

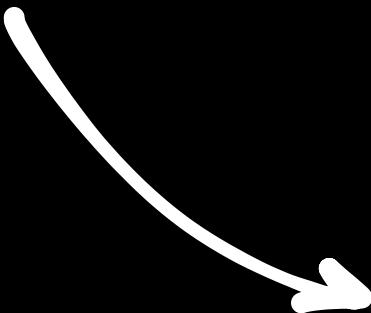
# Advanced Threat Modeling with GenAI

Vladimir Fedotov, EPAM

December 2023

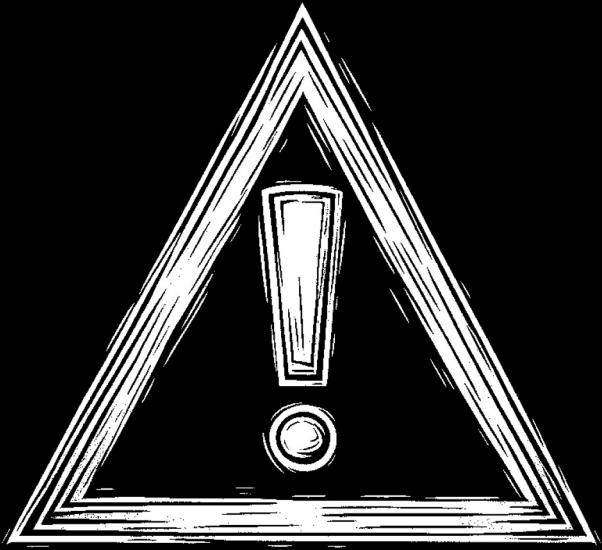


# Agenda



- Set up expectations
- Quick recap on Threat Modeling
- Well, what is Advanced Threat Modeling?
- Challenges?
- Can GenAI help?
- Enough of theory – show POC!

# DISCLAIMER

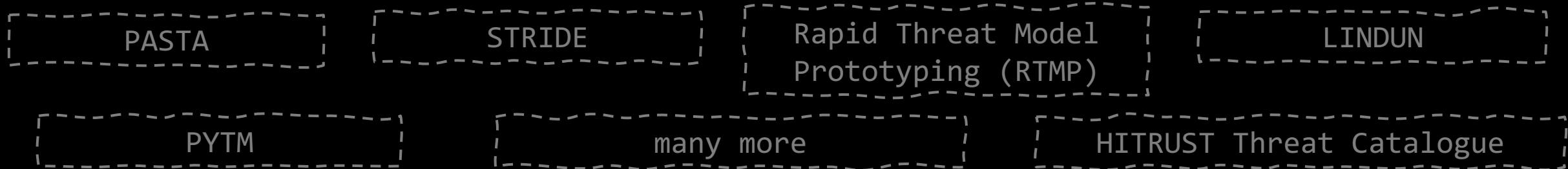


- The presentation doesn't aim to deliver a production-ready solution and focuses on the “What if...?” approach
- Uploading confidential data to LLM Providers may introduce security risks, so don't do it w/o evaluation
- Advanced is “modern and recently developed,” so it doesn't mean always good
- Looking forward to your critic

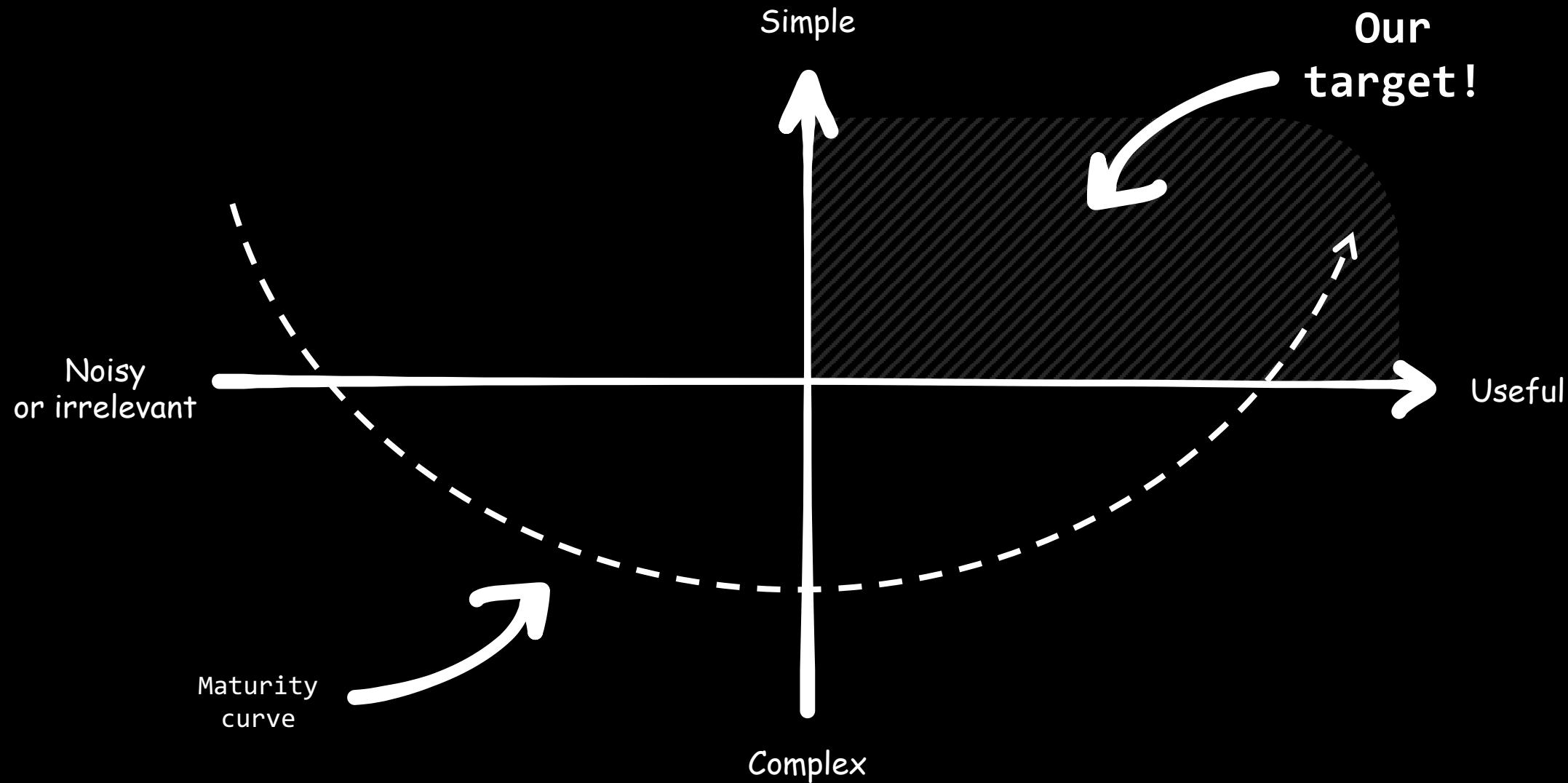
# What is Threat Modeling?

Threat modeling is a structured, repeatable process used to gain actionable insights into the security characteristics of a particular system.

1. What are we working on? → Application Decomposition
2. What can go wrong? → Threat Identification and Ranking
3. What are we going to do about it? → Response and Mitigations
4. Did we do a good enough job? → Review and Validation



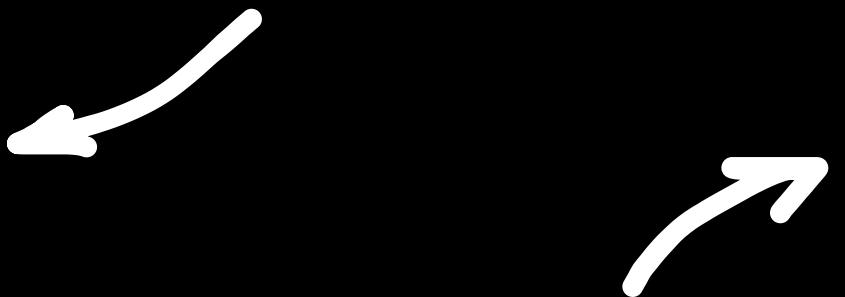
# Advanced Threat Modeling





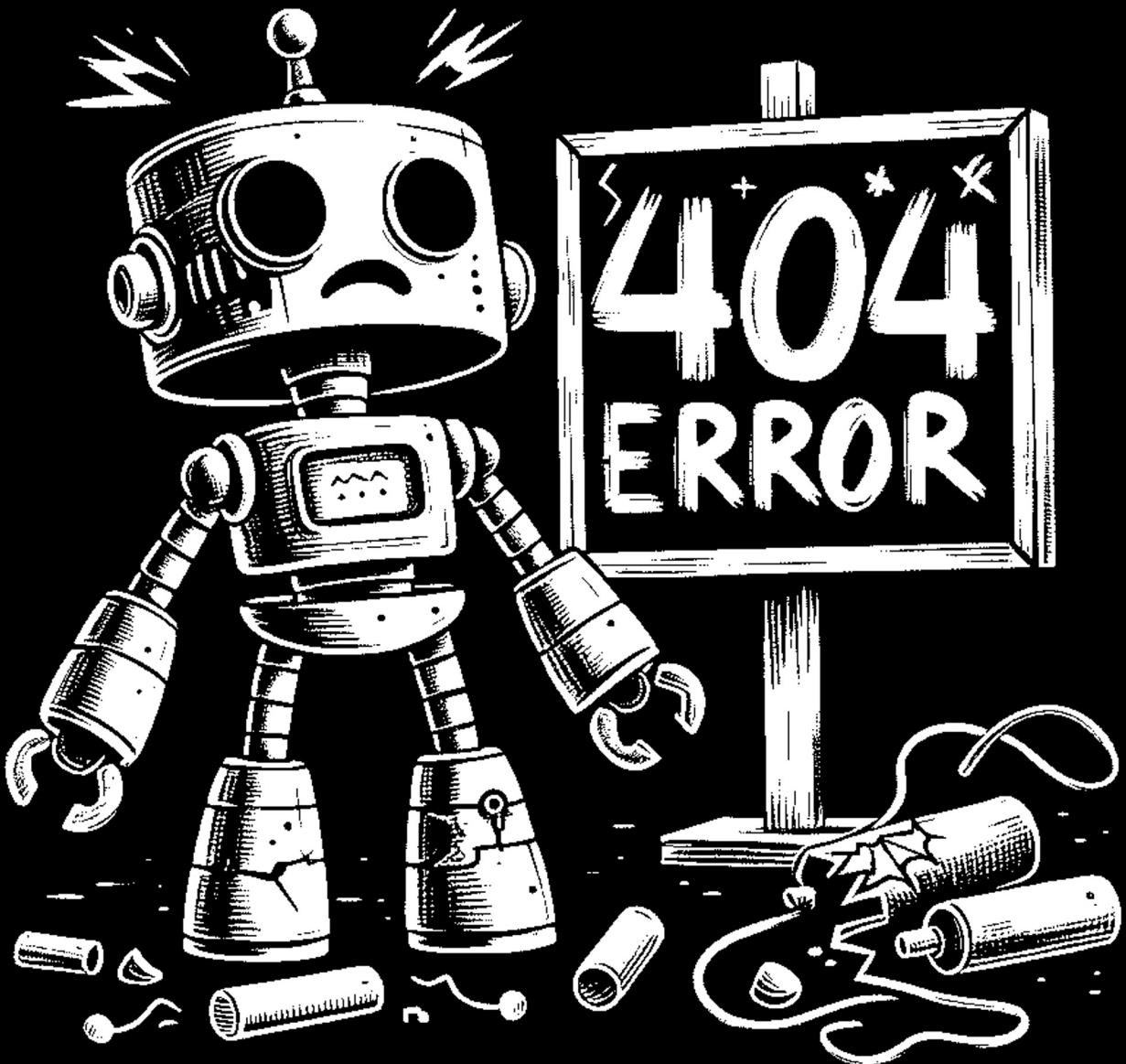
# Threat Modeling challenges

Limited time for comprehensive threat modeling due to agile development



The overwhelming output from threat modeling tools which lacks context





It cannot

*/\* just kidding \*/*

It will take  
your job :)



# Theories

## 1. Application Decomposition

- LLM provides a textual description of the architectural diagrams
- LLM can generate DFD based on diagrams or textual description

## 2. Threat Identification and Ranking

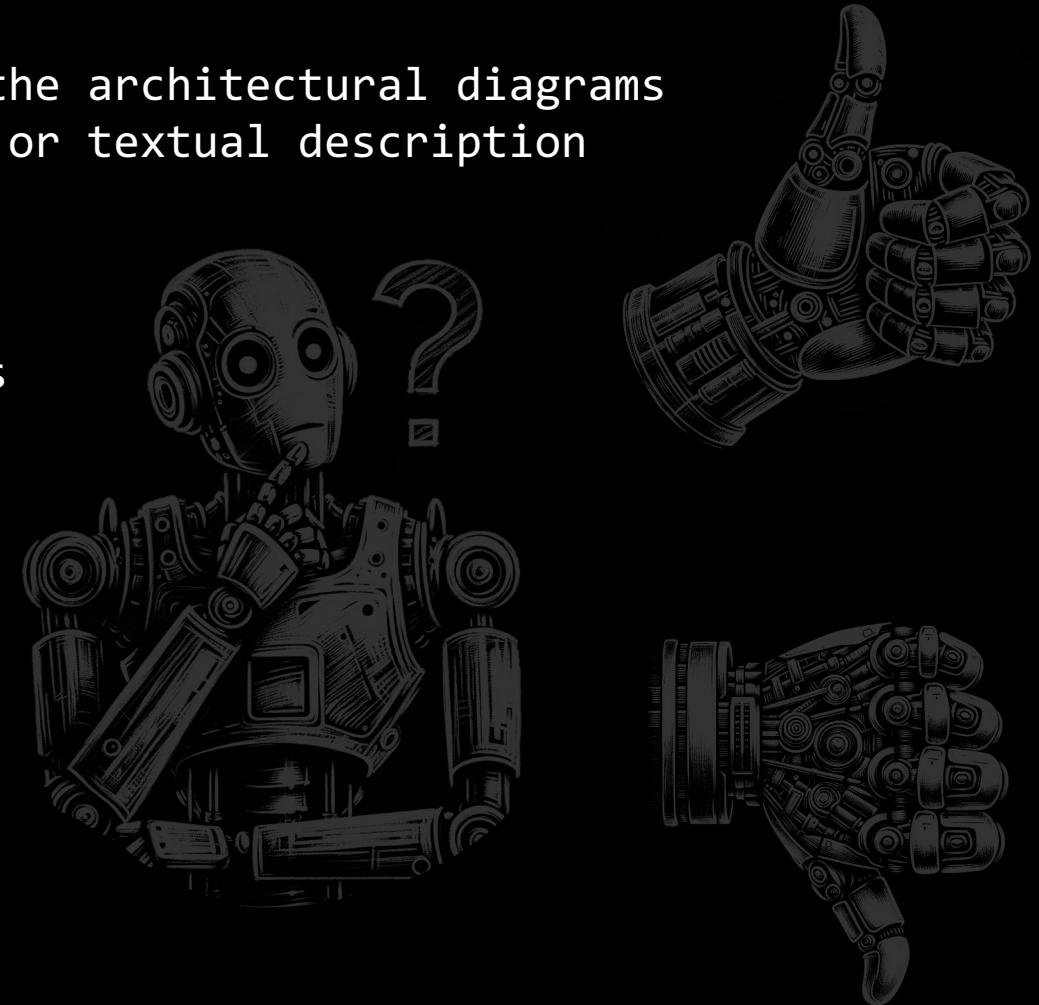
- LLM can triage autogenerated threats
- LLM can contextualize generated threats
- LLM can generate threats

## 3. Response and Mitigations

- LLM can generate mitigations
- LLM can contextualize mitigations

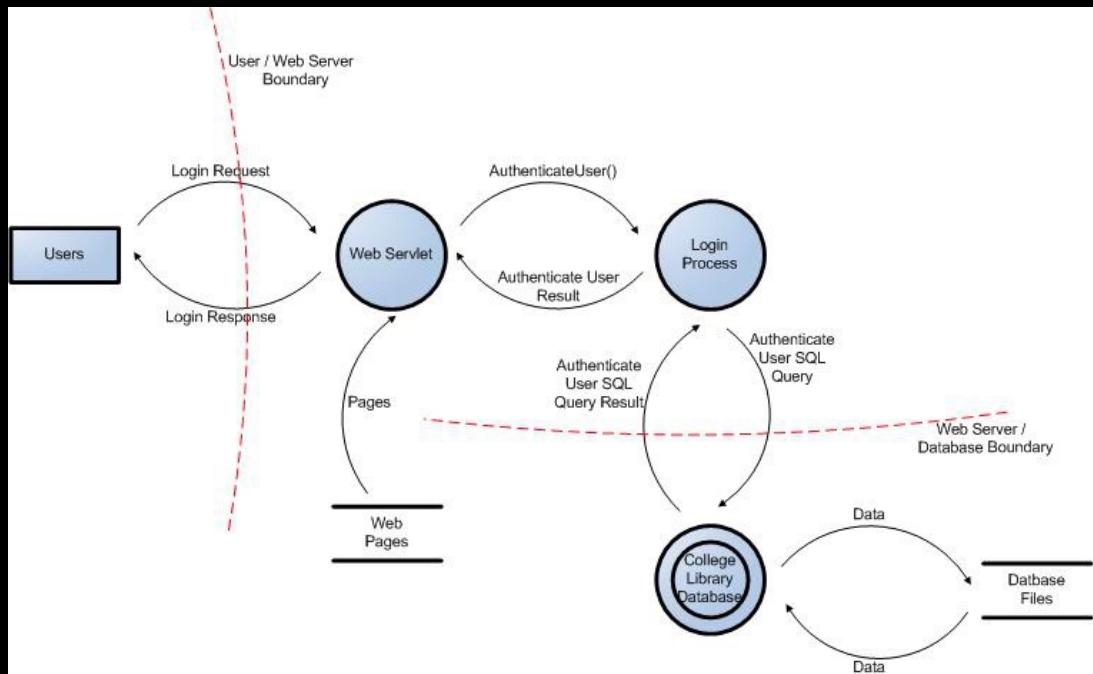
## 4. Review and Validation

- LLM can evaluate our threat models



# Describe DFD

LLM provides a textual description of the architectural diagrams



Legacy DFD -> GPT-4 -> Description

ChatGPT

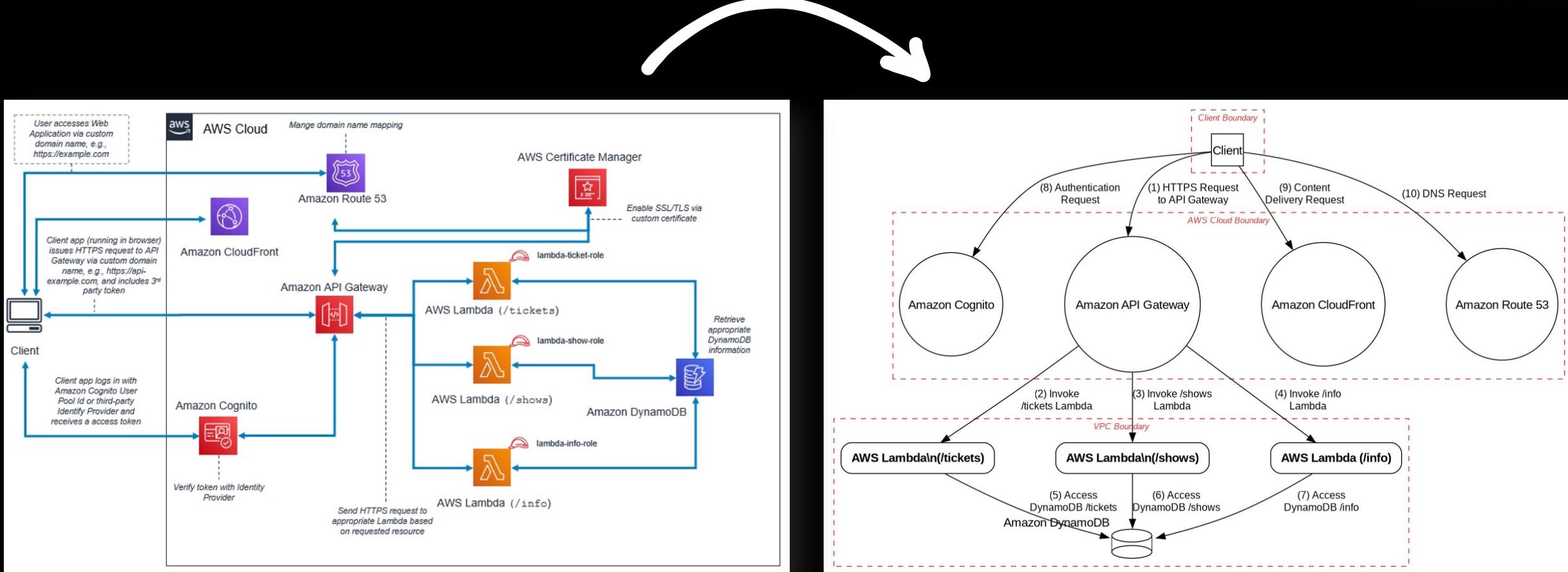
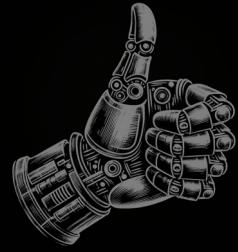
The diagram you've uploaded appears to be a flowchart representing the process of a user authentication system, typically used in the context of a web application. Here is a detailed description of its elements and data flows:

1. **Users** - Represented by a rectangle, indicating the start of the process. This likely refers to the end-users trying to access the system.
2. **User / Web Server Boundary** - A dotted line that demarcates the boundary between the user's domain and the web server's domain.
3. **Login Request** - An arrow pointing from the "Users" to the "Web Servlet", which signifies that users send a login request to the web server.
4. **Web Servlet** - A circle that acts as a node representing a component on the server that handles the login request.
5. **AuthenticateUser()** - An arrow pointing from the "Web Servlet" to the "Login Process", indicating a function or method call to authenticate the user.
6. **Authenticate User Result** - An arrow pointing back to the "Web Servlet" from the "Login Process", suggesting that the result of the authentication process is returned to the servlet.
7. **Login Process** - Another circle that likely represents a server-side process or function that handles user authentication.
8. **Authenticate User SQL Query** - An arrow from the "Login Process" to a point outside the diagram, indicating an SQL query is being made to authenticate the user.
9. **Authenticate User SQL Query Result** - An arrow pointing back towards the "Login Process", implying that the results of the SQL query are returned to the login process.
10. **Web Server / Database Boundary** - A dotted line that separates the web server's domain from the database domain.
11. **College Library Database** - A circle representing the database where user credentials might be stored.
12. **Data** - Two bi-directional arrows between the "College Library Database" and "Database Files", indicating data exchange.
13. **Database Files** - A rectangle that likely represents the physical or logical files where the database is stored.
14. **Login Response** - An arrow pointing back to the "Users" from the "Web Servlet", indicating that a response to the login request is sent back to the users.
15. **Pages** - An arrow from the "Web Servlet" to "Web Pages", indicating that based on the authentication result, the web servlet serves web pages.
16. **Web Pages** - A rectangle that probably stands for the actual web pages that are served to the user upon successful authentication.

The flowchart essentially outlines the steps taken when a user attempts to log into a web application, from sending a login request to receiving a response after authentication, with interactions with a database to verify user credentials.

# Create DFD

LLM can generate DFD based on diagrams or textual description



AWS Diagram -> GPT-4 -> PyTM code -> DFD + Threats

LLM can triage autogenerated threats



# Triage threats



Assets	
Name	Amazon Cognito
Description	
In Scope	True
Type	Process
Finding Count	34
<b>Threats</b>	
<ul style="list-style-type: none"> <li>▶ 1 – INP02 – Overflow Buffers</li> <li>▶ 2 – AA01 – Authentication Abuse/Bypass</li> <li>▶ 3 – DE02 – Double Encoding</li> <li>▶ 4 – AC01 – Privilege Abuse</li> <li>▶ 5 – INP07 – Buffer Manipulation</li> <li>▶ 6 – DO01 – Flooding</li> <li>▶ 7 – DO02 – Excessive Allocation</li> <li>▶ 8 – INP08 – Format String Injection</li> <li>▶ 9 – INP12 – Client-side Injection-induced Buffer Overflow</li> <li>▶ 10 – INP13 – Command Delimiters</li> <li>▶ 11 – INP14 – Input Data Manipulation</li> <li>▶ 12 – CR03 – Dictionary-based Password Attack</li> <li>▶ 13 – AA02 – Principal Spoof</li> <li>▶ 14 – INP20 – iFrame Overlay</li> <li>▶ 15 – INP23 – File Content Injection</li> <li>▶ 16 – AC12 – Privilege Escalation</li> <li>▶ 17 – AC13 – Hijacking a privileged process</li> <li>▶ 18 – AC14 – Catching exception throw/signal from privileged block</li> <li>▶ 19 – INP24 – Filter Failure through Buffer Overflow</li> <li>▶ 20 – INP25 – Resource Injection</li> <li>▶ 21 – INP26 – Code Injection</li> <li>▶ 22 – INP27 – XSS Targeting HTML Attributes</li> <li>▶ 23 – INP28 – XSS Targeting URI Placeholders</li> <li>▶ 24 – INP29 – XSS Using Doubled Characters</li> <li>▶ 25 – INP30 – XSS Using Invalid Characters</li> <li>▶ 26 – INP31 – Command Injection</li> <li>▶ 27 – INP32 – XML Injection</li> <li>▶ 28 – INP33 – Remote Code Inclusion</li> <li>▶ 29 – INP35 – Leverage Alternate Encoding</li> <li>▶ 30 – AC15 – Schema Poisoning</li> <li>▶ 31 – AC18 – Session Hijacking - ClientSide</li> <li>▶ 32 – INP41 – Argument Injection</li> <li>▶ 33 – AC20 – Reusing Session IDs (aka Session Replay) - ClientSide</li> <li>▶ 34 – AC21 – Cross Site Request Forgery</li> </ul>	

Threat ID and Name	Applicability	Justification
INP02 – Overflow Buffers	Not Applicable	AWS Cognito is managed by AWS and uses languages and architectures that inherently manage buffer overflows, making traditional overflow attacks less relevant.
AA01 – Authentication Abuse/Bypass	Partially Applicable	While AWS Cognito provides robust authentication mechanisms, potential misconfigurations or weak user-defined security policies might lead to authentication abuse.
DE02 – Double Encoding	Partially Applicable	AWS Cognito has measures to prevent XSS, but double encoding attacks might bypass these measures if not properly configured or if client-side vulnerabilities exist.
AC01 – Privilege Abuse	Partially Applicable	AWS Cognito's role-based access control can mitigate this risk, but misconfigurations or policy oversights could allow privilege abuse.
INP07 – Buffer Manipulation	Not Applicable	Similar to INP02, the managed nature of AWS Cognito and the use of safe programming practices minimize the risk of buffer manipulation.
DO01 – Flooding	Applicable	AWS Cognito could be subject to flooding attacks, impacting availability, though AWS has mitigation measures like scaling and rate limiting.
DO02 – Excessive Allocation	Partially Applicable	AWS infrastructure can scale to mitigate some impact, but poorly configured Cognito instances may still be vulnerable to resource exhaustion.
INP08 – Format String Injection	Not Applicable	Format string vulnerabilities are typically associated with lower-level programming languages, which are not directly applicable in the context of AWS Cognito.
INP12 – Client-side Injection-induced BO	Not Applicable	AWS Cognito is not directly vulnerable as it's a managed service, but client applications interacting with it could be, depending on their implementation.
INP13 – Command Delimiters	Not Applicable	Command delimiter issues are generally relevant to systems where command-line processing occurs, which is not a direct concern for AWS Cognito.
INP14 – Input Data Manipulation	Partially Applicable	AWS Cognito provides validation mechanisms, but incorrect configurations or insufficient validation in client applications can lead to issues.
CR03 – Dictionary-based Password Attack	Partially Applicable	Cognito's security depends on password policies. Weak policies might make it vulnerable to dictionary attacks.

# Triage threats (2)



## ▼ 16 – AC12 – Privilege Escalation

### Targeted Element

Amazon Cognito

### Severity

High

### Example Instances

The software does not properly assign, modify, track, or check privileges for an actor, creating an unintended sphere of control for that actor. As a result, the program is indefinitely operating in a raised privilege state, possibly allowing further exploitation to occur.

### Mitigations

Very carefully manage the setting, management, and handling of privileges. Explicitly manage trust zones in the software. Follow the principle of least privilege when assigning access rights to entities in a software system. Implement separation of privilege - Require multiple conditions to be met before permitting access to a system resource.

### References

<https://capec.mitre.org/data/definitions/233.html>, <http://cwe.mitre.org/data/definitions/269.html>

### Threat Description:

- **Privilege Escalation:** The threat involves improper assignment, modification, tracking, or checking of privileges for an actor, leading to an unintended sphere of control. This can result in the software operating with elevated privileges indefinitely, potentially allowing further exploitation.

### Targeted Element:

- **Amazon Cognito:** AWS Cognito is a service that provides authentication, authorization, and user management for web and mobile applications. It is responsible for handling user identities and granting the correct level of access to users.

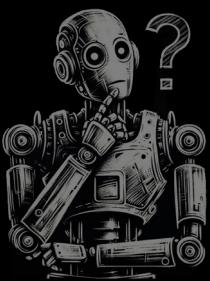
### Analysis:

1. **Default Configuration and Usage:** By default, AWS Cognito is designed with security in mind. It handles user authentication and authorization, ensuring that users have the appropriate access rights. If properly configured, it should prevent privilege escalation by design.
2. **Customization and Misconfiguration:** The risk of privilege escalation becomes significant if AWS Cognito is not correctly configured. For instance, assigning excessive privileges to a user or role, or not correctly managing user groups and permissions, can lead to privilege escalation issues.
3. **Mitigations in Place:** AWS Cognito supports the principle of least privilege and allows for the explicit management of trust zones and privileges. However, the effectiveness of these mitigations depends on how they are implemented in the specific context of the application.

### Applicability:

- **Partially Applicable:** Based on the analysis, the threat of Privilege Escalation (AC12) is partially applicable to AWS Cognito in the context of a ticketing application. While Cognito inherently supports mechanisms to prevent privilege escalation, the risk exists if it is not correctly configured or if the application's custom logic interacting with Cognito introduces vulnerabilities.

In conclusion, the threat is not inherent to AWS Cognito itself but rather depends on how it is implemented and managed within the application's architecture. Proper configuration, regular security audits, and adherence to best practices in user management are critical to mitigating this threat.



# Contextualization

```
{  
  "description": "Authentication Abuse/ByPass",  
  
  "details": "An attacker obtains unauthorized access to an application, service or device either through knowledge of the inherent weaknesses of an authentication mechanism, or by exploiting a flaw in the authentication scheme's implementation. In such an attack an authentication mechanism is functioning but a carefully controlled sequence of events causes the mechanism to grant access to the attacker. This attack may exploit assumptions made by the target's authentication procedures, such as assumptions regarding trust relationships or assumptions regarding the generation of secret values. This attack differs from Authentication Bypass attacks in that Authentication Abuse allows the attacker to be certified as a valid user through illegitimate means, while Authentication Bypass allows the user to access protected material without ever being certified as an authenticated user. This attack does not rely on prior sessions established by successfully authenticating users, as relied upon for the Exploitation of Session Variables, Resource IDs and other Trusted Credentials attack patterns.",  
  
  "example": "An adversary that has previously obtained unauthorized access to certain device resources, uses that access to obtain information such as location and network information.",  
  
  "mitigations": "Use strong authentication and authorization mechanisms. A proven protocol is OAuth 2.0, which enables a third-party application to obtain limited access to an API."  
}
```

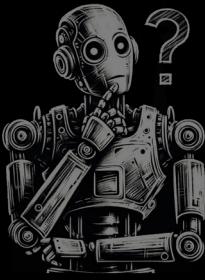
LLM can contextualize generated threats  
LLM can contextualize mitigations

```
{  
  "description": "Authentication Abuse/ByPass in Spring Boot Web Application",  
  
  "details": "In this context, an attacker targets the Spring Boot web application due to its inadequate authentication controls. They may exploit weak or absent authentication mechanisms to impersonate legitimate users, gaining unauthorized access to application functionalities. This could include accessing or modifying sensitive user comments or exploiting the application's functionalities for malicious purposes. The threat is exacerbated by the application's failure to implement strong authentication schemes, leaving endpoints like '/comments/create' and '/comments/update' vulnerable to unauthorized access and manipulation. This attack can lead to data breaches, injection of malicious content, and compromise of user trust.",  
  
  "example": "An adversary exploits weak authentication in the Spring Boot application to gain unauthorized access. They target the '/comments/create' endpoint to post malicious comments, using stolen or guessed credentials. The attacker leverages this access to inject harmful content or scripts, disrupting the application's normal operation.",  
  
  "mitigations": "Implement strong, multi-factor authentication mechanisms. Integrate Spring Security with OAuth 2.0 for robust API protection. Regularly update Spring Boot and its dependencies to patch known vulnerabilities. Enforce strict input validation on all endpoints, especially those handling user-generated content, to prevent injection attacks."  
}
```

PyTM Asset + Description + Finding -> GPT-4 -> Finding in context of description

# TM with GPT-4

LLM can generate threats  
LLM can generate mitigations



ID	STRIDE Category	Threat Name	Description and Impact	Impacted Component	Suggested Mitigation
1	Spoofing	Unauthorized Access via Spoofing	An attacker might spoof a user's identity to access the application, leading to unauthorized access to sensitive data or operations.	AWS Lambda, API Gateway, Amazon Cognito	Implement multi-factor authentication in Amazon Cognito. Ensure token validation in Lambda functions is robust.
2	Tampering	Data Tampering in DynamoDB	An attacker might tamper with data in DynamoDB, leading to corrupted data for tickets or shows.	Amazon DynamoDB	Use strong access control policies. Enable DynamoDB Streams to monitor and audit changes.
3	Repudiation	Lack of Audit Trails	Without proper logging, malicious activities or errors might go unnoticed, and users can deny actions they performed.	AWS Lambda, API Gateway	Implement comprehensive logging and monitoring using AWS CloudTrail and CloudWatch.
4	Information Disclosure	Sensitive Data Exposure	Sensitive data like user information or payment details might be exposed due to insecure API or database access.	AWS Lambda, Amazon DynamoDB	Encrypt sensitive data in transit and at rest. Implement fine-grained access controls.
5	Denial of Service	DoS via API Overload	The application could be susceptible to a DoS attack, where an attacker sends numerous requests to overload the server.	AWS Lambda, API Gateway, Amazon CloudFront	Use AWS WAF and rate limiting in API Gateway. Utilize CloudFront's DDoS protection capabilities.
6	Elevation of Privilege	Privilege Escalation via Lambda	An attacker could exploit a vulnerability in a Lambda function to gain higher privileges.	AWS Lambda	Follow the principle of least privilege in IAM roles for Lambda functions. Regularly update and audit Lambda functions for vulnerabilities.
7	Spoofing	Domain Spoofing	An attacker could use a similar domain to trick users, leading to phishing attacks.	Amazon Route 53, AWS Certificate Manager	Use DNSSEC in Route 53 to protect against DNS spoofing. Ensure SSL/TLS certificates are valid and up-to-date.
8	Tampering	Code Injection in Lambda Functions	An attacker might inject malicious code into Lambda functions, altering their behavior.	AWS Lambda	Validate all inputs rigorously. Use static code analysis tools to detect vulnerabilities.
9	Information Disclosure	Insecure Direct Object References	Directly referencing objects in APIs could lead to unauthorized access to data.	AWS Lambda, API Gateway	Implement strong input validation and access control checks in Lambda functions.
10	Denial of Service	Resource Exhaustion in Lambda	Lambda functions could be overwhelmed by high request volumes, leading to service unavailability.	AWS Lambda	Implement concurrency limits and scaling policies for Lambda functions.
11	Repudiation	Log Forgery	An attacker could manipulate logs to cover tracks or create false evidence.	AWS CloudTrail, CloudWatch	Ensure log integrity and security. Restrict access to logs.

# GPT review of PYTM

LLM can evaluate our threat models



This threat model represents an AWS cloud architecture for a web application using services like API Gateway, Lambda, DynamoDB, etc.

**Dataflow Diagram - Level 0 DFD**

The diagram illustrates the data flow between the Client, Amazon Cognito, Amazon API Gateway, Amazon CloudFront, and Amazon Route 53. The Client initiates several requests:

- (1) HTTPS Request to API Gateway
- (2) Invoke /tickets Lambda
- (3) Invoke /shows Lambda
- (4) Invoke /info Lambda
- (5) Access DynamoDB /tickets
- (6) Access DynamoDB /shows
- (7) Access DynamoDB /info
- (8) Authentication Request
- (9) Content Delivery Request
- (10) DNS Request

**Data Flows**

Name	Description	Sink	Source	Is Response	In Scope	Finding Count
H	C	P	A	F	T	7

**Threats**

- 165 – DE01 – Interception
- 166 – AC05 – Content Spoofing
- 167 – DE03 – Sniffing Attacks
- 168 – CR06 – Communication Channel Manipulation
- 169 – CR08 – Client-Server Protocol Manipulation
- 170 – DS06 – Data Leak
- 171 – DR01 – Unprotected Sensitive Data

**Dataflows**

Name	From	To	Data	Protocol	Port
(1) HTTPS Request to API Gateway	Client	Amazon API Gateway	Access Token, User Authentication Data	HTTPS	-1
(2) Invoke /tickets Lambda	Amazon API Gateway	AWS Lambda (/tickets)	Access Token, Payment Transaction Data, Ticketing Data	AWS SDK	-1
(3) Invoke /shows Lambda	Amazon API Gateway	AWS Lambda (/shows)	Access Token, Show Data	AWS SDK	-1
(4) Invoke /info Lambda	Amazon API Gateway	AWS Lambda (/info)	Access Token, Personal Identifiable Information (PII)	AWS SDK	-1
(5) Access DynamoDB /tickets	AWS Lambda (/tickets)	Amazon DynamoDB	Payment Transaction Data, Ticketing Data	AWS SDK	-1
(6) Access DynamoDB /shows	AWS Lambda (/shows)	Amazon DynamoDB	Show Data	AWS SDK	-1
(7) Access DynamoDB /info	AWS Lambda (/info)	Amazon DynamoDB	Personal Identifiable Information (PII)	AWS SDK	-1
(8) Authentication Request	Client	Amazon Cognito	User Authentication Data	HTTPS	-1
(9) Content Delivery Request	Client	Amazon CloudFront		HTTPS	-1
(10) DNS Request	Client	Amazon Route 53		DNS	-1

**Data Dictionary**

Name	Description	Classification	Carried	Processed
Access Token	Used for authenticating and authorizing user actions within the application.	SECRET	HTTPS Request to API Gateway AWS Lambda (/info) AWS Lambda (/shows) AWS Lambda (/tickets) Invoke /tickets Lambda Invoke /shows Lambda Invoke /info Lambda	Amazon API Gateway AWS Lambda (/info) AWS Lambda (/shows) AWS Lambda (/tickets) Amazon API Gateway Client
User Authentication Data	Includes usernames, passwords, and authentication tokens handled by Amazon Cognito.	SECRET	Authentication Request HTTPS Request to API Gateway	Amazon Cognito Client

**Threats**

- 172 – DE01 – Interception
- 173 – AC05 – Content Spoofing
- 174 – DE03 – Sniffing Attacks
- 175 – CR06 – Communication Channel Manipulation
- 176 – CR08 – Client-Server Protocol Manipulation
- 177 – DS06 – Data Leak
- 178 – DR01 – Unprotected Sensitive Data

**Invoke /shows Lambda**

Name	Description
Description	Routes show-related requests to the /shows Lambda/AWS Lambda (/shows)
Sink	Lambda/AWS Lambda (/shows)
Source	Process(Amazon API Gateway)
Is Response	False
In Scope	True
Finding Count	7

**Invoke /info Lambda**

Name	Description
Description	Routes user profile and help information requests to the /info Lambda function.
Sink	Lambda/AWS Lambda (/info)
Source	Process(Amazon API Gateway)
Is Response	False
In Scope	True
Finding Count	7

## ChatGPT

The threat modeling report focuses on an AWS cloud architecture for a web application, detailing data flows, data dictionaries, actors, boundaries, and assets. It includes a comprehensive analysis of various threats, each with a targeted element, severity rating, example instances, mitigations, and references. Key elements include Amazon Cognito, Lambda, DynamoDB, and API Gateway. The threats are categorized into areas like buffer overflow, authentication abuse, and SQL injection, with detailed mitigation strategies and external references for each.

For improvements:

- Clarity and Detail in Descriptions:** The descriptions of data flows, actors, and boundaries could be more detailed, offering clearer insight into the system's architecture and interaction between components.
- Risk Assessment and Prioritization:** Prioritize threats based on their likelihood and potential impact. This helps focus on the most critical areas first.
- Real-World Scenarios:** Include more real-world examples or case studies to illustrate how these threats might manifest in practical scenarios.
- Regular Updates:** Regularly update the report to reflect new threats and vulnerabilities as they emerge.
- Visual Aids:** Enhance the report with more diagrams or visual aids for better understanding of complex architectures and data flows.
- Compliance and Standards:** Reference relevant security standards and compliance requirements to ensure the architecture meets industry best practices.
- User and Role Definitions:** Expand on the roles and responsibilities of different actors within the system, including users, admins, and external entities.
- Performance Impact of Mitigations:** Discuss the potential performance impact of each mitigation strategy, helping to balance security with efficiency.
- Feedback Mechanism:** Implement a feedback mechanism to continuously improve the threat model based on user and expert input.
- Automated Testing and Tools:** Suggest automated tools and practices for continuous monitoring and testing of the security measures.



# STRIDE-GPT TM

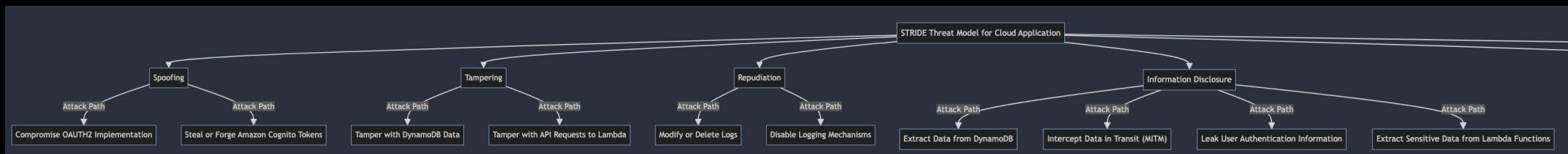
LLM can generate threats  
LLM can generate mitigations

Threat Model		
Threat Type	Scenario	Potential Impact
Spoofing	An attacker could create a fake OAuth2 provider to mimic the legitimate authentication provider and trick users into entering their credentials.	Users' credentials and sensitive data could be compromised, leading to unauthorized access and potential identity theft.
Tampering	Without proper input validation on the Lambda functions, an attacker could submit malicious data resulting in stored XSS or SQL-like injection attacks in DynamoDB.	Attackers could manipulate ticket or show data, disrupt service operations, or extract sensitive information from the database.
Repudiation	Due to lack of proper logging mechanisms, a user or an attacker might deny performing transactions or actions in the app, making it difficult to prove malfeasance.	This makes it difficult to track and audit user actions, complicating incident response and forensic investigations.
Information Disclosure	If API Gateway or Lambda functions aren't configured to handle errors properly, they could leak sensitive information through error messages.	This information could be used to facilitate further attacks, leading to a breach of confidential information.
Denial of Service	An attacker might exploit the serverless architecture by flooding the Lambda functions with high volumes of traffic, causing them to scale rapidly and incur high costs for the application owner.	This could render the application unusable for legitimate users and impose significant financial costs.
Elevation of Privilege	Without privileged access management, an attacker could exploit a misconfiguration or vulnerability in the application to gain higher privileges than intended.	This would potentially give the attacker the same level of access as admins or other high-level user roles, enabling them to alter application logic, access sensitive data or disrupt service availability.

Improvement Suggestions	
<ul style="list-style-type: none"> <li>Specify the mechanisms in place for validating and sanitizing input data to prevent tampering and injection attacks.</li> <li>Describe the logging and monitoring solutions utilized for tracking user activities and identifying malicious actions.</li> <li>Provide details on error handling practices to understand how sensitive information leaks are prevented.</li> <li>Clarify the implementation of rate limiting or other protective measures against DoS attacks.</li> <li>Detail the access control measures in place, particularly how privileges are managed and what privileged access management system is used if any.</li> </ul>	

Threat Type	Scenario	Suggested Mitigation(s)
Spoofing	An attacker could create a fake OAuth2 provider to mimic the legitimate authentication provider and trick users into entering their credentials.	<ul style="list-style-type: none"> <li>Implement multi-factor authentication (MFA) to reduce the risk of credential theft.</li> <li>Use certificate pinning to ensure that the application communicates only with the genuine authentication provider.</li> <li>Educate users on phishing tactics and how to identify legitimate sign-in pages.</li> </ul>
Tampering	Without proper input validation on the Lambda functions, an attacker could submit malicious data resulting in stored XSS or SQL-like injection attacks in DynamoDB.	<ul style="list-style-type: none"> <li>Perform rigorous input validation and sanitization on any data received by Lambda functions.</li> <li>Use parameterized queries or prepared statements with DynamoDB to prevent injection attacks.</li> <li>Implement least privilege access controls on the Lambda functions to limit the potential damage of a successful attack.</li> </ul>
Repudiation	Due to lack of proper logging mechanisms, a user or an attacker might deny performing transactions or actions in the app, making it difficult to prove malfeasance.	<ul style="list-style-type: none"> <li>Implement comprehensive logging of all user actions and keep secure audit trails.</li> <li>Use timestamping and securely store logs to prevent tampering.</li> <li>Implement non-repudiation mechanisms such as digital signatures where critical transactions are involved.</li> </ul>
Information Disclosure	If API Gateway or Lambda functions aren't configured to handle errors properly, they could leak sensitive information through error messages.	<ul style="list-style-type: none"> <li>Implement proper error handling that does not expose stack traces or sensitive information to the users.</li> <li>Configure API Gateway and Lambda functions to return generic error messages.</li> <li>Regularly review and update access policies to enforce principle of least privilege.</li> </ul>
Denial of Service	An attacker might exploit the serverless architecture by flooding the Lambda functions with high volumes of traffic, causing them to scale rapidly and incur high costs for the application owner.	<ul style="list-style-type: none"> <li>Implement rate limiting and throttling on API Gateway.</li> <li>Use AWS WAF (Web Application Firewall) to filter out malicious traffic patterns.</li> <li>Set budget alerts and scaling limits to control cost exposure during an attack.</li> </ul>
Elevation of Privilege	Without privileged access management, an attacker could exploit a misconfiguration or vulnerability in the application to gain higher privileges than intended.	<ul style="list-style-type: none"> <li>Regularly audit and update IAM roles and policies to enforce the principle of least privilege.</li> <li>Use role-based access control (RBAC) to manage user permissions effectively.</li> <li>Incorporate automated tools to monitor for misconfigurations or unusual activity patterns indicating privilege escalation attempts.</li> </ul>



# Theories review

## 1. Application Decomposition

- LLM provides a textual description of the architectural diagrams +
- LLM can generate DFD based on diagrams or textual description +

## 2. Threat Identification and Ranking

- LLM can generate threats +/-
- LLM can triage autogenerated threats +
- LLM can contextualize generated threats +

## 3. Response and Mitigations

- LLM can generate mitigations +/-
- LLM can contextualize mitigations +

## 4. Review and Validation

- LLM can evaluate our threat models +/-

# Conclusion

1. GenAI can help with the challenges
2. Tools, processes, people are needed
3. There are challenges in applying GenAI itself
4. Feature is coming...

# Thank you !

## Advanced Threat Modeling with GenAI

Vladimir Fedotov

<https://www.linkedin.com/in/vafedotov/>

December 2023

