Palo Alto XSOAR Integration Guide for Vectra XDR (RUX)

Version: November 2025

Table of Contents

Introduction

Vectra XDR for Palo Alto Networks XSOAR (formerly Demisto) empowers the SOC to create and manage incidents using Vectra AI's Attack Signal Intelligence for the Respond User Experience.

This integration allows the security operations center to create and manage incidents based on prioritized entities, powered by Vectra AI's Attack Signal Intelligence. Integrating Vectra and XSOAR enables security teams to synchronize Vectra XDR Entities with XSOAR incidents in real time, making it feasible to manage operations from a single place.

Integration value is achieved by injecting Vectra's integrated signal into the security operations center in a structured and highly efficient approach to ultimately transform the analyst experience to enable:

- Rich Prioritization
- Incident Management
- Detailed Investigations
- Enrichment
- Enforcement
- Resolution
- Reporting

What's New in v1.2.0

New Commands

- **vectra-detection-tag-list** – Returns the list of tags for a specified detection.
- **vectra-detection-tag-add** – Adds one or more tags to a detection.
- **vectra-detection-tag-remove** – Removes one or more tags from a detection.
- **vectra-entity-detections-mark-asclosed** – Marks all detections for a specified entity as closed using the provided entity ID.
- **vectra-detections-mark-asclosed** – Marks detections as closed using the provided detection IDs.
- **vectra-detections-mark-asopen** – Reopens detections using the provided detection IDs.

Behavioral Changes

- Updated the **List Detections** logic to retrieve detections based on **Entity ID** and **Entity Type**, instead of relying on comma-separated Detection IDs.

Fixes

- Fixed an issue with **Outgoing Mirroring** when invalid tags are provided.

Docker Image

- Updated the Docker image to: demisto/python3:3.12.12.5490952.

What's New in v1.1.0

Integrations – Vectra XDR

- **Re-fetch Closed Incidents via Mirroring**
  - Added support for re-fetching closed incidents using a mirroring parameter.
    - When selected, re-fetched incidents are created as **new incidents**.
    - When not selected, the integration **reopens previously closed incidents**.

Note: This flow is triggered only when the relevant entity is still active, and the previously fetched incident is closed.

- **Enhanced Entity Listing**
  - Added support for the name argument in the vectra-entity-list command to enable more granular entity retrieval.
- **Docker Image Update**
  - Updated the Docker image to: demisto/python3:3.12.11.4508456.

Playbooks

- **New: Detections Assessment – Vectra XDR**
  - This playbook runs a detections assessment for a given entity and stores the results in the incident context data for downstream use and reporting.
- **New: MDR Escalation Process – Vectra XDR**
  - This playbook:
    - Retrieves the MDR ticket number associated with the given entity by parsing its notes.
    - Collects the entity's active detections.
    - Performs a detections assessment.
    - Generates and sends the assessment results to the designated recipient via email to support MDR escalation workflows.

Scripts

- **New: VectraXDRGenerateMailBody**
  - Added a new script, VectraXDRGenerateMailBody, which generates an email body based on the detections assessment results for use in the MDR escalation process.
- **Updated: VectraXDRDisplayEntityDetections**
  - Updated the external URL pivot to use the latest pack version.
  - Updated the Docker image to: demisto/python3:3.12.11.4508456.

Document and Release Information

This section provides key metadata for the Vectra RUX integration guide, including document versioning, supported platforms, intended audience, and the purpose of the document. It serves as a reference point to ensure that users are working with the correct release information before proceeding with deployment or configuration activities.

Field	Details
Document Title	Palo Alto XSOAR Integration Guide for Vectra XDR (RUX)
Document Version	November 2025
Package Version	1.2.0
Supported Platform	Vectra Respond UX (RUX) and Palo Alto Networks Cortex XSOAR
Audience	SOC Analysts, Incident Responders, XSOAR Administrators, Security Architects
Purpose	This document provides comprehensive guidance for deploying, configuring, and operationalizing the Vectra RUX integration within Cortex XSOAR. It covers installation prerequisites, implementation steps, incident handling, playbook operations, workflows, troubleshooting procedures, and best practices to support effective SOC investigation and response.

Terminology

There are several terms that can be used interchangeably. The following table provides the XSOAR term as well as other terms that may be used to refer to the same.

XSOAR Term	Additional Terms
App	Vectra XDR for Palo Alto XSOAR
Instance	Configuration Profile, Asset
Action	Function, Command
Incident	Event, Alert, Container

Architecture

Integrating Vectra with XSOAR utilizes the REST API in a PULL model. The Vectra app resides inside the XSOAR platform and uses REST API calls to pull the appropriate data following the operator configured polling interval (figure 1).

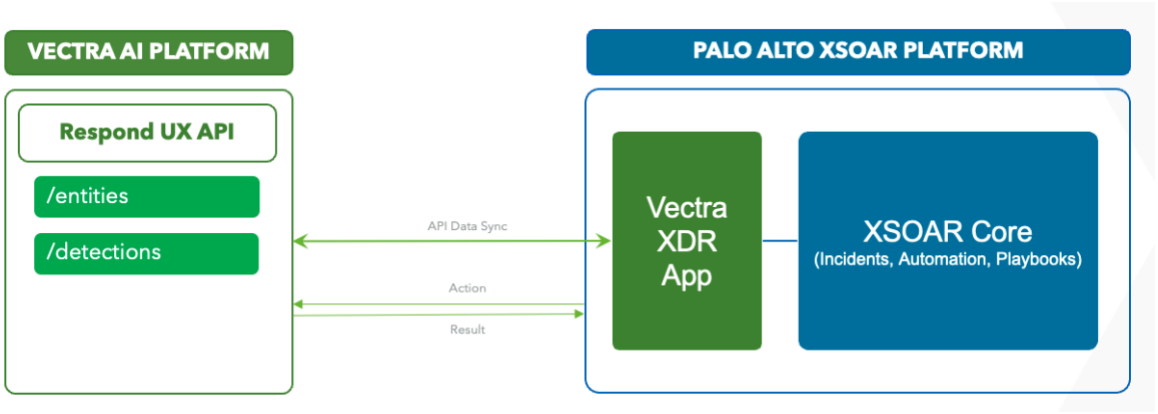


Figure 1 - Simplified Architecture

One or more Vectra Respond UX tenants provide the source of the data. Configuration Profiles (instances) are used to configure the details of how to communicate with a specific tenant (i.e., tenant URL and API credentials), polling schedule, as well as API error handling. The integration is designed to retrieve entity and detection data (along with all associated components such as notes, tags, and assignments) and ingestion filters enable the operator to fine tune the data that is considered for ingestion.

Multiple instances are required when there are multiple Vectra tenants but can also be used when there is a single tenant that requires competing ingestion filters. Figure 2 demonstrates a scenario where the operator wishes to ingest all prioritized entities from Vectra Tenant 1 as well as entities of any priority that has at least one exfiltration detection.

Once data begins ingestion, an app-embedded de-duplication mechanism controls if something is new and unique or if an update is warranted. If a previous incident does not exist a new incident will be created. If there is an existing incident, then the appropriate updates are made to ensure no duplicates.

The final components of the app include the supported actions as well as automations and playbooks. These will be covered later in this document.

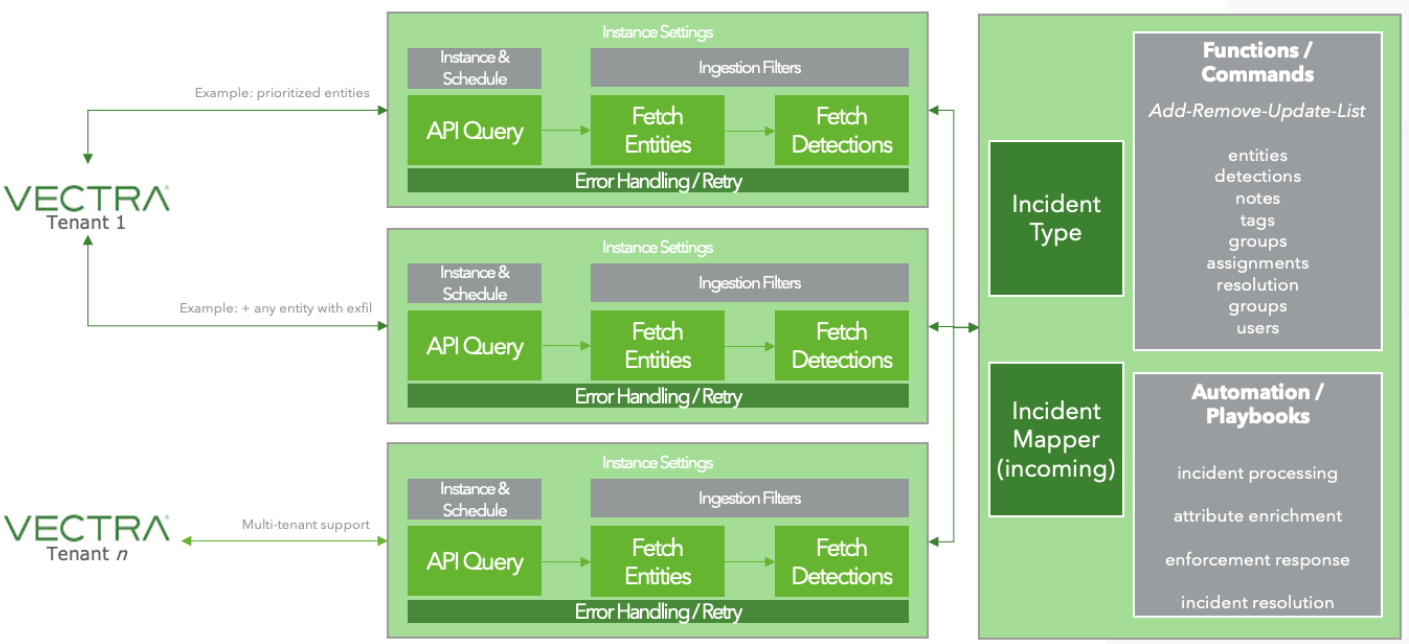


Figure 2 - Block Diagram

The next order of detail includes the operational components. As data makes it through ingestion as per the configuration profile, incidents are created. Each incident will have incident information in the form of context data. The context data includes the entity, detection and assignment data received from the Vectra platform. The integration includes support for several actions where each action is a command that can be issued to communicate with the Vectra platform (ex. add tag). Playbooks are used to define automation flows and can consist of multiple commands as well instructions for interfacing with other apps configured in the XSOAR environment.

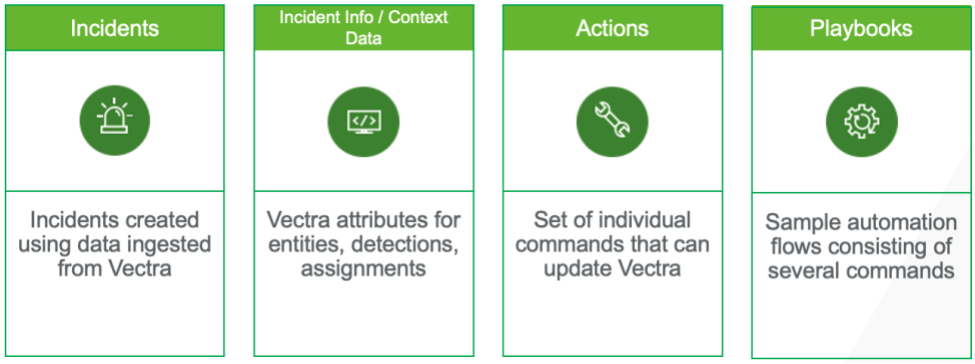


Figure 3 - Operational Components

Incidents are a foundational component of the integration as that is the starting point for any investigative or response workflow. Vectra employs Attack Signal Intelligence to conduct ruthless prioritization to ensure operational efficiency. Best practice is to implement a configuration profile that generates incidents based on Vectra prioritized entities. These incidents are funneled into an analyst queue in XSOAR. Incidents are generated at the Vectra entity level only and detections are associated with an entity. An incident includes all the Vectra attributes that make up the entity and its detections. There are several other XSOAR components that are part of the incident layout. In addition to the Incident Info and Detections, these components include the War Room and Workplan, as well as several other components that may be blank if

not utilized (ex. evidence board, related items, canvas). It's possible to attach a workplan to an incident, run individual actions or launch playbooks.

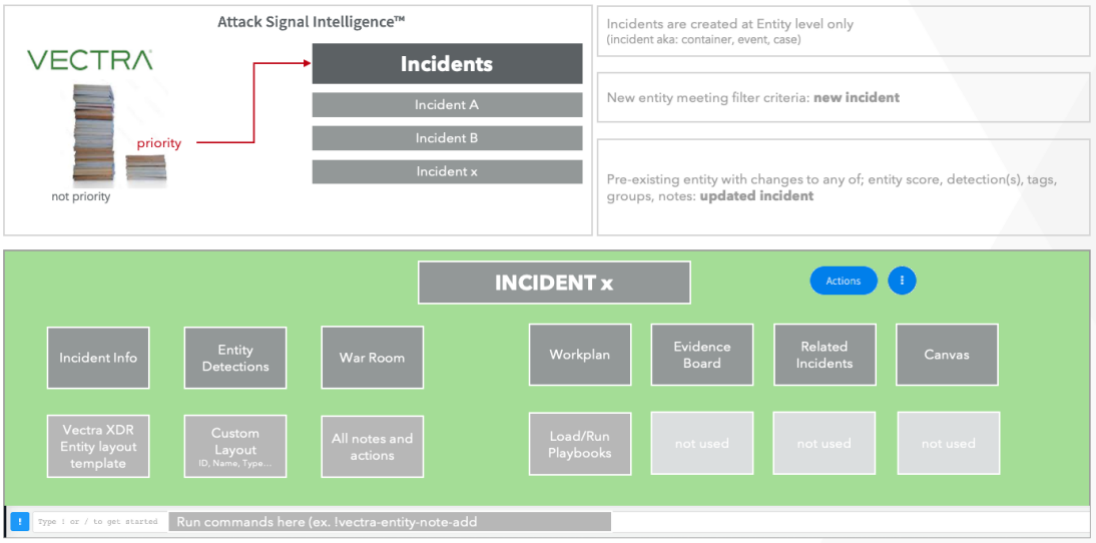


Figure 4 - Incident Structure

Implementation

Vectra Pre-requisites

The minimum requirements on the Vectra side to configure this integration include:

- Vectra Respond UX Tenant running API version 3.3 or higher.

  If you aren't sure which analyst experience is being utilized, please refer to this support article: https://support.vectra.ai/s/article/KB-VS-1673

- API client ID and secret configured with the Security Analyst Role.

  To create the API client:

  1. Log in to your Vectra Respond UX, navigate to *Manage* > *API Clients*, and click "Add API Client".
  2. Enter a name for the client you are creating, select the role, optionally enter a description, and then click "Generate Credentials".
  3. Copy the client ID and the secret and store in a safe location and then click "Done".

XSOAR Pre-requisites

The minimum requirements on the XSOAR side to configure this integration include:

- Palo Alto account that has access to download content from XSOAR Marketplace.
- XSOAR software installed (on-prem or cloud) v6.8.0 or higher.
- XSOAR local account with access to install apps.
- XSOAR platform must be able to communicate with the Vectra tenant over port 443.

The Vectra integration does not impose any specific modifications or requirements of the XSOAR platform. Please refer to the XSOAR documentation for recommendations on system requirements.

Downloading and Installing the App

The integration (app) is available for download from the XSOAR Marketplace at this location:

https://cortex.marketplace.pan.dev/marketplace/details/VectraXDR/

Publisher: Vectra
App Version: 1.2.0
Content Pack: 1.2.0
Supported Versions: Vectra Respond UX with Vectra API v3.3+

For easiest implementation, navigate to Marketplace from within your XSOAR instance and search for Vectra XDR. Select the Vectra XDR application and then click the install button.

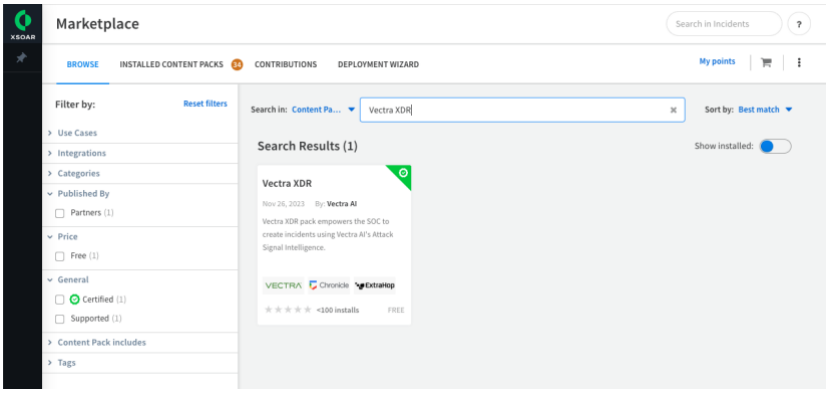


Figure 5 - App Install

Implementation Checklist

Prior to configuring a new instance, it may be helpful to review the following checklist as there are several parameters that may be required for configuration. Mandatory parameters are prefixed with a *.

Parameter	Content
* Name	Ex. Vectra_*tenantID*
* Fetching	Enable or disable fetch
Classifier	Not required for integration
* Incident Type	Vectra XDR Entity (included with app)
* Mapper (incoming)	Vectra XDR – Incoming Mapper (included with app)
Mapper (outgoing)	Not required for integration
* Server URL	Ex. https://999999999999.xx1.portal.vectra.ai
* Client ID	Generated in Vectra – Manage > API Clients
* Client Secret Key	Generated in Vectra – Manage > API Clients
Certificate and Proxy	If using proxy, must be first configured in the system
Fetch	Max is 200 at a time, first fetch is historical lookback
* Mirroring	Best practice is incoming and outgoing, note is the XSOAR tag needed to write
* Entity types to poll	Host and/or account (best practice is both)
* Prioritized entity polling	Yes or No (best practice is Yes and blank is both)
Tag filtering	CSV list – only entities with matching tags will be ingested
Detection category filtering	Single selection (Botnet, C2, Recon, Lateral, Exfil, Info, All)
Detection type filtering	Single item free text – only entities with the detection specified will be ingested
* Low severity level	Vectra entities with urgency score up to this value is mapped to Low

* Medium severity level	Vectra entities with urgency score up to this value is mapped to Medium.
* High severity level	Vectra entities with urgency score up to this value is mapped to High so anything over is mapped Critical.
* Polling interval	Specific fetch interval time
Engine	Not required for integration

When working with severity mapping levels it's best practice to align it with Vectra priority thresholds. The priority threshold is configured within the Vectra UI under *Hunt > Manage Prioritization Threshold.* In the example below (Figure 6), entities with an urgency score of 50 or higher are considered prioritized. To map this to XSOAR severity settings low would be set to 29, medium to 49 and high to 79. This will align Vectra prioritized entities with XSOAR severities of high and critical.

Figure 6 - Severity Mapping

Initial Configuration of a New Instance

An asset configuration is required to allow XSOAR to communicate with the Vectra platform and pull data.

With the **Vectra XDR for Palo Alto XSOAR** pack installed:

- In the XSOAR UI, go to Settings → Integrations → Instances.
- Locate the Vectra XDR integration and click Add instance (see *Figure 7 – Configure New Instance*).

Note:

- You can configure **multiple instances** for the same or different tenants.
- Multiple instances are required when you have:
  - Multiple Vectra tenants, or
  - Different/overlapping filters that cannot be represented in a single instance.
- Follow these procedures for configuring each instance as needed.

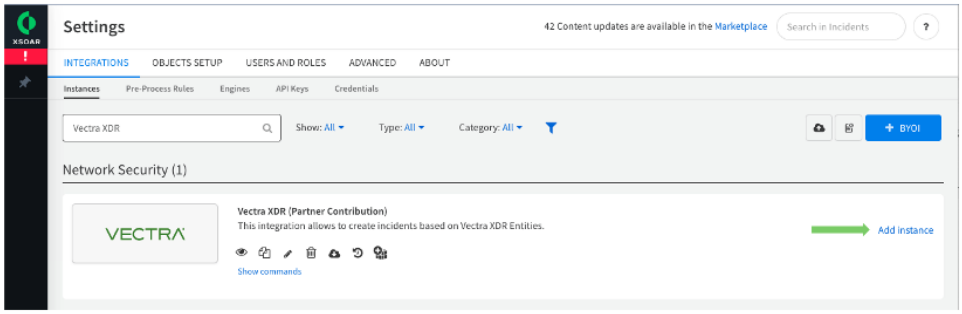


Figure 7 - Configure New Instance

Instance Configuration Page

After clicking **Add instance**, the **New Instance** configuration page is displayed (Figure 8).

Although the instance configuration is presented as a **single page element**, it is logically organized into multiple sections (for documentation purposes). The first section contains the base application settings, including the instance name.

- When choosing an instance name, it is recommended to include an indicator for the **Vectra tenant** being queried (for example, Vectra-XDR-EU-Tenant or Vectra-XDR-Prod).
- A common best practice is to:
  - Configure **all settings** with **"Do not fetch"** enabled.
  - **Save** the configuration.
  - Run **Test** to validate connectivity and credentials.
  - Only after successful testing, **enable fetching**.

To save your work, click the **Save** icon before initiating a **Test**. Once saved, the full instance configuration view is displayed (Figure 8), including all sections relevant to connectivity, authentication, mirroring, and fetching.

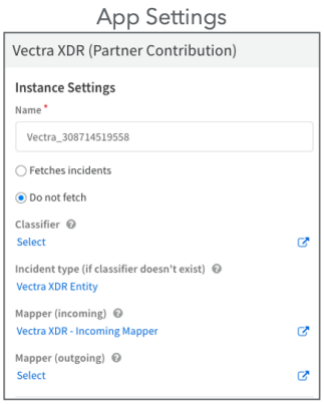


Figure 8a – Instance Settings

Classification and Mapping Requirements

For Vectra XDR incidents to be created and enriched correctly in XSOAR, the following settings are required on the instance:

- Incident Type:

- Must be set to Vectra XDR Entity.

- Incoming Mapper:
  - Must be set to Vectra XDR.
- Classifier / Outgoing Mapper:
  - A classifier and outgoing mapper are not required for basic operation.
  - They can be configured if you need custom routing, normalization, or outbound mapping to other systems.

Configure these values before enabling fetching to ensure that incidents are created with the correct type and mapped fields.

Tenant Settings and Connectivity

The next block of settings covers **tenant connectivity** and defines how XSOAR reaches the Vectra platform, including the API credentials.

These settings typically include:

- Server URL / Vectra Tenant URL
- API Credentials (Client ID / Client Secret Key)
- Certificate and Proxy Options

Certificates

The Vectra tenant uses **valid certificates** by default.

- In most environments, you **should not enable** the option to “trust any certificate”.
- Only enable “trust any certificate” if:
  - You are using an interception proxy or non-standard TLS setup **and**
  - You fully understand the security implications of trusting arbitrary certificates.

Proxy Configuration

If XSOAR must use a **proxy** to reach the (external) Vectra tenant:

- First, configure the proxy at the **XSOAR server level**:
  - Go to **Settings** → **About** → **Troubleshooting** → **Server Configuration**.
  - Define the proxy settings there (host, port, authentication, etc.).
- After the server proxy is configured, return to the Vectra XDR instance settings and:
  - Enable the **“Use system proxy”** (or equivalent **proxy checkbox**) to instruct this integration to use that proxy configuration.

This ensures all communication from the Vectra XDR integration to the Vectra platform is routed correctly through your enterprise proxy.

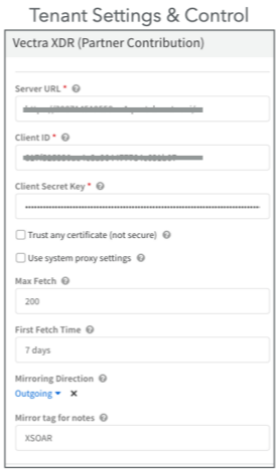


Figure 9b – Instance Settings

Max Fetch

Max Fetch controls the maximum number of entities retrieved from Vectra during each fetch cycle.

- This value defines the upper limit per poll.
- If you configure a value greater than 200, it will be internally capped at 200.
- The maximum effective value is 200.

Recommendation:

- Start with a value such as **50–100** during initial deployment to observe volume and performance.
- Increase towards **200** only if:
  - Your environment has a high number of prioritized entities, and
  - Your XSOAR infrastructure can handle the additional incident volume per fetch.

First Fetch Time

First Fetch Time defines the **starting point in time** from which XSOAR begins fetching entities from Vectra.

You can specify this as either:

- A **relative time** (e.g., last few hours/days), or
- An **absolute timestamp**.

Supported formats include:

- Relative:
  - 2 minutes
  - 2 hours
  - 2 days
  - 2 weeks
  - 2 months
  - 2 years
- Absolute:
  - yyyy-mm-dd
  - yyyy-mm-ddTHH:MM:SSZ

Examples:

- 2 days
- 01 May 2023
- 01 Mar 2021 04:45:33
- 2022-04-17T14:05:44Z

Recommendations:

- For **new deployments**, use a **recent relative time** (for example, 3 days) to avoid flooding XSOAR with historical data on first run.
- For **migration / replay scenarios**, use an **absolute timestamp** aligned with your cutover date so that incidents start from a controlled point in time.

Once the first fetch has completed, subsequent fetches will continue from the **last fetched time** regardless of the original First Fetch Time setting.

Mirroring Settings

Mirroring controls the direction of data synchronization between **XSOAR** and **Vectra XDR**.

- **Outgoing mirroring**
  - Sends updates from XSOAR to Vectra, such as:
    - Tags
    - Assignments / ownership
    - Resolution / status
  - These changes made in XSOAR are reflected back on the Vectra side.
- **Incoming mirroring**
  - Pulls updates from Vectra to XSOAR, such as:
    - Tags
    - Assignments / ownership
    - Resolution / status
    - Detections
  - These changes made in Vectra are mirrored into the corresponding XSOAR incidents.

Recommended:

- In most deployments, **both Incoming and Outgoing** are enabled to maintain **bi-directional synchronization** between XSOAR and Vectra.

Re-Fetch Closed Incidents via Mirroring

The **Re-Fetch closed incidents via mirroring** option controls how XSOAR handles entities that become active again in Vectra XDR after their related incidents have already been closed in XSOAR.

This behavior is driven by a single checkbox in the Vectra XDR integration instance:

Checkbox: Re-Fetch closed incidents via mirroring

- **If selected** – new incidents will be created (**via Outgoing Mirroring**).
- **If not selected** – it reopens previously closed incidents (**via Incoming Mirroring**).

Note: This flow is triggered only when the relevant entity is still active and the previously fetched incident is closed.

When This Flow Is Triggered

The re-fetch logic runs as part of the normal mirroring / fetch cycle. It applies only when the following are true:

- The entity exists in **Vectra XDR** and is currently **active**.
- XSOAR has previously created an incident for that entity from the **Vectra XDR integration**.
- That previously created incident is now **closed** in XSOAR.

If any of these conditions are not met (for example, the entity is no longer active, or there was never an incident for it), the re-fetch flow does **not** run, and the mirroring behaves in the standard way.

Choose Your Desired Behavior

A. Create New Incidents (Checkbox Selected)

- Check **Re-Fetch closed incidents via mirroring.**
- Resulting behavior:
    - When an entity with a previously closed incident becomes active again:
        - XSOAR **creates a new incident** for that entity (**via Outgoing Mirroring**).
        - The old incident remains closed.
- Use this setting if:
    - You want **separate incidents** for each new wave of activity.
    - Different teams / shifts own different time windows.
    - Reporting is incident-based (e.g., count incidents per month, SLA metrics, ticket queues).

B. Reopen Existing Incidents (Checkbox Not Selected)

- Leave **Re-Fetch closed incidents via mirroring unchecked.**
- Resulting behavior:
    - When an entity with a previously closed incident becomes active again:
        - XSOAR reopens the previously closed incident for that entity (via Incoming Mirroring).
        - New detections and context are appended to the same incident.
- Use this setting if:
    - You prefer a single continuous investigation per entity.
    - Analysts should see full history over time in one incident.
    - You track long-lived or recurring adversary behavior under a single case.

Save & Validate

- Click **Save** on the integration instance.
- After the next mirroring/fetch cycle:
    - Close a test incident tied to an active entity.
    - Trigger new detections for the same entity in Vectra XDR.
    - Confirm that XSOAR either **created a new incident** or **reopened the existing one**, according to your checkbox selection.

Example Flows

Example 1 – Checkbox Selected (New Incident via Outgoing Mirroring)

- An entity in Vectra XDR generates detections → XSOAR creates **Incident #101**.
- Analysts investigate and **close Incident #101**.
- Later, the same entity becomes active again with new detections.
- During mirroring:
  - The entity is still active.
  - The last related incident (Incident #101) is closed.
  - The checkbox is **selected**.
- XSOAR creates a new incident (e.g., Incident #145) via Outgoing Mirroring.

Result:

- Incident #101 = prior investigation (closed).
- Incident #145 = new investigation for the same entity.

Example 2 – Checkbox Not Selected (Incident Reopened via Incoming Mirroring)

- An entity in Vectra XDR generates detections → XSOAR creates **Incident #202**.
- Analysts investigate and **close Incident #202**.
- Later, the same entity becomes active again.
- During mirroring:
  - The entity is still active.
  - The last related incident (Incident #202) is closed.
  - The checkbox is **not selected**.
- XSOAR **reopens Incident #202** via **Incoming Mirroring** and adds the new detections/context.

Result:

- Analysts see **one incident (#202)** containing both the original and the new activity.

Mirror Tag for Notes

Mirror tag for notes controls which XSOAR notes are sent (mirrored) to Vectra.

- The value you configure here is a **tag string** (prefix) that must appear in the **note body** in XSOAR.
- Only notes that **start with** this tag value will be mirrored from XSOAR to Vectra as entity notes.

The **default tag value is** note.

Filtering Settings

The **Filtering** section is critical, as it determines **which Vectra entities become XSOAR incidents**.

Vectra defines **two entity types**:

- Accounts
- Hosts

Entity Type

- Leaving the entity type field blank means all entity types are included.

- Best practice: Explicitly select Accounts and Hosts to avoid ambiguity and to be explicit about what you are ingesting.
- You can also choose to ingest from only one entity type if required by the use case (for example, only accounts for identity-focused workflows).

Prioritized

The **Prioritized** parameter is a **Yes/No** option:

- **Best practice:** Set **Prioritized = Yes** to poll only **prioritized entities** from Vectra.
  - This ensures XSOAR incidents are created only for entities Vectra has scored as important.
- If your use case requires ingesting all entities:
  - Leave this parameter blank (no value), which effectively removes the prioritized filter.

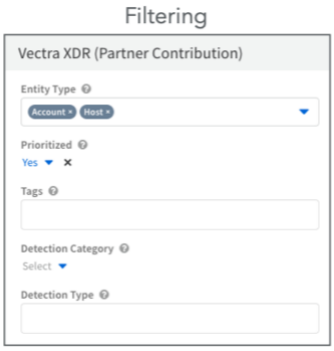


Figure 10c – Instance Settings

Additional Filters

The following filters allow for **fine-tuning** which entities and detections are ingested into XSOAR. These are typically used in **secondary instances** that target more specific use cases.

Tags

Retrieve entities that contain **any** of the specified tags.

- Supports comma-separated values.
- If multiple tags are provided, an entity that matches at least one of the tags is included.

Detection Category

Retrieve detections belonging to a specified **detection category**.

This is a **single-select** field. Example categories include:

- Command & Control
- Botnet
- Reconnaissance
- Lateral Movement
- Exfiltration
- Info

Use this when you want an instance focused on **specific stages of the attack lifecycle** (for example, one instance dedicated to **Command & Control** and **Exfiltration**).

Detection Type

Retrieve detections belonging to a specified **detection type**.

- Use this to target **specific detection names/types**
- This is typically used when a use case or playbook is designed to respond to a **very specific detection** rather than an entire category.

Severity Mapping and Polling Interval

The final section covers **severity mapping** and the **polling (fetch) interval**.

Severity Mapping

Different tools often use different terminology or scales for **severity**, which can be confusing for analysts.

- Vectra uses **Urgency** to describe the importance of an entity.
- XSOAR uses **Severity** to classify incidents.

To reduce confusion and ensure consistent triage:

- Map Vectra Urgency values to XSOAR Severity levels so that:
  - High-urgency entities in Vectra become high-severity incidents in XSOAR.
  - Lower-urgency entities map to appropriate lower severity values.

This mapping helps ensure that incidents are **elevated and represented consistently** across both tools.

Fetch / Polling Interval

The **fetch interval** controls how frequently XSOAR polls Vectra for new or updated entities:

- Configure this value according to your SOC's operational needs and platform capacity.
- XSOAR can fetch as frequently as **every 1 minute** for near real-time updates.
- In many environments, a slightly longer interval (e.g., 3–5 minutes) can be chosen to balance **timeliness** with **load**.

Make sure the chosen interval is aligned with:

- Your SOC's responsiveness expectations.
- The volume of entities and detections in your Vectra deployment.
- Any broader platform performance considerations.

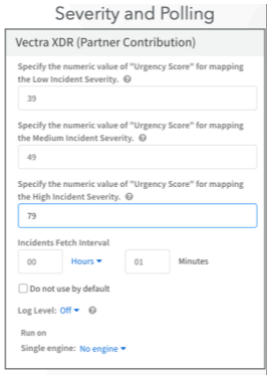


Figure 11d – Instance Settings

Operational Components

At this stage, the Vectra app is installed, configured and the system should be receiving data. The architecture and implementation are complete so next we will take a deeper look into the operational perspective in more detail.

Incidents

Also referred to as containers, events, and alerts, an incident is the starting point for incident management, workflow, and automation. Vectra incidents are accessible via the Incidents menu in the XSOAR UI (figure 9).

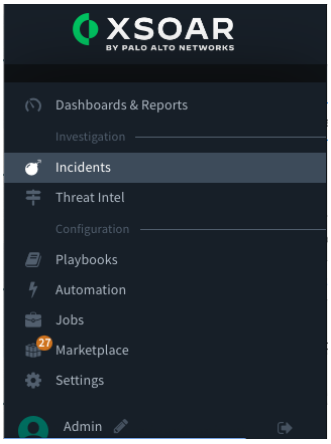


Figure 9 – Incidents

The ingestion mechanism includes de-duplication logic natively to ensure that if an incident doesn't already exist a new one is created. Conversely, if an incident already exists then the incident is updated rather than duplicated. Security analysts should strive to resolve events as they appear as they are already prioritized via AI-driven Prioritization. The list of incidents can be viewed from the incidents page and the operator can specify filters, column, and sort ordering. The recommended column order (ID, Status, Entity Name, Urgency Score, Attack Rating, Entity Velocity, Attack Profile, Entity Importance, Assigned to, Playbook, Run Status, Last Modified Timestamp, Incident Link, Source Instance) is shown in the figure below (Figure 10).

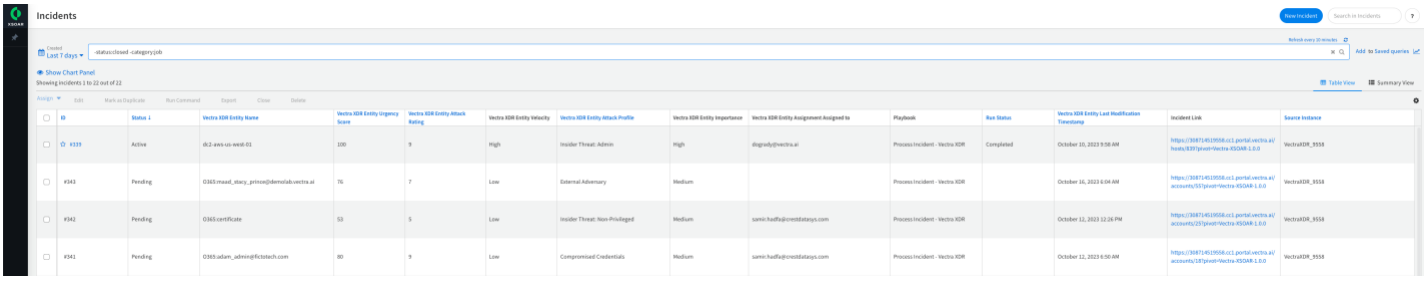


Figure 10 – Incidents Page

Customizing the column order is done using the gear icon. The columns will appear in the order they are selected so it's recommended to deselect all columns and then select the columns required in the order you want them to appear from left to right (Figure 11).

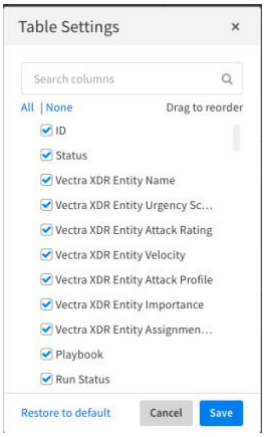


Figure 11 - Column Order

Incident Template

The Vectra XDR integration for XSOAR includes a pre-defined incident layout template. This layout template includes multiple tabs as shown in Figure 12. This layout is displayed as each incident is accessed. Details for the relevant tabs are covered next.

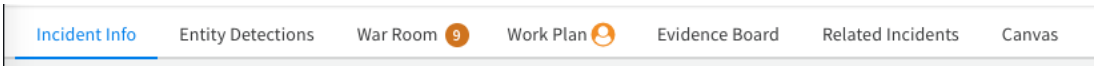


Figure 12 - Layout Template

Incident Info

To view the details of a single incident, select the incident ID field (Figure 10). Selecting a single incident will bring up the incident layout (Figure 13) starting with the Incident Info. Incident Info is broken into several sections and provides a single page view with access to the most relevant data and commands.

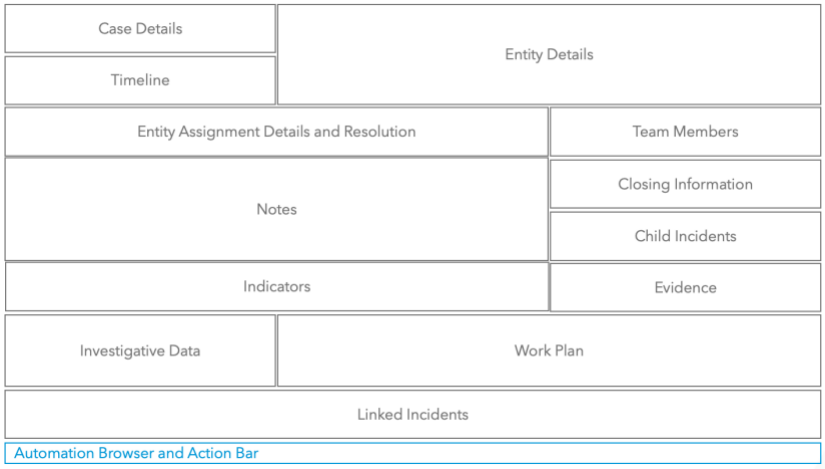


Figure 13 - Incident Layout

Entity Detections

Each incident is tied to an entity and each entity can have one or more detections. The detections that are associated with an entity are displayed in table form within the Detections tab of the incident template.

War Room

Each incident has its own war room. The war room facilitates real-time investigation and keeps track of all activities related to an incident.

Work Plan

The work plan for each Vectra incident will include the Process Incident – Vectra XDR playbook. The work plan will show the run status of the playbook. The operator can also change the playbook to run from within the work plan.

Remaining Tabs

The remaining tabs (Evidence Board, Related Incidents, Canvas) aren't initially engaged as part of Vectra incidents. These tabs may be used as required during the lifecycle of the incident.

Context Data

The layout contains the high-level data for an incident but additional data for Vectra and XSOAR attributes are available as "Context Data" (Figure 14). This data is accessed by selecting "Context Data" from the menu. Context data is expandable JSON that is searchable, and it contains all attributes relevant to Vectra and XSOAR. Vectra data includes all API attributes returned from calls to the following endpoints: (/entities, /detections, /assignments). Elements including notes, tags, groups, etc.… are included in these endpoints. Context Data is available from several locations and not just the Incident Info view.

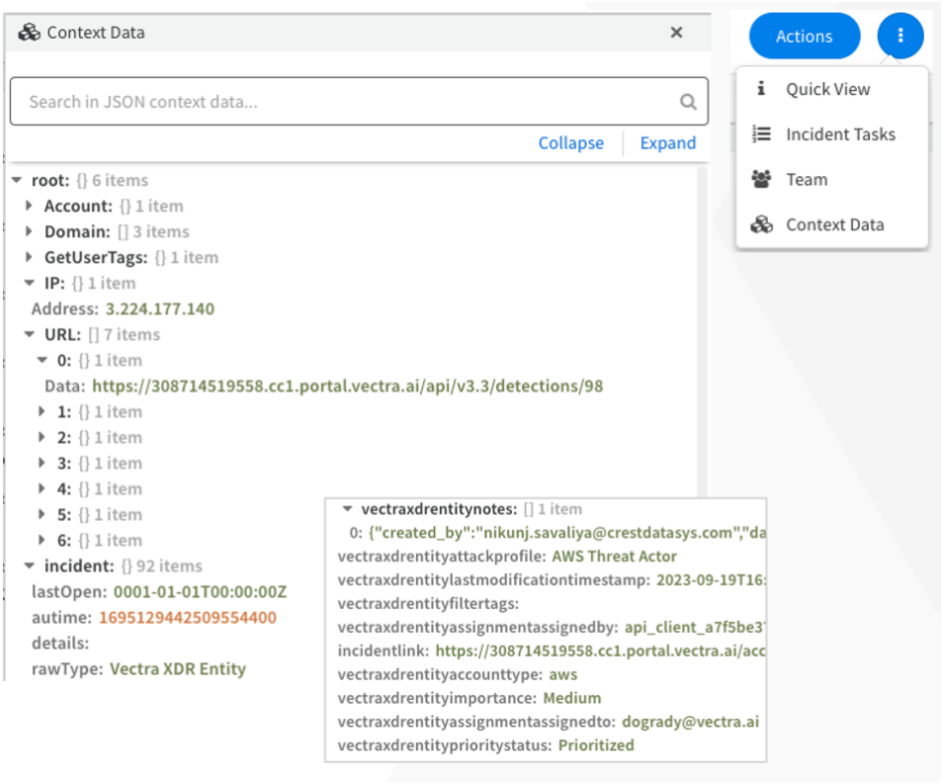


Figure 14 - Context Data

Indicators

Artifacts such as domain names, URLs, accounts, and IP addresses are automatically mapped by XSOAR as indicators. These indicators are present in the Context Data (Figure 12) in JSON as well as in the indicators section of the Incident Layout (Figure 15). The default XSOAR indicator settings are employed so there is not a custom incoming indicator mapper.

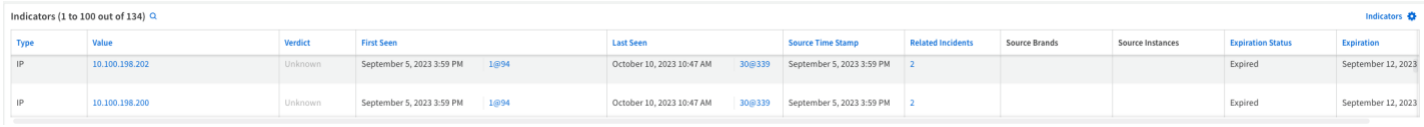


Figure 15 - Indicators

Actions

Actions are commands that can be run against the Vectra AI Platform. The following table (Table 1) outlines all the supported commands, a description of what the command does, as well as what mandatory parameters are required for running the action.

Table 1 - Actions List

Action	Description	Parameters (summary)
vectra-user-list	Returns a list of all users (operators).	username, role, last_login_timestamp
vectra-entity-list	Returns a list of all entities.	prioritized, entity_type, name, tags, state, ordering, last_detection_timestamp, page, page_size, last_modified_timestamp

vectra-entity-describe	Returns attributes for a single entity.	entity_id, entity_type
vectra-entity-detection-list	Returns a list of detections for a single entity.	entity_id, entity_type, page, page_size, detection_category, detection_type, last_timestamp, detection_name, state, tags
vectra-detection-describe	Returns detailed information for a single detection.	detection_id
vectra-entity-detections-mark-fixed	Marks all detections as fixed for a single entity.	entity_id, entity_type
vectra-detections-mark-fixed	Marks one or more detections as fixed.	detection_ids
vectra-detections-unmark-fixed	Clears the fixed flag from one or more detections.	detection_ids
vectra-entity-detections-mark-asclosed	Closes all detections for a given entity with a close reason.	entity_id, entity_type, close_reason
vectra-detections-mark-asclosed	Closes one or more detections with a close reason.	detection_ids, close_reason
vectra-detections-mark-asopen	Reopens one or more detections.	detection_ids
vectra-detection-pcap-download	Downloads the PCAP file for a detection (if available).	detection_id
vectra-detection-tag-list	Returns the list of tags on a detection.	detection_id
vectra-detection-tag-add	Adds tags to a detection.	detection_id, tags
vectra-detection-tag-remove	Removes tags from a detection.	detection_id, tags
vectra-entity-note-list	Lists notes for a specific entity.	entity_id, entity_type
vectra-entity-note-add	Adds a note to an entity.	entity_id, entity_type, note
vectra-entity-note-update	Updates an existing note on an entity.	entity_id, entity_type, note_id, note
vectra-entity-note-remove	Removes a note from an entity.	entity_id, entity_type, note_id
vectra-entity-tag-list	Lists tags for a specific entity.	entity_id, entity_type
vectra-entity-tag-add	Adds one or more tags to an entity.	entity_id, entity_type, tags
vectra-entity-tag-remove	Removes one or more tags from an entity.	entity_id, entity_type, tags
vectra-assignment-list	Returns a list of all assignments.	entity_ids, entity_type, resolved, assignees, resolution, page, page_size, ...
vectra-assignment-outcome-list	Returns a list of all entity assignment outcomes.	page, page_size
vectra-entity-assignment-add	Adds an assignment to an entity.	entity_id, entity_type, user_id
vectra-entity-assignment-update	Changes the assignee for an existing assignment.	assignment_id, user_id
vectra-entity-assignment-resolve	Resolves an assignment with an outcome and optional triage details.	assignment_id, outcome, note, triage_as, detection_ids (csv)
vectra-group-list	Returns a list of groups.	group_type, importance, last_modified_timestamp, last_modified_by, group_name
vectra-group-assign	Assigns members to the specified group.	group_id, members
vectra-group-unassign	Unassigns members from the specified group.	group_id, members

Vectra XDR Commands Overview

These commands are available from the Cortex XSOAR War Room, automations, and playbooks. Each command talks to the Vectra XDR API and writes data into the incident context (under Vectra.*) and renders a human-readable table.

vectra-user-list

- **Purpose:** List Vectra XDR users/operators.
- **Description:** Returns users with optional filters for username, role and last login time.
- **Input Arguments:**
  - username – *(Optional)* Filter by username.
  - role – *(Optional)* Filter by role. Values: Admin, Read-Only, Restricted Admin, Security Analyst, Setting Admin, Super Admin.
  - last_login_timestamp – *(Optional)* Return only users whose last login is on/after this timestamp (supports relative like “2 days” and absolute formats).

vectra-entity-list

- **Purpose:** List entities (accounts/hosts) with rich filtering.
- **Description:** Returns entities and their urgency, severity, tags, notes, etc., with pagination and time filters.
- **Input Arguments:**
  - prioritized – *(Optional)* Only entities above the priority threshold (true / false).
  - entity_type – *(Optional)* account or host.
  - name – *(Optional)* Filter by entity name.
  - tags – *(Optional)* Filter by one or more tags (comma separated).
  - state – *(Optional)* active or inactive.
  - ordering – *(Optional)* Sort by urgency_score, last_detection_timestamp, etc. Supports -field for descending.
  - last_detection_timestamp – *(Optional)* Only entities with last detection at/after this time.
  - last_modified_timestamp – *(Optional)* Only entities modified at/after this time.
  - page – *(Optional)* Page number (default 1).
  - page_size – *(Optional)* Page size (max 5000, default 50).

vectra-entity-describe

- **Purpose:** Get full details of a single entity.
- **Description:** Returns a detailed profile for one account/host by ID, including detections, scores, notes, tags and metadata.
- **Input Arguments:**
  - entity_id – *(Required)* ID of the entity.
  - entity_type – *(Required)* host or account.

vectra-entity-detection-list

- **Purpose:** List detections for a specific entity.
- **Description:** Returns detections associated with an entity, with pagination and rich filtering on category, type, state, tags, etc.
- **Input Arguments:**
  - entity_id – *(Required)* ID of the entity.
  - entity_type – *(Required)* account or host.
  - page – *(Optional)* Page number (default 1).
  - page_size – *(Optional)* Page size (max 5000, default 50).
  - detection_category – *(Optional)* e.g. Command & Control, Botnet, Reconnaissance, Lateral Movement, Exfiltration, Info.
  - detection_type – *(Optional)* Filter by detection type.
  - last_timestamp – *(Optional)* Only detections with last timestamp on/after this value (supports relative & absolute formats).
  - detection_name – *(Optional)* Filter by detection name.
  - state – *(Optional)* Filter by state (default active).
  - tags – *(Optional)* Filter by one or more tags.

vectra-detection-describe

- **Purpose:** Get details of one or more detections by ID.
- **Description:** Returns detailed detection records for the specified detection ID list.
- **Input Arguments:**
  - detection_ids – *(Required)* Single ID or comma-separated detection IDs.
  - page – *(Optional)* Page number (default 1).
  - page_size – *(Optional)* Page size (max 5000, default 50).

vectra-entity-note-add

- **Purpose:** Add a note to an entity.
- **Description:** Creates a new note attached to an account/host entity (used heavily for mirroring notes between XSOAR and Vectra).
- **Input Arguments:**
  - entity_id – *(Required)* Entity ID.
  - entity_type – *(Required)* host or account.
  - note – *(Required)* Note text to add.

vectra-entity-note-update

- **Purpose:** Update an existing entity note.
- **Description:** Modifies the content of a note already attached to an entity.
- **Input Arguments:**
  - entity_id – *(Required)* Entity ID.

- entity_type – *(Required)* host or account.
- note_id – *(Required)* ID of the note to update.
- note – *(Required)* New note content.

vectra-entity-note-remove

- **Purpose:** Remove a note from an entity.
- **Description:** Deletes a specific note by ID from the given entity.
- **Input Arguments:**
  - entity_id – *(Required)* Entity ID.
  - entity_type – *(Required)* host or account.
  - note_id – *(Required)* ID of the note to remove.

vectra-entity-note-list

- **Purpose:** List notes for an entity.
- **Description:** Returns all notes attached to a specified account/host, including who created/modified them.
- **Input Arguments:**
  - entity_id – *(Required)* Entity ID.
  - entity_type – *(Required)* host or account.

vectra-entity-tag-add

- **Purpose:** Add tags to an entity.
- **Description:** Adds one or more tags to the specified host/account.
- **Input Arguments:**
  - entity_id – *(Required)* Entity ID.
  - entity_type – *(Required)* host or account.
  - tags – *(Required)* Comma-separated list of tags to add.

vectra-entity-tag-list

- **Purpose:** List tags on an entity.
- **Description:** Returns tags currently attached to the specified entity.
- **Input Arguments:**
  - entity_id – *(Required)* Entity ID.
  - entity_type – *(Required)* host or account.

vectra-entity-tag-remove

- **Purpose:** Remove tags from an entity.

- **Description:** Detaches one or more tags from a given entity.
- **Input Arguments:**
  - entity_id – *(Required)* Entity ID.
  - entity_type – *(Required)* host or account.
  - tags – *(Required)* Comma-separated list of tags to remove.

vectra-detections-mark-fixed

- **Purpose:** Mark detections as fixed.
- **Description:** Sets one or more detections to “fixed” state in Vectra.
- **Input Arguments:**
  - detection_ids – *(Required)* Single ID or comma-separated list of detection IDs to mark as fixed.

vectra-detections-unmark-fixed

- **Purpose:** Clear the “fixed” flag on detections.
- **Description:** Reverts detections previously marked as fixed.
- **Input Arguments:**
  - detection_ids – *(Required)* Single or comma-separated detection IDs to unmark.

vectra-entity-detections-mark-fixed

- **Purpose:** Mark all detections for an entity as fixed.
- **Description:** For the given entity, marks its associated detections as fixed.
- **Input Arguments:**
  - entity_id – *(Required)* Entity ID.
  - entity_type – *(Required)* account or host.

vectra-entity-assignment-add

- **Purpose:** Assign an entity to a user.
- **Description:** Creates a new assignment record linking an entity to a user.
- **Input Arguments:**
  - entity_id – *(Required)* Entity ID.
  - entity_type – *(Required)* account or host.
  - user_id – *(Required)* User ID to assign to.

vectra-entity-assignment-update

- **Purpose:** Reassign an existing entity assignment.
- **Description:** Updates an assignment to point to a different user.
- **Input Arguments:**

- assignment_id – *(Required)* Assignment ID.
- user_id – *(Required)* New user ID to assign to.

vectra-entity-assignment-resolve

- **Purpose:** Resolve an assignment with an outcome.
- **Description:** Closes an assignment and records outcome, note, triage action and optional detection IDs.
- **Input Arguments:**
  - assignment_id – *(Required)* Assignment ID to resolve.
  - outcome – *(Required)* Outcome title. Built-ins include Benign True Positive, Malicious True Positive, False Positive.
  - note – *(Optional)* Resolution note (default “Updated by XSOAR”).
  - triage_as – *(Optional)* Triage rule/category to apply.
  - detection_ids – *(Optional)* Detection IDs (comma-separated) affected by this resolution.

vectra-assignment-list

- **Purpose:** List entity assignments.
- **Description:** Returns assignments with filters for entity, resolution status, assignees, outcome, creation time, etc.
- **Input Arguments:**
  - entity_ids – *(Optional)* Comma-separated entity IDs.
  - entity_type – *(Optional)* account or host.
  - resolved – *(Optional)* Filter by resolution status (True / False).
  - assignees – *(Optional)* Filter by assignee user IDs (comma-separated).
  - resolution – *(Optional)* Filter by outcome IDs (comma-separated).
  - created_after – *(Optional)* Only assignments created after this timestamp (supports relative & absolute formats).
  - page – *(Optional)* Page number (default 1).
  - page_size – *(Optional)* Page size (default 50).

vectra-assignment-outcome-list

- **Purpose:** List all available assignment outcomes.
- **Description:** Returns built-in and custom outcomes that can be used when resolving assignments.
- **Input Arguments:**
  - page – *(Optional)* Page number (default 1).
  - page_size – *(Optional)* Page size (default 50).

vectra-detection-pcap-download

- **Purpose:** Download PCAP for a detection.

- **Description:** Retrieves the PCAP file associated with a detection as a file entry.
- **Input Arguments:**
  - detection_id – *(Required)* Detection ID whose PCAP should be downloaded.

vectra-entity-detections-mark-asclosed

- **Purpose:** Close all detections for an entity.
- **Description:** Marks all detections on the given entity as closed with a specified close reason.
- **Input Arguments:**
  - entity_id – *(Required)* Entity ID.
  - entity_type – *(Required)* account or host.
  - close_reason – *(Required)* Close reason – benign or remediated.

vectra-detections-mark-asclosed

- **Purpose:** Close specific detections by ID.
- **Description:** Marks one or more detections as closed with a reason.
- **Input Arguments:**
  - detection_ids – *(Required)* Single or comma-separated detection IDs.
  - close_reason – *(Required)* benign or remediated.

vectra-detections-mark-asopen

- **Purpose:** Re-open detections.
- **Description:** Re-opens detections previously closed in Vectra.
- **Input Arguments:**
  - detection_ids – *(Required)* Single or comma-separated detection IDs to re-open.

vectra-group-list

- **Purpose:** List Vectra groups (host/account/IP/domain).
- **Description:** Returns groups with member information and supports rich filtering by type, members, importance, description and last modification.
- **Input Arguments:**
  - group_type – *(Optional)* account, host, ip, or domain.
  - account_names – *(Optional)* Comma-separated account names (only valid when group_type=account).
  - domains – *(Optional)* Comma-separated domains (only valid when group_type=domain).
  - host_ids – *(Optional)* Comma-separated host IDs (only valid when group_type=host).
  - host_names – *(Optional)* Comma-separated host names (only valid when group_type=host).
  - importance – *(Optional)* high, medium, low, never_prioritize.
  - ips – *(Optional)* Comma-separated IPs (only valid when group_type=ip).
  - description – *(Optional)* Filter by group description.

- last_modified_timestamp – *(Optional)* Only groups modified at/after this time.
- last_modified_by – *(Optional)* Filter by user ID who last modified the group.
- group_name – *(Optional)* Filter by group name.

vectra-group-assign

- **Purpose:** Add members to a group.
- **Description:** Assigns accounts/hosts/domain patterns/IPs to a group.
- **Input Arguments:**
  - group_id – *(Required)* Group ID.
  - members – *(Required)* Comma-separated member values (e.g. host IDs, domains, IPs depending on group type).

vectra-group-unassign

- **Purpose:** Remove members from a group.
- **Description:** Unassigns one or more members from the specified group.
- **Input Arguments:**
  - group_id – *(Required)* Group ID.
  - members – *(Required)* Comma-separated members to remove.

vectra-detection-tag-list

- **Purpose:** List tags for a detection.
- **Description:** Returns tags attached to the given detection.
- **Input Arguments:**
  - detection_id – *(Required)* Detection ID.

vectra-detection-tag-add

- **Purpose:** Add tags to a detection.
- **Description:** Appends one or more tags to the specified detection.
- **Input Arguments:**
  - detection_id – *(Required)* Detection ID.
  - tags – *(Required)* Comma-separated tags to add.

vectra-detection-tag-remove

- **Purpose:** Remove tags from a detection.
- **Description:** Removes one or more tags from the given detection.
- **Input Arguments:**
  - detection_id – *(Required)* Detection ID.
  - tags – *(Required)* Comma-separated tags to remove.

Playbooks

Playbooks are used to define automation flows and can consist of multiple actions as well instructions for interfacing with other apps configured in the XSOAR environment. There are some playbooks included as they are used with the integration. These playbooks will be covered later but the included playbooks also serve as a starting point that demonstrate key techniques. The operator can take specific techniques from the included playbooks to create their own automations based on their individual use cases. The included playbooks, their descriptions, and the techniques demonstrated are outlined in Table 2. A deeper dive into playbooks is covered later in this document.

Table 2 - Included Playbooks

Playbook Name	Description	Techniques Demonstrated
Prepare Incident – Vectra XDR	Starting point to demonstrate workflow to manage incidents	Changing incident state, prompting for input, calling another playbook and passing data to it
Dispatch Incident – Vectra XDR	Prepare detections and assign to analyst	Fetch detection details, adding or updating assignment, adding notes
Add Note – Vectra XDR	Looks up entity, checks if it's a host or account and writes note to the entity	Attribute lookup, condition check, adding notes
New: Detections Assessment – Vectra XDR	This playbook conducts a detection assessment and saves the result in context data.	Running detection assessment, aggregating detection data, writing structured results to context for reuse
New: MDR Escalation Process – Vectra XDR	This playbook retrieves the MDR ticket number associated with the given entity by parsing its notes. It then collects the entity's active detections, performs a detection assessment, and emails the results.	Parsing notes to extract MDR ticket, collecting active detections, chaining detection assessment, email notification

Operations

Incident Creation Philosophy

Best practice: Generate incidents on an entity-by-entity basis versus detection-by-detection.

Why: A key pillar of Vectra AI's value proposition to organizations is SOC efficiency. Vectra accomplishes this by attributing behavioral detections to entities (currently, hosts & accounts), by leveraging AI to compute an urgency score that considers multiple factors such as detections, velocity of progression, significance of the entity itself, and finally bringing all detection and non-detection context together in one prioritized place. For this reason, we promote the generation of incidents based on entities in external tools to mirror the Vectra AI Platform value proposition which results in decreased ticket volume, alert fatigue, and false positives. The following tables show a real-world difference between a detection-centric (Table 3) approach (which competitors employ) to an entity-centric (Table 4) approach with Vectra. The result is an average reduction of 80% in ticket load, all while being laser-focused on what is most urgent.

Table 3 - Detection Centric

Month	# of Detections	Avg # of Tickets / Day
June 2023	418	14
July 2023	472	15
Aug 2023	762	25

Table 4 - Entity Centric

Month	# of Entities	Avg # of Tickets / Day	% change of tickets created
June 2023	107	4	-74%
July 2023	107	3	-77%
Aug 2023	88	3	-88%

With Vectra's entity-centric prioritization, the detections are still available and relevant but instead of managing each detection as an isolated incident, they are managed holistically at the entity.

Incident Priority

The recommended starting point is to configure the incident list column order to match figure 11. With that order defined, sort the Vectra XDR Entity Urgency Score column from highest to lowest. Incidents should then be investigated from top down. The default filter `-status:closed -category:job` will not show any closed incidents so the incidents will either be "Active" or "Pending". All incidents start as pending and when you select that incident ID, it will automatically initiate a playbook and will turn active. After sorting, the first incident in our list that requires attention is #347 (Figure 16).

Figure 16 - Incident Priority

Investigate

Selecting the pending incident will initiate the investigate process which will open up the incident layout. The initial tab contains all the incident info details grouped into various sections. There is a section in the incident

info layout titled Work Plan and upon first run there should be one work item waiting user attention (Figure 17). The work plan is waiting for the operator to provide the user ID (in Vectra) to assign the incident to. It's possible to view in Task or Work Plan but viewing in the War Room is preferred.

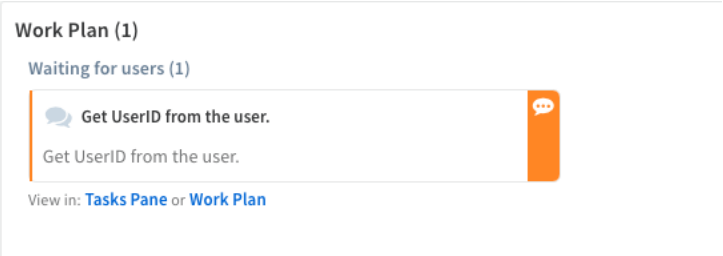
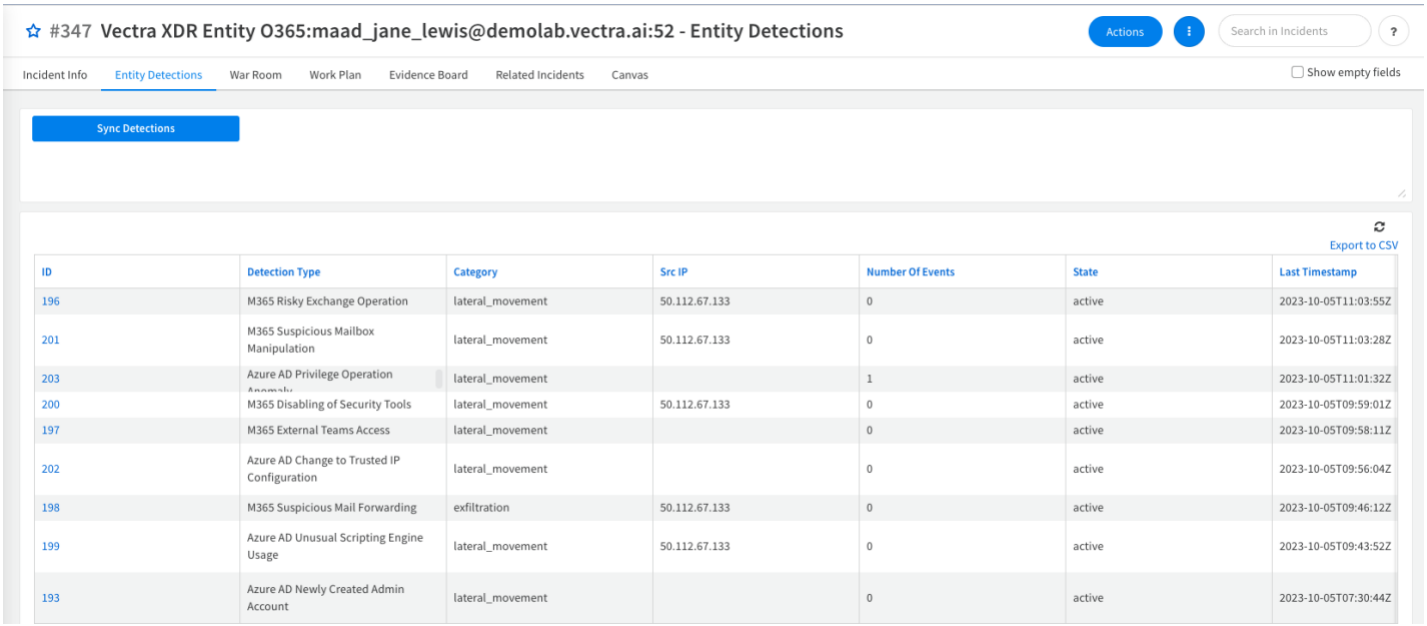


Figure 17 - Work Plan

Detections

Prior to assigning (dispatching) the incident, the operator may wish to review the detections associated with the incident. The detection tab will outline all the detections (Figure 18). Selecting any detection ID will spawn a new tab directed at the specific detection inside the Vectra UI so the operator can review specific details.

The Sync Detections button can be used to force XSOAR to check Vectra for any new or updated detections attributed to the entity. The detections will automatically update during the polling window but if the polling interval is very long or polling is disabled, this button can be used to refresh the data.



ID	Detection Type	Category	Src IP	Number Of Events	State	Last Timestamp
196	M365 Risky Exchange Operation	lateral_movement	50.112.67.133	0	active	2023-10-05T11:03:55Z
201	M365 Suspicious Mailbox Manipulation	lateral_movement	50.112.67.133	0	active	2023-10-05T11:03:28Z
203	Azure AD Privilege Operation Anomaly	lateral_movement		1	active	2023-10-05T11:01:32Z
200	M365 Disabling of Security Tools	lateral_movement	50.112.67.133	0	active	2023-10-05T09:59:01Z
197	M365 External Teams Access	lateral_movement		0	active	2023-10-05T09:58:11Z
202	Azure AD Change to Trusted IP Configuration	lateral_movement		0	active	2023-10-05T09:56:04Z
198	M365 Suspicious Mail Forwarding	exfiltration	50.112.67.133	0	active	2023-10-05T09:46:12Z
199	Azure AD Unusual Scripting Engine Usage	lateral_movement	50.112.67.133	0	active	2023-10-05T09:43:52Z
193	Azure AD Newly Created Admin Account	lateral_movement		0	active	2023-10-05T07:30:44Z

Figure 18 - Detections

War Room

Loading the War Room and scrolling to the bottom (most recent information) the operator will notice that a task named 'List Available Users for Assignment' has run and a user table is displayed. This table provides the corresponding User ID for each user as the user id is required for assignment. Below that, the operator

will notice that an execution has been paused, waiting for manual input. This is the task to assign the Vectra user to the incident (Figure 19). The operator can complete in the task pane or navigate to the Work Plan to follow the workflow.

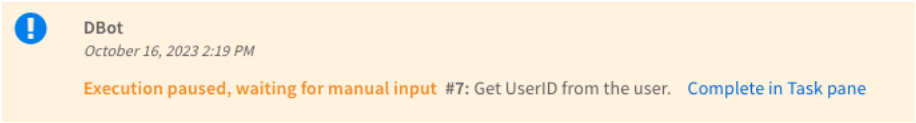


Figure 19 - User Assignment

Work Plan

The initial playbook to process the incident begins with incident investigation. The playbook runs automatically until it reaches a block where it requests input from the operator. After the operator supplies a user ID for assignment, the playbook will continue (Figure 20).

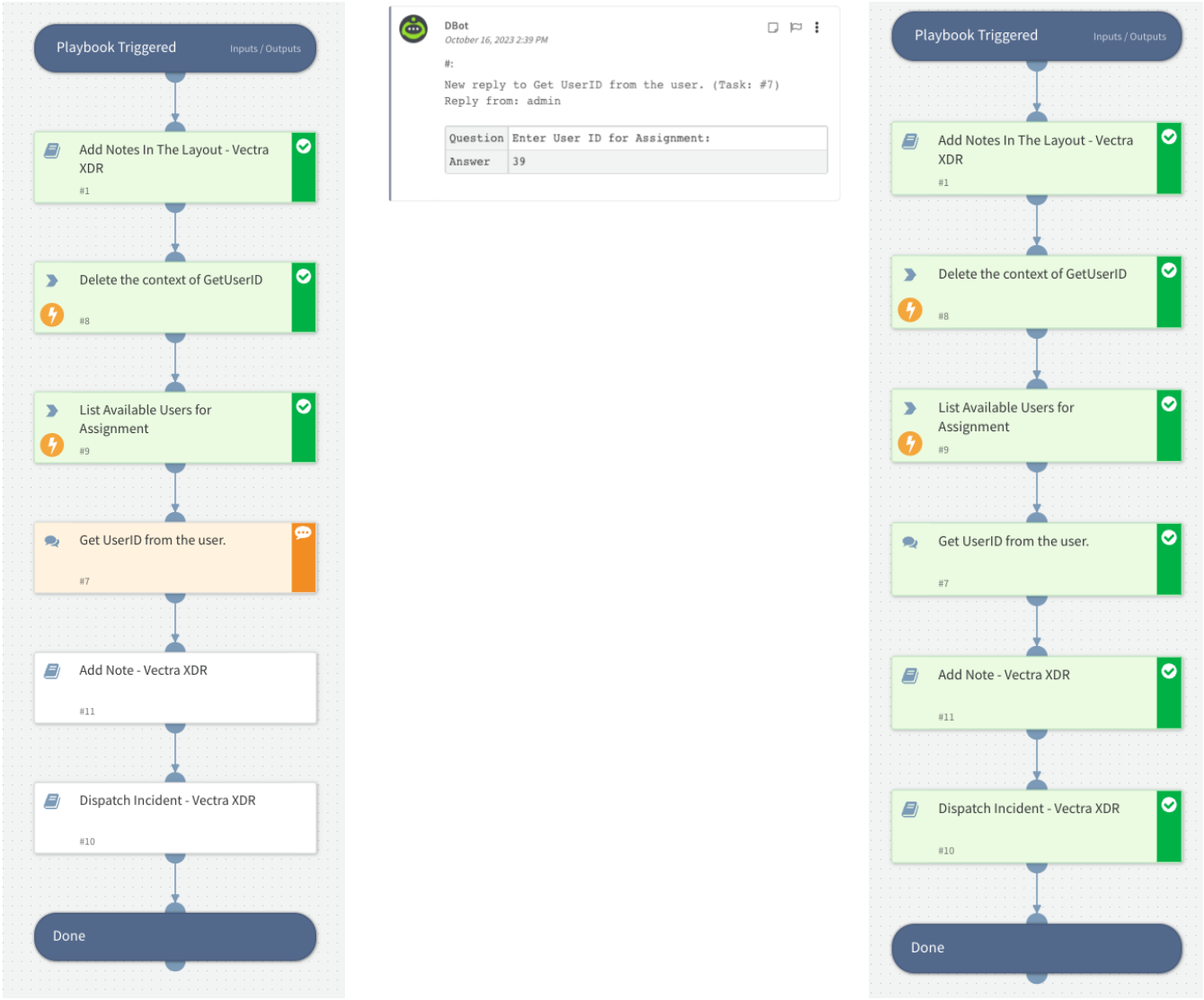


Figure 20 - Process Incident Work Plan

Sync Assignment

The incident info page contains a section on assignment. The assigned user will automatically update on the next poll but the **Sync Assignment** button can be used to force the system to refresh the information.

Basic Workflow

The following diagram (Figure 21) outlines a starter workflow that can be employed. The top line refers to the three XSOAR status labels (pending, active, closed) that are tied to an incident. The second line provides more color on the stage of the incident life cycle.

The status of an un-opened incident is set to pending by default and this equates to the new stage since no analyst is working the incident yet. Once the analyst (or incident commander) starts working the incident (investigate), the Process Incident playbook automatically kicks off and this will automatically change the status to active. Until the analyst supplies the user ID for assignment, the incident run status will be shown as Waiting.

The event is now in the active state and should progress through the investigation and/or remediation stages which are outside the scope of this document. The intent is to adjudicate the incident, and this may involve manual or automated processes.

Once the incident has been adjudicated it can move to the resolution stage. In the resolution stage the operator should run the resolve assignment action to close the incident on Vectra. Finally, the operator should set the XSOAR event status to "Closed".

Figure 21 - Recommended Basic Workflow

Intermediate Workflow

The following diagram (Figure 22) outlines an alternative workflow that is more intermediate. This workflow closely follows the basic workflow, but the resolution is more automated. This requires the playbook Resolve Incident – Vectra XDR to be downloaded from the Vectra GitHub location (refer to the section on Working with Playbooks). When an incident is ready for closing (resolution), navigate to the Work Plan tab within the incident and change the playbook to point to the resolve incident playbook. After confirming in the affirmative, select the "Get Detection IDs" task and it will open the results which will show a comma separated list of the detection IDs for the entity in question. That data is required for input so copy just the CSV data to the clipboard. Select the next task and paste the output into the requested area. Continue to the next task and provide input for the closing details. Once that is entered, the playbook will continue to run, and any errors

will be highlighted in red. If everything is green, then the entity has been resolved on Vectra and the incident will be marked as closed in XSOAR.

There is a limitation that if overlooked may result in a failure to resolve the assignment. The 'Get Detection IDs' task will return all detections associated with an entity and this can include detections that have already been triaged. Any previously triaged detections cannot be included in the csv list, or it will result in a failure to resolve. To avoid this potential error, prior to initiating the playbook, use the incident link URL in the incident info to pivot into Vectra to check if any detections have been triaged. Collect those detection IDs and make sure to exclude them in the csv list when prompted. This will be addressed in a future release.

Figure 22 - Intermediate Workflow

Running Actions - General

It's possible to run actions from within the incident container. Where the action is run from will depend on the required parameters, but the output of actions will be displayed in the 'War Room. For the most part, the action names include the artifact type they apply to. For instance, actions such as 'describe entity' or 'list entity detections', apply to entity artifacts, whereas 'mark detection' or 'describe detection', applies to detection artifacts. Table 1 identifies the artifact type by the content in the requires column. In the following example (Figure 23), I wish to list the detections for an entity, and I know from Table 1 that this requires entity_id and entity_type so I will run this action from within 'Incident Info' because the data exists there.

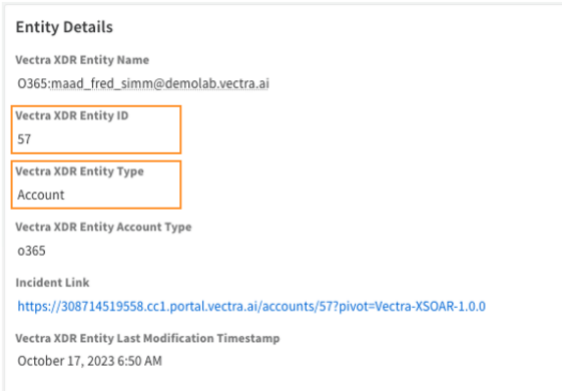


Figure 23 - Action Requirements

The action is initiated from the command bar at the bottom of the screen. Once you start typing a command that is prefixed with "!", autocomplete will assist and it will also provide context to the required parameters (Figure 24).

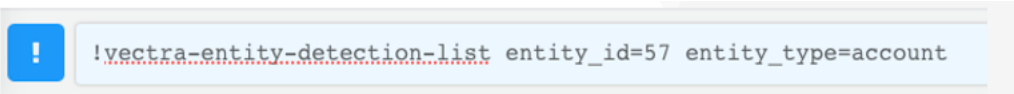


Figure 24 - Run Action

The output of the action is displayed in the 'War Room (Figure 25).

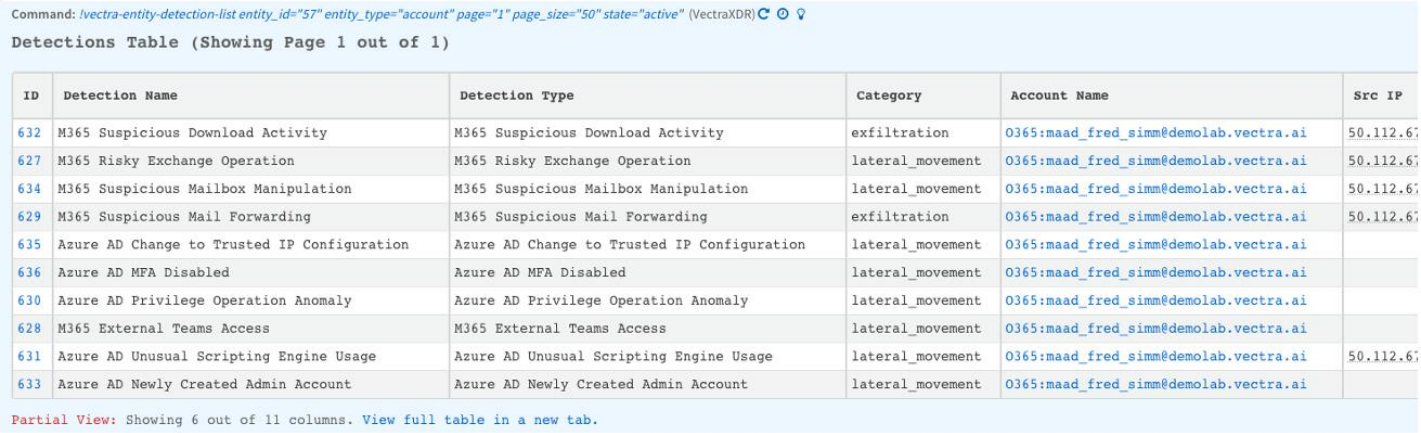


Figure 25 - Action Output

Running Actions – Resolve Assignment

Most actions are self-explanatory and are relatively finite so it's not necessary to cover each one in detail. The action that requires the most explanation is the 'Resolve Assignment' action. This action covers several components and is very powerful, but it does require some prep work. When run, this action will take an entity that has been assigned and will mark it resolved including adding a resolution outcome, adding a note for the Vectra operational metrics report, optionally triage the detections associated with the entity and place the desired label on the triaged detections. The following table (Table 5) highlights the input that is required to run the action (Figure 26) and where to find the relevant data.

Table 5 - Resolve Assignment Parameters

Parameter	Format	Source
Assignment ID	Single integer	Incident Info – entity assignment details
Outcome	Pick List	Benign True Positive, Malicious True Positive, False Positive
Note	Free Form String	Free form – this note only appears in operational metrics report
Triage As	Free Form String	Short label that shows up beside the triaged detection in Vectra
Detection ID's	Multiple integers csv	Run describe entity detections action and obtain the data

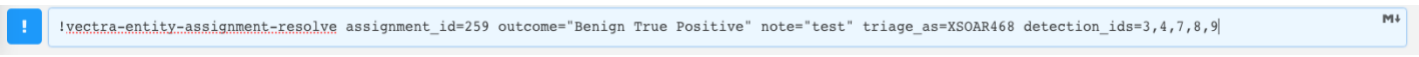


Figure 26 - Resolve Assignment Action

Working with Playbooks

Playbooks will evolve over time and new content will be created to support specific use cases. They playbooks created or curated by Vectra aren't necessarily intended to function out of the box. In most cases some form of customization is required to operate inside the destination environment. The playbooks provided offer the starting point and structure to ease any customization burden.

Playbooks

Vectra provided playbooks and corresponding documentation are available for download from the following GitHub repository:

https://github.com/vectranetworks/palo_xsoar_vectra_xdr

This repository includes documentation pertaining to using playbooks with the Vectra platform as well as several playbooks. Please refer to the repository for additional details on the topic of playbooks.

Known Limitations

Detection Notes and Tags

It is not possible to add notes or tags directly to detections with this integration. This is by design so that the entire entity is investigated. Where appropriate, notes and tags should be added to the entity level and/or inside XSOAR incident management. All detections are attributed to the entity and are accessible via the entity itself.

Mirroring (Synchronization)

Deleting a tag from an entity in Vectra will not remove the tag from XSOAR even with incoming mirroring enabled. This is a known limitation because XSOAR Tags is a protected field.

Multiple Tenants

When working with multiple Vectra tenants the actions (including automations inside playbooks) need to know which tenant to communicate with. This is accomplished by incorporating the 'using' parameter within the command '*using=instance_name'*. To automate this, the instance name to use can be retrieved from the 'sourceinstance' context data.

Troubleshooting

No incidents

If no events are displaying after configuration this could be the result of several things.

An incorrect key could have been entered during configuration. Use the 'test connectivity' button under instance settings to validate.

The instance may not be configured to fetch incidents (set to 'Do not fetch').

The filters that have been configured restricts the data that is initially received and if the filters are too restrictive, it's possible there is no matching data. Modify the "Instance Settings" filters to poll all entities for prioritized and remove any other filters to isolate the issue.

Review the polling settings under "Incident Fetch Interval" to ensure that a polling interval or schedule has been set correctly.

The instance configuration itself may be disabled. When it's running it should show "Disable" as per the image below as clicking that icon will instruct the system to disable the configuration.

Playbook not running properly

When troubleshooting playbooks use "Playbook Debugger Panel" to isolate testing. In the debugger panel it's possible to select an existing incident to use as test data.

Worldwide Support Contact Information

- Support portal: https://support.vectra.ai/ (preferred contact method)
- Email: support@vectra.ai
- Additional information: https://www.vectra.ai/support