as specific behaviors and operational constraints, including performance expectations and policy limitations. Despite many discussions surrounding them, they are often not given the attention they deserve.

Glinz [2] suggests categorizing functional and non-functional requirements to ensure that they are inherently considered during application development. Alexander [3] points out that the language used to describe requirements is essential, noting that words ending in "-ility," such as reliability and verifiability, often refer to NFRs. Much of this research focuses on identifying NFRs. Our work builds on these foundations by applying domain-specific models using our proposed modeling technique.

Ranabahu and Sheth [4] explore four different modeling semantics to represent cloud application requirements: data, functional, non-functional, and system. Their work primarily addresses functional and system requirements, with some overlap in non-functional requirements from a system perspective. They built upon research conducted by Stuart, who defined semantic modeling languages for modeling cloud computing requirements throughout the three phases of the cloud application life cycle: development, deployment, and management. Our work fills in the gap regarding the semantic category of non-functional requirements.

Ranabahu and Sheth [4] use Unified Modeling Language (UML) to model only functional requirements. UML [6] is a standardized notation for representing software systems' interactions, structures, and processes. It consists of various diagram types, with individual diagrams linked to different perspectives of the same part of a software system. We utilize UML to express non-functional requirements as a secondary step following the PERTD models.

Integrating UML Sequence, Activity, and Class diagrams can enhance the semantics of our models. UML offers extensibility mechanisms that allow designers to add new semantics to a model. One such mechanism is a stereotype, which helps extend the vocabulary of UML to represent new model elements. Traditionally, software developers interpret these semantics and manually translate them into program code in a hard-coded manner. In our book [6], we marry the models generated by each phase of the software development lifecycle into with threat modeling and risk mitigation techniques.

The Object Constraint Language (OCL) [8] is part of the official Object Management Group (OMG) standard for UML. An OCL constraint specifies restrictions for the semantics of a UML specification and is considered valid as long as the data is consistent. Each OCL constraint is a declarative statement in the design model that signifies correctness. The expression of the constraint occurs at the class level, while enforcement happens at the object level. Although OCL has operations to observe the system state, it does not include functions to modify it.

JSON [9] stands for "JavaScript Object Notation," a simple data interchange format that began as a notation for the World Wide Web. Since most web browsers support JavaScript, and JSON is based on JavaScript, it is straightforward to support it there, which stands for "JavaScript Object Notation," a simple format used for data interchange that originated as a notation for the World Wide Web. Since most web browsers support JavaScript and JSON is based on JavaScript, it is easy to work with in web environments. Many cloud-based web services now exchange data in JSON format. JSON Schemas [10] define correctness for data passed in JSON format. We utilize an extended form of JSON schemas on the aggregated data from several web services.

Our contribution to secure software development involves new Threat Modeling techniques, coupled with modeling standards, such as UML and OCL, utilizing their extensibility mechanism of stereotypes to model non-functional requirements effectively.

III. WORKFLOW ENGINES

Workflow engines like Zapier [11] and Power Automate [12] are powerful automation tools that enable users to create and manage workflows for integrating and automating tasks across various applications and services, whether in the cloud or on-premises.

Zapier is a popular cloud-based automation platform that allows users to connect to different web applications and automate their workflows. It operates on a simple "trigger-action" model, where an event in one application triggers an action in another. Users can create "Zaps" (automated workflows) by selecting a trigger and defining the subsequent actions. For example, when a new email arrives in Gmail (trigger), the attachments can be automatically saved to Google Drive (action).

Zapier supports numerous apps and services, including well-known ones like Gmail, Slack, Salesforce, and Trello. It features a user-friendly interface, pre-built Zap templates for everyday use cases, and advanced options like filters, delays, and data transformations. Additionally, Zapier allows for multi-step Zaps, making it possible to create complex workflows with multiple actions and conditions.

Power Automate is a cloud-based service from Microsoft that allows users to automate workflows and integrate applications and services within the Microsoft ecosystem and beyond. It offers connectors for various applications, including Microsoft 365 apps (such as Outlook and SharePoint), Dynamics 365, Azure services, and third-party services like Salesforce, Dropbox, and Twitter.

Power Automate features a visual design interface where users can create workflows by combining triggers, actions, and conditions. Available triggers include email arrivals, button clicks, data changes, and scheduled events. Actions can involve sending emails, creating tasks, updating records, etc. Power Automate offers advanced capabilities like loops, parallel branches, and approval processes.

Both Zapier and Power Automate provide extensive libraries of pre-built templates and connectors, making it easier for users to begin automating tasks. They offer options to monitor and manage workflows, handle errors, and track activity logs. These platforms cater to users with varying technical expertise, from business users to developers, and help automate repetitive tasks, streamline processes, and enhance productivity.

IV. STRIDE Threat Modeling

STRIDE [12] is a threat modeling framework that offers a structured approach for identifying and analyzing threats in software systems. It helps security practitioners and developers understand potential risks and implement appropriate security controls. STRIDE is an acronym representing six categories of threats:

1. Spoofing Identity: This category involves attackers impersonating legitimate users or entities to gain unauthorized access or deceive the system. For instance, attackers may spoof a user's identity by stealing credentials or manipulating authentication mechanisms.

2. Tampering with Data: Tampering threats involve the unauthorized modification or alteration of data within the system. Attackers may tamper with data in transit, modify stored data, or manipulate system parameters to achieve desired outcomes. For example, an attacker could alter the contents of a database, inject malicious code into an application, or change parameters to bypass security checks.

3. Repudiation: Repudiation threats allow users to deny their involvement in specific transactions or activities, posing challenges for auditing and accountability. For instance, an attacker might modify logs or manipulate transaction records to evade detection or deny their actions.

4. Information Disclosure: This category addresses threats related to unauthorized exposure or disclosure of sensitive information. Attackers may exploit vulnerabilities to access confidential data, such as personal information, financial records, or intellectual property. This can happen through insecure data transmission, weak access controls, or information leakage via error messages.

5. Denial of Service: Denial of Service (DoS) threats aim to disrupt or degrade a system's availability or performance. Attackers may overload resources, exhaust system capacity, or exploit vulnerabilities to cause a service outage, rendering the system unresponsive or unusable for legitimate users.

6. Elevation of Privilege: Elevation of Privilege threats involve attackers gaining unauthorized access to higher privileges or permissions than they should have. By exploiting vulnerabilities or design flaws, attackers can bypass security controls and gain elevated access rights, leading to unauthorized data access, system compromise, or further exploitation.

When applying the STRIDE framework, security practitioners and developers analyze the software system from the perspective of each threat category. They identify potential vulnerabilities and develop corresponding mitigation strategies to address the threats. This analysis facilitates informed decisions regarding security controls, system design improvements, and the prioritization of security efforts.

V. DREAD Threat Modeling

DREAD is a threat modeling framework designed to assess and prioritize software vulnerabilities based on their potential impact. The acronym DREAD stands for five key factors used to evaluate threats:

1. Damage Potential: This factor refers to the extent of harm that could be caused if a vulnerability is exploited. It evaluates the impact, which can range from minor inconveniences to severe consequences like data breaches, system compromises, or financial losses.

2. Reproducibility: This measures how easily an attacker can reproduce or exploit a vulnerability. Vulnerabilities that are consistently easy to exploit are considered more dangerous than those that require complex or unpredictable conditions for exploitation.

3. Exploitability: This factor assesses the level of skill or effort needed to exploit a vulnerability. Vulnerabilities easily exploited with readily available tools or techniques pose a higher risk. Conversely, vulnerabilities that are difficult to exploit or require specialized knowledge are considered lower risk.

4. Affected Users: This evaluates the number of users or systems a vulnerability could impact. A vulnerability affecting numerous users or critical systems is considered more significant than one impacting only a limited subset of users.

5. Discoverability: This assesses how likely an attacker is to find the vulnerability. Vulnerabilities that are easily discoverable—through public disclosures, known attack techniques, or automated scanning tools—are riskier than those that are harder to find or require advanced reconnaissance.

Using the DREAD framework, each factor is scored on a scale from 0 to 10, with 0 being the least concerning and ten being the most critical. These scores help prioritize vulnerabilities and allocate resources for mitigation efforts. Higher scores indicate a higher priority for addressing the identified vulnerabilities.

While DREAD is a valuable tool for assessing and prioritizing vulnerabilities based on their potential impact, it should be used alongside other threat modeling techniques and considerations to ensure a comprehensive security analysis and informed decision-making.

VI. Distributed Model Motivating Example

The challenge with the STRIDE and DREAD threat models is that they primarily focus on vulnerabilities associated with malicious user activities. However, many risks arise from architecture, the environment, or human error.

Consider a common architecture used by many businesses today: data generated by an online transaction processing (OLTP) system, either stored on-premises or logically on-premises, is synchronized to a cloud system considered off-premises and beyond the organization's control. This scenario is not uncommon in today's business landscape.

Consider a large performing arts venue employing a local SQL Server-based system for ticketing and donation transactions. Meanwhile, its marketing department uses a cloud-based email and SMS marketing system. The OLTP data must be extracted, translated, uploaded, and loaded regularly for the marketing system to function correctly.

Various issues can arise when multiple processes and data are transferred across networks that span domain boundaries. A UML activity diagram illustrates the steps involved in moving data from the on-premises OLTP system to the cloud-based system used by the marketing team. This model shows that activities occur in both environments. The challenge with the STRIDE and DREAD threat models is that the vulnerabilities modeled and the matching remediations target malicious user activities. Many times, risks come from architecture, environment, or human error.

A motiving example is an architecture that is used in many businesses today where data that is generated in OLTP systems that are either stored on-premises or logically on-premises is synchronized to a cloud system that is considered off-premises and outside the domain of control of the organization. To understand this better, consider a large performing arts venue that utilizes a local SQL Server-based system to process ticketing and donation transactions. The marketing department uses a cloud-based system for email and SMS marketing. The OLTP data must be extracted, translated, uploaded, and loaded regularly for the marketing system to be functional.

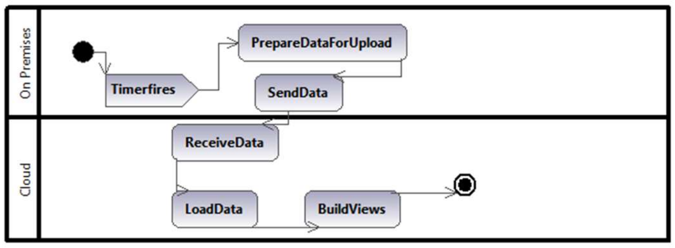


Figure 1 - Upload Activity

Understanding the data transfer process is crucial to prevent potential risks. Figure 1 shows a UML activity diagram for moving data from the on-premises OLTP system to the cloud-based system used by the marketing folks. In the model, you will see that activities happen in both partitions.

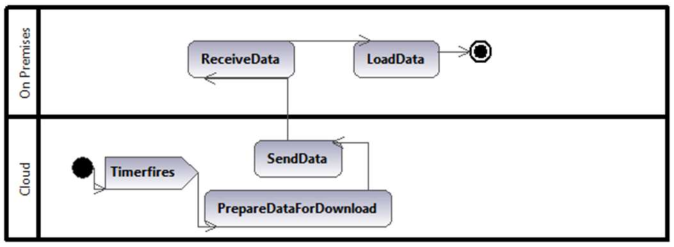


Figure 2 - Download Activity

Table 1 - Upload Activity STRIDE Model

Action	S	T	R	I	D	E
Timerfires						
PrepareDataForUpload						
SendData	X	X		X	X	
ReceiveData						
LoadData						
BuildViews						

Figure 2 presents a model outlining the execution path for retrieving data from the cloud system. The data includes sending activity for both emails and SMS text messages. This sending activity can be substantial, encompassing tuples for sends, opens, clicks, and bounces. Additionally, information regarding communication preferences and unsubscribed data is retrieved.

The marketing department requires service availability and data integrity for its business operations. For instance, NFRs could specify that the system must be available 99.999% of the time or that the data must be no more than 24 hours old. Whenever a distributed system is proposed, a model should be developed to represent these NFRs and the threats to the system's ability to meet them.

Unfortunately, the focus of STRIDE and DREAD on malicious users does not adequately address many of the risks in our motivating example. Table 1 Illustrates a STRIDE model corresponding to the update activity depicted in Figure 1, while Table 2 shows the STRIDE model related to the download activity from Figure 2. In the STRIDE model, actions are at risk from malicious users; however, many steps are also vulnerable to environmental issues that can impact the system's availability and integrity. Examples of these issues include network and system outages, concurrent computational usage on equipment, and a lack of control over the quality of source data.

Table 2 - Download Activity STRIDE Model

Action	S	T	R	I	D	E
Timerfires						
PrepareDataForDownload						
SendData	X	X		X	X	
ReceiveData						
LoadData						

VII. PERTD MODEL

We developed the PERTD Model to better assess the risks associated with distributed applications [13]. This model addresses four main environmental risk categories for distributed systems:

1. PARTITION

Activities vulnerable to partition errors will fail if a network is partitioned between on-premises devices and the cloud. Risk reduction strategies include:

- Pausing the complete workflow and retrying

- Utilizing previous execution data

- Employing alternative data sources

2. EXECUTION

Activities that are susceptible to execution errors may fail due to ambiguous code requirements, leading to runtime or tooling errors. For example, queries that generate data might fail with future datasets. Risk reduction measures include:

- Utilizing previous execution data (most systems create a copy before execution)

- Using alternative data sources

3. REQUISITE

Table 3 - Upload Activity PERTD Model

Action	P	E	R	T	D
Timerfires		X			
PrepareDataForUpload		X			
SendData	X	X	X	X	
ReceiveData	X	X	X	X	
LoadData		X	X	X	X
BuildViews		X	X		

Activities with requisite vulnerabilities depend on prerequisite activities. If a prerequisite fails, the dependent activity becomes stale. Risk reduction can involve:

- Utilizing previous execution data

- Employing alternative data sources

4. TIMING

Activities at risk due to timing need to finish within a specific time window or under a threshold duration. Risk reduction strategies include:

- Utilizing previous execution data (most systems create a copy before execution)

- Using alternative data sources

5. DATA

Activities are at risk because data are often combined from different sources. Unfortunately, schema correctness specifiers only apply to one data source. Risk reduction strategies include:

- Additional workflow steps to verify correctness

In Tables 3 and 4, we apply our PERTD model to analyze risks associated with uploading and downloading activities. The PERTD model captures significantly more risks than the STRIDE model.

After identifying NFRs in the PERTD model, we develop standard UML Class, Sequence, and Activity Diagrams. The threats to the system are modeled using UML stereotypes. UML stereotypes extend the standard UML language by introducing custom or specialized elements, properties, and behaviors. They allow adding domain-specific annotations, constraints, or semantics to UML elements, enhancing expressiveness and tailoring modeling to specific contexts. Stereotypes are indicated by guillemets (<< >>) placed above the name of the stereotyped element.

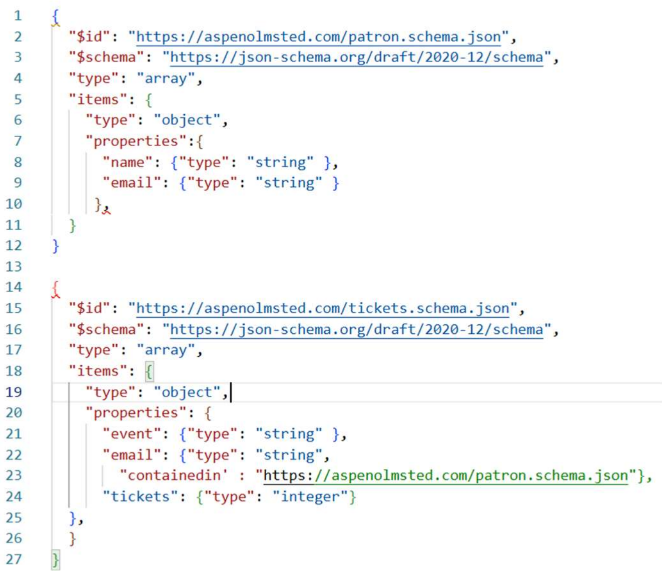
Stereotypes can be attached to classes, messages, attributes, and activities. With the PERTD model, we incorporated the four risk categories as stereotypes: <<PARTITION>>, <<EXECUTION>>, <<REQUISITE>>, <<TIMING>> and <<DATA>>. These stereotypes are then tagged to messages in UML Sequence and Activity diagrams, while data classes and individual attributes can also be tagged if they are susceptible to these risks.

Table 4 - Download Activity PERTD Model

Action	P	E	R	T	D
Timerfires		X			
PrepareDataForDownload		X			
SendData	X	X	X	X	
ReceiveData	X	X	X	X	
LoadData		X	X		X

Additionally, OCL is included to specify invariants that can define additional semantics related to the correctness of method calls, classes, or attributes. For instance, if data in a particular class must be no older than three days, this can be expressed using the last_update attribute.

To verify data from when it is vulnerable, we utilize an extended version of JSON Schemas [10]. Our extension allows the Schema to reference different data sources. JSON schema supports a CONTAINS operator to verify the existence of an element in a collection. We added a CONTAINEDIN operator to span across schemas represented by different data sources in the distributed system. We also added a NOTCONTAINEDIN to verify the absence of an element. Figure 3 shows two sample schemas. The top schema is a simplified version of a patron, the bottom schema is a simplified version of a ticket. They share an email field which is designated in the tickets schema to require the existence in the patron data.



```
{
  "$id": "https://aspenolmsted.com/patron.schema.json",
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "type": "array",
  "items": {
    "type": "object",
    "properties":{
      "name": {"type": "string" },
      "email": {"type": "string" }
    },
  }
}

{
  "$id": "https://aspenolmsted.com/tickets.schema.json",
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "type": "array",
  "items": {
    "type": "object",
    "properties": {
      "event": {"type": "string" },
      "email": {"type": "string",
        "containedin' : "https://aspenolmsted.com/patron.schema.json"},
      "tickets": {"type": "integer"}
  },
  }
}
```

Figure 3 - Sample Schema

To mitigate the risk of data integrity issues, we validate the data against the specified schemas as part of the data workflow.

VIII. DATABASE MODEL MOTIVATING EXAMPLE

The challenge with the STRIDE and DREAD threat models for database application security is that the vulnerabilities modeled and the matching remediations target malicious user activities. Many times, risks come from architecture, environment, or human error.

To explore a motivating example, consider an event ticketing system used in a consortium of performing arts centers, such as a symphony, opera, ballet, and Broadway venue. A transaction processing (OLTP) system stores the data generated from the patron transactions on-premises for ticket purchases. Data needs to be logically partitioned so patrons are shared, but only the transaction data appropriate to the organization is visible on software screens and reports. Patrons can go to individual constituent box offices to work with an agent on transactions related to that constituent organization, or they can use self-service on a web or mobile interface that allows them to purchase tickets to any constituent organization or see all their individual transaction histories.

To illustrate a weakness in the traditional threat models when applied to database security, we picked out four use cases from our motivating example:

- A constituent box office agent accessing a patron's ticket history. In this case, the data rows must be confidential, so only those related to the constituent organization should be visible.
- A constituent box office agent selecting a patron's general admission (GA) seats. In this use case, the agent should only have access to constituent inventory and not exceed capacity.
- A patron accessing their ticket history through a self-service web page. In this case, the data rows must be confidential to just the patron's data and not make other patrons' histories available.
- A patron selects general admission (GA) seats through a self-service web page. In this use case, the patron should not exceed the venue capacity.

Unfortunately, the focus of STRIDE and DREAD on the malicious user does not account for many of the risks in our motivating example. Table **5** shows a STRIDE model to match the four use cases we are analyzing. Table 6 shows the equivalent DREAD model. In both models, we can see a large surface area of vulnerabilities to malicious user attacks. As stated earlier, most security risks to database software and software generally come from not understanding and enforcing security requirements in the software engineering process. This lack of enforcement leaves an application vulnerable to user error and malicious attacks.

Table 5 - STRIDE Model for Database Software

Action	S	T	R	I	D	E
Agent View of Ticket History	X			X	X	X
Agent Reserving GA Seat Selection		X	X		X	X
Self-Service View of Ticket History	X			X	X	X
Self-Service Reserving GA Seat Selection		X	X		X	X

IX. CRIRTA MODEL

We developed the CRIRTA Model to better model the risks associated with database applications. The model covers a database system's seven main environmental risk categories.

A. *Column Confidentiality*

Activities vulnerable to exposing column confidentiality have data with sensitive data, where some users should have restricted read and write access. Other users will need access to the column values. Risk reduction can include:

- Removing all access at the table level for users who should not have access.
- Creating database views without the column values and granting access to these user groups

B. *Row Confidentiality*

Activities vulnerable to row confidentiality have user access that must be restricted to specific values for some user groups. Examples include self-service apps where users can only see their data, or departmental users can only see department records. Risk reduction can consist of:

- Removing all access at the table level for users who should not have access.
- Creating database views with where statements that restrict the rows that are visible to these user groups

C. Column Inference

Activities vulnerable to column inference allow users to infer other values based on values in these columns. Gender or race are often examples of this when the greater data population is relatively homogenous. Risk reduction can include:

- Limiting access to the column to essential users (see column confidentiality reductions above)

D. Relationship Correctness

Activities vulnerable to relationship correctness issues have correctness that spans a relationship between tables. An example would be an order that needs a shipping address that exists in a customer's address. Risk reduction can include:

- Utilizing database foreign keys
- Utilizing database check constraints with queries that check for the existence of rows or values in related tables.
- Utilizing database triggers

E. Table Correctness

Activities vulnerable to table correctness issues have column values restricted based on other columns or sets of column values. Risk reduction can include:

- Utilizing database check constraints with predicates that check row values
- Utilizing database triggers

F. Availability

Activities vulnerable to availability issues often use locks to access one process at a time. The exclusivity is done with the database isolation level. Risk reduction can include:

- Minimize code with higher restriction levels
- Table design changes

TABLE 6 - DREAD Model for Database Software

Action	D	R	E	A	D
Agent View of Ticket History	2	3	7	1	5
Agent Reserving GA Seat Selection	3	3	9	9	2
Self-Service View of Ticket History	6	6	8	3	5
Self-Service Reserving GA Seat Selection	9	6	9	9	2

We model the risks to the motivating example system in Table **7** - *CRICRTA* Model that utilizes our CRIRTA model. As you can see, the CRIRTA model captures many more risks than the STRIDE and DREAD models.

Table 7 - *CRICRTA* Model

Action	C	R	I	R	T	A
Agent View of Ticket History		X		X	X	
Agent Reserving GA Seat Selection			X	X	X	
Self-Service View of Ticket History		X		X	X	
Self-Service Reserving GA Seat Selection			X	X	X	

After identifying NFRs in the CRIRTA model, standard UML Class, Sequence, and Activity Diagrams are developed. The threats to the system are modeled by utilizing UML Stereotypes. UML (Unified Modeling Language) stereotypes are a way to extend the standard UML language by introducing custom or specialized elements, properties, and behaviors. Stereotypes allow you to add domain-specific annotations, constraints, or semantics to UML elements, making them more expressive and tailored to specific modeling contexts. Stereotypes are denoted by guillemets (<< >>) placed above the name of the element being stereotyped.

Stereotypes can be attached to classes, messages, attributes, and activities. With the CRIRTA model, we added the six CRIRTA categories as stereotypes: << ColumnConfidentiality >>, <<RowConfidentiality >>,<< Column Inference >>,<< RelationshipCorrectness>>,<<TableCorrectness>>,<<Availability>>. These stereotypes are then tagged to messages in UML Sequence and Activity diagrams; data classes and individual attributes can be tagged with the stereotype if the data in the class or attribute is susceptible to the risk.

OCL is added to provide invariants that can specify additional semantics related to the correctness of a method call, class, or attribute. An example is if data in a particular table must have a related table with a specific attribute range.

X. AI/ML MODEL MOTIVATING EXAMPLE

The challenge with the STRIDE and DREAD threat models for applications that consume ML and AI algorithms is that the vulnerabilities modeled and the matching remediations are aimed at malicious user activities attacking a system. Many times, risks come from architecture, environment, or human error.

A motivating example is an architecture that is used in many businesses today, where data that is generated in online transaction processing (OLTP) systems that are either stored on-premises or logically on-premises is synchronized to a cloud system that is considered off-premises and outside the domain of control of the organization. Once the data is in the cloud, it is augmented utilizing AI or ML algorithms. To understand this better, consider a large performing arts venue that operates a local SQL Server-based system to process ticketing and donation transactions. The marketing department uses a cloud-based system for email and SMS marketing. The OLTP data must be extracted, translated, uploaded, and loaded regularly for the marketing system to be functional.

Once the data is in the cloud, the data is augmented for several purposes, including segmentation, lookalike matching, data flows, and personalized advertising. Segmentation arranges potential customers into groups based on attributes and activities in the input data. Lookalike matching is used to match the attributes and activities of new customers to similar customers who have been with the organization for a more extended period. Input data is utilized to design interactions with the new prospects to increase their engagement with the organization. The input data from the attribute or activities is also used to design personalized advertisements to motivate the new prospect to engage in that next activity.

Unfortunately, the input data may have been entered incorrectly in a self-service fashion, such as through a mobile application, web page, or kiosk. A tired or poorly trained customer service representative may have entered the data incorrectly. A third option for poor input data is the data may have come incorrectly from an external organization, such as a biographical data service provider that may suggest updates to addresses, phone numbers, or email addresses. A fourth option could be a malicious user intentionally polluting the data in revenge for some wrong they feel was done to them by the organization.

The best case is that an improper email is sent or the prospect displays and ignores an advertisement or offer. Worse case, an organization's reputation is damaged, and sales and relationships with customers are lost. It would be best to discover the vulnerabilities early in the design process or software development to minimize the risk.

Table 8 – AI/ML Activity STRIDE Model

Action	S	T	R	I	D	E
Segmentation						
LookALike						
Data Flow	X					
Personalized Advertisements						

Table **8** shows the four activities that utilize AI or ML in the cloud arranged in a STRIDE Model. We marked the columns for spoofing with the data flow activity, as we can envision a scenario when a user may want to qualify for a new customer offer by creating a fake account. The rest of the model is left blank as they are inappropriate threats to the activities. The small number of threats in our model could lead a software development team to believe the application is not at risk based on the limited representation in the STRIDE model.

XI. BIRFS Threat Modeling

To better model the risks associated with applications that utilize AI or ML algorithms, we developed the BIRFS Model [14]. The BIRFS Model covers a process or system's five leading risk categories that utilize AI or ML algorithms.

B-potential biases in output

The B in BIRFS comes from the risk of potential biases in the AL or ML algorithm output. AI algorithms can exhibit biases in their production due to various reasons. These biases can arise from the data used to train the models, the design of the algorithms, or both. Here are some familiar sources of biases in AI algorithms:

- **Biased Training Data:** If the training data used to train the AI model is biased, the model will learn and perpetuate those biases. For example, if historical data used for training reflects societal biases, the model may replicate and amplify those biases.
- **Data Sampling Bias:** If the training data does not represent the entire population, the model may be biased toward the overrepresented groups. The sampling bias can lead to inaccurate predictions or decisions for underrepresented groups.
- **Labeling Bias:** Biases in labeling training data can also contribute to biased models. If the labels used for training data are biased, the model may learn and propagate those biases.
- **Algorithmic Bias:** The design and structure of the algorithm itself can introduce bias. The bias may happen if the algorithm uses feature proxies for sensitive attributes (e.g., using ZIP code as a proxy for race) or if the algorithm inherently reflects certain societal biases.
- **Implicit Bias in Training Examples:** Biases in the examples used to train the model can also contribute to biased outputs. For instance, if a model is trained on text from the internet, it may learn and reproduce the biases present in that text.
- **Lack of Diversity in Development Teams:** The composition of the teams developing AI models can also influence biases. If development teams lack diversity, there may be a lack of perspectives and awareness regarding potential biases in the models.
- **Feedback Loop Bias:** Biases can be reinforced through feedback loops. For example, biased predictions can lead to biased outcomes, which are then used as input for future predictions, creating a self-reinforcing cycle.
- **Contextual Bias:** The context in which the AI system is deployed may introduce bias. For instance, a model trained on data from a specific cultural or geographical context may not generalize well to other contexts.

Addressing biases in AI algorithms is a complex and ongoing challenge. It requires careful consideration at every stage of the AI development process, including data collection, model training, and deployment. Strategies such as diverse and representative data collection, regular audits of model outputs, and involving diverse teams in AI development can help somewhat mitigate biases.

I - input is outside the domain of control.

The I in BIRFS comes from the risk when the input data comes from outside the organization's control. When it comes to AI models, some factors and inputs are outside the direct control of the developers or operators. These external influences can affect the performance and behavior of AI models in various ways. Here are some examples:

- **External Data Changes:** AI models are often trained on historical data, and if the real-world data changes over time, it can impact the model's performance. For example, sudden shifts in user behavior, market dynamics, or other external factors might lead to a mismatch between the training data and the current environment.
- **Changing User Expectations:** User expectations and preferences can evolve. AI models trained to meet specific user needs may become less effective if user expectations change, and these changes are beyond the direct control of the AI developers.
- **Regulatory Changes:** Changes in regulations or legal frameworks can significantly impact how AI models operate, especially in highly regulated industries. Developers may need to adapt models to comply with new laws or regulations.
- **External Security Threats:** AI systems can be vulnerable to external security threats. Malicious actors may attempt to manipulate or compromise AI models, leading to undesirable outcomes. These threats are often beyond the control of the AI developers and require ongoing security measures.
- **Environmental Factors:** The performance of AI models may be affected by environmental factors, such as changes in weather, network conditions, or other external variables. For example, a model designed for specific weather conditions might perform differently in a completely different climate.
- **Integration with External Systems:** AI models are often integrated into larger systems and workflows. Changes or issues in these external systems, which are not under the direct control of AI developers, can impact the overall performance of the AI model.
- **Global Events and Catastrophes:** Unforeseen global events, such as natural disasters, economic crises, or pandemics, can have widespread effects on various industries. AI models operating in affected domains may face challenges due to disruptions caused by such events.
- **User Feedback and Interactions:** User interactions and feedback can influence the behavior of AI models. Users providing biased or unrepresentative feedback may impact the model's learning and performance. User behavior is often outside the direct control of AI developers.

Addressing the challenges posed by external factors requires robust system design, ongoing monitoring, and adaptability. Developers should consider building models that adapt to environmental changes and implement continuous monitoring and update mechanisms.

R - output result does not deviate from a reasonable range

The R in BIRFS comes from the risk when the output is wrong but in a reasonable range for the input data. Ensuring that the output results of AI algorithms fall within a reasonable and expected range is crucial for the reliability and safety of AI systems. Deviations outside a reasonable range can pose risks and challenges. Here are some considerations related to the risk of output results deviating from a reasonable range in AI algorithms:

- **Out-of-Distribution Data:** If an AI model encounters data significantly differently from what it was trained on (out-of-distribution data), it may produce unpredictable and unreliable results. The new data ranges can happen if the model encounters scenarios or inputs not adequately represented in the training data.
- **Overfitting:** Overfitting occurs when a model learns the training data too well but fails to generalize to new, unseen data. In such cases, the model may perform well on the training set but poorly on real-world data, leading to outputs that deviate from a reasonable range.
- **Data Quality Issues:** Poor-quality or biased training data can contribute to the model learning incorrect patterns or making inaccurate assumptions. If the data used to train the model is not representative or contains errors, the model's outputs may deviate from what is considered reasonable.
- **Lack of Explainability:** If an AI model is too complex or lacks interpretability, it may be challenging to understand why it produces a particular output. This lack of transparency can make identifying and addressing deviations from a reasonable range challenging.
- **Concept Drift:** Over time, the underlying patterns in data may change, a phenomenon known as concept drift. If an AI model is not regularly updated to adapt to these changes, its outputs may become less accurate and fall outside the expected range.
- **Adversarial Attacks:** Adversarial attacks involve intentionally manipulating input data to deceive an AI model and produce incorrect outputs. If an AI system is vulnerable to such attacks, the outputs may deviate from the reasonable range, posing security and reliability risks.
- **Uncertainty and Confidence Estimation:** AI models should ideally provide measures of uncertainty and confidence in their predictions. If a model is overly confident in its outputs, even in situations where it should be uncertain, it may lead to outputs that deviate from a reasonable range.
- **Monitoring and Validation:** Monitoring and validating model outputs against real-world data are essential. If the model's performance degrades or deviates from expected behavior, it should trigger alerts for further investigation and potential retraining.

It's vital to employ good practices in data collection, preprocessing, model training, and ongoing monitoring to mitigate the risk of output results deviating from a reasonable range. Additionally, incorporating human oversight and feedback mechanisms can enhance the system's robustness and help identify and correct possible deviations. Regular updates and retraining of models based on new and relevant data are also crucial for maintaining performance in dynamic environments.

F - forensics or logging to defend results

The F in BIRFS comes from the risk when the algorithm process cannot be audited to discover why it produced a particular output based on a set of input data. Forensics and logging play crucial roles in defending the results generated by AI algorithms. They provide a means to trace, understand, and validate the processes and decisions made by AI models. Here are critical aspects of forensics and logging in the context of AI:

- **Data Logging:** Record all relevant data in the AI model's training and inference processes. The logging includes input data, preprocessing steps, feature engineering, and any transformations applied to the data. Having a detailed log helps in understanding the context and ensuring transparency.
- **Model Configuration and Hyperparameters:** Document and log the configuration settings and hyperparameters used during the training of AI models. This information is crucial for reproducibility and understanding the model's setup.
- **Model Training Logs:** Record information about the training process, such as training loss, accuracy metrics, and other relevant statistics. The logs help track the model's performance during training and identify potential issues.
- **Algorithm Versions and Updates:** Keep track of the versions of algorithms and models used. When updates or changes are made to the algorithm, logging ensures you can trace which version was used for specific results.
- Timestamps and Versioning: Timestamps in logs can be critical for establishing a chronological order of events. Additionally, versioning of data, models, and algorithms helps associate specific results with the corresponding versions used.
- **Explainability and Interpretability Logs:** Record explanations and interpretations provided by the AI model, especially in cases where explainability is crucial. The logs can include feature importance, attention weights, or any other information that aids in understanding the model's decisions.
- **User Interactions and Feedback:** If the AI system involves user interactions, log relevant user inputs and system responses. This information is valuable for analyzing user experiences and addressing issues from the user's perspective.
- **Monitoring and Anomaly Detection Logs:** Implement monitoring logs to track the model's performance in real-time. Logging anomalies or unexpected behavior helps identify when the model's outputs deviate from expected ranges.
- **Security Logs:** Incorporate security logs to capture information about potential adversarial attacks or unauthorized access. Security logs can help identify and mitigate security risks.
- **Compliance and Regulations:** Ensure that logging practices align with relevant compliance requirements and regulations. Some industries and regions may have specific guidelines regarding data handling and logging.
- **Forensic Tools and Techniques:** Develop or utilize forensic tools and techniques to investigate issues or discrepancies in AI model results. These tools can help conduct detailed analyses of model behavior and decision-making processes.
- **Human-in-the-Loop Logging:** If human reviewers are involved in decision-making, log their interactions and decisions. This information can be valuable for understanding the role of human oversight in the system.

By incorporating comprehensive logging and forensic practices, developers and operators of AI systems can enhance transparency, accountability, and the ability to defend the results produced by AI algorithms. These practices are critical in applications where the stakes are high, such as healthcare, finance, and critical infrastructure.

S - Sensitive or private data needs to be protected

The S in BIRFS comes from the risk associated with protecting private or sensitive input data. Protecting sensitive or private data is a critical aspect of using AI algorithms, and there are several techniques and best practices to ensure data privacy and security. Here are key considerations:

- **Data Encryption:** Implement encryption mechanisms to protect data at rest and in transit. Encryption ensures the data remains unreadable even if unauthorized access occurs without the appropriate decryption keys.
- **Secure Data Storage:** You should store sensitive data in secure, access-controlled environments. Utilize secure databases and storage systems with proper access controls to restrict unauthorized access to sensitive information.
- **Data Masking and Anonymization:** Before using data in AI algorithms, consider techniques such as data masking and anonymization to replace or generalize sensitive information. Masking and anonymization reduce the risk of exposing private details introduced during model training.
- **Federated Learning:** Consider federated learning when data cannot be centralized. This approach allows models to be trained across decentralized devices or servers without exchanging raw data, preserving privacy.
- **Differential Privacy:** Implement differential privacy techniques to add noise or randomness to data, making it harder to link specific data points to individuals. Differential privacy protects individual privacy while still allowing meaningful analysis.
- **Access Controls and Authorization:** Implement strict access controls and authorization mechanisms to ensure that only authorized personnel can access sensitive data. Regularly review and audit access permissions.
- **Secure Model Deployment:** When deploying AI models, ensure the inference process is conducted securely. Limit access to the model and its outputs to authorized users and systems.
- **Secure APIs:** If AI models are accessed through APIs (Application Programming Interfaces), secure the APIs by implementing authentication and authorization mechanisms and encrypting data transmitted between the client and the API.
- **Regular Security Audits:** Conduct regular security audits to identify vulnerabilities in the system. The regular security

audits include the AI models and the infrastructure supporting them.

- **Compliance with Privacy Regulations:** Ensure your AI system complies with relevant privacy regulations such as GDPR, HIPAA, or other industry-specific standards. Understand the legal requirements for handling sensitive data and incorporate necessary safeguards.
- **Data Lifecycle Management:** Establish clear policies for the entire data lifecycle, including collection, storage, processing, and disposal. Regularly review and update these policies to align with evolving privacy and security standards.
- **Educate and Train Personnel:** Train personnel handling sensitive data on privacy best practices and security protocols. Human error is a common source of data breaches, so awareness and education are crucial.
- **Incident Response Plan:** Develop a comprehensive incident response plan to address potential data breaches. This plan should outline the steps during a security incident, including communication strategies and legal obligations.
- **Transparent Communication:** Communicate to users and stakeholders how their data will be used, processed, and protected. Transparent communication builds trust and helps users understand the measures to safeguard their privacy.

By incorporating these measures, organizations can significantly reduce the risks of handling sensitive or private data in the context of AI algorithms. Data privacy should be a fundamental consideration throughout the entire AI development and deployment lifecycle. Table **9** displays the same four AI/ML activities discussed earlier in the BIRFS model. As you can see from the model, many more risks and vulnerabilities are represented by the model than in the previous STRIDE model. The mitigations discussed, along with the BIRFS model elements can next be applied to reduce the risks.

Table 9 - AI/ML Activity BIRFS Model

Action	B	I	R	F	S
Segmentation	X	X	X	X	
LookALike	X	X	X	X	X
Data Flow	X		X	X	X
Personalized Advertisements	X		X	X	

XII. Conclusions and Future Works

In this work, we provide three domain-specific modeling methodologies to handle issues in cloud, database, and AI/ML software related to NFRs in distributed systems. We present modeling methodologies aimed at uncovering issues related to NFRs in the software development lifecycle. Our models enable us to identify significantly more fine-grained risks associated with the development of these systems than traditional threat modeling techniques. Additionally, we have enhanced the modeling of functional requirements by employing UML stereotypes to represent the NFRs identified in the models. Future work will incorporate code generation to mitigate the risks identified and modeled throughout this process.

References

[1] A. Olmsted, "PERTD - Cloud Application Threat Modeling," in *Proceedings of CLOUD COMPUTING 2025, The Sixteenth International Conference on Cloud Computing, GRIDs, and Virtualization*, Valencia, Spain, 2025.

[2] C. J. Pavlovski and J. Zou, "Non-functional requirements in business process modeling," *Proceedings of the Fifth on Asia-Pacific Conference on Conceptual Modelling, vol. 79,* pp. 1-10, 2008.

[3] M. Glinz, "Rethinking the Notion of Non-Functional Requirements," *Third World Congress for Software Quality, Munich, Germany,* pp. 1-10, 2005.

[4] Alexander, I, "Misuse Cases Help to Elicit Non-Functional Requirements," *Computing & Control Engineering Journal, 14, 40-45,* pp. 1-10, 2003.

[5] R. Ajith and A. Sheth, "Semantic Modeling for Cloud Computing, Part I," *Computing,* vol. May/June, pp. 81-83, 2010.

[6] Object Management Group, "Unified Modeling Language: Supersturcture," 05 02 2007. [Online]. Available: http://www.omg.org/spec/UML/2.1.1/. [Accessed 11 Nov 2025].

[7] A. Olmsted, Security-Driven Software Development: Learn to analyze and mitigate risks in your software projects, Birmingham, UK: Packt Publishing, 2024.

[8] Object Management Group, "OMG Formally Released Versions of OCL," 02 2014. [Online]. Available: http://www.omg.org/spec/OCL/. [Accessed 11 Nov 2025].

[9] JSON.org, "Introducing JSON," 2024. [Online]. Available: https://www.json.org/json-en.html. [Accessed 11 Nov 2025].

[10] Open Collective, "JSON Schema," 2024. [Online]. Available: https://json-schema.org/. [Accessed 11 Nov 2025].

[11] Zapier Inc., "Automate without limits," 2024. [Online]. Available: https://zapier.com/. [Accessed 11 Nov 2025].

[12] Microsoft, "Power Automate," 2024. [Online]. Available: https://www.microsoft.com/en-us/power-platform/products/power-automate. [Accessed 11 Nov 2025].

[13] R. Khan, D. Laverty, D. McLaughlin and S. Sezer, "STRIDE-based threat modeling for cyber-physical systems,," in *2017 IEEE PES Innovative Smart Grid Technologies Conference Europe (ISGT-Europe)*, Turin, Italy, 2017.

[14] A. Olmsted, "BIRFS is a Threat Model for Software Systems That Utilize Artificial Intelligence or Machine Learning Algorithms," *International Conference on Artificial Intelligence and its Application,* pp. 23-37, 2024.