PERTD - Cloud Application Threat Modeling

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

***Abstract*— This research work investigates the problem of developing secure cloud software applications. Currently, proposed solutions focus on data flow across so-called trust boundaries. The challenge with the current approach is that many of our applications' threats are not from malicious users. Many threats come from poor design, misunderstanding of use cases, and a lack of planning for environmental changes. This research focuses on the challenges of developing secure cloud software applications through a modeling process that allows us to identify risks to the cloud software during the design phase and implement strategies to mitigate those risks in the coding and implementation phase.**

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

I. INTRODUCTION

Secure software development stands at the intersection of innovation and protection, emphasizing the design, implementation, and maintenance of software systems with a robust focus on security. By embedding security measures and best practices throughout the development lifecycle, we can transform vulnerabilities into resilient defenses against potential threats. Here are some inspiring research areas in secure software development:

1. Secure Coding Practices: This area champions identifying and promoting vital coding techniques that empower developers to write secure code. By exploring common programming errors and vulnerabilities, we can equip ourselves with tools like static code analysis and automated vulnerability detection, paving the way for robust software security.

2. Threat Modeling: In a proactive approach, threat modeling illuminates potential threats and vulnerabilities early in development, giving you a sense of preparedness and control. This research area enables developers to refine their techniques and effectively chart pathways to improved security through tools like attack tree analysis and risk assessment methodologies.

3. Security Testing: Evaluating software for weaknesses becomes a quest for excellence, driving us to improve constantly. Innovative techniques such as penetration testing and fuzz testing serve as guardians of security, while automated processes revolutionize how we ensure the integrity of our software systems.

4. Secure Software Architectures: Research in this field aspires to design architectures that withstand attacks, safeguarding sensitive information with secure component integration and effective communication protocols.

5. Secure Software Development Processes: Methodologies become a fortress by embedding security at every stage of the software development lifecycle, from requirements engineering to incident response planning, forming an unshakeable foundation of trust.

6. Secure DevOps and Agile Development: In the fast-paced realms of DevOps and agile methodologies, research navigates the exciting intersection of speed and security, integrating practices that ensure rapid innovation without compromise.

7. Secure Software Analytics: This area of research uncovers patterns and anomalies in software-related data, harnessing the power of machine learning and data mining to predict vulnerabilities and bolster our defenses.

8. Security Education and Training: Elevating security education for developers transforms knowledge into action, fostering a culture of security awareness that resonates within software development teams.

These research areas advance secure software development and inspire a collective drive to protect our digital world and mitigate risks associated with cyber threats and attacks. Our paper's focus on threat modeling is a call to action, aiming to reduce risks to software functionality, regardless of the source of potential danger.

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 study. Section VII describes our modeling methodology. We conclude and discuss future work in Section VIII.

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 their groups 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 [3] 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 [3] use Unified Modeling Language (UML) to model only functional requirements. UML [5] 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) [7] 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 [8] 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 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 [9] 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 for cloud applications involves a new Threat Modeling technique, coupled with modeling standards, such as UML and OCL, utilizing their extensibility mechanism of stereotypes to model non-functional requirements effectively. We allow for an aggregated JSON Schema with our extensions to validate the combined data format.

III. WORKFLOW ENGINES

Workflow engines like Zapier [10] and Power Automate [11] 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 aids security practitioners and developers in understanding potential risks and implementing 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.

TABLE 1 - UPLOAD ACTIVITY STRIDE MODEL

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

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. 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.

Understanding the data transfer process is crucial to prevent

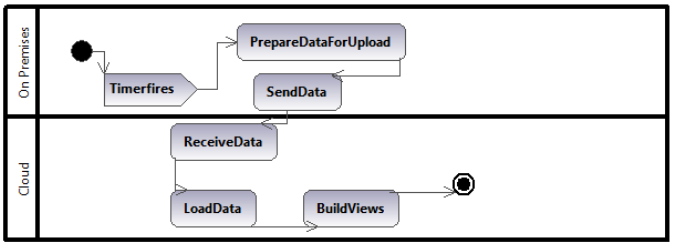

Figure 1 - Upload Activity

potential risks. Figure 1 shows a UML activity diagram executed to move data from the OLTP system on-premises to the system in the cloud used by the marketing folks. In the model, you will see that activities happen in both partitions.

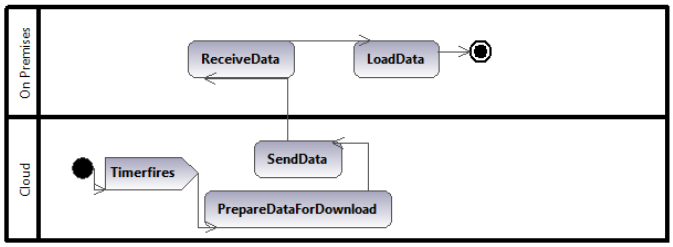

Figure 2 - Download Activity

Figure 2 presents a model that outlines the execution path when data is retrieved 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 lack of control of the quality of source data.

VII. PERTD MODEL

We developed the PERTD Model to assess better the risks associated with distributed applications. 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

TABLE 2 - DOWNLOAD ACTIVITY STRIDE MODEL

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

2. Execution

Activities that are susceptible to execution errors may fail due to ambiguous code requirements, which can lead 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 4 - UPLOAD ACTIVITY PERTD MODEL

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

TABLE 3 - DOWNLOAD ACTIVITY PERTD MODEL

Action	P	E	R	T	D
Timerfires		X			
PrepareDataForDownload		X			
SendData	X	X	X	X	
ReceieveData	X	X	X	X	
LoadData		X	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 due to data often being 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 the risks related to 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 the addition of domain-specific annotations, constraints, or semantics to UML elements, enhancing expressiveness and tailoring modeling for 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.

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 [9]. 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. CONCLUSIONS AND FUTURE WORKS

In this work, we provide a modeling methodology to handle issues in cloud software development related to NFRs in distributed systems. We show that in this work, we present a modeling methodology aimed at addressing issues related to NFRs in distributed systems during software development. Our PERTD model enables us to identify significantly more fine-grain risks associated with distributed systems. Additionally, we have enhanced the modeling of functional requirements by employing UML stereotypes to represent the NFRs identified in the PERTD model. Future work will incorporate code generation to mitigate the risks identified and modeled throughout this process. Utilizing our PERTD model makes identifying many more risks to a distributed system possible. We extended the modeling of functional requirements by using UML stereotypes to model the NFRs identified in the PERTD model. Implementation of cross-data source validation is provided to ensure data integrity. In our future work, we will add code generation to reduce the risks identified and modeled in the process.

REFERENCES

[1] 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.

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

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

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

[5] Object Management Group, "Unified Modeling Language: Supersturcture," 05 02 2007. [Online]. Available: http://www.omg.org/spec/UML/2.1.1/. [Accessed 20 Feb 2025].

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

[7] Object Management Group, "OMG Formally Released Versions of OCL," 02 2014. [Online]. Available: http://www.omg.org/spec/OCL/. [Accessed 20 02 2025].

[8] JSON.org, "Introducing JSON," 2024. [Online]. Available: https://www.json.org/json-en.html. [Accessed 20 Feb 2025].

[9] Open Collective, "JSON Schema," 2024. [Online]. Available: https://json-schema.org/. [Accessed 20 Feb 2025].

[10] Zapier Inc., "Automate without limits," 2024. [Online]. Available: https://zapier.com/. [Accessed 20 Feb 2025].

[11] Microsoft, "Power Automate," 2024. [Online]. Available: https://www.microsoft.com/en-us/power-platform/products/power-automate. [Accessed 20 Feb 2025].

[12] 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.