Threat Modeling, Automation and AI

The human side

About Me

David Robert

- Founder of SecurityArchitect.AI - Application Security Review platform
- 20+ years primarily in application security, previous roles include Principal Application Security Engineer in Amazon (Secure AI Foundations, Payments, Robotics).
- Passionate about scaling application security reviews while keeping a high bar.

Agenda

- Threat Modeling
- Discuss automation techniques
- Explore how LLMs can help
- Some demos

Disclaimers

- I am a security engineer, not an AI Specialist or Data scientist.
- The threat models, prompt, and other examples are intentionally simplified.


Automatic Threat Modeling in minutes
Context diagram
Code analysis
Full automation
Deep dive with
Context Diagram

THREAT MODELING MANIFESTO

What is threat modeling?

Threat modeling is analyzing representations of a system to highlight concerns about security and privacy characteristics.

At the highest levels, when we threat model, we ask four key questions:

1. What are we working on?
2. What can go wrong?
3. What are we going to do about it?
4. Did we do a good enough job?

Why threat model?

When you perform threat modeling, you begin to recognize what can go wrong in a system. It also allows you to pinpoint design and implementation issues that require mitigation, whether it is early in or throughout the lifetime of the system. The output of the threat model, which are known as threats, informs decisions that you might make in subsequent design, development, testing, and post-deployment phases.

Threat Modeling vs Vulnerabilities detection

Aspect	Threat Modeling	Vulnerability Detection
What it asks	*"What could go wrong?"*	*"What is already wrong?"*
Stage	Early (design / architecture phase)	Later (code / implementation phase)
Scope	System-level risks, trust boundaries, data flow	Code flaws, misconfigurations, exposed secrets
Tools	DFDs, STRIDE, attack trees, graph modeling	SAST, DAST, IaC scanners, linters
Output	Potential threats	Actual, actionable findings

Some bold statements

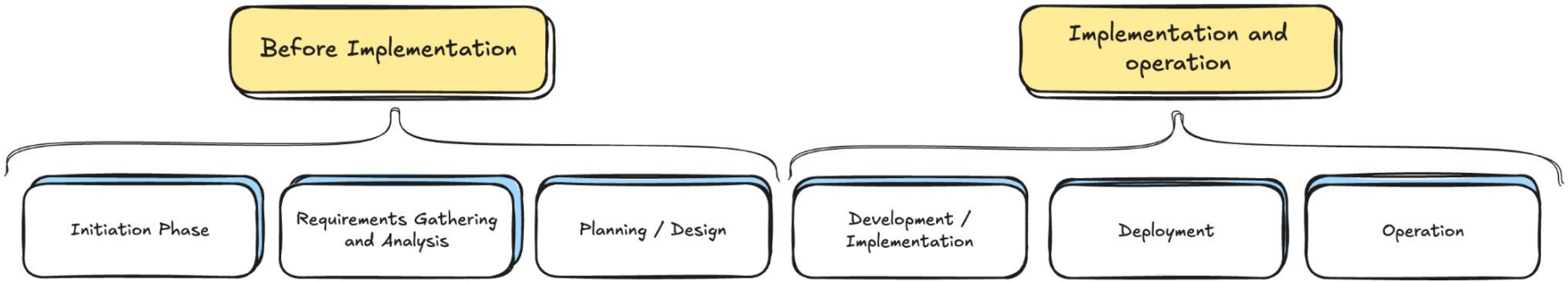
The main reason software engineers introduce vulnerabilities is the lack of tailored security requirements.

The effective method to create these security requirements is Threat Modeling

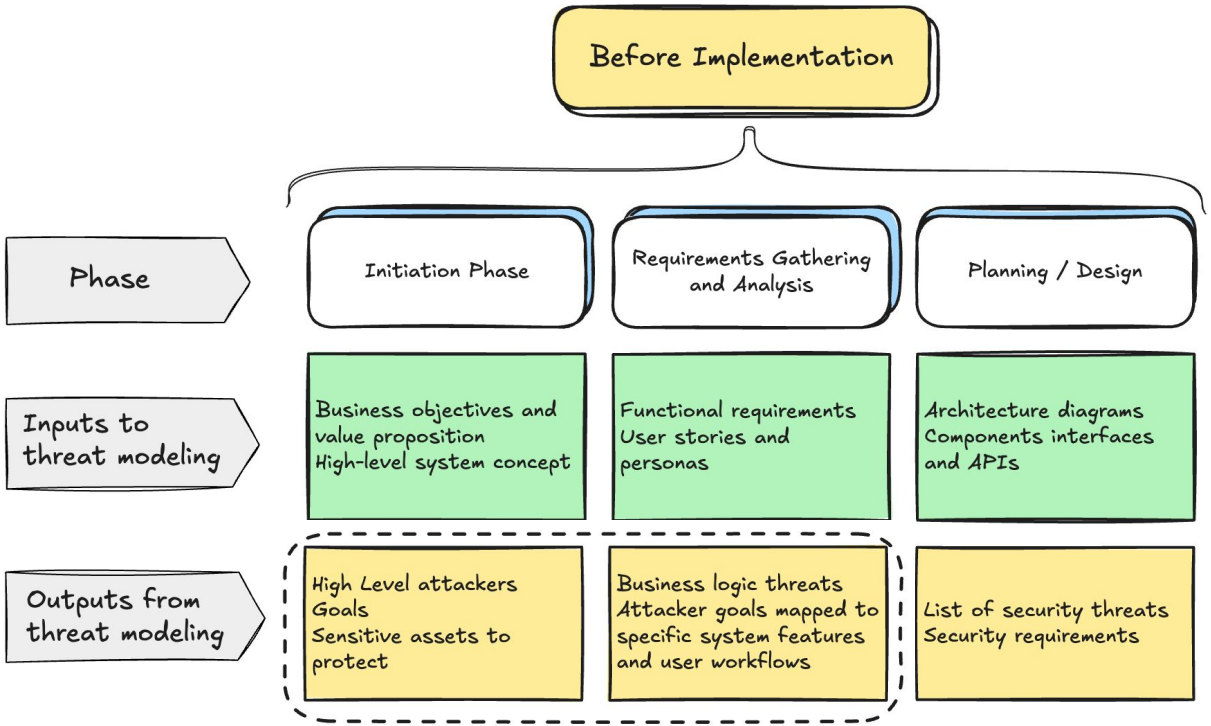
Understanding the applications' business logic, features is Critical

It's not about shifting left security, it is about putting security where it belongs.

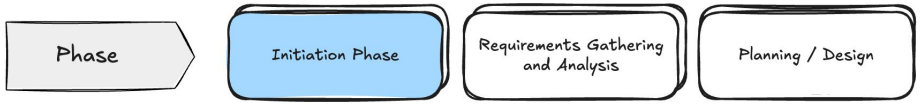
Threat modeling in different phases



Threat modeling during design

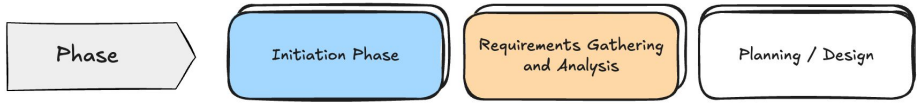


Initiation



“We propose building a system that analyzes camera footage to track customer movement and shopping behaviors in real-time. The platform will generate heatmaps, count foot traffic, and provide actionable insights through dashboards, enabling store managers to optimize layouts, improve staffing, and enhance customer experience while driving sales growth.”	Attacker Goals - Obtain raw camera footage - Obtain customer behavioral profiles - Tamper with real-time analytics

Requirements



User stories:

As a store manager, I can login to the Web UI to manage the application.

As a store manager, I can access live streams via the Web UI

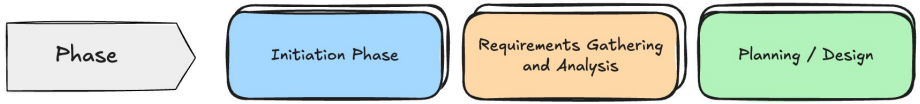
As a store manager, I can share raw recordings to other stores via the share feature in the Web UI

As a store manager, I can generate and export heatmap reports showing customer movement patterns

Attacker Goals

- Obtain raw camera footage
  - Via the live stream feature
  - Via the recording sharing feature
- Obtain customer behavioral profiles
  - Via the export feature
- Unauthorized access to store manager account
- Tamper with real-time analytics

Design



Design document, Architecture Diagram:

List of components (e.g.)

- IP Cameras
- Vision System
- API Gateway
- Database
- …

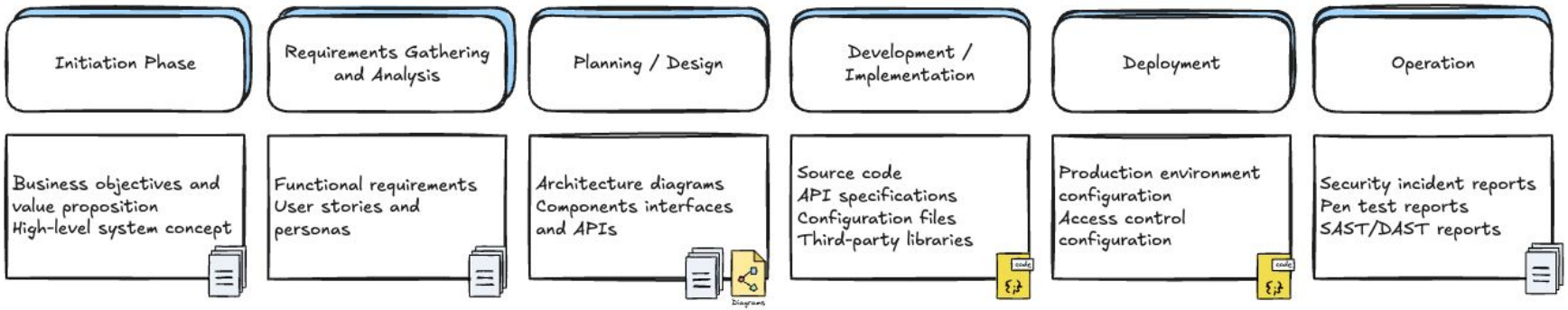
List of APIs
Authentication mechanisms details

Attacker Goals
- Obtain raw camera footage
  - Via the live stream feature
    - Improper auth in GET /api/v1/stream API
    - Web client vulnerability in React Web UI
  - Via the recording sharing feature
    - Improper validation of recipient email in /api/v1/share API
  - By direct IP camera access
    - Using default camera credentials
  - By accessing database
- Unauthorized access to store manager account
  - Exploiting JWT validation vulnerabilities
- ⋮

Other important outputs to threat modeling

- Security requirements
  - Preventive controls
  - Detective controls
- Testing plan

Automation



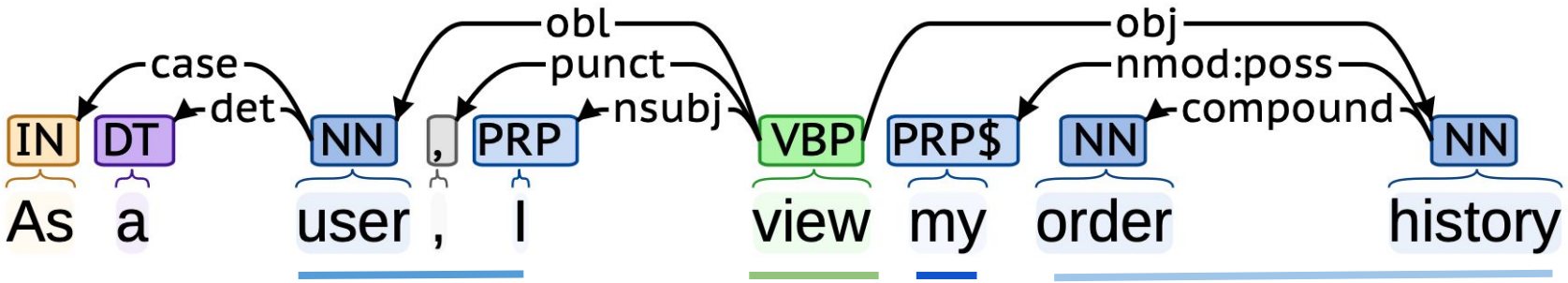
Automation of attacker goals, assets, technical threats:

- Data extraction / reasoning from unstructured data
- If the DFD is represented in a machine-readable format, graph analysis methods can be applied.
- Static analysis and LLMs (agents) can be used when input is code (e.g. IaC)

Threat Identification from Unstructured Data/Text

User story: “As a user, I view my order history”

Before LLMs, used traditional NLP (e.g. Core-NLP):



Entity-Action-Object extraction

```
function extract_EAO_with_possession(parse_tree):
    action = find_token_with_relation(parse_tree, "root")
    subject = find_dependent(parse_tree, action, "nsubj")

    # role entity ("As a user") vs pronoun subject ("I")
    role = find_dependent(parse_tree, action, "obl")
    entity = role if role else subject

    # object
    obj = find_dependent(parse_tree, action, "obj")

    # possession check
    possessor = find_dependent(parse_tree, obj, "nmod:poss")
    if possessor:
        owner = possessor.text
    else:
        owner = None

    # object expansion with modifiers
    full_object = expand_object(obj, parse_tree)

    return (entity.text, action.text, full_object, owner)
```

Abuse cases generation
Assets detection
etc.

Difficult:
Fuzzy matching problem
Not all text relates directly to the application

Threat Identification from Unstructured Data/Text

Much has changed with transformer architectures and LLMs.

- Greatly improved entity-action-object extraction and semantic variation.
- New ways of obtaining insights simply with prompting (and context engineering).
- Examples we'll show use foundation models, but specialised transformers models can be used for different tasks.

Threat Identification from Unstructured Data/Text

Extract the entity, action, and object from the user story below. Return your answer as JSON. User story: "As a user, I view my order history."

json Copy code

```
{
  "entity": "user",
  "action": "view",
  "object": "order history"
}
```

Generate abuse cases for this user story, focusing solely on conceptual security risks at the business logic level. Frame each abuse case in terms of fundamental security principles (confidentiality, integrity, availability) without referencing any specific technologies, implementation methods, or technical vulnerabilities. Consider that this functionality could be delivered through any interface or system architecture. Return your answer as JSON with only an abuse_cases key.

Copy code

```
"abuse_cases": [
  {
    "principle": "confidentiality",
    "description": "An unauthorized person accesses another user's order history, exp
  },
  {
    "principle": "integrity",
    "description": "A user manipulates the order history data to create false records
  },
  {
```

Assets Identification from Unstructured Data/Text

You are a security analyst tasked with identifying sensitive assets and their data classifications from a feature description. Your goal is to comprehensively catalog all assets that could be of interest to attackers, including data assets, privileged operations, and system capabilities.

Instructions

Analyze the provided feature description and identify all sensitive assets, categorizing them according to the data classification catalog provided. Consider both tangible data assets and intangible operational assets that could be exploited.

Data Classification Catalog
{DATA_CLASSIFICATION_CATALOG}

Asset Categories to Consider

Data Assets
- Personal Identifiable Information (PII)
- Financial data
- Authentication credentials
- Business intelligence data
- Customer data
[...]

Assets identification
From software feature description

Automation: Data Flow Diagram

Threat Identification from Data Flow Diagram

A DFD is essentially a directed graph


```
// Components (e.g. Neo4j)
CREATE (browser:Component { name: "Web Browser SPA", technology: "React", type: "Client" });
CREATE (api:Component { name: "API Server", technology: "FastAPI", type: "Server" });
CREATE (db:Component { name: "SQL Database", technology: "PostgreSQL", type: "Database" });

// Data Flows
CREATE (browser)-[:SENDS_REQUEST {protocol:"HTTPS"}]->(api);
CREATE (api)-[:QUERIES {queryType:"SQL"}]->(db);
```

Threat Identification from Data Flow Diagram

Graph queries can identify threats:

Threat	STRIDE	Client	VulnerableServer	TargetDatabase
SQL Injection	Tampering	Web Browser SPA	API Server	SQL Database

```
MATCH (c:Component {type:"Client"})-[:SENDS_REQUEST]->(s:Component {type:"Server"})
MATCH (s)-[:QUERIES {queryType:"SQL"}]->(d:Component {type:"Database"})
RETURN "SQL Injection" AS Threat, "Tampering" AS STRIDE, c.name AS Client, s.name AS
VulnerableServer, d.name AS TargetDatabase;
```

Threat Identification from Data Flow Diagram

Can you use LLMs to help with this?

- LLMs to help build your directed graph from documents
  - Vision model for image (“Can you create a directed graph from the image attached? [... more instructions…]”)
  - Text model for documents (prompts chaining)
- LLMs to help maintain your threat database (graph queries)
  - Use LLM to help creating your threat database including its graph queries.
  - Ideal when coupled with deep research agent to identify latest threats.
- LLMs to directly identify threats from directed graph
  - Require prompt engineering, context management, etc.
  - Recommended in addition to threats identified via threat database.

Prompt for building threat queries

You are a cybersecurity research assistant and a threat modeling expert.
You are given a document containing descriptions of the latest attacks and vulnerabilities targeting Agentic AI.

Your task is to:

1. Read and extract all novel attack vectors.
2. For each attack, identify:
   - The attacker input or source (e.g., user prompt, API request)
   - The vulnerable component(s) (e.g., AI Agent, API Server, Database)
   - The impact or target (e.g., data exfiltration, command execution, privilege escalation)
   - STRIDE category (Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, Elevation of Privilege)

3. Translate each attack pattern into a Neo4j Cypher query that can detect the risk in a DFD graph. Use the following assumptions:
   - Components are labeled as :Component with properties {name, type, technology}
   - Data flows are relationships like :SENDS_REQUEST, :QUERIES, :HAS_ACCESS with relevant properties (protocol, queryType, dataType, etc.)
   - Your Cypher query should match clients sending input to vulnerable components and highlight potential sensitive targets.

Output format for each attack:

- Attack Name:
- Description:
- STRIDE Category:
- Example Cypher Query:

Only provide relevant attack patterns and queries suitable for automated threat detection in a DFD graph. Be concise but precise.

Attack Name: Excessive Functionality Exploitation

Description: An attacker exploits LLM agents that have access to unnecessary extensions or functions beyond their intended scope, allowing unintended system operations. **STRIDE Category:** Elevation of Privilege **Example Cypher Query:**

Cypher

```
MATCH (client:Component)-[:SENDS_REQUEST]->(llm:Component {type: "LLM_Agent"})
-[:HAS_ACCESS]->(ext:Component {type: "Extension"})
-[:QUERIES|SENDS_REQUEST]->(target:Component)
WHERE target.type IN ["Database", "FileSystem", "EmailService", "ExternalAPI"]
AND ext.functionality = "open_ended"
RETURN client, llm, ext, target,
       "Excessive functionality risk detected" as risk_type
```

Attack Name: Excessive Permissions Abuse

Description: Attackers leverage LLM extensions that have overprivileged access to downstream systems, enabling actions beyond the intended read-only or limited scope. **STRIDE Category:** Elevation of Privilege **Example Cypher Query:**

Cypher

```
MATCH (llm:Component {type: "LLM_Agent"})-[:HAS_ACCESS]->(ext:Component {type:
-[:QUERIES]->(db:Component {type: "Database"})
WHERE ext.permissions IN ["admin", "write", "full_access"]
AND ext.intended_operation = "read_only"
RETURN llm, ext, db,
       "Excessive permissions detected - " + ext.permissions as risk_details
```

Attack Name: Autonomous High-Impact Actions

How can I help you?

+ Attach

Claude Sonnet 4

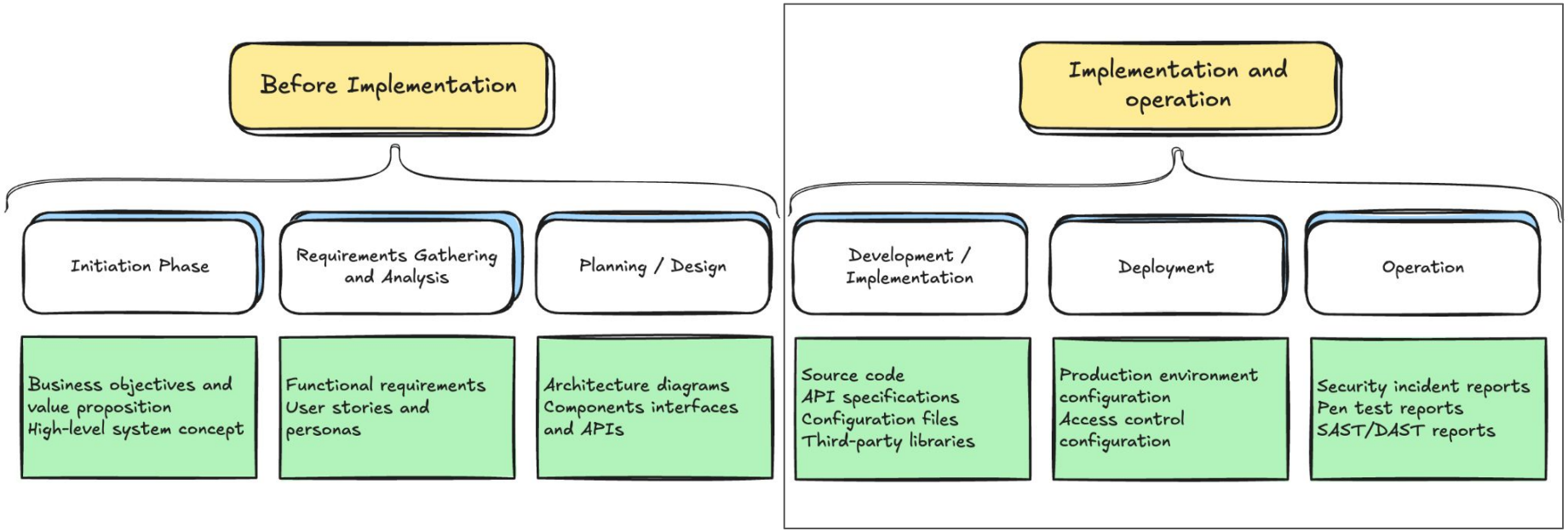
2_0_vulns/LLM06_Exce...

106 lines · 9 KB

```
21
22  ### Common Examples of Risks
23
24  #### 1. Excessive Functionality
25
26    An LLM agent has access to extensions which include functions that are not needed
    for the intended operation of the system. For example, a developer needs to grant
    an LLM agent the ability to read documents from a repository, but the 3rd-party
    extension they choose to use also includes the ability to modify and delete
    documents.
27
28  #### 2. Excessive Functionality
29
30    An extension may have been trialled during a development phase and dropped in
    favor of a better alternative, but the original plugin remains available to the LLM
    agent.
31
32  #### 3. Excessive Functionality
33
34    An LLM plugin with open-ended functionality fails to properly filter the input
    instructions for commands outside what's necessary for the intended operation of
    the application. E.g., an extension to run one specific shell command fails to
    properly prevent other shell commands from being executed.
35
36  #### 4. Excessive Permissions
37
38    An LLM extension has permissions on downstream systems that are not needed for
    the intended operation of the application. E.g., an extension intended to read data
    connects to a database server using an identity that not only has SELECT
    permissions, but also UPDATE, INSERT and DELETE permissions.
39
40  #### 5. Excessive Permissions
41
42    An LLM extension that is designed to perform operations in the context of an
    individual user accesses downstream systems with a generic high-privileged
    identity. E.g., an extension to read the current user's document store connects to
    the document repository with a privileged account that has access to files
    belonging to all users.
43
44  #### 6. Excessive Autonomy
45
```

OWASP/www-project-top-10-for-large-language-model-applications/2_0_vulns/LLM06_ExcessiveAgency.md

Threat modeling in different phases



Threat Modeling after implementation

- Very valuable to review the security of an existing application
- Threat model are a great input to pen test plan.
- Findings from pen tests, incidents, and other security activities should inform:
  - Your threat model
  - Your threat modeling process (e.g. include new threats in your threat library)

Threat Modeling after implementation

- What is done differently?
  - The code is the ultimate source of truth for many details that otherwise are provided in documents.
  - This is true for application code, but also IaC

- What does it mean for automation?
  - There are great opportunities to obtain application details automatically (e.g. list of APIs along with their authentication properties).

Obtaining details (e.g. exposed APIs) from code:

- Traditional techniques
  - Pattern matching & regular expressions
  - Abstract Syntax Tree (AST) parsing
  - Static code analysis (formal semantic analysis)
- AI-powered semantic techniques:
  - Natural language code querying (via embeddings & similarity search).
    Use specialised encoding models like voyage-code-3 and CodeSage-large.
  - LLM code understanding
- AI Agent orchestration:
  - Combines multiple analysis techniques
  - Example: Langraph + tools
  - Off the shelf: Claude Code SDK or CLI

Agentic AI

- AI Agent with tools:
  - Code embeddings, static code analysis:
    Can answer natural language questions about your code (application code and infrastructure code)
  - Graph queries (GraphRAG):
    Can answer natural language question about your data flows, your components and other application information.

- Lots of possibilities:
  - Chatbot helping security reviewer understand the applications and identify threats.
  - Autonomous agent running campaigns to identify systemic issues.

Demos

We’re running jupyter notebook to demonstrate:

📊 DFD Analysis → Turn data flow diagrams from images or text description into neo4j graphs to surface threats

📚 Dynamic Threat Libraries → Automatically generate libraries from deep research agents

🤖 Automated Threat Analysis → Apply these libraries to uncover risks in DFDs in neo4j

🏗️ Architecture Doc Generation → Use code analysis agents (app + IaC) to auto-build architecture artifacts

Code will be available soon on my GitHub:https://github.com/castlebbs

```python
# Analyze diagram with Claude AI
cypher_representation = None

if local_image_path and os.path.exists(local_image_path):
    print("🤖 Sending to Claude for analysis...")
    cypher_representation = claude_analyzer.analyze_data_flow_diagram(local_image_path)

    print("\n" + "="*50)
    print("🧠 CLAUDE'S CYPHER ANALYSIS")
    print("="*50)
    print(cypher_representation)
    # print(cypher_representation[:500] + "..." if len(cypher_representation) > 500 else cypher_representation)
    print("="*50)
else:
    Click to collapse the range.  valid image to analyze")
```

[4]

Turn DFD into graph

```
🤖 Sending to Claude for analysis...
Sending image to Claude Sonnet for analysis...
✓ Received Cypher representation from Claude

==================================================
🧠 CLAUDE'S CYPHER ANALYSIS
==================================================
Looking at this Math Wiz App Schema diagram, I can identify the following components and their relationships:

**Components:**
- User Query (user input)
- Application Frontend (web interface)
- Agent (central AI orchestrator with lightbulb icon)
- LLM (language model)
- Tool 1: Wikipedia Tool (for searching information)
- Tool 2: Calculator (for numeric calculations)
- Tool 3: Reasoning Tool (for logical reasoning)

**Flow Analysis:**
The user query flows through the frontend to an AI agent, which coordinates with an LLM and uses three specialized tools based on the query type.

```cypher
MATCH (n) DETACH DELETE n

CREATE (user_query:UserInput {name: 'User Query', type: 'user_input'})
```
```

```python
# Step 5: Parse and structure results
if analysis_result:
    parsed_queries = analyzer.query_parser.parse_claude_analysis(analysis_result)

    if parsed_queries:
        print(f"✅ Parsed {sum(len(queries) for queries in parsed_queries.values())} queries")
        print("📊 Query categories:")
        for category, queries in parsed_queries.items():
            print(f"  • {category}: {len(queries)} queries")
    else:
        print("❌ Failed to parse analysis results")
else:
    print("⚠️ No analysis result to parse")
    parsed_queries = {}
```

```
🔧 Parsing Claude analysis for Cypher queries...
📊 Found 7 potential threat sections
  ✅ Threat 1: Excessive Functionality in AI Extensions (Elevation of Privilege) - 2 queries
  ✅ Threat 2: Excessive Permissions in AI Database Access (Elevation of Privilege) - 2 queries
  ✅ Threat 3: Excessive Autonomy in AI Operations (Tampering) - 2 queries
  ✅ Threat 4: Indirect Prompt Injection via Data Sources (Tampering) - 2 queries
  ✅ Threat 5: AI Service Privilege Escalation (Elevation of Privilege) - 2 queries
  ✅ Threat 6: Insecure AI Extension Communication (Information Disclosure) - 2 queries
🎯 Successfully parsed queries for 3 STRIDE categories
  • Elevation of Privilege: 6 queries
  • Tampering: 4 queries
  • Information Disclosure: 2 queries
✅ Parsed 12 queries
📊 Query categories:
  • Elevation of Privilege: 6 queries
  • Tampering: 4 queries
  • Information Disclosure: 2 queries
```

Turn research papers, guidances, or result from deep research agents into threat libraries

Load the generated queries into our analysis engine.

```python
# Initialize the query loader
loader = CypherQueryLoader(queries_directory=".")

# Load queries from the most recent LLM-generated file
queries_file = "llm_threat_queries_20250918_200145.cypher"
# queries_file = "threat_library/llm_threat_queries_20250917_180146.cypher"
threat_queries = loader.load_cypher_queries(queries_file)

print(f"\n📋 Loaded queries summary:")
if threat_queries:
    for category, queries in threat_queries.items():
        print(f"   • {category.replace('_', ' ').title()}: {len(queries)} queries")
else:
    print("   ⚠️ No queries loaded - check file path and format")
```

```
📂 Loading queries from: llm_threat_queries_20250918_200145.cypher
📄 File size: 8,495 characters
✅ Loaded 12 queries
📊 Categories found:
  • ELEVATION_OF_PRIVILEGE: 4 queries
  • TAMPERING: 4 queries
  • SPOOFING: 2 queries
  • INFORMATION_DISCLOSURE: 2 queries

📋 Loaded queries summary:
   • Elevation Of Privilege: 4 queries
   • Tampering: 4 queries
   • Spoofing: 2 queries
   • Information Disclosure: 2 queries
```

Result of analysis:
DFD was an image converted into a graph
Queries were generated from unstructured research document

==========
🎯 ELEVATION OF PRIVILEGE THREATS
==========

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

	#	Threat Type	Component	Severity	Description
0	1	Potential Excessive Functionality	Math Wiz Agent	MEDIUM	AI service "Math Wiz Agent" requires review for excessive agency vulnerabilities including functionality, permissions, and autonomy controls.

==========
🎯 SPOOFING THREATS
==========

	#	Threat Type	Component	Severity	Description
0	1	Indirect Prompt Injection Risk	Math Wiz Agent	HIGH	AI service "Math Wiz Agent" processes external/user data from "Application Frontend" and has access to "LLM". This creates indirect prompt injection attack surface.
1	2	Indirect Prompt Injection Risk	Math Wiz Agent	HIGH	AI service "Math Wiz Agent" processes external/user data from "Application Frontend" and has access to "Wikipedia Tool". This creates indirect prompt injection attack surface.
2	3	Indirect Prompt Injection Risk	Math Wiz Agent	HIGH	AI service "Math Wiz Agent" processes external/user data from "Application Frontend" and has access to "Calculator". This creates indirect prompt injection attack surface.
3	4	Indirect Prompt Injection Risk	Math Wiz Agent	HIGH	AI service "Math Wiz Agent" processes external/user data from "Application Frontend" and has access to "Reasoning Tool". This creates indirect prompt injection attack surface.
4	5	Data Flow Manipulation Risk	Math Wiz Agent	MEDIUM	AI service "Math Wiz Agent" is in a data processing chain. Implement input sanitization and output validation to prevent manipulation attacks.

API Analysis

Analyze the cloned repository to identify available APIs.

Auto generation of API description from AI Analysis Agent.

```
import pandas as pd
import json
import re

def analyze_apis():
    try:
        red prompt for API analysis
        structured_prompt = """
        Analyze this codebase and provide a structured list of all available
        Return the response as a JSON array where each API endpoint is an ob
        - "endpoint": the API endpoint path
        - "method": HTTP method (GET, POST, PUT, DELETE, etc.)
        - "description": brief description of what this endpoint does
        - "authenticated": boolean indicating if the API requires authentica

        Format your response as valid JSON only, no additional text.
        """
```

Click to add a breakpoint

	endpoint	method	description	authenticated
0	/api/users	POST	Create a new user account	False
1	/api/users/login	POST	Login with user credentials	False
2	/api/user	GET	Get current authenticated user details	True
3	/api/user	PUT	Update current user profile	True
4	/api/articles	GET	Get paginated list of articles with optional f...	False
5	/api/articles/feed	GET	Get paginated feed articles from followed users	True
6	/api/articles	POST	Create a new article	True
7	/api/articles/:slug	GET	Get a specific article by slug	False
8	/api/articles/:slug	PUT	Update an existing article	True
9	/api/articles/:slug	DELETE	Delete an article	True
10	/api/articles/:slug/comments	GET	Get comments for a specific article	False
11	/api/articles/:slug/comments	POST	Add a comment to an article	True
12	/api/articles/:slug/comments/:id	DELETE	Delete a specific comment	True
13	/api/articles/:slug/favorite	POST	Mark an article as favorite	True
14	/api/articles/:slug/favorite	DELETE	Remove article from favorites	True
15	/api/profiles/:username	GET	Get user profile by username	False
16	/api/profiles/:username/follow	POST	Follow a user	True
17	/api/profiles/:username/follow	DELETE	Unfollow a user	True
18	/api/tags	GET	Get list of popular tags	False

LLMs challenges in practice

Technical Limitations: Non-deterministic outputs, hallucinations, context window limits

Domain Knowledge Gaps: Foundation models vs. specialized security expertise

Operational Concerns: Cost, latency, prompt engineering complexity

Bottom Line: LLMs are powerful tools, not magic solutions. Success requires understanding their limitations and designing systems accordingly.

Takeaways

Security Starts with Understanding

Understand the applications, real features, real use cases and you'll identify real threats.

Threat Modeling identifies threats, developers need actionable requirements

Threat Models need to be followed by concrete actions.

Large Language Models are a game changer for threat modeling automation

Know the limits, keep humans in control, embrace the potential.

Thank you

You can contact me:

- david@securityarchitect.ai
- http://linkedin.com/in/castlebbs/

Questions?