INDRAPRASTHA COLLEGE FOR WOMEN

UNIVERSITY OF DELHI

SOFTWARE ENGINEERING

PROJECT

DEVICE CODE PHISHING THREAT HUNTER

Name	Dhruvika Verma	Jaya
College roll no.	23/CS/20	23/CS/31
Examination roll no.	23029570019	23029570030
Course	B.Sc. Computer Science (Hons.)	
Semester	V	
Unique Paper Code	2342013503	
Submitted to	Ms. Vimala Kumari	

ACKNOWLEDGMENT

It is our privilege to express our sincerest regards to my Software Engineering professor, **Ms. Vimala Kumari**, for her invaluable inputs, able guidance, encouragement, whole hearted cooperation and constructive criticism without which this project would not have been able to sail through. We deeply feel honored and blessed for getting this opportunity of being guided by her. We express our sincere thanks to her for being there constantly for help and supporting, encouraging and guiding me to successfully complete and present this project on the topic: "Device Code Phishing Threat Hunter"**.**

Dhruvika Verma
Jaya

CERTIFICATE

This is to certify that **Ms. Dhruvika Verma** & **Ms. Jaya** have successfully carried out the completion of the project entitled "Device Code Phishing Threat Hunter" under my supervision. The project has been submitted as per the requirements of lab based on Software Engineering in the fifth semester of B.Sc. (H) Computer Science.

Project-in-charge
(**Ms. Vimala Kumari**)

INDEX

PROJECT DESCRIPTION

INTRODUCTION

The Device Code Threat Hunter project focuses on a growing cybersecurity problem called **Device Code Phishing**, where attackers trick users into entering verification codes on fake websites or malicious devices. These attacks are hard to detect because the user believes they are completing a normal login process, and the attacker silently captures the code to access the victim's account. Since device-code authentication is commonly used in TVs, consoles, and workplace apps, this threat is becoming more dangerous.

To address this problem, the project creates **simulated authentication logs** and uses a Python-based detection engine to find patterns that look suspicious. The system checks for behaviours like too many code requests, failed verification attempts, login attempts from unusual IP addresses, strange user agents, or verification on unknown domains. These patterns are common indicators of real phishing or account takeover attempts. By applying simple rule-based detection methods, the project helps users understand how threat hunting works in a real environment. It shows how to analyze logs, flag abnormal activity, and generate an alert report.

OBJECTIVES

- Design, implement, and test a Python-based threat hunting tool to detect Device Code Phishing activity.
- To analyze authentication logs for anomalous device code flows.
- Provision of simulated real-time alerting and reporting of potential threats.
- Improve the understanding of MITRE ATT&CK techniques: T1566.002 - Phishing: Device Code Phishing.
- To create a foundation for future automation and integration with SIEM systems.

HARDWARE & SOFTWARE REQUIREMENTS

Hardware Requirements:

1. **RAM:** 4 GB or higher
2. **Storage (ROM / HDD / SSD):** 200–300 MB free disk space
3. **Processor:** Intel Core i3 or higher
4. **Operating System:** Windows 7 / 8 / 10 / 11 (or Linux with Python support)
5. **Internet Access:** Required only for downloading logs or updates
6. **Device:** Laptop / Desktop with basic log-processing capability

Software Requirements:

- **Front-End:** Python
- **Editors, IDEs:** VS Code
- **Back-End:** Python 3.x , JSON file, TXT log file
- **Storage:** File-based data stores (JSON/TXT for logs, alerts, events)

PROCESS MODEL

<u>Incremental Process Model</u>

The **Incremental Model** is a development approach where the system is created step-by-step in smaller, manageable parts rather than all at once. The entire project is divided into separate functional modules, and each module is developed through the usual phases like requirements gathering, design, coding, and testing. Every completed module represents a working portion of the software, and as more increments are added, the system gradually grows into its full form.

This approach works especially well for projects that have clearly defined goals but may need adjustments or enhancements along the way. In the case of cybersecurity tools particularly threat detection systems, requirements often evolve because new attack techniques emerge. The incremental model supports this reality by allowing the detection logic to be improved, updated, and expanded over time without disrupting the entire system. It also fits naturally with projects built from multiple independent components, since each part can be developed and tested individually before being integrated into the main system.

Phases of the Incremental Model

The incremental development process for the **Device Code Phishing Threat Hunter** involves the following phases:

1. Requirement Analysis
In this phase, requirements for the entire system are collected at a high level. Individual module-level requirements are defined for each increment, such as:

- Log parsing
- Known IP management
- Anomaly detection
- Alerting mechanism
- Report generation

These requirements may evolve as testing reveals new threats or logic improvements.

2. System Design
The overall architecture is prepared in this phase. Each increment has its own design inputs and outputs.
For example:

- **Increment 1:** Design log parser → defines log format
- **Increment 2:** Design detection logic → defines anomaly rules
- **Increment 3:** Alert system → defines notification templates
- **Increment 4:** Reporting → defines incident summary output

The entire system blueprint is defined early, while detailed internal designs evolve incrementally.

3. Implementation of Each Increment
Each module is coded independently and integrated with the previously developed modules.
Example implementation sequence:

- **Increment 1:** Build log reader & parser
- **Increment 2:** Build suspicious IP detection
- **Increment 3:** Build alert system (simulated email/SMS alert)
- **Increment 4:** Build final incident summary generator
- **Increment 5:** Add advanced logic (time correlation, multiple users, multi-log support)

Each increment results in a **working and testable version** of the software.

4. Testing

Each increment undergoes individual testing before integration:

- Unit Testing
- Module Testing
- Integration Testing
- Functional Testing

This reduces debugging complexity and ensures system reliability at every phase.

5. Integration

Increments are assembled step-by-step. Every new component is integrated with previous increments, ensuring smooth compatibility and functional consistency.

6. Deployment (Final Integration)

After all increments are fully tested and combined, the final version of the system is prepared. For this project, deployment refers to running the system on a local environment or presenting it for evaluation.

Why We Are Using the Incremental Model?

1. Cybersecurity Requirements Are Evolving

Phishing techniques and authentication bypass methods change frequently. The incremental model allows updating detection logic without redesigning the entire system.

2. Modular Project Architecture

The system consists of well-defined modules such as:

- Log Parsing
- Threat Detection
- Incident Alerting
- Reporting

Each module can be built and tested individually, making incremental development ideal.

3. Early Delivery and Feedback

Even after the first increment, a basic version of the tool can parse logs and detect suspicious IP activity. This enables early demonstration and feedback for iterative improvement.

REQUIREMENT ANALYSIS

SOFTWARE REQUIREMENTS SPECIFICATION (SRS)

Introduction

It is a formal document, specifying the functional and non-functional requirements of the system. An SRS for DeviceCode-Log-Hunter specifies how detection software will process log files, identify suspicious authentication patterns, and simulate alert generation.
This document serves as the blueprint for system design and implementation, ensuring that all requirements are well-defined, testable, and traceable.

SRS outlines how software interacts with system files, parses data, detects anomalies, and conveys output to the user. This is also supposed to document system constraints, external interface requirements, and performance expectations.

The SRS for the project includes:

- Functional requirements expressed in simple and structured language.
- Non-functional requirements that cover security, performance, and portability.
- Log file and alert report input/output format.
- High-level design of how the logs flow through the system.

Purpose

This SRS shall describe the requirement to develop a Threat Hunting Tool for detecting Device Code Phishing using simulated authentication logs. This document describes all system functions, design constraints, and performance criteria necessary to implement the software successfully.

It acts as a guide to:
- **Developers**, to understand the logic flow and detection algorithms to be implemented.
- **Cybersecurity learners and analysts**, to use the tool for demonstration and analysis.
- **Project evaluators**, to assess the completeness and quality of software design and testing.

Overall Description

The idea here is the DeviceCode-Log-Hunter project maintains three major components:

- **Log Analyzer Module:** This module reads simulated log files, parses events, and identifies key fields such as timestamp, user, source IP, and event type.
- **Detection Engine:** Contains the logic to detect suspicious DeviceCodeStart and DeviceCodeSuccess patterns, taking into account IP and time correlation.
- **Alert and Reporting Module:** It generates immediate alerts for suspicious events and prints a summarized incident report with the details of detected anomalies.

Functional Requirements

1. The system shall read authentication log entries from a text file.
2. The system shall identify and extract log components, including timestamp, username, source IP, event type, and status.
3. The system shall maintain a list of known safe IP addresses from a JSON file.
4. The system shall flag any DeviceCodeStart event originating from an unknown IP as suspicious.
5. The system shall correlate DeviceCodeSuccess events within a defined time window (by default, 10 minutes) of a suspicious start event.
6. The system shall produce immediate printed warnings for suspicious events.
7. The system should generate a summary report listing all the alerts detected during execution.

Non-Functional Requirements

1. **Performance**: A log file containing over 500 entries should be processed in a matter of seconds on an average desktop environment.
2. **Portability**: Software should run on any platform with Python 3.x installed.
3. **Usability**: The tool should be easy to run and understand, without any external dependencies or databases.
4. **Maintainability**: Modular design for easy updates of detection rules or log formats.
5. **Security**: The system shall safely simulate the alerts without connecting to any real authentication systems or dispatching real emails.

Performance Requirements

- Should flag suspicious events instantly after log parsing.
- Should handle multiple users and events simultaneously in simulation.
- The system shall provide clear display of alerts and final summary without lag or loss of data.

System Constraints

- Works with structured text logs in a defined format: Timestamp - User - IP -Event_Type - Status.
- Assumes known IPs are provided manually by the analyst.
- Does not connect to external servers or real-time authentication APIs (simulation only).

Benefits of the System

Current Situation	With Device Code Log Hunter
Manual log review is slow and error-prone.	Automated parsing and correlation improves detection speed.
Script immediately flags anomalies	Script immediately flags anomalies.
No emulated SOC reaction	Instant printed alerts imitate real-life notifications
Limited awareness of MITRE ATT&CK linkage	Project explicitly maps detections to T1566.002 - Device Code Phishing.

DATA FLOW DIAGRAM (DFD)

A Data Flow Diagram (DFD) is a graphical tool used to represent the flow of data through a system. It shows how data enters the system, how it moves between different processes, where it is stored, and how it leaves the system. It explains the system in terms of **input, process, and output**.

In a DFD, data comes from external sources, gets processed inside the system, may be stored in data stores, and then flows out as output. This diagram helps in understanding the logical working of the system without focusing on technical details.

DFD Components

1. External Entities
External entities are sources or destinations of data outside the system boundary.
They either provide input or receive output.
They are represented using **rectangles**.

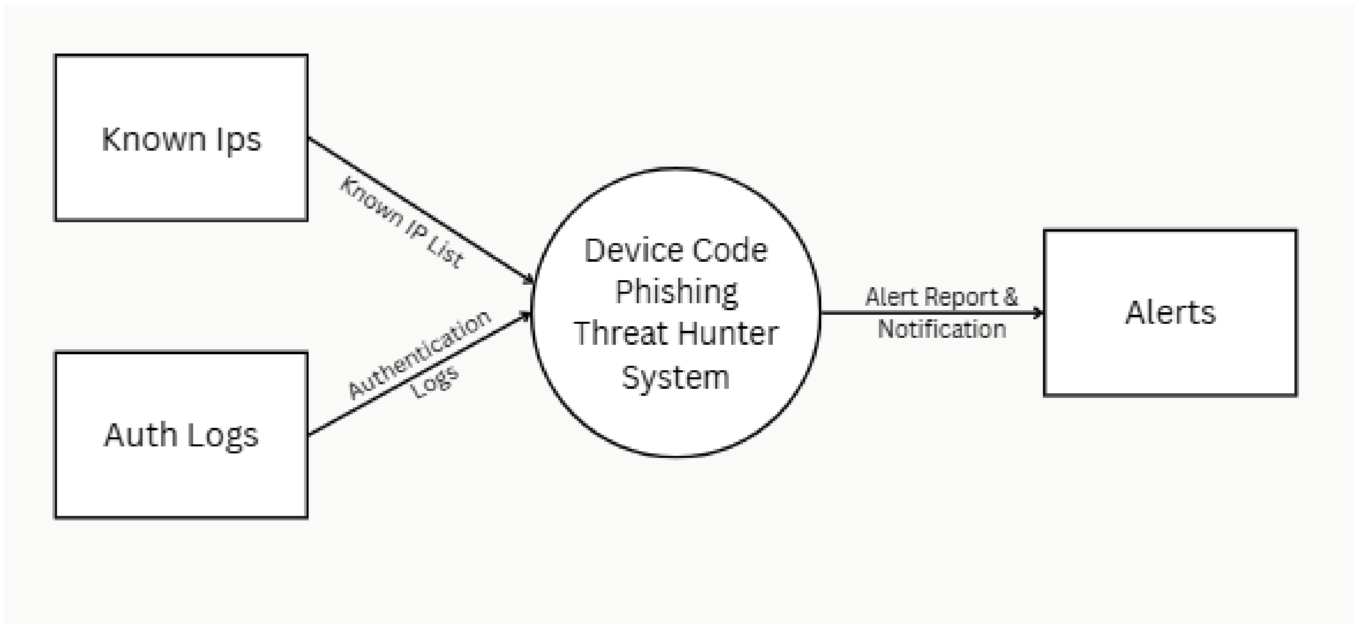
2. Processes
Processes represent actions performed on data.
They convert input data into output data.
Processes are shown using **circles** or **rounded rectangles.**

3. Data Flow
Data flow shows the movement of data between entities, processes, or data stores.
It is represented using **arrows**.
Each arrow must be labelled clearly.
4. Data Store
A data store represents locations where data is kept temporarily or permanently.
It is represented using **two horizontal parallel lines**.

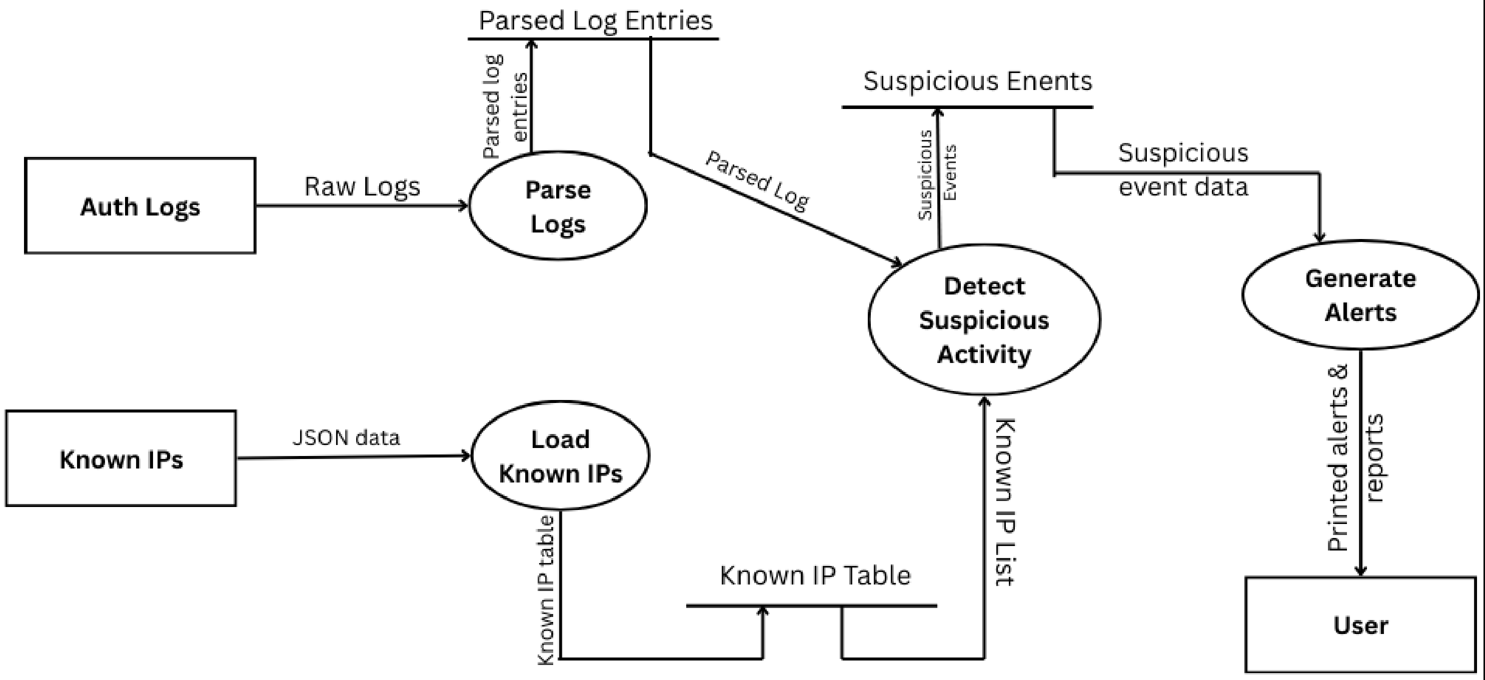
DFD Guidelines

- The **Level-0 DFD** should represent the whole system as a single process.
- All major inputs and outputs must be clearly shown at Level-0.
- In the next level, the major process is broken into smaller processes to show detailed data flow.
- All arrows, processes, and data stores should be given meaningful names.
- Continuity must be maintained between successive levels.
- Only one process should be expanded at a time to avoid confusion.
- No new external entity should appear suddenly in later levels.

CONTEXT LEVEL DFD

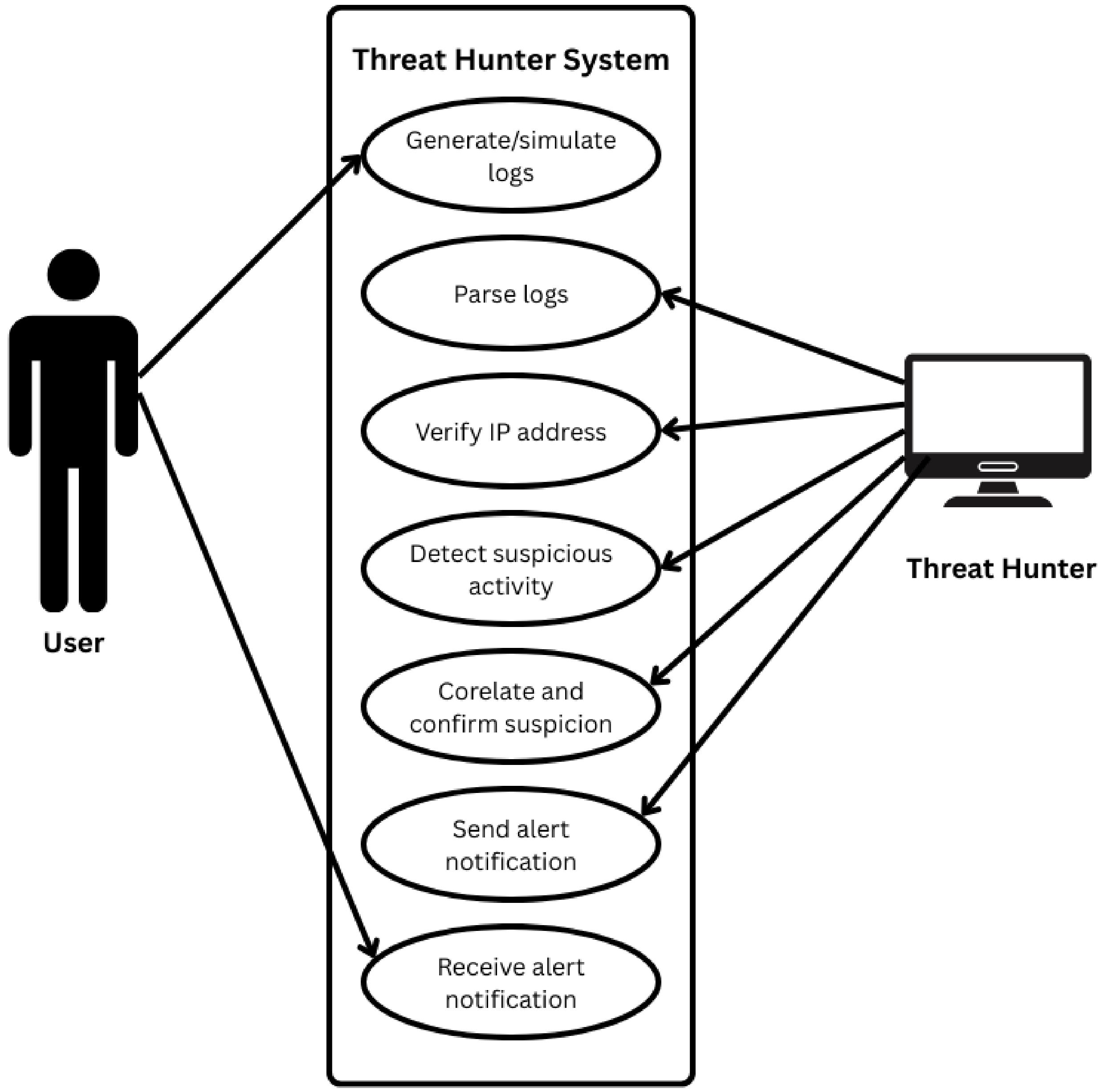

Known Ips
Known IP List
Auth Logs
Authentication Logs
Device Code Phishing Threat Hunter System
Alert Report & Notification
Alerts

1ST LEVEL DFD



USE CASE DIAGRAM

A use case diagram is a graphical depiction of a user's possible interactions with a system. A use case diagram shows various use cases and different types of users the system has and will often be accompanied by other types of diagrams as well. Hence, a use case diagram captures the functional aspects of system.



USE CASES

Use case 1: Provide logs for analysis

1.	**Actor**	User
2.	**Pre-Condition**	User activity has been recorded and the auth_logs.txt file exists
3.	**Main Success Scenario**	1. User provides the auth_logs.txt file containing authentication events (DeviceCodeStart, DeviceCodeSuccess) to the system 2. System locates and opens the file for processing 3. System accepts the input stream for parsing • System raises a FileNotFoundError or simply processes an empty list (stops execution)
4.	**Exception Scenarios**	2. a) The log file does not exist (FileNotFoundError) 3. a) The log file is empty

Use Case 2: Threat Hunter System Detects Logs (Combines: Parse Logs, Verify IP, Correlate Suspicion, Send Alerts)

1.	**Actor**	User
2.	**Pre-Condition**	User activity has been recorded and the auth_logs.txt file exists
3.	**Main Success Scenario**	1. **[Parse Logs]** System reads the text file, splits lines by the | delimiter, and converts timestamp strings into datetime objects 2. **[Verify IP]** System compares the IP from DeviceCodeStart events against the known_ips.json whitelist 3. **[Correlate]** System identifies a DeviceCodeStart from an unknown IP and temporarily stores it as suspicious 4. **[Correlate]** System detects a DeviceCodeSuccess for the same user and calculates the time difference 5. **[Confirm]** System validates that the success event happened within 600 seconds (10 minutes) of the start event 6. **[Send Alerts]** System appends the incident to the ALERTS list, prints a console warning, and triggers the notification function • System catches the ValueError, returns None, and skips that specific line • System catches JSONDecodeError, prints "Error", and defaults to treating all IPs as unknown • System treats the session as timed out/unrelated and removes the user from the tracking dictionary without sending a final alert

4.	**Exception Scenarios**	1. a) Log file contains an invalid timestamp format 2. a) The known_ips.json file is malformed or missing 5. a) The time difference is greater than 600 seconds

Use Case 3: Receive Notification

1.	**Actor**	User
2.	**Pre-Condition**	The Threat Hunter System has flagged the User's activity as suspicious
3.	**Main Success Scenario**	1. User receives an "Early Notification" email warning of a login attempt from an unknown location 2. User verifies if they are currently traveling or using a new network 3. User receives a "Final Notification" alert confirming a successful login from that unknown IP 4. User follows the instructions to revoke active sessions and change their password • User ignores the warning if the activity was legitimate
4.	**Exception Scenarios**	1. a) User is actually the one logging in (False Positive)

DATA DICTIONARY

A data dictionary is a collection of detailed information about the data used in the system. It includes the **name of the data**, **its alias**, **where it is used**, and **its description**.
It represents the structure and content of data that flows between various processes of the system. The data dictionary helps in understanding the system clearly by defining each data item that appears in the Data Flow Diagram.

The data dictionary normally contains:

1. **Name** – The name of the data element.
2. **Alias** – Other names used for the same data item.
3. **Where Used** – Processes where the data is used.
4. **Description** – Explanation of the purpose and meaning of the data.

DATA DICTIONARY TABLE

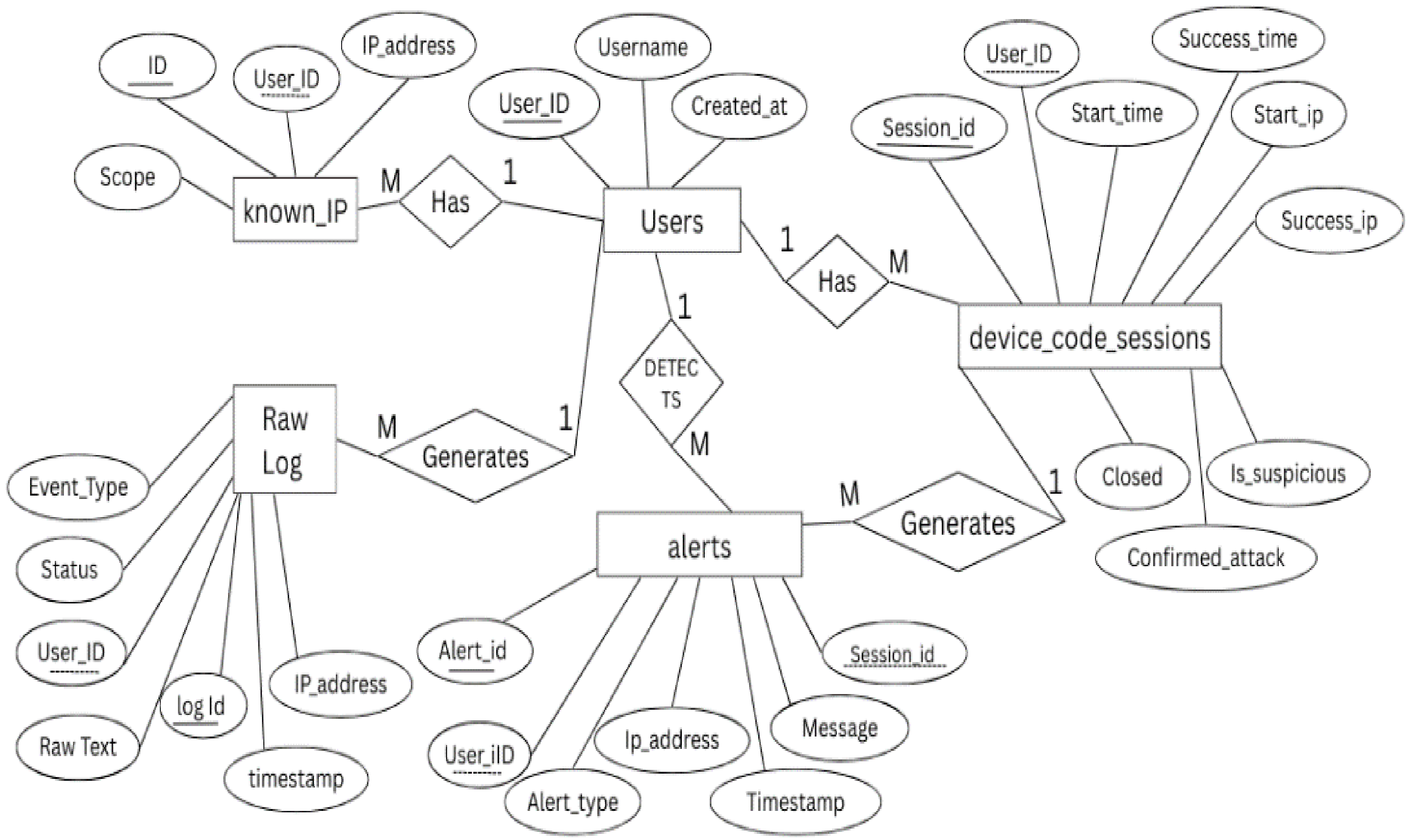
S.no	**Name of data**	**Alias**	**Where used**	**Description**
1	Authentication log file	Log file	Log parser	Raw authentication logs stored in text form.
2	Known IP list	Trusted IP list	IP checker	Stores known safe IPs for each user.
3	Parsed log entries	Parsed logs	Detection engine	Stores all cleaned and structured log entries.
4	Username	User	Parser & detection	The user who attempted login.
5	Source IP address	IP	Parser & detection	Ip address from which the login attempt was made.
6	Event	Event type	Detection engine	Shows whether event is devicecodestart or devicecodesuccess
7	Login status	Status	Detection engine	Indicates success or failure of event.
8	Stored start events	Start event record	Correlation module	Stores suspicious device code start details.
9	Alert list	Alert records	Reporting	Contains detected suspicious/confirmed alerts.
10	Time window	Correlation time limit	Correlation module	10-minute limit for matching events.

ENTITY RELATIONSHIP DIAGRAM

An Entity Relationship Diagram (ER Diagram) pictorially explains the relationship between entities to be stored in a database. Fundamentally, the ER Diagram is a structural design of the database. It acts as a framework created with specialized symbols for the purpose of defining the relationship between the database entities. ER diagram is created based on three principal components: entities, attributes, and relationships.

<u>Symbols Used in ER Diagrams</u>

▪ Rectangles: This Entity Relationship Diagram symbol represents entity types.
▪ Ellipses: This symbol represents attributes.
▪ Diamonds: This symbol represents relationship types.
▪ Lines: It links attributes to entity types and entity types with other relationship types.
▪ Primary key: Here, it underlines the attributes.
▪ Double Ellipses: Represents multi-valued attributes.



PROJECT MANAGEMENT

FUNCTION POINT

The function point is a *unit of measurement* used to express the amount of business functionality an information system provides to a user. Function points are used to compute the functional size of software applications.
In this project, the function points represent the amount of functional work performed by the Threat Hunter System, such as log collection, parsing, suspicious IP detection, correlation, and alert generation.

Functionality is calculated in terms of the number of **inputs**, **outputs**, **inquiries**, **internal logical files**, and **external interface files**.

NUMBER OF EXTERNAL INPUTS: Each external input that provides distinct application oriented data to the software is counted separately.

NUMBER OF EXTERNAL OUTPUTS: Each external output that provides application oriented information to the user is counted. In this context output refers to reports, screen, and error messages, etc. Individual data items within a report are not counted separately.

NUMBER OF EXTERNAL INQUIRIES: An external inquiry is defined as an on line input that results in the generation of some immediate software response in the form of an on-line output. Each distinct inquiry is counted.

NUMBER OF INTERNAL LOGICAL FILES: Each internal logical file (i.e., a logical grouping of data that may be one part of a large database or a separate file) is counted.

NUMBER OD EXTERNAL INTERFACE FILES: All external interfaces (e.g., data files on storage media) that are used to transmit information to another system are counted.

INFORMATION DOMAIN VALUE	**COUNT**		**WEIGHTING FACTOR**				**TOTAL**
			SIMPLE	**AVERAGE**	**COMPLEX**		
External Inputs (EI)	2	X	3	4	6	=	8
External Outputs (EO)	2	X	4	5	7	=	10
External Inquiries (EQ)	0	X	3	4	6	=	0
Internal Logical Files (ILF)	3	X	7	10	15	=	30
External Interface Files (EIF)	0	X	5	7	10	=	0
COUNT TOTAL							**48**

To compute Function Point (FP), the following relationship is used:
F.P. = Total Count * [0.65 + (0.01 x Σ F_i)]
Here, total count is the sum of all FP entries obtained from the above table.

The F_i (i=1 to 14) are "Complexity Adjustment Values" based on response to the following questions

VALUE ADJUSTMENT FACTORS

Backup and recovery required	– **2**
Special data communication needed	– **1**
Distributed processing	– **2**
Performance critical	– **4**
Runs in heavily loaded environment	– **3**
Online data processing	– **5**
Multi-step input screens	– **3**
ILFs updated online	– **3**
Inputs/outputs complex	– **2**
Internal processing complex	– **3**
Reusable components	– **4**
Conversion/installation	– **1**
Multi-environment installation	– **2**
Designed for easy change	– **3**

$\Sigma F_i = 38$

Final Function Point (FP)

Formula: FP = Total Count * [0.65 + (0.01 × $\sum F_i$)]

Putting values:

FP = 48 * (0.65 + 0.01×38)

FP = 48 * (0.65 + 0.38)

FP = 48 * (1.03)

FP = 49.44

Function Point (FP) = 49.44

EFFORT ESTIMATION (FINAL)

Effort Estimation is used to predict the duration of a software engineering project based on its functional size.

It helps approximate the total person-months required for development.

For this project, the effort is calculated using the standard formula:

Effort Formula : Effort(E) = −37 + (0.96 × FP)

E = -37 + (0.96 × 49.44)

E = -37 + (47.46)

E = 10.46 = 10 (appx.)

TOTAL MONTHS = 40% OF EFFORT
= 40% * 10
= 4 months (Approx)

As two persons are working on this project, hence effort is estimated as 2 months per person (Approx).

RISK TABLE

Risk is a probabilistic event i.e. it may occur or may not occur. We frequently have optimistic tendency to simply not see risk or wish they will not occur, which later on leads us to trouble. Hence it is advisable to identify the critical areas, assess their probabilities, estimate the impact and plan the contingency plan. The first step is Risk identification. Next each risk is analyzed to determine the likelihood that it will occur and the damage it will do if it occurs. Risks are then, ranked by probability and impact. Finally, a plan is developed to manage those risks with high probability, moderate as well as low impact.

These risk analysis activities assist us in developing a strategy for dealing risks.
An effective strategy is the RMMM plan.

It Includes:
- **Risk identification**
- **Risk monitoring**
- **Risk management and contingency plan**

Impact Values:
- Negligible (4)
- Marginal (3)
- Critical (2)
- Catastrophic (1)

S.No	**RISK**	**CATEGORY**	**PROBABILITY**	**IMPACT**	**RMMM** (Risk Mitigation, Monitoring & Management)
1	Log file format mismatch	Technical	**40%**	**3**	Validate log format before parsing; add error handling
2	known IP list file corrupted	Technical	**30%**	**3**	User error handling parsing, add error handling
3	Logs may contain sensitive IP or user data	Security/Privacy	**30%**	**4**	Mask sensitive details, restrict storage of logs

4	Fake or forged logs may cause false alerts	Security/Privacy	**35%**	**3**	Validate timestamps, IP formats, and reject abnormal logs
5	Log file missing during execution	Operational	**35%**	**2**	Check file existence before running; notify user clearly
6	System crash due to unexpected input format	Operational	**25%**	**3**	Add strong validation and input error handling
7	User may not understand alert messages	User Experience	**40%**	**2**	Provide simple clear messages and explanations
8	User may find the tool difficult to use	User Experience	**20%**	**2**	Provide brief instructions or on-screen guidance
9	Known IP list not updated regularly	Compliance	**25%**	**2**	Remind user to update the list, check for outdated entries
10	Logs may not follow required compliance format	Compliance	**30%**	**2**	Apply strict log structure checks and validation

TIMELINE CHART

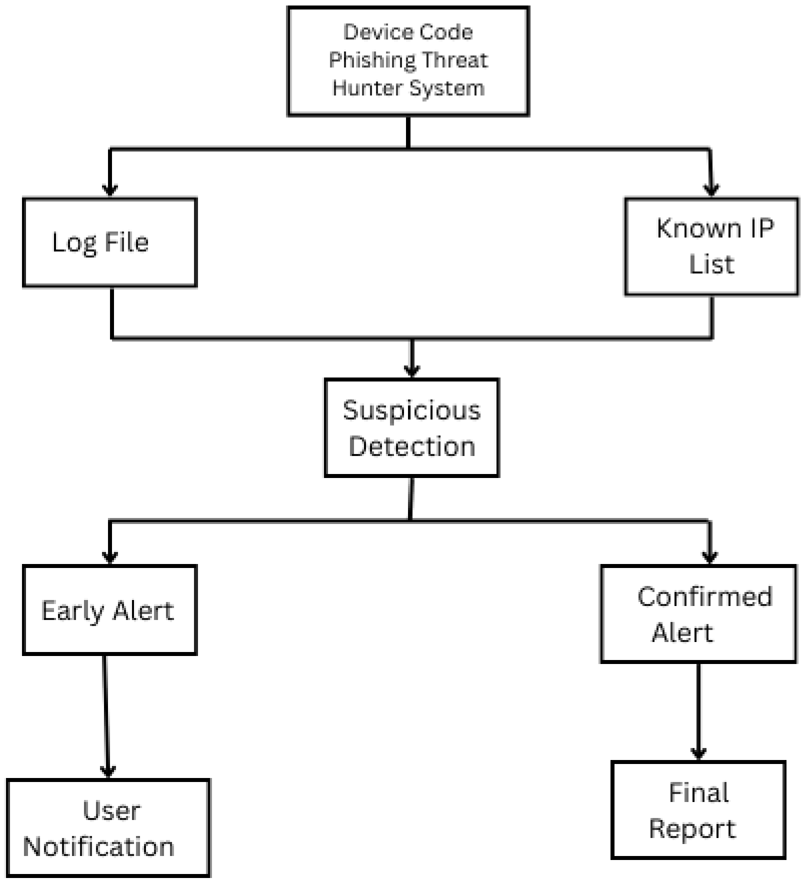
A Timeline is a chart which displays a project plan schedule in chronological order. A Timeline is used in project management to depict project milestones and visualize project phases, and show project progress. The graphic form of a timeline makes it easy to understand critical project milestones, such as the progress of a project schedule.

S. no.	Work Tasks	Week 1	Week 2	Week 3	Week 4	Week 5	Week 6	Week 7	Week 8	Week 9
1.	**PROJECT DESCRIPTION**									
2.	Introduction	■								
3.	Objectives	■								
4.	Hardware And Software Requirements		■							
5.	Process Model		■							
6.	**Milestone- Project Decription**		◆							
7.	**REQUIREMENT ANALYSIS**									
8.	Software Requirement Specification (Srs)			■						
9.	Data Flow Diagram (Dfd)			■						
10.	Use Case Diagram				■					
11.	Use Case Template				■					
12.	Data Dictionaries					■				
13.	Er Diagram					■				
14.	**Milestone- Requirement Analysis**					◆				
15.	**PROJECT MANAGEMENT**									
16.	Function Point						■			
17.	Effort Estimation						■			
18.	Risk Table						■			
19.	**Milestone- Project Management**						◆			
20.	**ENGINEERING DESIGN**									
21.	Architectural Design							■		
22.	Database Design							■		
23.	**Milestone- Engineering Design**							◆		
24.	**CODING AND TESTING**									
25.	Pseudocode								■	
26.	Testing								■	
27.	Flow Graph									■
28.	Cyclomatic Complexity									■
29.	Independent Path Testing									■
30.	**Milestone- Testing**									◆

ENGINEERING DESIGN

ARCHITECTURAL DESIGN

Architectural design in software engineering is the process of breaking down a system into interacting components. It is shown as a block diagram that defines an overview of the system structure, component characteristics, and how these components connect with one another to share data. It specifies the components required for constructing a computer-based system and their communication, i.e. the interaction between these components. It outlines the structure and attributes of the system's components, as well as the interrelationships between these components.



DATABASE DESIGN

Database design can be generally defined as a collection of tasks or processes that enhance the designing, development, implementation, and maintenance of enterprise data management system. Designing a proper database reduces the maintenance cost thereby improving data consistency and the cost-effective measures are greatly influenced in terms of disk storage space. Therefore, there has to be a brilliant concept of designing a database.

Table 1: users
Stores all users found in logs.

Column Name	Type	Description
user_id (PK)	INT	Auto-increment user ID
username	VARCHAR	Email/username from logs
created_at	DATETIME	First time seen

Table 2: known_ips
Stores trusted IPs per user.

Column Name	Type	Description
id (PK)	INT	Auto-increment
user_id (FK → users.user_id)	INT	Owner user
ip_address	VARCHAR	Known/trusted IP
scope	ENUM('user', 'global')	User-specific or global trusted IP

Table 3: raw_logs
Stores every parsed line of auth_logs.txt.

Column Name	Type	Description
log_id (PK)	INT	Auto-increment
timestamp	DATETIME	t_str converted
user_id (FK)	INT	Log's user
ip_address1	VARCHAR	IP from log
event_type	VARCHAR	DeviceCodeStart / DeviceCodeSuccess
status	VARCHAR	Success / Fail
raw_text	TEXT	Full original log line

Table 4: device_code_sessions
Tracks the login attempt from DeviceCodeStart → DeviceCodeSuccess.

Column Name	Type	Description
Session_id (PK)	INT	Auto-increment
User_id (FK)	INT	User involved
Start_time	DATETIME	DeviceCodeStart timestamp
Start_ip	VARCHAR	IP used at start

Success_time	DATETIME	DeviceCodeSuccess timestamp
Success_ip	VARCHAR	IP used at success
Is_suspicious	BOOLEAN	Unknown IP?
Confirmed_attack	BOOLEAN	Completed suspicious login?
Closed	BOOLEAN	Session ended and resolved

Table 5: alerts
Stores all alerts generated.

Column name	**Type**
Alert_id (PK)	INT
User_id (FK)	INT
Alert_type	ENUM
Message	TEXT
Ip_address	VARCHAR
Timestamp	DATETIME
Session_id	INT

CODE DEVELOPMENT

PSEUDO-CODE

```
START

SET device_code_starts = empty dictionary
SET ALERTS = empty list
SET TIME_WINDOW = 600 seconds

FUNCTION parse_logs(line):
    clean the line
    split line into: time_str, user, ip, event, status
    convert time_str to datetime (if fails, return null)
    RETURN record {time, user, ip, event, status}

FUNCTION get_known_ips():
    TRY to open known_ips.json
    IF file missing OR JSON wrong:
        RETURN empty dictionary
    ELSE:
        RETURN dictionary of known IPs

FUNCTION is_known_ip(user, ip, known_ips):
    user_list = known_ips[user] or empty list
    global_list = known_ips["_global"] or empty list

    IF ip in user_list:
        RETURN TRUE
    ELSE IF ip in global_list:
        RETURN TRUE
    ELSE:
        RETURN FALSE

FUNCTION detect_logs(parsed_logs, known_ips):

    FOR each log_entry IN parsed_logs:

        user = log_entry.user
        ip = log_entry.ip
        event = log_entry.event
        time = log_entry.time
        status = log_entry.status

        IF event == "DeviceCodeStart" AND status == "Success":
            CALL is_known_ip(user, ip)

            IF result is FALSE:
```

```
            suspicious = TRUE
            send_early_notification(user, ip)
            ADD early alert to ALERTS
         ELSE:
            suspicious = FALSE

         STORE device_code_starts[user] = {ip, time, suspicious}


      ELSE IF event == "DeviceCodeSuccess" AND status == "Success":
         start = device_code_starts[user]

         IF start exists:
            time_gap = time - start.time

            IF time_gap ≤ TIME_WINDOW:
               IF start.suspicious == TRUE:
                  send_final_notification(user, ip)
                  ADD final alert to ALERTS

            REMOVE device_code_starts[user]


FUNCTION send_early_notification(user, ip):
   PRINT "Early Warning: Suspicious device code for user"


FUNCTION send_final_notification(user, ip):
   PRINT "Alert: Device code phishing confirmed"


MAIN PROGRAM:

   known_ips = get_known_ips()

   parsed_log_list = empty list

   OPEN auth_logs.txt
   FOR each line:
      entry = parse_logs(line)
      IF entry not null:
         ADD entry to parsed_log_list

   CALL detect_logs(parsed_log_list, known_ips)

STOP
```

TESTING PHASE

Testing is the process of evaluating a software system to verify correctness, detect defects, and ensure it behaves as expected, and it is broadly divided into **White Box Testing** (structural) and **Black Box Testing** (functional).

White Box Testing
White Box Testing examines the internal code, logic, and control flow of the software, ensuring each path, condition, and loop executes correctly.

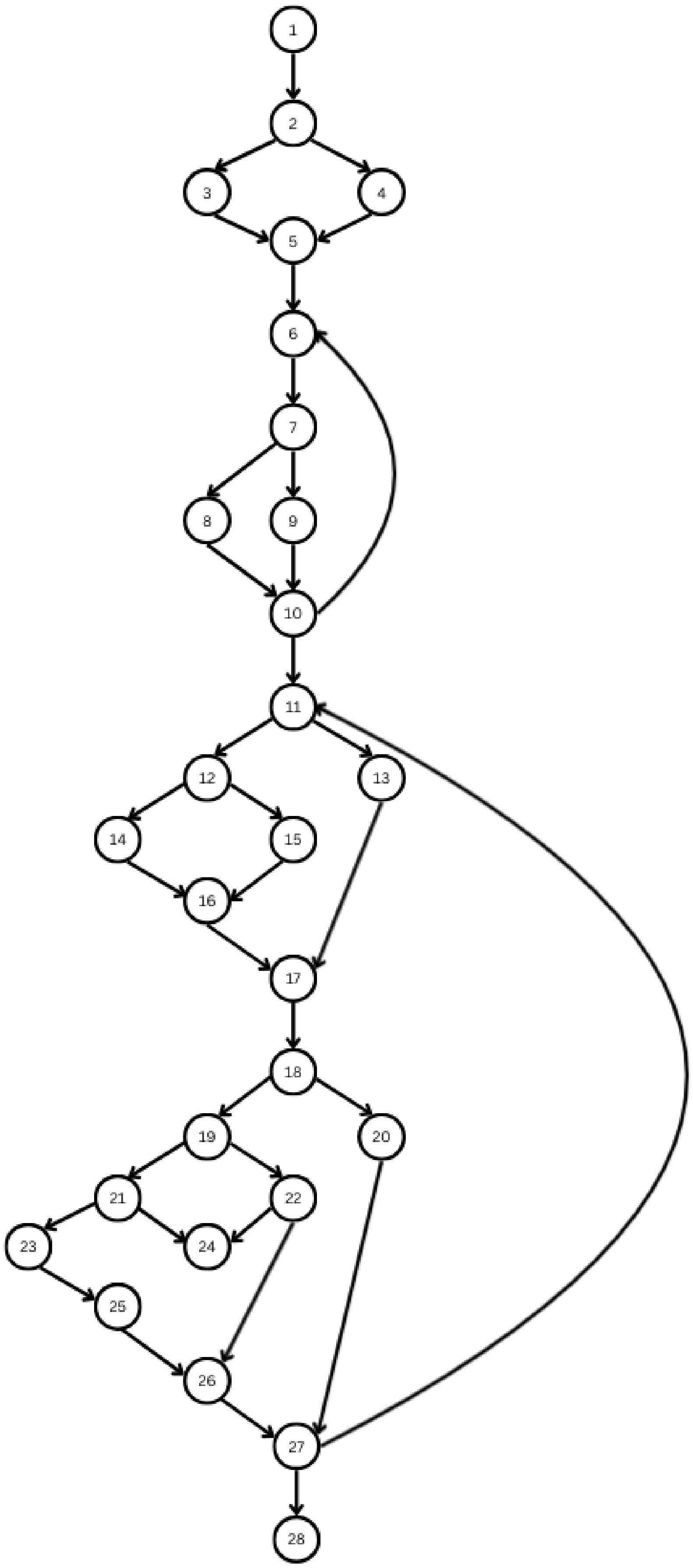
Black Box Testing
Black Box Testing evaluates the system solely based on inputs and outputs, ensuring the functionality meets requirements without viewing internal code.

Chosen Method: White Box Testing
Path Testing – Introduction
Path Testing focuses on identifying independent execution paths within the code, ensuring that all possible routes through the logic are covered at least once.

Flow Graph
A flow graph (control flow graph) represents the program using nodes for statements and edges for control transitions, helping visualize logical structure for path identification.

Approach
The approach involves calculating cyclomatic complexity, enumerating all independent logical paths, and designing targeted test cases to execute each path systematically.

FLOW GRAPH



CYCLOMATIC COMPLEXITY

Cyclomatic Complexity in Software Testing is a testing metric used for measuring the complexity of a software program. It is a quantitative measure of independent paths in the source code of a software program. Cyclomatic complexity can be calculated by using control flow graphs.

Mathematically, it is set of independent paths through the graph diagram. The Code complexity of the program can be defined using the formula:-

Method-1| **V(G) = E - N +2**,
Where, E = number of edges
N = number of Nodes

Here, E= 36 , N= 28
Therefore, V(G) = 36 - 28 + 2 = **10**

Method-2 | **V(G) = P + 1**,
Where, P = number of predicate nodes (nodes containing conditions).

Here, P = 9
Therefore, V(G) = 9 + 1 = **10**

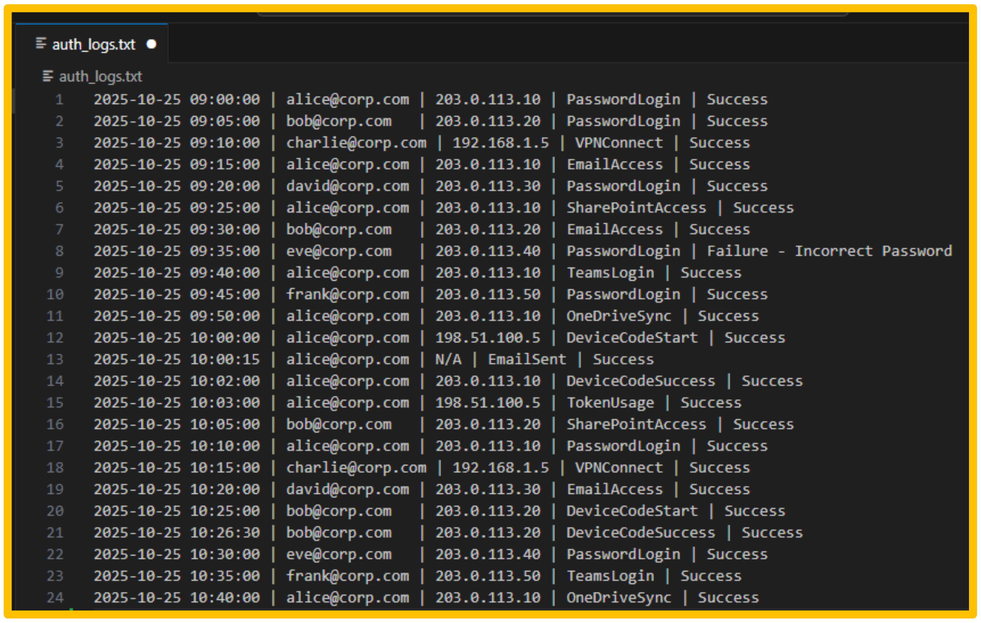
Method-3 | **V(G) = no. of bounded region + 1**
Here, no of bounded region = 9
Therefore, V(G) = 9 + 1 = **10**

INDEPENDENT PATH TESTING

S. No.	**Path**	**Condition**	**Test cases**	**Statement**	**Status**
1.	1, 2, 3, 5, 6, 7, 8, 10, 11, 12, 14, 16, 17, 18, 19, 21, 23, 25, 26, 27, 28	if suspicious: (Inside Start)	YES	User IP is NOT in known_ips	Alert Triggered: Early Notification sent
2.	1, 2, 4, 5, 6, 7, 8, 10, 11, 12, 14, 16, 17, 18, 19, 21, 23, 25, 26, 27, 28	try/except json.JSONDecodeError	NO	json.load(f) fails due to bad syntax	Exception Handled: Returns empty dict, prints "Error"
3.	1, 2, 3, 5, 6, 7, 9, 10, 11, 12, 14, 16, 17, 18, 19, 21, 23, 25, 26, 27, 28	try/except ValueError (Date Parse)	NO	datetime.strptime fails on invalid format	Skip Entry: Returns None, loop continues

4.	1, 2, 3, 5, 6, 7, 8, 10, 6, 7, 8, 10, 11, 12, 14, 16, 17, 18, 19, 21, 23, 25, 26, 27, 28	for line in f (File Loop)	YES	File contains multiple lines of logs	Loop Continues: Reads next line from file
5.	1, 2, 3, 5, 6, 7, 8, 10, 11, 13, 17, 18, 19, 21, 23, 25, 26, 27, 28	for l in parsed_logs	NO	parsed_logs list is empty	Process Ends: No logs to detect
6.	1, 2, 3, 5, 6, 7, 8, 10, 11, 12, 15, 16, 17, 18, 19, 21, 23, 25, 26, 27, 28	if event == "DeviceCodeStart"	NO	Event is "DeviceCodeSuccess" (goes to elif)	Check Next Condition: Proceed to elif block
7.	1, 2, 3, 5, 6, 7, 8, 10, 11, 12, 14, 16, 17, 18, 20, 27, 28	elif event == "DeviceCodeSuccess"	NO	Event is neither Start nor Success (e.g., "Logoff")	Ignore: Log entry is irrelevant, continue loop
8.	1, 2, 3, 5, 6, 7, 8, 10, 11, 12, 14, 16, 17, 18, 19, 22, 26, 27, 28	t - start < TIME_WINDOW	NO	Time difference is > 600 seconds	Timeout: Session expired, remove user tracking
9.	1, 2, 3, 5, 6, 7, 8, 10, 11, 12, 14, 16, 17, 18, 19, 21, 24, 26, 27, 28	if start["suspicious"]	NO	Original Start event was from Known IP	Safe: No alert needed, remove user tracking
10.	1, 2, 3, 5, 6, 7, 8, 10, 11, 13, 17, 18, 20, 27, 11, 13, 17, 18, 20, 27, 28	for l in parsed_logs (Main Loop)	YES	Multiple parsed logs exist in list	Loop Continues: Analyze next log entry

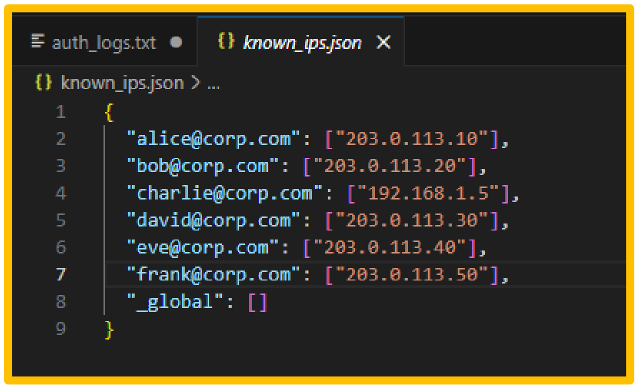
PROTOTYPE



auth_logs.txt ●

auth_logs.txt

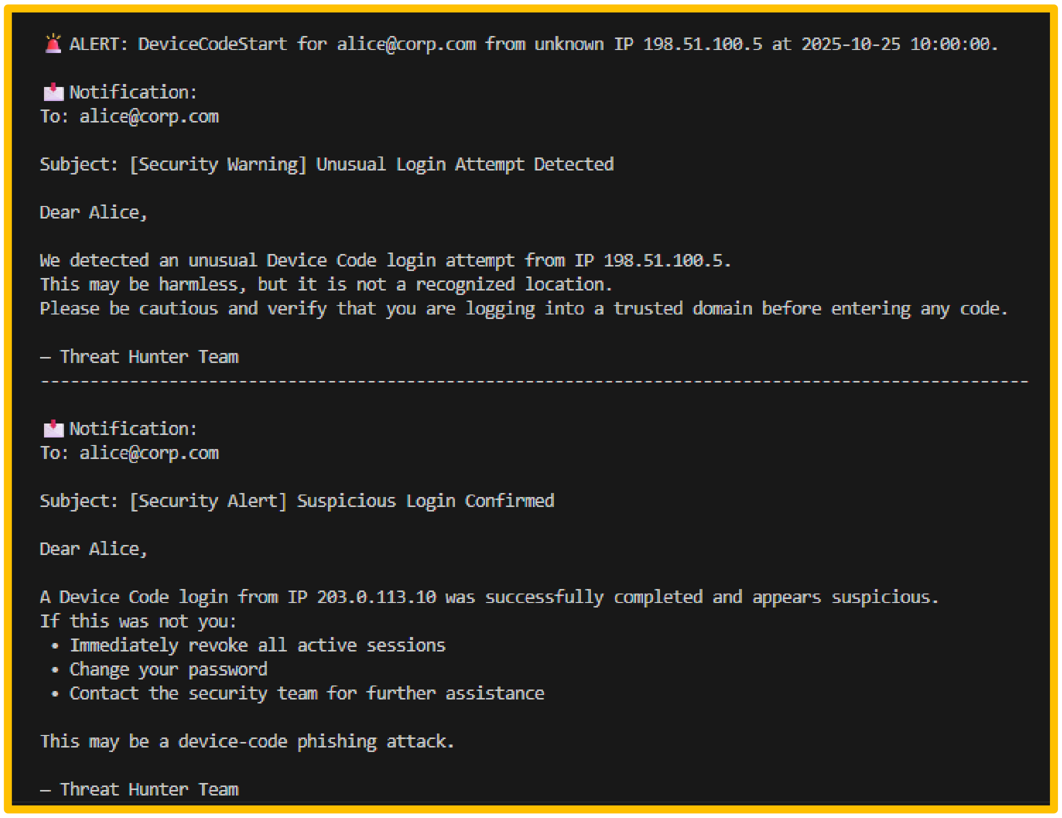
```
2025-10-25 09:00:00 | alice@corp.com | 203.0.113.10 | PasswordLogin | Success
2025-10-25 09:05:00 | bob@corp.com   | 203.0.113.20 | PasswordLogin | Success
2025-10-25 09:10:00 | charlie@corp.com | 192.168.1.5 | VPNConnect | Success
2025-10-25 09:15:00 | alice@corp.com | 203.0.113.10 | EmailAccess | Success
2025-10-25 09:20:00 | david@corp.com | 203.0.113.30 | PasswordLogin | Success
2025-10-25 09:25:00 | alice@corp.com | 203.0.113.10 | SharePointAccess | Success
2025-10-25 09:30:00 | bob@corp.com   | 203.0.113.20 | EmailAccess | Success
2025-10-25 09:35:00 | eve@corp.com   | 203.0.113.40 | PasswordLogin | Failure - Incorrect Password
2025-10-25 09:40:00 | alice@corp.com | 203.0.113.10 | TeamsLogin | Success
2025-10-25 09:45:00 | frank@corp.com | 203.0.113.50 | PasswordLogin | Success
2025-10-25 09:50:00 | alice@corp.com | 203.0.113.10 | OneDriveSync | Success
2025-10-25 10:00:00 | alice@corp.com | 198.51.100.5 | DeviceCodeStart | Success
2025-10-25 10:00:15 | alice@corp.com | N/A | EmailSent | Success
2025-10-25 10:02:00 | alice@corp.com | 203.0.113.10 | DeviceCodeSuccess | Success
2025-10-25 10:03:00 | alice@corp.com | 198.51.100.5 | TokenUsage | Success
2025-10-25 10:05:00 | bob@corp.com   | 203.0.113.20 | SharePointAccess | Success
2025-10-25 10:10:00 | alice@corp.com | 203.0.113.10 | PasswordLogin | Success
2025-10-25 10:15:00 | charlie@corp.com | 192.168.1.5 | VPNConnect | Success
2025-10-25 10:20:00 | david@corp.com | 203.0.113.30 | EmailAccess | Success
2025-10-25 10:25:00 | bob@corp.com   | 203.0.113.20 | DeviceCodeStart | Success
2025-10-25 10:26:30 | bob@corp.com   | 203.0.113.20 | DeviceCodeSuccess | Success
2025-10-25 10:30:00 | eve@corp.com   | 203.0.113.40 | PasswordLogin | Success
2025-10-25 10:35:00 | frank@corp.com | 203.0.113.50 | TeamsLogin | Success
2025-10-25 10:40:00 | alice@corp.com | 203.0.113.10 | OneDriveSync | Success
```



auth_logs.txt ● {} known_ips.json ✕

{} known_ips.json > ...

```json
{
  "alice@corp.com": ["203.0.113.10"],
  "bob@corp.com": ["203.0.113.20"],
  "charlie@corp.com": ["192.168.1.5"],
  "david@corp.com": ["203.0.113.30"],
  "eve@corp.com": ["203.0.113.40"],
  "frank@corp.com": ["203.0.113.50"],
  "_global": []
}
```



```
🚨 ALERT: DeviceCodeStart for alice@corp.com from unknown IP 198.51.100.5 at 2025-10-25 10:00:00.

📩 Notification:
To: alice@corp.com

Subject: [Security Warning] Unusual Login Attempt Detected

Dear Alice,

We detected an unusual Device Code login attempt from IP 198.51.100.5.
This may be harmless, but it is not a recognized location.
Please be cautious and verify that you are logging into a trusted domain before entering any code.

— Threat Hunter Team
----------------------------------------------------------------------------------------------------

📩 Notification:
To: alice@corp.com

Subject: [Security Alert] Suspicious Login Confirmed

Dear Alice,

A Device Code login from IP 203.0.113.10 was successfully completed and appears suspicious.
If this was not you:
 • Immediately revoke all active sessions
 • Change your password
 • Contact the security team for further assistance

This may be a device-code phishing attack.

— Threat Hunter Team
```

FUTURE SCOPE

- **Integration with Real SIEM Tools (Splunk, ELK, QRadar, Azure Sentinel):**
  The detection logic can be converted into SIEM-friendly rules so that alerts appear in a real security dashboard. Instead of reading JSON files, the system can collect logs directly from SIEM indexes. This would allow analysts to visualize attacks in real time, apply filters, generate timelines, and correlate device-code phishing with other security events from the same user or IP.

- **Using Real User Authentication Logs:**
  The project can be upgraded to process actual authentication logs from Microsoft Azure AD, Google Workspace, Okta, or internal corporate identity systems. By mapping fields such as IP address, device ID, timestamp, browser, and location, the tool can detect more accurate patterns. This helps in understanding real-world login behavior, making detections more reliable and industry-ready.

- **Building an Automated Alerting System:**
  Instead of saving alerts in a text file, the system can send notifications via email, Slack, Microsoft Teams, or a ticketing system like Jira. This would make the tool behave like a real SOC (Security Operations Center) component and help teams respond to phishing attempts faster.

- **Adding Machine Learning–Based Anomaly Detection:**
  The project can move beyond rule-based logic and use ML models to learn normal user behavior. For example, the system could learn usual login countries, device types, and times for each user. When unusual behavior occurs—like a sudden login from another country—it can generate more accurate alerts without fixed rules.

- **Mapping Detections to MITRE ATT&CK Framework:**
  Each detection can be linked to an ATT&CK technique (for example, “Valid Accounts," “Phishing," or “User Interaction"). This gives the project more professional value, since SOC teams rely on ATT&CK mapping for investigations and reporting.

- **Creating a Dashboard for Real-Time Monitoring:**
  A small web dashboard can be added using Flask, Django, or Streamlit. It can show graphs of suspicious events, trend lines, top risky IPs, and user-based alerts. This converts the script into a mini security product that can be demonstrated easily during interviews or internships.

- **Dynamic Threat Intelligence Integration:**
  The project can pull known malicious IPs, URLs, and domains from public threat-intel feeds. Matching these with the login logs will help detect phishing attempts more accurately and keep the tool updated with new threats.

- **Containerization and Deployment:**
  Docker or Kubernetes support can be added for easy deployment in enterprise environments. This helps run the detection engine at scale, processing thousands of logs per minute.

REFERENCES

1. Pressman, R.S., & Maxim, B.R. (2020). Software Engineering:A Practitioner’s Approach. 9th ed. McGraw-Hill

2. Aggarwal, K. K., & Singh, Y. (2007). Software Engineering. 3rd ed. New Age International Publishers.

3. Jalote, P. An Integrated Approach to Software Engineering, 3rd ed., Narosa Publishing House, 2005.

4. Sample projects