Key Models and Features

Accounts

Code links:

1. Models
2. Serializers
3. Views

This section describes the primary models and their corresponding features in the Accounts app of the SIEM. The models manage user accounts, roles, permissions, and employee information, ensuring secure access and operations within the platform.

User Model

The User model represents the system's users, including their authentication details, roles, and status.

Key Features

- **Custom User Manager**: Supports user creation with email as the unique identifier, enforcing email validation and password management.
- **Role Association**: Each user is linked to a specific role, defining their permissions and access within the SIEM.
- **Active Status**: Users can be marked as active or inactive, allowing for easy management of user accounts.
- **Staff Status**: The `is_staff` attribute determines access to administrative functions, controlling who can access sensitive areas of the platform.
- **Permission Handling**: Users can have permissions derived from their assigned role, facilitating fine-grained access control.

Key Fields

- **user_id**: Unique identifier for each user (UUID).
- **email**: User's email address (unique).
- **role**: Foreign key to the Role model.
- **is_active**: Boolean indicating if the user account is active.
- **is_staff**: Boolean indicating if the user has staff access.

Role Model

The Role model defines various roles within the system, such as Admin and Analyst, and manages the permissions associated with each role.

Key Features

- **Role Management**: Allows for the definition of various roles and associated permissions within the SIEM, supporting role-based access control.
- **Dynamic Permissions**: Roles can have multiple permissions assigned, providing flexibility in managing user capabilities.
- **Permission Check**: The `has_permission` method enables checking if a role has a specific permission, facilitating access validation.

Employee Model

The Employee model associates additional employee details with the User model, capturing relevant information for organizational purposes.

Key Features

- **One-to-One Relationship**: Each employee is linked to a user, ensuring a direct correlation between user credentials and employee data.
- **Employee ID Generation**: Automatically generates a unique employee ID when creating a new employee record, ensuring each employee can be uniquely identified.
- **Custom Save Method**: Overrides the save method to manage the automatic generation of employee IDs based on existing records.

Key Fields

- **user**: One-to-one link to the User model.
- **employee_id**: Unique employee identifier.
- **first_name**, **last_name**: Employee's personal information.

Permission Model

The Permission model defines specific permissions that can be assigned to roles, enabling a more granular approach to access control.

Key Features

- **Unique Permissions**: Each permission is uniquely defined, allowing roles to be assigned multiple permissions without duplication.
- **Easy Integration with Roles**: Permissions can be easily linked to roles through the RolePermission model, streamlining permission management.

Key Fields

- **permission_id**: Unique identifier for each permission (UUID).
- **permission_name**: Name of the permission (e.g., "view_logs").

RolePermission Model

The RolePermission model serves as a junction table between roles and permissions, managing their many-to-many relationship.

Key Features

- **Unique Combinations**: Ensures that each combination of role and permission is unique within the system, preventing duplicates.
- **Facilitates Role Management**: Simplifies the management of which permissions are assigned to which roles.

Key Fields

- **role**: Foreign key to the Role model.
- **permission**: Foreign key to the Permission model.

Summary of Key Features Across Models

- **Robust User Management**: The user model provides a secure framework for managing users, utilizing email-based authentication and strong password practices.
- **Dynamic Role and Permission Structure**: The role and permission models facilitate role-based access control, enabling customized permissions for different user roles.
- **Employee Integration**: The employee model links user accounts to organizational details, enhancing management and reporting capabilities.
- **Logging and Auditing**: The incorporation of logging in various methods ensures that actions are tracked, aiding in auditing and compliance efforts.

Alerts

Code links:

1. Models
2. Serializers
3. Views

This section describes the primary models and their corresponding features in the Alerts app of the SIEM.

AlertSeverity (Enum Class)

- **Purpose**: Defines various levels of severity for alerts.
- **Key Features**:
  - Severity levels include INFO, LOW, MEDIUM, HIGH, and CRITICAL.

InvestigationStatus (Enum Class)

- **Purpose**: Defines the status of an investigation.
- **Key Features**:
  - Statuses include OPEN, IN PROGRESS, and CLOSED.

Alert Model

- **Purpose**: Represents an alert generated by a rule based on event data.
- **Key Features**:
  - **id**: Unique identifier for the alert (AutoField).
  - **rule**: Foreign key relationship to the Rule model indicating which rule triggered the alert.
  - **event**: Foreign key relationship to the BronzeEventData model for event details.
  - **severity**: CharField representing the severity of the alert (with choices defined by AlertSeverity).
  - **comments**: Optional text field for additional notes on the alert.
  - **String Representation**: Returns a descriptive string about the alert, including its ID, severity, associated event, and rule name.
  - **Meta Options**: Alerts are ordered by their creation date in descending order.

InvestigateAlert Model

- **Purpose**: Represents an investigation associated with an alert.
- **Key Features**:
  - **id**: Unique identifier for the investigation (AutoField).
  - **alert**: One-to-one relationship with the Alert model.
  - **assigned_to**: Foreign key referencing the user to whom the alert is assigned.
  - **status**: CharField indicating the status of the investigation (with choices defined by InvestigationStatus).
  - **notes**: Optional text field for additional notes on the investigation.
  - **String Representation**: Returns a descriptive string about the investigation, including the associated alert, assigned user, and status.

Serializers

AlertSerializer

- **Purpose**: Serializes the Alert model for API responses.
- **Key Features**:
  - Includes nested representations of the related event and rule.
  - Custom representation for formatting event and rule data.

InvestigateAlertSerializer

- **Purpose**: Serializes the InvestigateAlert model for API responses.
- **Key Features**:
  - Includes nested representation of the associated alert and the user assigned_to.

ViewSets

AlertViewSet

- **Purpose**: Handles API requests related to alerts.
- **Key Features**:
  - Supports filtering by severity and rule name.
  - Custom actions for assigning alerts to analysts and retrieving the latest alerts.

InvestigateAlertViewSet

- **Purpose**: Handles API requests related to investigations of alerts.
- **Key Features**:
  - Supports filtering by assigned analyst email and alert severity.
  - Custom actions for updating investigations and counting the status of investigations.
  - Logic to limit investigations returned based on the user's role (ADMIN or ANALYST).

Summary

The alerts module is designed to manage alerts generated by specific rules based on event data. It includes clear severities for the alerts, a structured investigation model to track who is working on each alert, and comprehensive API views to interact with these models effectively. The combination of models, serializers, and viewsets provides a robust foundation for alert management and investigation workflows in your application.

Core

1. Models
2. Serializers
3. Views

Rule Model

- **Purpose**: Defines a set of rules that can trigger alerts based on conditions.
- **Key Features**:
  - **Fields**:
    - `id`: Auto-incrementing primary key for the rule.
    - `name`: CharField representing the name of the rule.

    - `description`: A brief description of the rule.
    - `conditions`: A TextField that holds the logic or conditions which the rule evaluates (JSON).
    - `severity`: CharField that defines the severity level of the rule, with choices including INFO, LOW, MEDIUM, HIGH, and CRITICAL. The default is set to MEDIUM.
  - **String Representation**: Displays the name of the rule followed by its severity level.

Serializers

RuleSerializer

- **Purpose**: Serializes the Rule model for API responses and requests.
- **Key Features**:
  - **Fields**:
    - `id`: The unique identifier of the rule.
    - `name`: The name of the rule.
    - `description`: The description of the rule.
    - `conditions`: The logic or conditions of the rule.

ViewSets

RuleViewSet

- **Purpose**: Provides the view and API handling logic for interacting with the Rule model.
- **Key Features**:
  - **Queryset**: Retrieves all Rule instances.
  - **Serializer**: Utilizes the RuleSerializer to serialize and deserialize Rule data.

Summary

The core app is focused on managing rules that dictate how alerts are triggered based on specific conditions. The Rule model encapsulates the rule logic and associated severity. A corresponding RuleSerializer is used for API interaction, and the RuleViewSet provides a view for managing rules through standard API actions.

This simple and extensible design supports rule-based alert generation in your SIEM system, forming the core logic for event processing and alerting.

Logs

1. Models
2. Serializers
3. Views

BronzeEventData Model

- **Purpose**: Captures detailed event logs related to a system or application. Likely to represent low-level events (bronze-tier data) in a SIEM architecture.
- **Key Features**:
  - **Fields**:
    - `created_at`: Timestamp when the event was created.
    - `priority`, `h_version`, `iso_timestamp`, `hostname`, `app_name`, `process_id`, `Keywords`, `EventType`, etc.: These fields capture various metadata and attributes of an event such as the process ID, event type, hostname, and timestamps.
    - `message`: Stores the actual event message.
    - `processed`: Integer flag to indicate whether the event has been processed.
  - **String Representation**: Displays a combination of the `iso_timestamp`, `app_name`, and `EventType` for better identification of the event log.

RouterData Model

- **Purpose**: Represents log data captured from network devices (e.g., routers).
- **Key Features**:
  - **Fields**:
    - `created_at`: Timestamp when the log was recorded.
    - `severity`, `date_time`, `hostname`, `process`: Captures router log attributes like the severity, date, hostname, and process involved.
    - `message`: The router log message.
  - **String Representation**: Combines `date_time`, `hostname`, and `process` to uniquely describe the log entry.

Serializers

BronzeEventDataSerializer

- **Purpose**: Serializes the BronzeEventData model for API requests and responses.
- **Fields**: Exposes key fields such as `iso_timestamp`, `hostname`, `EventType`, `EventID`, `AccountName`, and `message`.

RouterDataSerializer

- **Purpose**: Serializes the RouterData model for network-related logs.
- **Fields**: Includes fields like `date_time`, `hostname`, `process`, and `message`.

LogCountSerializer

- **Purpose**: A serializer for log count statistics (such as percentages).
- **Key Feature**: It rounds the `windows_os_percentage` and `network_percentage` fields to two decimal places in the response.

ViewSets

BronzeEventDataViewSet

- **Purpose**: Manages read-only access to the BronzeEventData entries.
- **Key Features**:
  - **Pagination**: Implements `StandardResultsSetPagination` to handle large datasets.
  - **Filtering**: Supports filtering by query, event type, start and end time, and ordering by `iso_timestamp` or `event_id`.
  - **Search**: Allows searching through fields like `hostname`, `AccountName`, and `message`.
  - **Permissions**: Requires authentication via `IsAuthenticated`.
  - **Custom Actions**:
    - `export_pdf`: Exports filtered event data to a PDF report.
    - `count`: Returns the total count of BronzeEventData entries.

RouterDataViewSet

- **Purpose**: Manages read-only access to RouterData entries.
- **Key Features**:
  - Similar functionality to BronzeEventDataViewSet, but tailored for router logs.
  - **Custom Actions**:
    - `router_log_count`: Returns the total count of RouterData entries.
    - `export_pdf`: Exports router data to a PDF report.

LogPercentageViewSet

- **Purpose**: Calculates and returns the percentage distribution of log types (e.g., Windows OS events vs. network events).
- **Key Features**:
  - `log_percentages`: Aggregates and returns the percentage of logs related to Windows OS (from BronzeEventData) and network logs (from RouterData).

LogAggregationViewSet

- **Purpose**: Aggregates log data for analysis, specifically by hour.
- **Key Features**:
  - `logs_per_hour`: Aggregates both BronzeEventData and RouterData logs by hour for better time-based analysis of events.

EventsToday

- **Purpose**: Counts events logged today.
- **Key Feature**: Uses the local timezone (Australia/Melbourne) to calculate the events that occurred on the current day, then converts it to UTC to perform the filtering.

HostnameCountViewSet

- **Purpose**: Returns the count of distinct hostnames (devices) from both BronzeEventData and RouterData.
- **Key Features**: Aggregates unique hostnames to give a total device count.

Summary

- The logs app provides comprehensive logging functionality, with models that capture both system and network events. It integrates useful APIs for querying, filtering, and aggregating log data, and includes custom actions like PDF exporting and log percentage calculation. This design ensures both flexibility and performance for managing large-scale logs in a SIEM context.

Reports

1. Models
2. Serializers
3. Views

Models

IncidentReport The IncidentReport model captures incident report data and tracks the lifecycle of reports related to security incidents, network traffic, user activity, system performance, and compliance audits.

- **Fields**:
  - `title` (CharField): Title of the incident report.
  - `type` (CharField): Type of the report. Choices include:
    - `security_incident`
    - `network_traffic`
    - `user_activity`
    - `system_performance`
    - `compliance_audit`
  - `status` (CharField): Status of the report. Choices include:
    - `draft`
    - `open`
    - `pending`
    - `approved`
    - `rejected`

    * archived
  - rules (ManyToManyField): Many-to-many relationship with the Rule model.
  - user (ForeignKey): Reference to the User who created the report.
  - description (TextField): Description of the incident or report.
  - pdf_file (FileField): An optional field to store the generated PDF report file.
- **Meta Options**:
  - ordering: The reports are ordered by the created_at field in descending order.
- **Methods**:
  - __str__(): Returns a string representation of the incident report.

Serializers

IncidentReportSerializer Handles serialization of IncidentReport objects.

- **Fields**:
  - source: A nested serializer to display log data (uses the BronzeEventDataSerializer).
  - user: A nested serializer to display user details (uses the IncidentReportUserSerializer).
  - rules: A nested serializer to display rule details (uses the IncidentReportRuleSerializer).
  - pdf_url: A method field that returns the URL of the generated PDF file if available.
  - rule_ids: A write-only field that accepts a list of rule IDs for linking Rule objects to the incident report.
- **Methods**:
  - get_pdf_url(): Returns the URL of the report's PDF file.
  - create(): Handles the creation of an IncidentReport and associates rules with the report.
  - update(): Updates the incident report and its associated rules.

Supporting Serializers:

- **IncidentReportUserSerializer**: Serializes the user data (user_id, email).
- **IncidentReportRuleSerializer**: Serializes the rule data (id, name, description, severity).

Filters

IncidentReportFilter Provides custom filtering capabilities for incident reports based on: - type: Filter reports by type. - status: Filter by the current status of the report. - user__email: Filter reports by the email address of the associated user. - last_update: Filter reports by their last updated timestamp.

Views

IncidentReportViewSet Handles view logic for managing IncidentReport objects. It extends `BaseViewThrottleSet` and implements the standard Django REST Framework (DRF) viewset actions (`list`, `retrieve`, `create`, `update`, `delete`).

- **Methods**:
  - `get_queryset()`: Customizes the queryset to support searching by title, description, rules, and user email.
  - `perform_create()`: Saves the report and associates the current user.
- **Custom Actions**:
  - `generate_pdf()`: Generates a PDF report for a specific incident report.
  - `delete_report()`: Deletes the selected report.

Summary

- The Reports module in the SIEM system is designed to capture, manage, and track the lifecycle of incident reports related to security incidents, network traffic, user activity, system performance, and compliance audits. The core model, IncidentReport, records critical details such as report type, status, associated rules, and user information, with the option to generate a PDF of the report. It is supported by serializers that manage data formatting and custom filtering capabilities to allow for flexible querying based on type, status, user email, and timestamps. The viewset provides standard actions for creating, retrieving, and updating reports, with additional features like PDF generation and custom deletion, ensuring comprehensive management of security incidents and audits.