Automated Identity Threat Response (SOAR)

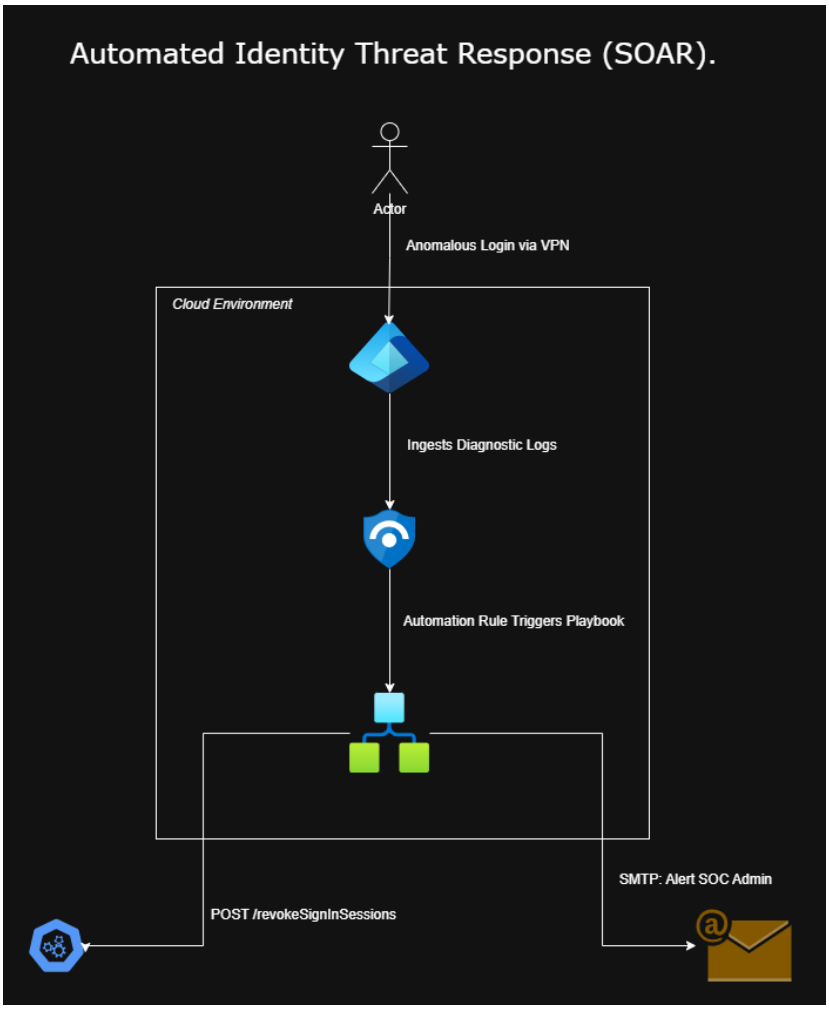
🚀 Building a Zero-Touch SOAR Pipeline: Automated Identity Threat Response

📌 The Problem: The "Impossible Travel" Dilemma

In modern cloud environments, speed is the ultimate deciding factor during an identity compromise. If a threat actor successfully authenticates using stolen credentials or hijacked session tokens, relying on a human SOC analyst to manually review the alert and revoke access gives the attacker too much time to pivot.

To reduce the Mean Time to Respond (MTTR) from minutes to milliseconds, I engineered a fully automated Security Orchestration, Automation, and Response (SOAR) pipeline using Microsoft Sentinel and Azure Logic Apps.

🏗️ Architecture Flow



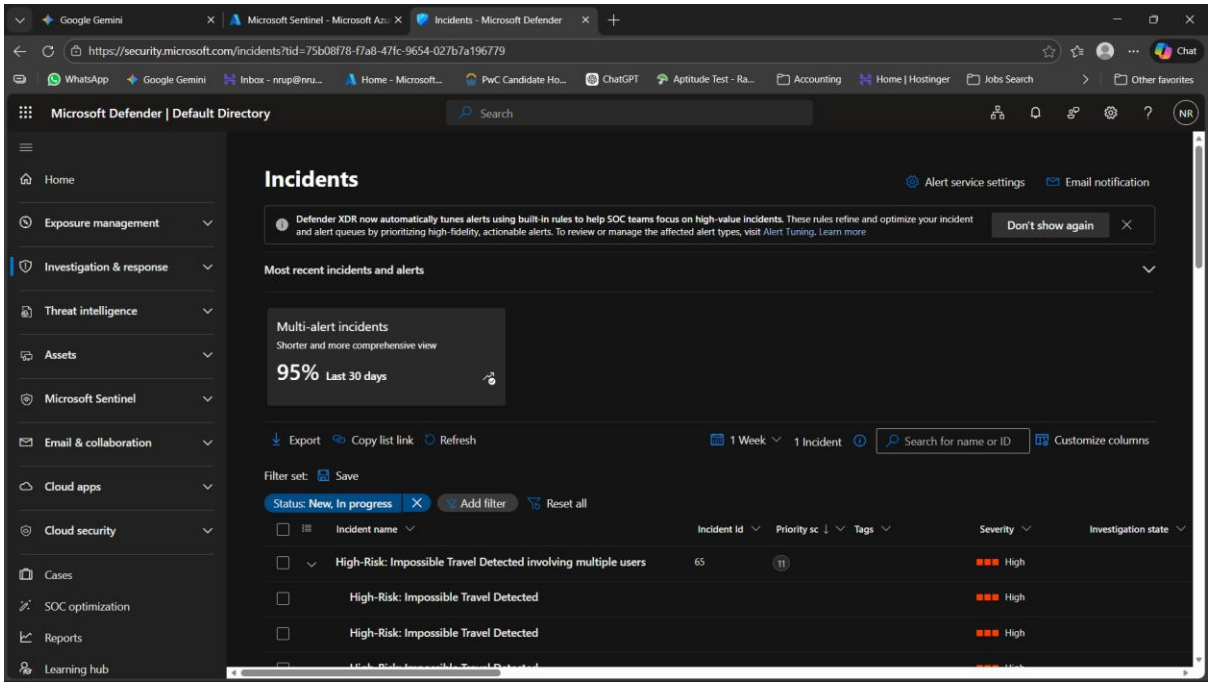
1. **Detection:** Microsoft Entra ID flags an anomalous "Impossible Travel" login (e.g., a simultaneous login from India and the US).

2. **Alerting:** Microsoft Sentinel ingests the logs and generates a High-Risk Incident.
3. **Trigger:** A Sentinel Automation Rule instantly fires an Azure Logic App playbook.
4. **Execution:** The Logic App parses the JSON payload, isolates the specific compromised Entra ID user, and sends a POST request to the Microsoft Graph API.
5. **Containment:** All active session and refresh tokens for the compromised user are instantly revoked.
6. **Notification:** An automated HTML email is sent via SMTP alerting the SOC team of the contained threat.

🛠️ Step-by-Step Build Guide

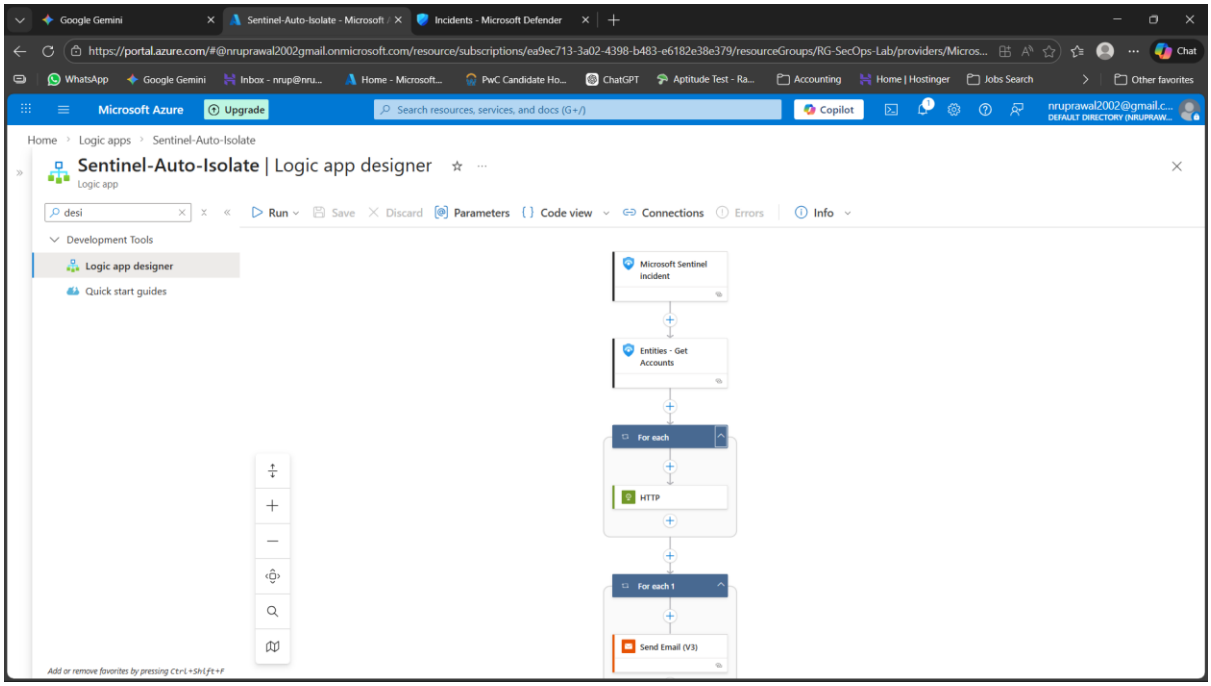
Step 1: Configuring the SIEM Trigger

The foundation of the automation is the **Microsoft Sentinel Incident** trigger. When an incident is created, Sentinel passes a massive JSON array containing all the metadata about the attack.



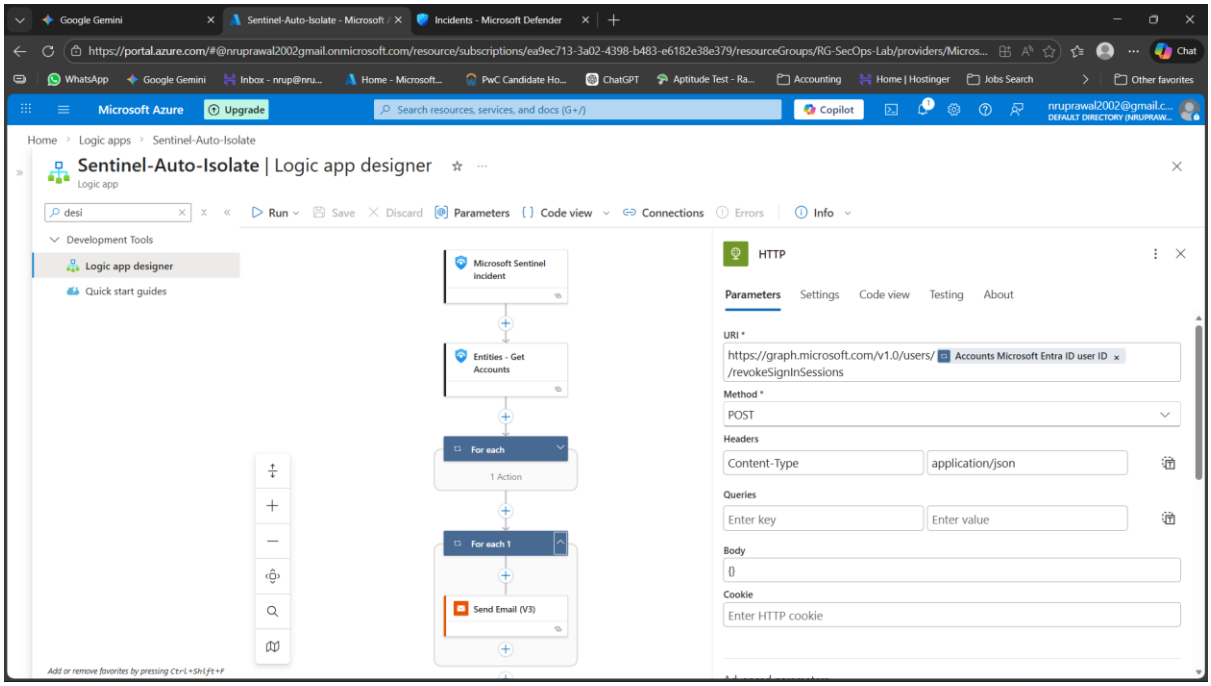
Step 2: The Logic App & Data Parsing

Because a single incident can involve multiple compromised accounts, I implemented an **Entities - Get Accounts** helper action followed by a **For each** loop. This ensures that the SOAR platform dynamically scales to isolate every single compromised identity found in the alert payload, rather than just the first one.



Step 3: The Graph API Remediation (The Muscle)

To forcefully kick the attacker out of the environment, the Logic App makes an authenticated HTTP call to the Microsoft Graph API using an OAuth 2.0 Service Principal.



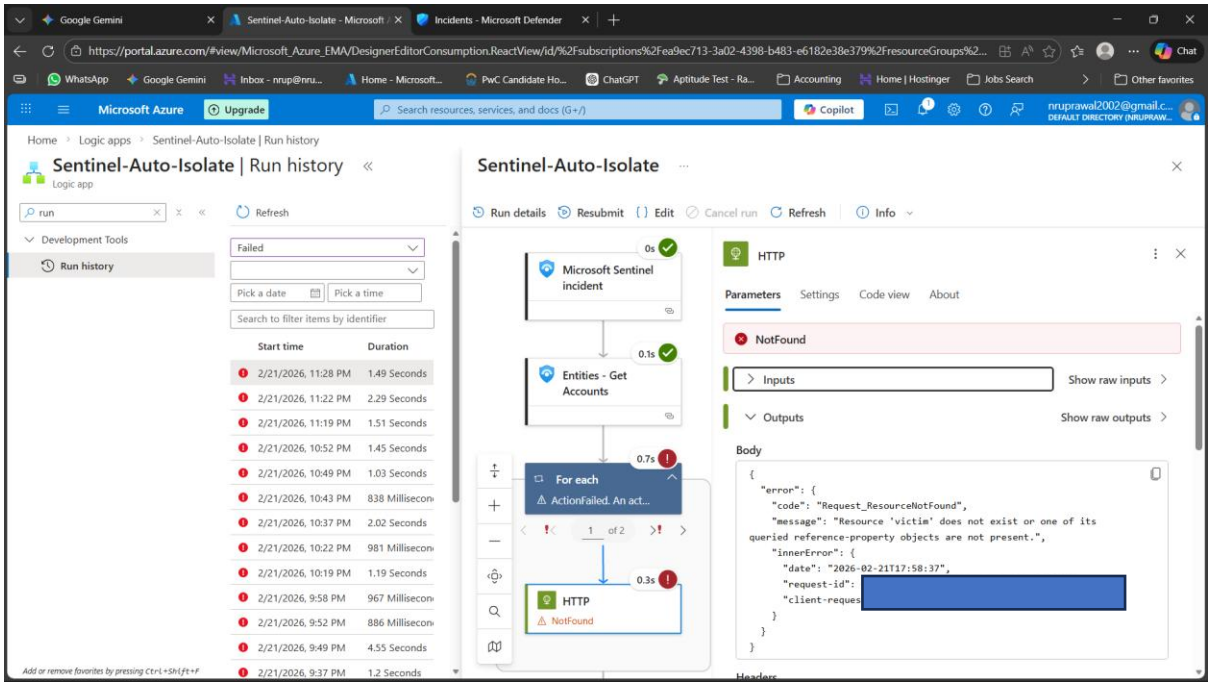
🚧 Overcoming Real-World Engineering Hurdles

Building this pipeline revealed several strict architectural requirements that standard tutorials often miss.

Hurdle 1: API Routing Strictness (404 Not Found & 400 Bad Request)

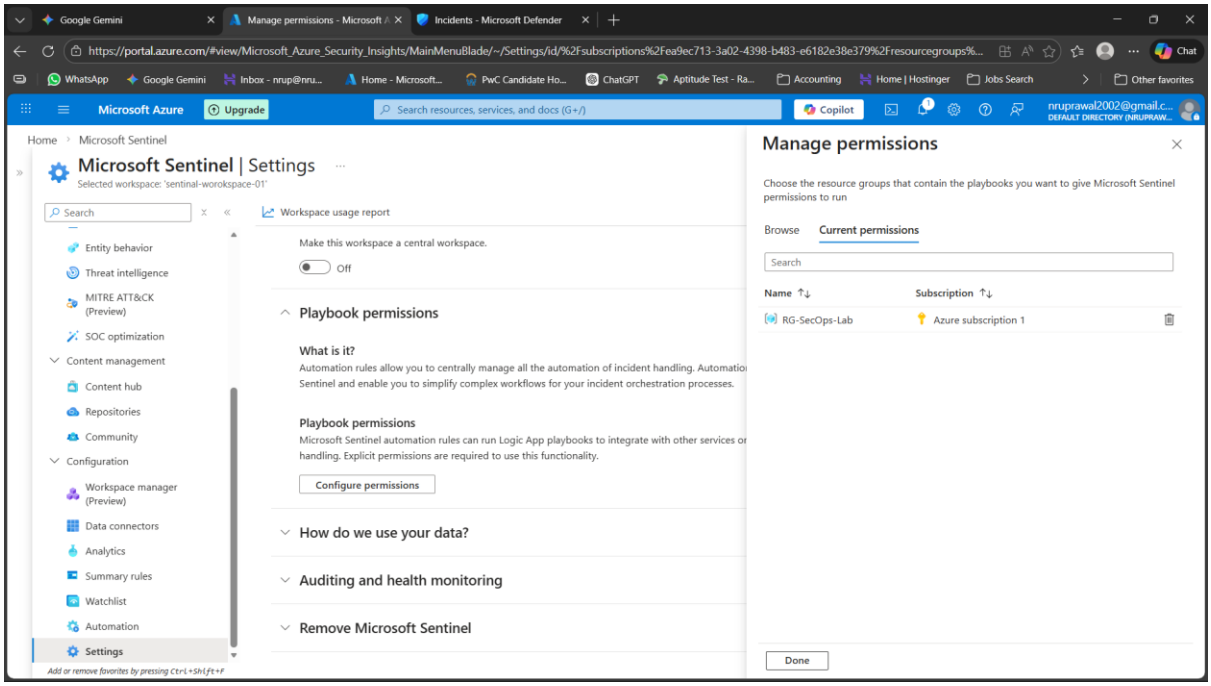
Initially, the Graph API rejected the POST commands. Through raw JSON log analysis, I identified two critical API rules:

1. **Endpoint Targeting:** You cannot pass a display name into the URI; the API requires the exact alphanumeric Entra ID Object ID to execute the session revocation.
2. **Formatting Strictness:** Even a single blank space in the dynamic URI string will break the web request. The URI must be a perfectly tight, unbroken string.



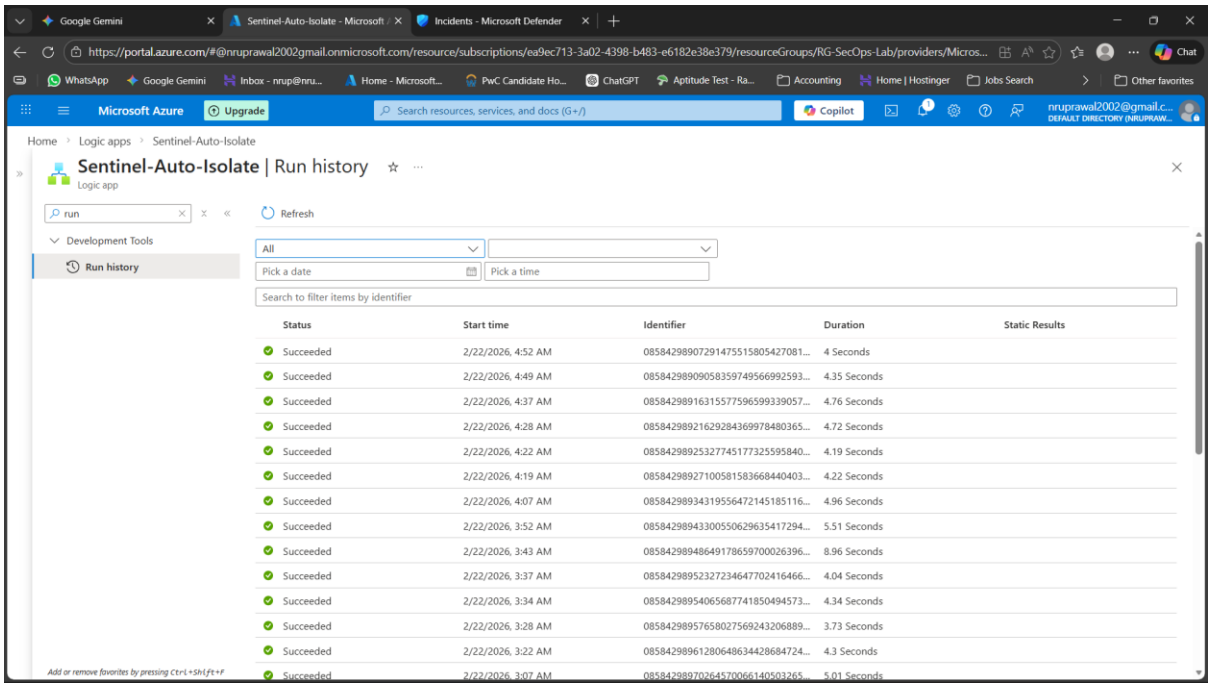
Hurdle 2: The "Phantom" RBAC Permission

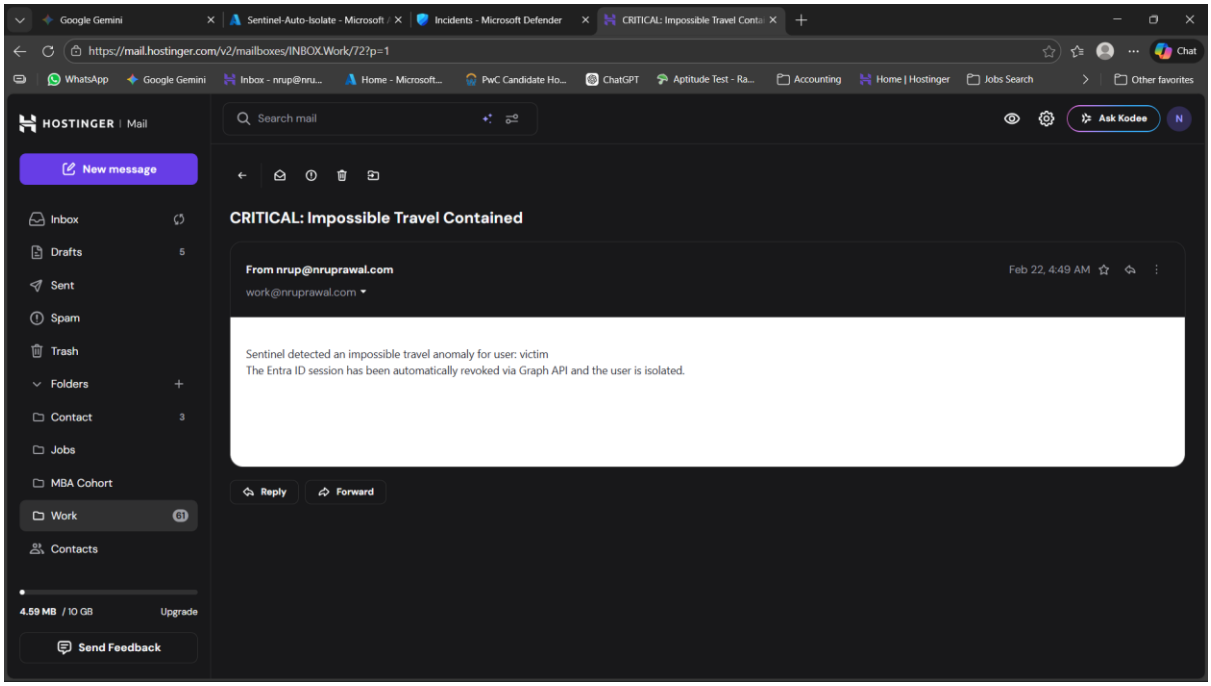
Even as an Azure Global Administrator, Sentinel initially refused to trigger the playbook, throwing a "No Microsoft Sentinel permissions" error. This occurs because the Microsoft Sentinel background service account (Azure Security Insights) requires its own explicit authority to reach into other resource groups. I resolved this by navigating to the global Sentinel Settings and manually assigning the **Microsoft Sentinel Automation Contributor** role to the specific resource group, securely bridging the SIEM and SOAR tools.



🎯 Final Outcome

By successfully connecting the SIEM directly to the Identity Provider via API automation, this project entirely removes the human element from the immediate containment phase. The threat actor is forcefully logged out of all active Microsoft sessions before an analyst even opens the ticket.





Note: The email says from nrup@nruprawal.com because it was set to sent from that email in SMTP automation