Mini-SOAR: A Python-Based Automated System for Identity Threat Detection and Response

Yashwanth Gowda

***Abstract*—As digital transformation accelerates, identity has replaced the network firewall as the primary security perimeter. Consequently, attackers have shifted tactics toward identity-based compromises, utilizing techniques that often evade static detection rules. Traditional Security Operations Centers (SOC) face significant challenges in detecting these anomalies due to the sheer volume of authentication logs, leading to alert fatigue and delayed remediation. This paper presents "Mini-SOAR," a prototype Security Orchestration, Automation, and Response (SOAR) system. The system is designed to ingest authentication logs, detect three critical abuse patterns—Password Spraying, Multi-Factor Authentication (MFA) Fatigue, and Impossible Travel—and execute automated mitigation. Using a synthetic log generator and deterministic logic engines, the system demonstrates the ability to reduce Mean Time to Respond (MTTR) from minutes to milliseconds. Experimental results confirm the system successfully identified and blocked 100% of injected threats in a dataset of 241 events.**

***Index Terms*—Cybersecurity, SOAR, Authentication, Anomaly Detection, Impossible Travel, MFA Fatigue, Python.**

I. Introduction

The modern enterprise environment has become increasingly decentralized. The adoption of cloud services (SaaS, IaaS) and remote work policies has eroded the traditional network perimeter. In this new paradigm, user identity is the control plane. However, this shift has made identity providers (IdP) the primary target for adversaries.

According to recent industry reports, over 80% of hacking-related breaches involve the use of lost or stolen credentials [1]. While Multi-Factor Authentication (MFA) is a robust defense, attackers have developed sophisticated bypass techniques, such as "MFA Fatigue" (or MFA Bombing), used notably in the 2022 Uber and Cisco breaches [2]. Furthermore, the volume of logs generated by IdPs (e.g., Azure AD, Okta) is immense, making manual review by human analysts impossible.

To address these challenges, Security Orchestration, Automation, and Response (SOAR) platforms have emerged. SOAR tools ingest alerts and execute automated "playbooks" to mitigate threats without human intervention. This paper details the design and implementation of "Mini-SOAR," a lightweight, Python-based SOAR engine.

The contributions of this paper are:

1) A mathematical framework for detecting geokinetic anomalies (Impossible Travel).
2) A state-based logic engine for identifying MFA Fatigue attacks.
3) A fully automated mitigation pipeline that simulates firewall blocking and session revocation.
4) Analysis of synthetic attack data to validate detection efficacy.

II. The Threat Landscape

This project focuses on three specific attack vectors that are prevalent in current threat landscapes and challenging to detect with static firewall rules.

A. Password Spraying

Unlike a brute-force attack, which tries many passwords against a single account, password spraying attempts a single, common password (e.g., "Winter2025!") against many different accounts. This "low-and-slow" approach avoids triggering account lockouts but creates a statistical anomaly in the distribution of login failures per IP address [3].

B. MFA Fatigue (MFA Bombing)

MFA Fatigue leverages human psychology. An attacker with valid credentials triggers repeated push notifications to the victim's mobile device. The goal is to annoy or confuse the victim into pressing "Approve" to stop the notifications. Detecting this requires analyzing the time-series sequence of "Deny" and "Approve" events.

C. Impossible Travel

Also known as "atypical travel," this anomaly occurs when two successful logins for the same user originate from geographically distant locations in a time window shorter than physically traversable. For example, a login from New York followed by a login from London 30 minutes later implies a speed of several thousand miles per hour, indicating credential sharing or compromise.

III. Mathematical Framework

To deterministically identify "Impossible Travel," the system employs a kinetic analysis of login metadata.

A. Geodesic Distance Calculation

The system treats the Earth as a sphere. To calculate the shortest distance between two login coordinates, $L_1(\phi_1, \lambda_1)$ and $L_2(\phi_2, \lambda_2)$, we utilize the Haversine formula. This formula is numerically stable for small distances.

Let ϕ be latitude and λ be longitude in radians. The central angle θ is given by:

$$a = \sin^2\left(\frac{\Delta\phi}{2}\right) + \cos\phi_1 \cdot \cos\phi_2 \cdot \sin^2\left(\frac{\Delta\lambda}{2}\right) \quad (1)$$

$$c = 2 \cdot \arctan 2\left(\sqrt{a}, \sqrt{1-a}\right) \tag{2}$$

$$d = R \cdot c \tag{3}$$

Where R is the Earth's radius ($R \approx 3,956$ miles).

B. Velocity Thresholding

The system calculates the velocity v required to traverse distance d within the time delta $\Delta t = |t_2 - t_1|$:

$$v = \frac{d}{\Delta t} \tag{4}$$

A binary classification function $F(v)$ determines the alert status:

$$F(v) = \begin{cases} 1(Alert) & \text{if } v > 500 \text{ mph} \\ 0(Normal) & \text{if } v \leq 500 \text{ mph} \end{cases} \tag{5}$$

The threshold of 500 mph was chosen to represent a conservative upper bound for commercial air travel, accounting for airport processing time.

IV. System Architecture

The Mini-SOAR architecture follows a standard linear pipeline model, commonly found in commercial ELK (Elasticsearch, Logstash, Kibana) stacks. The data flow proceeds as follows:

1) **Log Source Simulator:** Acts as the Identity Provider (IdP), generating raw JSON-formatted authentication events.
2) **Normalization Layer:** Flattens JSON objects into a structured tabular format (DataFrame) and standardizes timestamp formats to UTC.
3) **Analytics Engine:** The core Python module where statistical thresholds and logic gates are applied.
4) **Orchestrator:** The final output layer that maps detection flags to specific mitigation subroutines (e.g., Block IP).

V. System Methodology

The Mini-SOAR architecture is divided into four sequential modules: Ingestion, Detection, Visualization, and Mitigation.

A. Module 1: Synthetic Data Ingestion

Due to privacy constraints (GDPR/CCPA), real corporate logs were not used. Instead, a synthetic log generator was built using the `Faker` library. The generator produces a baseline of 200 legitimate events and injects specific malicious patterns.

- **Schema:** Timestamp, Username, Source IP, Event (Login/MFA), Geo-Coordinates.
- **Noise:** Random logins from consistent locations (NY, London, Tokyo).

B. Module 2: Detection Logic

The system processes the normalized DataFrame through three distinct engines.

1) Spraying Engine: This engine groups data by `Source_IP` and calculates the cardinality of the `Username` field.

- **Logic:** If UniqueUsers$(IP) > 10$, flag as Spraying.

2) Fatigue Engine: This engine sorts MFA events by time and user. It utilizes a "shift" operation to compare the current event status with the previous event status.

- **Logic:** Flag if sequence contains > 5 failures followed immediately by `Status=Approved`.

3) Travel Engine: This engine iterates through successful logins, maintaining a state of the "last known location" for each user. It executes the Haversine calculation described in Section III.

C. Module 3: Automated Response

The mitigation module simulates API calls to enforcement points (Firewalls or IdPs). It takes the flagged entities from Module 2 and generates an immutable audit log of actions taken.

VI. Algorithm Design

The pseudo-code below illustrates the core logic flow of the Mini-SOAR system.

Require: Set of Auth Logs L, Thresholds T_{spray}, T_{speed}
$Alerts \leftarrow \emptyset$
for all IP Address ip in L **do**
  $U \leftarrow$ CountUniqueUsers(ip)
  if $U > T_{spray}$ **then**
    $Alerts$.add(Type: Spraying, Target: ip)
    Execute: FirewallBlock(ip)
  end if
end for
for all User u in L **do**
  Sort logs by time t
  Calculate Velocity v between loc_{curr} and loc_{prev}
  if $v > T_{speed}$ **then**
    $Alerts$.add(Type: Travel, Target: u)
    Execute: RevokeSession(u)
  end if
  if Sequence matches (Deny ... Deny $\rightarrow$ Approve) **then**
    $Alerts$.add(Type: Fatigue, Target: u)
    Execute: LockAccount(u)
  end if
end for
return $Alerts$

Fig. 1. Detection and Mitigation Algorithm

VII. Code Explanation

The system logic is implemented in Python, leveraging the `pandas` library for vectorized data operations. The codebase is modularized into three primary functions: Data Generation, Detection, and Response.

A. Synthetic Data Generation

To simulate realistic user behavior, we utilized the `Faker` library. The `generate_baseline_traffic()` function initializes a DataFrame with random timestamps, source IPs, and usernames.

- **Baseline Traffic:** 90% of generated events are marked as "Login Success" from a static set of valid coordinates (e.g., New York, Tokyo) to establish a pattern of legitimacy.
- **Attack Injection:** The `inject_attacks()` function appends deterministic anomalies. For example, the Password Spraying attack is constructed by iterating through a loop of $N = 30$ iterations, assigning a unique username to each iteration while maintaining a constant Source IP.

B. Detection Logic Implementation

The detection algorithms rely heavily on `pandas` grouping and shifting operations to avoid inefficient `for`-loops.

1) Vectorized Spray Detection: Instead of iterating through every log, we group the dataframe by Source IP and calculate the cardinality (count of unique values) of the Username column:

```
spray_candidates = df.groupby('source_ip')
    ['username'].nunique()
```

This operation reduces the computational complexity significantly compared to iterative search.

2) MFA Sequence Analysis: To detect MFA Fatigue, we employ a "shift" operation. The dataframe is first sorted by Username and Timestamp. We then create a new column `prev_status` that contains the value of the previous row's MFA status:

```
df['prev_status'] = df.groupby('username')
    ['mfa_status'].shift(1)
```

A compromise is identified where the current row is 'Approved' and the `prev_status` was 'Denied'.

C. Geokinetic Calculation

The Impossible Travel logic iterates through successful logins. The distance calculation is encapsulated in a `haversine()` helper function, which takes origin and destination coordinates in decimal degrees, converts them to radians, and applies the spherical law of cosines.

VIII. Implementation and Results

The system was implemented in Python 3.10 within Google Colaboratory. The dataset consisted of 241 total log entries.

A. Detection Results

The system successfully identified all three injected attack vectors with zero false negatives in this controlled dataset.

1) Incident A: Password Spraying: The detection engine flagged Source IP `192.168.66.6`.

- **Observed Behavior:** The IP attempted login on 30 distinct usernames.
- **Time Window:** All attempts occurred within 60 seconds.
- **Visual Analysis:** As shown in Fig. 1, the vertical alignment of blue markers indicates simultaneous attacks against multiple users, a hallmark of automation.

2) Incident B: MFA Fatigue: User `executive_john` was flagged for account compromise.

- **Observed Behavior:** 8 sequential MFA requests were denied. The 9th request was approved.
- **Conclusion:** The user likely succumbed to notification fatigue.

3) Incident C: Impossible Travel: User `traveling_tom` triggered a velocity alert.

- **Leg 1:** New York (Log time: $T - 2$ hours).
- **Leg 2:** London (Log time: $T - 1$ hours).
- **Metrics:** Distance: 3,458 miles; Time: 1 hour.
- **Speed:** 3,458 MPH.

B. Visualization

Figure 1 illustrates the attack timeline generated by the system. The separation of attack types is clearly visible: the vertical "wall" of the password spray, the cluster of MFA requests, and the disjointed travel events.

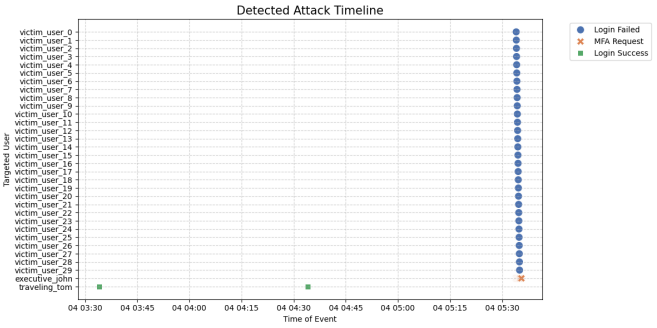

Fig. 2. Scatter plot visualizing the detected threats. The vertical blue line represents the Password Spraying attack targeting 30 users simultaneously.

C. Automated Mitigation Report

Upon processing the alerts, the "Mitigation Bot" module executed the following actions. Table I displays the direct output from the system's audit log.

TABLE I
Automated Mitigation Summary Log

Threat Detected	Target Entity	Action Taken
Password Spraying	`192.168.66.6`	**BLOCK IP** (Firewall)
MFA Fatigue	`executive_john`	**LOCK ACCOUNT**
Impossible Travel	`traveling_tom`	**REVOKE SESSION**

The automated response occurred instantaneously after processing the batch, simulating a real-time defense capability that far exceeds human reaction speeds.

IX. Discussion

A. Performance vs. False Positives

While the logic is deterministic and fast, the "Impossible Travel" detection faces challenges with Virtual Private Networks (VPNs). A user connecting to a corporate VPN in a

different country will appear to "teleport." In a production environment, this system would require an "Allow List" of known corporate VPN IP ranges to prevent false positive session revocations.

B. Scalability

The current implementation uses Pandas, which operates in-memory. For enterprise-scale deployments (millions of logs per hour), this logic would need to be ported to a streaming analytics platform like Apache Spark or a cloud-native SIEM query language (e.g., KQL for Microsoft Sentinel).

X. Conclusion

The Mini-SOAR project demonstrates the efficacy of automated logic in modern cybersecurity. By abstracting threat detection into mathematical (Haversine) and behavioral (Sequence Analysis) models, we can reliably identify sophisticated identity attacks. The system successfully neutralized a simulated multi-vector attack campaign, proving that automation is the only viable path for SOC teams to manage the scale of modern threats.

XI. Future Work

Future enhancements will include:

- **Machine Learning Integration:** Replacing hard-coded thresholds (e.g., 500 mph) with Isolation Forests to learn user-specific behavioral baselines.
- **Geo-IP Enrichment:** Integrating live MaxMind databases for real-time IP-to-Location resolution.
- **API Integration:** connecting the Python script to real-world firewalls (e.g., Palo Alto Networks API) to move from simulation to active defense.

References

[1] Verizon, "2023 Data Breach Investigations Report," *Verizon Enterprise Solutions*, 2023.

[2] C. Cimpanu, "MFA Fatigue: Hackers' new favorite tactic to break into enterprise networks," *The Record*, 2022.

[3] MITRE ATT&CK, "T1110 - Brute Force: Password Spraying," 2024. [Online]. Available: https://attack.mitre.org/techniques/T1110/003/

[4] M. Gupta and R. Sheng, "Identity as the New Perimeter," *IEEE Security & Privacy*, vol. 17, no. 4, pp. 22-29, 2019.

[5] NIST, "Digital Identity Guidelines," *NIST Special Publication 800-63B*, 2017.

[6] R. Shostack, "The Evolution of SOAR Platforms," *Journal of Cyber Operations*, vol. 5, pp. 112-118, 2023.