Cloud SOAR Project

Detection and Automated Response to Cloud Security Events

Student: Hedir Bel Arbya

January 6, 2026

Contents

Chapter 1 Project Overview

The objective of this project is to design and implement a simplified yet functional **SOAR (Security Orchestration, Automation, and Response)** pipeline dedicated to cloud security monitoring and incident handling. The main goal is to demonstrate how security events can be detected, analyzed, and processed automatically using open-source tools in a cloud environment.

The project focuses on the centralization and analysis of security-related events generated within a simulated cloud context. These events represent typical security scenarios such as suspicious access attempts, sensitive configuration changes, or identity and access management (IAM) operations. By collecting and analyzing these events, the system aims to identify potentially risky behaviors that require further investigation.

A key aspect of the project is the automation of alert processing. Once a sensitive event is detected, predefined rules and workflows are applied to evaluate the risk level of the event. Automation is implemented using workflow orchestration, allowing the system to react consistently and efficiently without manual intervention. This approach reflects modern security operations practices, where automation is essential to handle large volumes of security data.

In addition, the project prepares and implements the integration with an incident response platform. Security alerts generated by the detection pipeline are forwarded to an incident management system, enabling structured tracking, investigation, and response. This step transforms raw security events into actionable incidents, which is a fundamental principle of SOAR architectures.

The entire environment is deployed on an AWS EC2 instance using containerized services. This deployment approach ensures modularity, scalability, and ease of integration between components. Although the project uses simulated security events, the architecture is designed to be compatible with real cloud security logs, making it adaptable to real-world use cases.

Overall, this project provides a practical demonstration of how a SOAR pipeline can be built using open-source technologies to improve cloud security monitoring, automate alert handling, and support incident response processes.

Chapter 2 Architecture and Tools

2.1 Infrastructure Overview

The project infrastructure is deployed on an **AWS EC2 instance (t3.large)** running Ubuntu Linux. This instance hosts all security components using **Docker** and **Docker Compose** to ensure modularity, portability, and ease of deployment.

Due to limited system resources, several optimizations were applied:

- Swap memory was configured to prevent out-of-memory crashes
- Containers were started sequentially to reduce CPU and RAM spikes
- Persistent Docker volumes were used to avoid data loss

This infrastructure simulates a realistic cloud SOC environment while remaining suitable for an academic project.

2.2 Containerized Architecture

All components were deployed as Docker containers, each responsible for a specific role in the security pipeline:

- **Graylog**: Central log management and SIEM platform
- **MongoDB**: Stores Graylog configuration and metadata
- **OpenSearch**: Indexes and stores log data
- **n8n**: Workflow automation and SOAR engine
- **(Planned) TheHive**: Incident response and case management

Docker networking was used to allow secure communication between containers while exposing only necessary services to the outside.

2.3 Tools and Their Roles

2.3.1 Graylog (SIEM)

Graylog is used as the central Security Information and Event Management (SIEM) platform. Its responsibilities include:

- Ingesting cloud security logs
- Parsing and indexing events
- Filtering and organizing logs into streams
- Generating alerts for suspicious activity

In this project, Graylog receives simulated AWS CloudTrail events related to IAM activity.

2.3.2 n8n (SOAR Automation)

n8n is used as the automation and orchestration layer of the project. It receives alerts from Graylog via HTTP webhooks and executes predefined workflows.

Its main roles include:

- Receiving security alerts
- Parsing event fields (user, IP address, risk level)
- Applying conditional logic
- Preparing automated response actions

This component represents the core of the SOAR pipeline.

2.3.3 Docker

Docker ensures that all tools run in isolated, reproducible environments. It also simplifies deployment and troubleshooting by allowing services to be stopped, restarted, or scaled independently.

2.3.4 AWS EC2

AWS EC2 provides the cloud infrastructure required to simulate a real-world cloud security environment. Security groups were configured to expose only required ports such as:

- 9000 (Graylog Web Interface)
- 9001 (Graylog GELF Input)
- 5678 (n8n Web Interface and Webhook)

2.4 Global Architecture Flow

The project follows a layered security architecture inspired by real SOC and SOAR implementations.

Simulated Cloud Logs $\downarrow$
Graylog (SIEM) $\downarrow$
Alert / Webhook $\downarrow$
n8n (Automation & Orchestration) $\downarrow$
TheHive (Incident Management)

2.5 Data Flow Explanation

1. Cloud security events are simulated using scripts and HTTP requests
2. Logs are ingested into Graylog through a GELF HTTP input
3. Events are filtered and organized using streams
4. Alerts are generated based on detection logic
5. Alerts are forwarded to n8n via webhook
6. n8n processes the alert and prepares automated response actions

This design ensures separation of concerns between detection, orchestration, and response.

2.6 Design Choices and Constraints

The project intentionally uses simulated data instead of a real CloudTrail integration in order to:

- Reduce cloud costs
- Maintain full control over test scenarios
- Focus on detection and automation logic

Despite this limitation, the architecture remains fully compatible with real cloud log sources.

2.7 Scalability and Future Improvements

The architecture was designed to be extensible. Future improvements include:

- Integration with TheHive for incident creation
- Enrichment of IP addresses and users
- Automated remediation actions
- Advanced detection rules and risk scoring

This makes the project a solid foundation for a complete SOAR platform.

Chapter 3 Graylog Configuration and SIEM Implementation

3.1 Graylog Deployment Architecture

Graylog was deployed on an AWS EC2 instance using Docker and Docker Compose. The deployment includes the following components:

- **Graylog**: SIEM platform responsible for log ingestion, search, and alerting
- **MongoDB**: Stores Graylog configuration and metadata
- **OpenSearch**: Stores indexed log data

To ensure data persistence and avoid configuration loss during container restarts, Docker volumes were configured for MongoDB, OpenSearch, and Graylog data directories.

The deployment was executed on a **t3.medium EC2 instance** with memory optimization and swap configuration to ensure system stability.

3.2 Container Configuration and Resource Management

During the deployment, memory-related issues were encountered due to limited system resources. To resolve this:

- Java heap size for OpenSearch was limited to 512 MB
- Swap memory was configured on the EC2 instance
- Containers were started in a controlled order (MongoDB → OpenSearch → Graylog)

These optimizations ensured stable operation of the SIEM environment.

3.3 Graylog Web Interface and Authentication

The Graylog web interface was exposed on port `9000` and accessed using the public IP address of the EC2 instance.

Administrator authentication was configured using:

- A secure password secret
- A SHA-256 hashed root password

After successful authentication, the Graylog dashboard was accessible and operational.

3.4 Input Configuration for Cloud Logs

To simulate AWS CloudTrail log ingestion, a **GELF HTTP Input** was created in Graylog with the following configuration:

- Input type: GELF HTTP
- Listening address: All network interfaces
- Port: 9001
- Scope: Global

Security group rules were updated on the EC2 instance to allow inbound traffic on the configured input port.

3.5 Simulation of CloudTrail Events

Since a real AWS CloudTrail integration was not used at this stage, cloud security events were simulated using HTTP requests. A test event representing an IAM access key creation was sent using the `curl` command.

The simulated log contained the following fields:

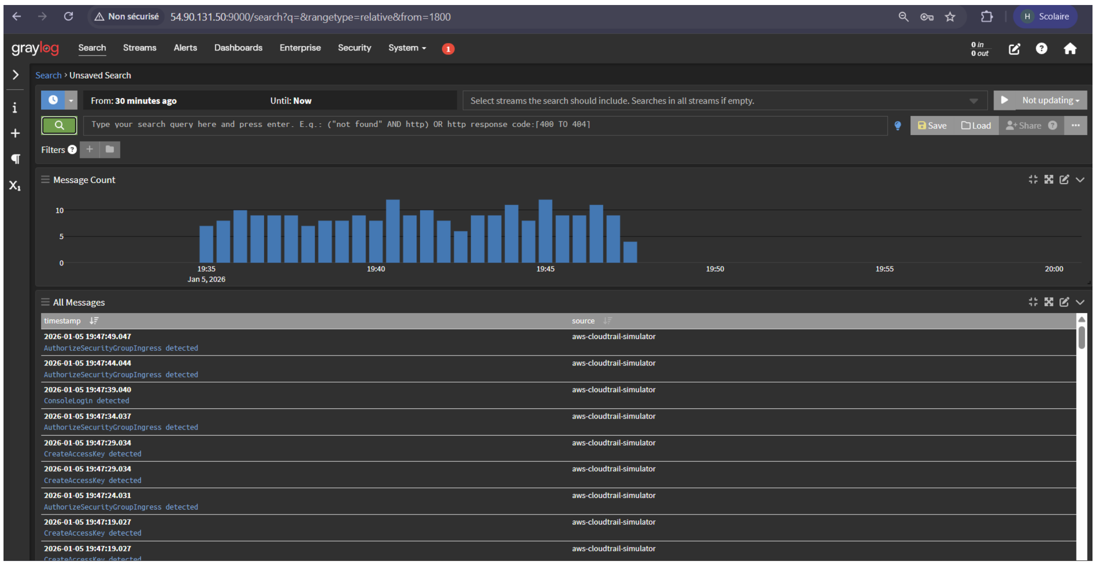
- eventName (CreateAccessKey)
- user
- source IP address
- AWS region

This approach allowed validation of the ingestion pipeline without requiring a real AWS account.

3.6 Log Ingestion Verification

After sending simulated events, logs were successfully received and indexed by Graylog. The events were visible in the search interface, confirming:

- Correct input configuration
- Proper network connectivity
- Successful parsing of JSON log fields



timestamp	source
2026-01-05 19:47:49.047 AuthorizeSecurityGroupIngress detected	aws-cloudtrail-simulator
2026-01-05 19:47:44.044 AuthorizeSecurityGroupIngress detected	aws-cloudtrail-simulator
2026-01-05 19:47:39.040 ConsoleLogin detected	aws-cloudtrail-simulator
2026-01-05 19:47:34.037 AuthorizeSecurityGroupIngress detected	aws-cloudtrail-simulator
2026-01-05 19:47:29.034 CreateAccessKey detected	aws-cloudtrail-simulator
2026-01-05 19:47:29.034 CreateAccessKey detected	aws-cloudtrail-simulator
2026-01-05 19:47:24.031 AuthorizeSecurityGroupIngress detected	aws-cloudtrail-simulator
2026-01-05 19:47:19.027 CreateAccessKey detected	aws-cloudtrail-simulator
2026-01-05 19:47:19.027 CreateAccessKey detected	aws-cloudtrail-simulator

Figure 3.1: Graylog receiving and displaying simulated CloudTrail logs

3.7 Stream Creation for Cloud Events

A dedicated stream named **AWS CloudTrail** was created to isolate cloud-related security events. The stream was configured using rules based on log fields such as:

```
eventName exists
```

This allowed structured separation of cloud security logs from other events.

3.8 Detection Preparation

The stream was used as the foundation for future detection rules, including:

- IAM access key creation
- Privilege escalation attempts

- Suspicious access patterns

Detected events from this stream were later forwarded to the automation layer (n8n) for further processing.

3.9 Integration Readiness

Graylog was successfully prepared to:

- Forward alerts via webhook
- Integrate with SOAR workflows
- Support incident response automation

This completed the SIEM layer of the project and enabled the transition to orchestration and response.

Chapter 4 n8n Workflow Automation and SOAR Logic

4.1 Role of n8n in the Project

n8n is used as the **Security Orchestration, Automation and Response (SOAR)** component of the project. Its role is to receive alerts generated by the SIEM (Graylog), analyze their content, and orchestrate automated response actions.

In this architecture, n8n acts as an intermediate layer between detection (Graylog) and response (future integration with TheHive).

4.2 Workflow Design

The n8n workflow was designed to process security alerts related to cloud identity and access management (IAM) activity. Alerts are transmitted from Graylog to n8n using an HTTP webhook.

The workflow follows a simple but realistic SOAR logic:

- Receive a security alert
- Parse event fields
- Evaluate the risk level
- Trigger appropriate response actions

This design mirrors real-world SOC automation pipelines.

4.3 Workflow Trigger: Webhook

The workflow begins with a **Webhook node**, which exposes an HTTP endpoint. Graylog (or manual tests) sends JSON-formatted alerts to this endpoint.

The webhook listens for HTTP POST requests containing security event data such as:

- Event name
- User involved
- Source IP address
- Risk level

This approach allows easy integration with any SIEM or log source.

4.4 Event Parsing and JSON Structure

Alerts received by n8n are formatted as JSON objects. An example of a simulated alert payload is shown below:

```
{
  "eventName": "CreateAccessKey",
  "user": "test-user",
  "source_ip": "41.227.0.1",
  "risk": "HIGH"
}
```

Each field in the JSON payload has a specific purpose:

- `eventName`: Type of detected cloud activity
- `user`: IAM user responsible for the action
- `source_ip`: IP address associated with the event
- `risk`: Severity level assigned by the detection logic

n8n automatically converts this JSON payload into internal workflow data, making each field accessible to subsequent nodes.

4.5 Conditional Logic and Decision Making

After parsing the event, a **conditional (IF) node** is used to evaluate the risk level. This node checks whether the value of the `risk` field matches predefined criteria.

For example:

- If `risk = HIGH`, the event is treated as critical
- If `risk = LOW`, the event may only be logged

This conditional logic allows the workflow to branch and apply different response strategies depending on the severity of the alert.

4.6 Automated Response Preparation

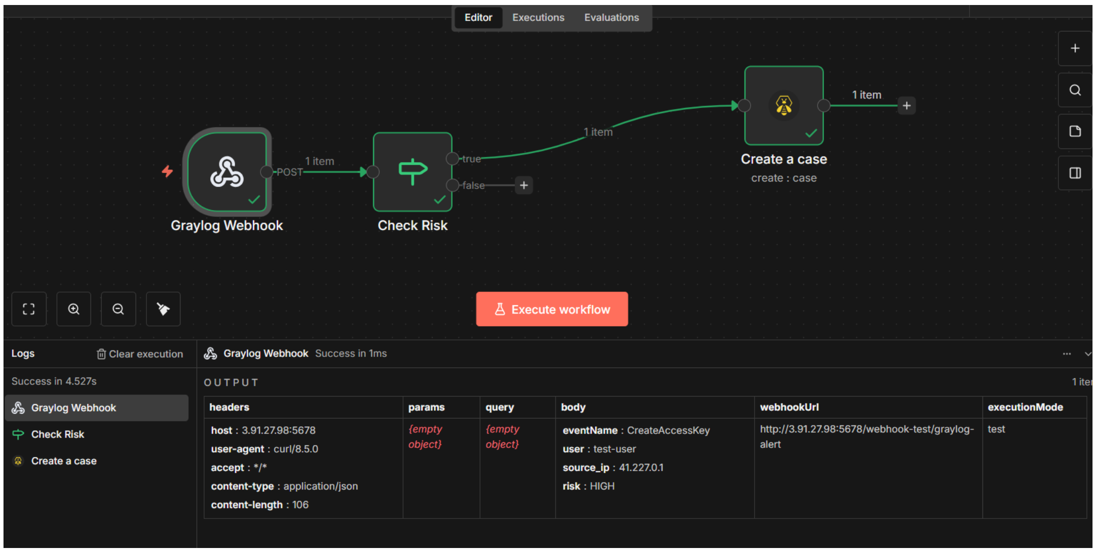
For high-risk events, the workflow prepares automated response actions. In the current project scope, these actions are simulated and include:

- Logging the alert as a critical incident
- Preparing incident data for external systems
- Generating structured output for future integration

This design ensures that the workflow can later be extended to trigger real remediation actions such as disabling IAM keys or creating incidents.

4.7 Workflow Visualization

The complete workflow is visualized in the n8n interface, showing the logical sequence of nodes from webhook trigger to response handling.



headers	params	query	body	webhookUrl	executionMode
host : 3.91.27.98:5678 **user-agent** : curl/8.5.0 **accept** : */* **content-type** : application/json **content-length** : 106	*{empty object}*	*{empty object}*	**eventName** : CreateAccessKey **user** : test-user **source_ip** : 41.227.0.1 **risk** : HIGH	http://3.91.27.98:5678/webhook-test/graylog-alert	test

Figure 4.1: n8n workflow processing Graylog security alerts

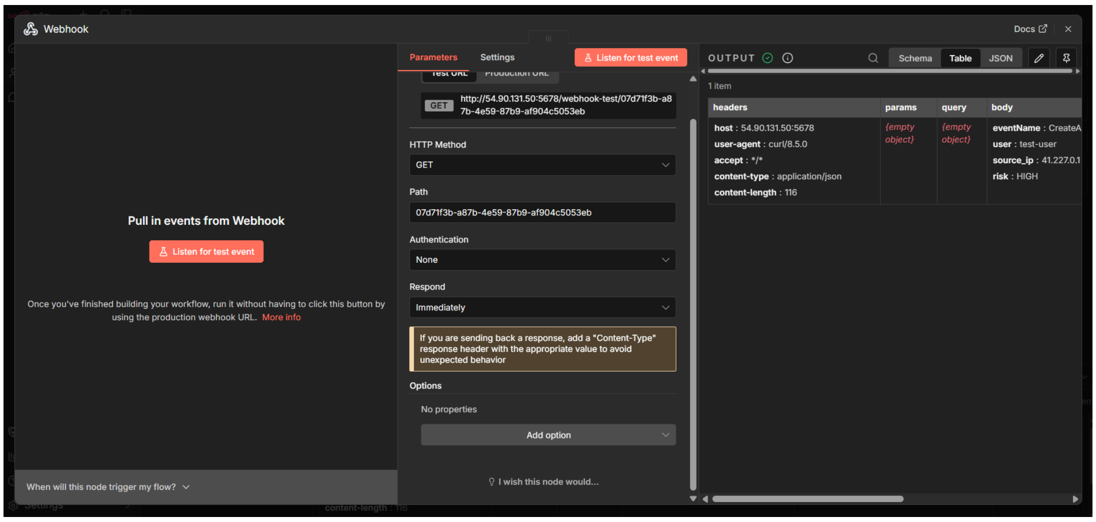
4.8 Workflow Testing and Validation

To validate the workflow, a manual test was performed by sending a POST request to the webhook endpoint using the `curl` command. The request contained a simulated security event with a high-risk level.

Upon execution:

- The webhook successfully received the event

- The JSON payload was parsed correctly
- The conditional logic was executed
- The workflow completed without errors



Once you've finished building your workflow, run it without having to click this button by using the production webhook URL. More info

GET http://54.90.131.50:5678/webhook-test/07d71f3b-a87b-4e59-87b9-af904c5053eb

HTTP Method: GET

Path: 07d71f3b-a87b-4e59-87b9-af904c5053eb

Authentication: None

Respond: Immediately

If you are sending back a response, add a "Content-Type" response header with the appropriate value to avoid unexpected behavior

Options

No properties

1 item

headers	params	query	body
host : 54.90.131.50:5678 user-agent : curl/8.5.0 accept : */* content-type : application/json content-length : 116	{empty object}	{empty object}	eventName : CreateA user : test-user source_ip : 41.227.0.1 risk : HIGH

Figure 4.2: Successful execution of the n8n workflow after webhook test

4.9 Integration Readiness and Extensibility

The n8n workflow was designed with extensibility in mind. It can easily be expanded to:

- Create incidents in TheHive
- Enrich IP addresses using threat intelligence
- Trigger automated remediation actions
- Send notifications to SOC analysts

This makes n8n a central component of the SOAR pipeline implemented in this project.

Chapter 5 TheHive Integration and Incident Management

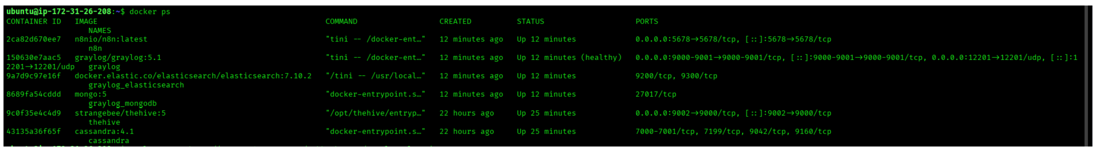
5.1 Objective

TheHive was integrated into the project as an **incident response and case management platform**. Its role is to automatically receive critical security events detected by Graylog and processed by n8n, and to convert them into structured security cases for analysis and tracking.

5.2 Deployment of TheHive

TheHive 5 was deployed on the same AWS EC2 instance as the other components using Docker. It runs alongside Cassandra as its backend database.

- Docker image: `strangebee/thehive:5`
- Web interface exposed on port `9002`
- Cassandra used for data persistence



```
ubuntu@ip-172-31-26-208:~$ docker ps
CONTAINER ID   IMAGE                                                  COMMAND                  CREATED          STATUS                    PORTS
                  NAMES
2ca82d670ee7   n8nio/n8n:latest                                       "tini -- /docker-ent…"   12 minutes ago   Up 12 minutes             0.0.0.0:5678→5678/tcp, [::]:5678→5678/tcp
                  n8n
150630e7aac5   graylog/graylog:5.1                                    "tini -- /docker-ent…"   12 minutes ago   Up 12 minutes (healthy)   0.0.0.0:9000-9001→9000-9001/tcp, [::]:9000-9001→9000-9001/tcp, 0.0.0.0:12201→12201/udp, [::]:1
2201→12201/udp   graylog
9a7d9c97e16f   docker.elastic.co/elasticsearch/elasticsearch:7.10.2   "/tini -- /usr/local…"   12 minutes ago   Up 12 minutes             9200/tcp, 9300/tcp
                  graylog_elasticsearch
8689fa54cddd   mongo:5                                                "docker-entrypoint.s…"   12 minutes ago   Up 12 minutes             27017/tcp
                  graylog_mongodb
9c0f35e4c4d9   strangebee/thehive:5                                   "/opt/thehive/entryp…"   22 hours ago     Up 25 minutes             0.0.0.0:9002→9000/tcp, [::]:9002→9000/tcp
                  thehive
43135a36f65f   cassandra:4.1                                          "docker-entrypoint.s…"   22 hours ago     Up 25 minutes             7000-7001/tcp, 7199/tcp, 9042/tcp, 9160/tcp
                  cassandra
```

Figure 5.1: TheHive 5 running in Docker on the AWS EC2 instance

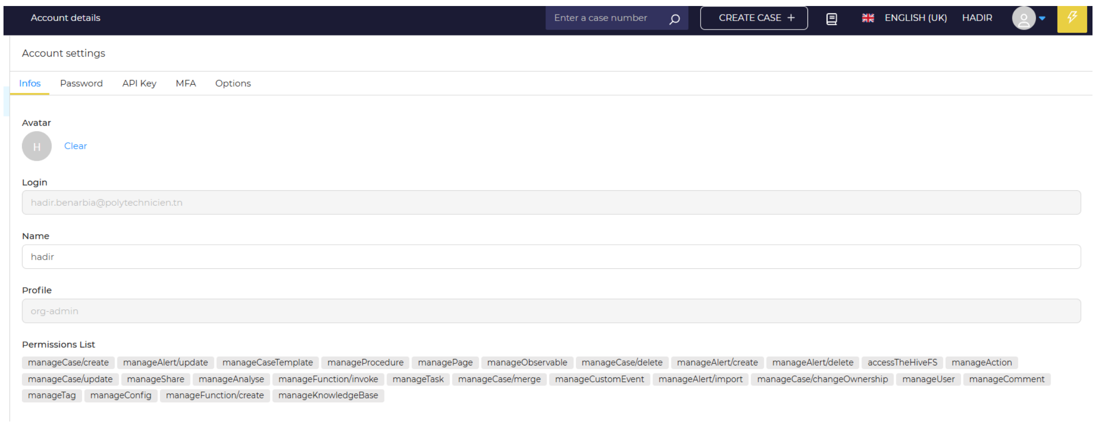
5.3 Initial Configuration

After deployment, the following configuration steps were performed:

- Creation of an organization
- Creation of an administrator account

- Generation of an API key for automation

These steps were mandatory in order to allow external tools (n8n) to create cases programmatically.



Account settings

Avatar

Login: hadir.benarbia@polytechnicien.tn

Name: hadir

Profile: org-admin

Permissions List

manageCase/create manageAlert/update manageCaseTemplate manageProcedure managePage manageObservable manageCase/delete manageAlert/create manageAlert/delete accessTheHiveFS manageAction manageCase/update manageShare manageAnalyse manageFunction/invoke manageTask manageCase/merge manageCustomEvent manageAlert/import manageCase/changeOwnership manageUser manageComment manageTag manageConfig manageFunction/create manageKnowledgeBase

Figure 5.2: Organization and administrator user creation in TheHive

5.4 Integration with n8n

The integration between n8n and TheHive was successfully implemented using the official **TheHive 5 node** in n8n.

The workflow logic is as follows:

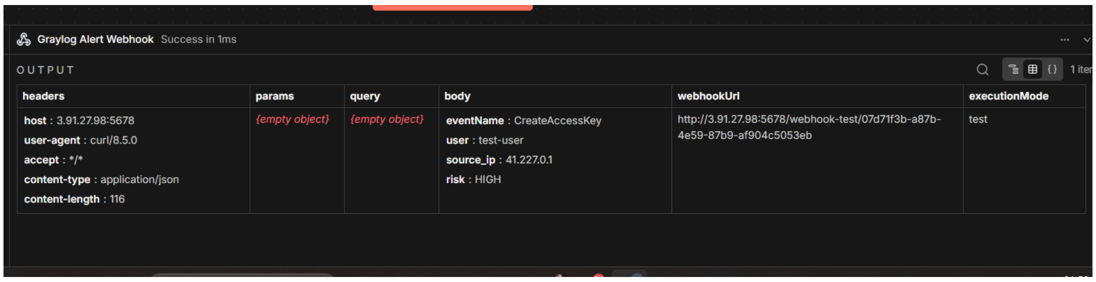
Graylog Alert → n8n Webhook → Risk Evaluation → TheHive Case Creation

Only events classified as **HIGH risk** trigger the creation of a case in TheHive.

5.5 Automated Case Creation

When a high-risk event is received (for example, IAM access key creation or suspicious activity), n8n automatically creates a case in TheHive with the following attributes:

- Case title
- Description including event details
- Severity level
- Tags related to the event type



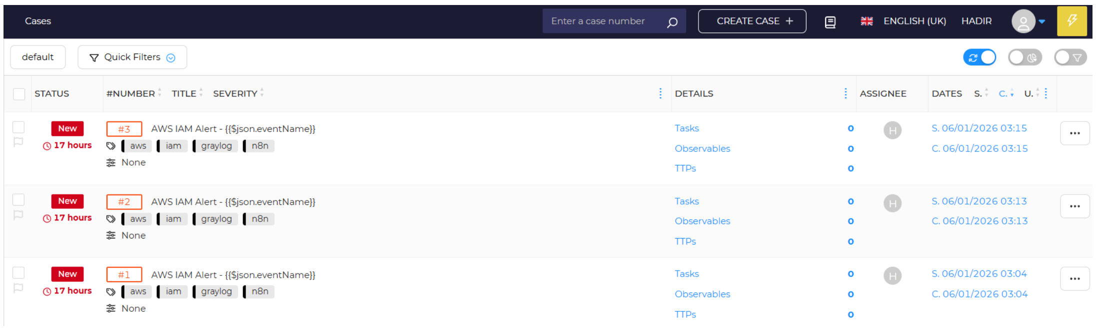
headers	params	query	body	webhookUrl	executionMode
host : 3.91.27.98:5678 **user-agent** : curl/8.5.0 **accept** : */* **content-type** : application/json **content-length** : 116	*{empty object}*	*{empty object}*	**eventName** : CreateAccessKey **user** : test-user **source_ip** : 41.227.0.1 **risk** : HIGH	http://3.91.27.98:5678/webhook-test/07d71f3b-a87b-4e59-87b9-af904c5053eb	test

Figure 5.3: Security case automatically created in TheHive

5.6 Case Visualization and Tracking

All created cases are visible in the TheHive interface, where analysts can:

- View event details
- Track the investigation lifecycle
- Assign severity and status
- Prepare response actions



STATUS	#NUMBER	TITLE	SEVERITY	DETAILS		ASSIGNEE	DATES S. C. U.
New 17 hours	#3	AWS IAM Alert - {{$json.eventName}} aws iam graylog n8n None		Tasks Observables TTPs	0 0 0	H	S. 06/01/2026 03:15 C. 06/01/2026 03:15
New 17 hours	#2	AWS IAM Alert - {{$json.eventName}} aws iam graylog n8n None		Tasks Observables TTPs	0 0 0	H	S. 06/01/2026 03:13 C. 06/01/2026 03:13
New 17 hours	#1	AWS IAM Alert - {{$json.eventName}} aws iam graylog n8n None		Tasks Observables TTPs	0 0 0	H	S. 06/01/2026 03:04 C. 06/01/2026 03:04

Figure 5.4: List of cases generated from automated alerts

5.7 Validation and Testing

The integration was validated by manually triggering test events through n8n and verifying:

- Successful workflow execution
- Correct risk-based filtering
- Automatic creation of cases in TheHive

This confirms the correct operation of the SOAR pipeline from detection to incident management.

Chapter 6 Conclusion

This project successfully demonstrated the design and implementation of a cloud security monitoring and automation pipeline using open-source technologies. The primary objective was to simulate a Security Orchestration, Automation and Response (SOAR) environment capable of detecting security-relevant events and responding to them automatically.

Throughout the project, a complete architecture was deployed on a cloud-based infrastructure, integrating Graylog as a Security Information and Event Management (SIEM) platform, n8n as an automation and orchestration engine, and TheHive as an incident management system. This integration enabled the centralized collection of security events, automated alert processing, and structured incident handling.

One of the key achievements of this work is the successful deployment and configuration of Graylog for log ingestion and analysis. Simulated cloud security events were generated and processed, allowing the definition of detection rules and alerting mechanisms that reflect realistic security scenarios, such as suspicious access or privilege-related actions.

The orchestration layer implemented with n8n played a crucial role in automating the response workflow. By using webhooks and conditional logic, security alerts were evaluated and escalated only when they met predefined risk criteria. This approach demonstrates how automation can reduce manual intervention and improve response efficiency in security operations.

In addition, the integration of TheHive provided a structured incident management capability. Security alerts generated by the system were transformed into alerts or cases within TheHive, allowing analysts to track, investigate, and manage incidents through a dedicated interface. This confirms the feasibility of implementing an end-to-end SOAR pipeline using open-source tools.

Although the project relied on simulated security events due to access limitations to real cloud audit logs, the implemented architecture remains fully compatible with real-world cloud environments. The solution is scalable and can be extended to ingest real cloud provider logs, enrich events with threat intelligence, and automate remediation actions.

In conclusion, this project provides a solid and practical foundation for advanced cloud security orchestration. It highlights the benefits of automation in security monitoring and incident response, and it demonstrates how open-source tools can be effectively combined to build a functional SOAR platform suitable for both academic and professional contexts.