Armis Centrix™

Google Chronicle

November 7, 2023

Disclaimer: This integration may not yet be available in your environment.

Please contact your account team.

Table of contents

1 Overview

- This document describes how to deploy and configure the ingestion script for Armis Chronicle Integration.
  - **Chronicle Platform**—Chronicle is a cybersecurity telemetry platform for threat hunting, and threat intelligence and is part of the Google Cloud Platform. Chronicle stores log events it receives in two formats: either as the original raw log or structured Unified Data Model (UDM) log. There are two critical elements to consider for parsing, Unified Data Model (UDM) which defines the schema for parsing, and Configuration Based Normalizers (CBN) which describes how to log data is transformed to the UDM schema.
  - **Armis Centrix™**—Armis offers the market's leading asset intelligence platform designed to address the new threat landscape that connected devices create. It provides passive and unparalleled cybersecurity asset management, risk management, and automated enforcement.
  - **Chronicle Integration for Armis**—The Chronicle integration for Armis enables the transfer and parsing of Armis Alerts, Activities, Devices, and Vulnerabilities in the Chronicle. These parsed events can be utilized for search, reporting, and visualization workflows.
- The ingestion script ingests the following types of event categories:
  - Armis Alerts
  - Armis Activities
  - Armis Devices
  - Armis Vulnerabilities

2 Prerequisites

- **customer_id** and **service_account** for the Chronicle.
- A **secret_key** and URL for Armis.
- Poll interval and Scheduler time should be the same. If the poll interval and scheduler time are not the same then data loss (**Poll interval** > **Scheduler time**) or data duplication (**Poll interval** > **Scheduler time**) may occur.
- Secret Manager should be configured and it should contain values of the **secret_key** of Armis and service account details of Chronicle.
- 4-core CPU is recommended for cloud function configuration.

3 Steps for deploying and executing the ingestion script

3.1 Add the secret in Secret Manager

1. Log in to the https://console.cloud.google.com/ using valid credentials.
2. Navigate to the **Secret Manager**.
3. Click **Create Secret**.
4. Provide the name for the secret in the **Name** field.
5. Upload the file if there is a file for the secret, or provide the secret value directly in the **Secret Value** field.
6. Click **Create Secret**.

For more information about how to create secrets, refer to this page.

Once the secrets are created in the Secret Manager, use the secret's resource ID as the value for environment variables. For example

```
CHRONICLE_SERVICE_ACCOUNT:
projects/{project_id}/secrets/{secret_id}/versions/{version_id}
```

3.2 Deploying the function on GCloud

1. Insert the necessary details into the **.env.yml** file.

```
CHRONICLE_CUSTOMER_ID:
CHRONICLE_REGION: "us"
CHRONICLE_SERVICE_ACCOUNT: <use the path of service account created in the
secret manager>
CHRONICLE_NAMESPACE:
POLL_INTERVAL: "10"
ARMIS_SERVER_URL:
ARMIS_API_SECRET_KEY: <use the path of the secret key created in the secret
manager>
HTTPS_PROXY:
CHRONICLE_DATA_TYPE:
```

2. Run the command for Gen1: (add the required function in the command).

   Command for deploying Cloud Function using CLI for Gen1

```
gcloud functions deploy <FUNCTION NAME>  --entry-point main --trigger-http
--runtime python39 --env-vars-file .env.yml
```

   Run the command for Gen2: (add the required function in the command).

   Command for deploying Cloud Function using CLI for Gen2

```
gcloud functions deploy <FUNCTION NAME> --gen2 --entry-point main --
trigger-http --
runtime python39 --env-vars-file .env.yml
```

3. **Verify the function manually for Gen1:**

   1. Click the deployed function.
   2. Navigate to the **Testing** module.
   3. Click the **Test the function** button.
   4. Navigate to the **Logs** module.

   Verify the function manually for Gen2:

   1. Click the deployed function.
   2. Navigate to the **Testing** module.
   3. Click the **TEST IN CLOUD SHELL** button.
   4. Hit to enters the cloud shell terminal.
   5. Navigate to the **Logs** module.

4. Cloud Function Default Specifications:

Variable	Default Value	Description
Memory	256 MB	Allocated memory for a specific cloud function.
Timeout	60 seconds	Time Interval for the termination of a cloud function.
Region	us-central1	Region for a cloud function.
Minimum instances	0	Minimum number of instances for a cloud function.
Maximum instances	100	Maximum number of instances for a cloud function.

- You can find the configuration documentation of the above variables here: link

3.3 Steps to create the scheduler

1. Log in to the https://console.cloud.google.com/ using valid credentials.
2. Navigate to **Cloud scheduler**.
3. Click **Create a Job**.
4. Give the name of the scheduler job.
5. Select the frequency for when the scheduler job should invoke.
6. Select the time zone and click **Continue**.
7. Select the target type as **HTTP**.
8. In the GCloud interface, navigate to trigger sections and copy the trigger URL of the function that the user needs to create a scheduler and paste it into the URL.
9. Select **Add OIDC Token** in Auth Header.
10. Select the Service account.
11. Click **Create**.

3.4 Setting up the directory

Create a new directory for the cloud function deployment and add the following files into that directory:

1. Contents of ingestion script (i.e., **armis**).
2. **common** directory.

3.5 Setting the required runtime environment variables

- Navigate to the **.env.yml** file which can be found in the folder of every event category.
- Edit the **.env.yml** file to populate all the required environment variables.

> NOTE: Information related to all the environment variables can be found in the README.md file.

- Below are environment variables that need to be added in the **.env.yml** file.

Variable	Description	Example dummy value	Required	Default	Secret
CHRONICLE_ CUSTOMER_ID	Chronicle customer ID.	00000000-0000-0000-0000-000000000000	Yes	-	No
CHRONICLE_ REGION	Chronicle region.	us	Yes	us	No
CHRONICLE_ SERVICE_ ACCOUNT	Path of the Google Secret Manager with the version, where the Chronicle Service Account is stored.	projects/{project_ id}/secrets/{secret_ id}/versions/{version_ id}	Yes	-	Yes
CHRONICLE_ NAMESPACE	The namespace that the Chronicle logs are labeled with.	dummy-namespace	No	-	No
POLL_ INTERVAL	Frequency interval at which the function executes to get additional log data (in minutes). This duration must be the same as the Cloud Scheduler job interval.	“10” Note: Value should be in quotes.	Yes	10	No
ARMIS_ SERVER_URL	Server URL of Armis platform.	https://lab-dummy.armis.com	Yes	-	No
ARMIS_API_ SECRET_KEY	Path of the Google Secret Manager with the version, where the Armis API Secret key is stored.	projects/{project_ id}/secrets/{secret_ id}/versions/{version_ id}	Yes	-	Yes
HTTPS_PROXY	Proxy server URL.	http://address:port	No	-	No
CHRONICLE_ DATA_TYPE	Chronicle data type to push data into the Chronicle.	ARMIS_ ALERTS,ARMIS_ ACTIVITIES,ARMIS_ DEVICES,ARMIS_ VULNRABILITIES	Yes	-	No

3.5.1 Configuring the namespace

- The namespace that the Chronicle logs are ingested into can be configured by setting the **CHRONICLE_ NAMESPACE** environment variable.

3.5.2 Using Secrets

- Environment variables marked as secret must be configured as secrets on Google Secret Manager. For more information, refer to REF.
- Once the secrets are created on Secret Manager, use the secret's resource ID as the value for environment variables.

For example:

```
CHRONICLE_SERVICE_ACCOUNT: projects/{project_id}/secrets/{secret_
id}/versions/{version_id}
```

4 View Parsed logs in the Chronicle UI

- Log in to https://crestdatasys.backstory.chronicle.security/.
- Type `.*` in the search field and click **Search**.
- Click on raw log search.
- Select **Run Query as Regex**.
- Set the time interval in which the logs are ingested.
- Select the log source and click search.
- Open any particular log to check.
- Parsed logs will be visible under the **UDM Event** section in Chronicle.

5 Mappings

5.1 Alerts field mapping

UDM Field Name	RawLog Field Name	Logic
metadata.event_type		This field is set to “GENERIC_EVENT”.
metadata.product_name		This field is set to “ARMIS”.
metadata.vendor_name		This field is set to “ARMIS”.
metadata.product_log_id	alertId	
metadata.description	description	
network.session_id	connectionIds	`connectionIds` is of type array. The values of an array will be converted into comma separated string and then mapped with the UDM field.
about.labels[activity_uuid]	activityUUIDs	
principal.asset.asset_id	deviceIds	First assignees object would be mapped in principal.asset.asset_id and remaining would be mapped in principal.labels
security_result.severity	severity	If Severity is equal to "High" then security_result.severity is set to "HIGH". Else If Severity is equal to "Low" then security_result.severity is set to "LOW". Else If Severity is equal to "Medium" then security_result.severity is set to "MEDIUM".

UDM Field Name	RawLog Field Name	Logic
security_result.severity_details	severity	
security_result.alert_state	status	If Status is equal to "Unhandled" then security_result.alert_state is set to "ALERTING". Else If Status is equal to "Resolved" or Status is equal to "Suppressed" then security_result.alert_state is set to "NOT ALERTING".
metadata.event_timestamp	time	
metadata.product_event_type	title	metadata.product_event_type is set to "ALERT: <value of title>".
security_result.category	type	If type is equal to "System Policy Violation" or type is equal to "Policy Violation" then security_result.category is set to "POLICY_VIOLATION".
security_result.category_details	type	

5.2 Activities field mapping

UDM Field Name	RawLog Field Name	Logic
metadata.event_type		This field is set to “GENERIC_EVENT”.
metadata.product_name		This field is set to “ARMIS”.
metadata.vendor_name		This field is set to “ARMIS”.
metadata.product_log_id	activityUUID	
metadata.event_timestamp	time	
metadata.description	title	
metadata.product_event_type	type	
network.session_id	connectionIds	`connectionIds` is of type array. The values of an array will be converted into comma separated string and then mapped with the UDM field.
security_result.description	content	
security_result.severity		If current_severity is equal to "High" then security_result.severity is set to "HIGH". Else If current_severity is equal to

UDM Field Name	RawLog Field Name	Logic
		"Low" then security_result.severity is set to "LOW". Else If current_severity is equal to "Medium" then security_result.severity is set to "MEDIUM".
security_result.severity_details		Extracted "current_severity" from "title" field using grok pattern and then mapped with the UDM field.
security_result.detection_fields["past_severity"]		Extracted "past_severity" from "title" field using grok pattern and then mapped with the UDM field.
security_result.detection_fields["past_risk_score"]		Extracted "past_risk_score" from "title" field using grok pattern and then mapped with the UDM field.
security_result.detection_fields ["present_risk_score"]		Extracted "present_risk_score" from "title" field using grok pattern and then mapped with the UDM field.
principal.asset.asset_id	deviceIds	First assignees object would be mapped in principal.asset.asset_id and remaining would be mapped with principal.labels
principal.asset.attribute.labels ["device_name"]		Extracted "device_name" from "title" field using grok pattern and then mapped with the UDM field.
target.ip		Extracted "target_ip" from "title" field using grok pattern and then mapped with the UDM field.
target.labels["interface_name"]		Extracted "interface_name" from "title" field using grok pattern and then mapped with the UDM field.
intermediary.asset.asset_id	sensor.name	`intermediary.asset.asset_id` is set to "ASSET_ID: <value of sensor.name>".
intermediary.asset.category	sensor.type	
about.labels["protocol"]	protocol	
about.labels["site"]	site	
extensions.vulns.vulnerabilities.cve_id		Extracted "cve_id" from "title" field using grok pattern and then mapped with the UDM field.

5.3 Devices field mapping

UDM Field Name	RawLog Field Name	Logic
metadata.product_name		This field is set to “ARMIS”.
metadata.vendor_name		This field is set to “ARMIS”.
metadata.entity_type		\`metadata.entity_type\` is set to "ASSET".
entity.asset.asset_type	type	If type is equal to "Mobile Phones" then entity.asset.asset_type is set to "MOBILE" Else If type is equal to "Personal Computers" then entity.asset.asset_type is set to "LAPTOP".
entity.asset.first_seen_time	firstSeen	
entity.asset.last_discover_time	lastSeen	
metadata.product_entity_id	id	
entity.ip	ipAddress,ipv6	
entity.mac	macAddress	
entity.asset.hardware.manufacturer	manufacturer	
entity.asset.hardware.model	model	
entity.asset.asset_id	name	\`entity.asset.asset_id\` is set to "ASSET_ ID: <value of name>".
entity.platform	operatingSystem	If operatingSystem is equal to "ios" then entity.platform is set to "IOS". Else "operating_system" will be mapped as a key for entity.labels with value of operatingSystem.
entity.platform_version	operatingSystem	
entity.user.userid	userIds	First assignees object would be mapped in entity.user.userid and remaining would be mapped with entity.labels
entity.labels["risk_level"]	riskLevel	
entity.labels["access_switch"]	accessSwitch	
entity.labels["boundaries"]	boundaries	
entity.labels["business_impact"]	businessImpact	
entity.asset.category	category	

UDM Field Name	RawLog Field Name	Logic
entity.labels["sensor_name"]	sensor.name	
entity.labels["sensor_type"]	sensor.type	
entity.labels["site_location"]	site.location	
entity.labels["site_name"]	site.name	
entity.labels["tags"]	tags	tags` is of type array. The values of an array will be converted into comma separated strings and then mapped with the UDM field.
entity.labels["visibility"]	visibility	
relations.entity.asset.first_seen_time	dataSources.firstSeen	
relations.entity.asset.last_discover_time	dataSources.lastSeen	
relations.relationship		This field is set to `OWNS`.
relations.entity_type	dataSources.name	If dataSources.name is equal to "User" then relations.entity_type is set to "USER" Else relations.entity_type is set to "RESOURCE".
relations.entity.asset.asset_id	dataSources.name	
relations.entity.labels[datasource_type]	dataSources.types	Type of dataSources.types is an array. The values of an array will be converted into comma separated string and then mapped with the UDM field.

5.4 Vulnerabilities field mapping

UDM Field Name	RawLog Field Name	Logic
metadata.event_type		This field is set to “GENERIC_EVENT”.
metadata.product_name		This field is set to “ARMIS”.
metadata.vendor_name		This field is set to “ARMIS”.
metadata.url_back_to_product	vulnerabilities_matches	
metadata.product_log_id	id	
metadata.description	description	
extensions.vulns.vulnerabilities.cve_id	cveUid	
security_result.detection_fields ["affected_devices_count"]	affectedDevicesCount	

UDM Field Name	RawLog Field Name	Logic
extensions.vulns.vulnerabilities.about.labels["attack_complexity"]	attackComplexity	
extensions.vulns.vulnerabilities.about.labels["attack_vector"]	attackVector	
extensions.vulns.vulnerabilities.about.labels["availability_impact"]	availabilityImpact	
extensions.vulns.vulnerabilities.about.labels["botnets"]	botnets	`botnets` is of type array. The values of an array will be converted into comma-separated strings and then mapped with the UDM field.
extensions.vulns.vulnerabilities.about.labels["cisa_due_date"]	cisaDueDate	
extensions.vulns.vulnerabilities.name	commonName	
security_result.detection_fields ["confidentiality_impact"]	confidentialityImpact	
extensions.vulns.vulnerabilities.cvss_base_score	cvssScore	
extensions.vulns.vulnerabilities.about.labels["epss_percentile"]	epssPercentile	
extensions.vulns.vulnerabilities.about.labels["epss_score"]	epssScore	
extensions.vulns.vulnerabilities.first_found	firstReferencePublishDate	
extensions.vulns.vulnerabilities.about.labels["first_weaponized_ reference_publish_date"]	firstWeaponizedReferencePublishDate	
extensions.vulns.vulnerabilities.about.labels["has_ransomware"]	hasRansomware	
extensions.vulns.vulnerabilities.about.labels["exploitability_score"]	exploitabilityScore	
security_result.detection_fields ["impact_score"]	impactScore	
security_result.detection_fields ["integrity_impact"]	integrityImpact	
extensions.vulns.vulnerabilities.about.labels["is_weaponized"]	isWeaponized	
extensions.vulns.vulnerabilities.about.labels["latest_exploit_	latestExploitUpdate	

UDM Field Name	RawLog Field Name	Logic
update"]		
security_result.detection_fields ["num_of_exploits"]	numOfExploits	
security_result.detection_fields ["number_of_threat_actors"]	numberOfThreatActors	
security_result.detection_fields ["avm_rating"]	avmRating	
security_result.detection_fields ["org_priority_manual_change_reason"]	orgPriorityManualChangeReason	
principal.user.userid	orgPriorityManualChangedBy	
principal.labels["org_priority_manual_update_time"]	orgPriorityManualUpdateTime	
security_result.detection_fields ["privileges_required"]	privilegesRequired	
extensions.vulns.vulnerabilities.about.labels["published_date"]	publishedDate	
extensions.vulns.vulnerabilities.about.labels["reported_by_google_zero_days"]	reportedByGoogleZeroDays	

UDM Field Name	RawLog Field Name	Logic
extensions.vulns.vulnerabilities.about.labels["scope"]	scope	
extensions.vulns.vulnerabilities.severity	severity	If severity is equal to "Critical" then extensions.vulns.vulnerabilities.severity is set to "CRITICAL" Else If severity is equal to "High" then extensions.vulns.vulnerabilities.severity.severity is set to "HIGH" Else if severity is equal to "Medium" then extensions.vulns.vulnerabilities.severity.severity is set to "MEDIUM" Else If severity is equal to "Low" then extensions.vulns.vulnerabilities.severity.severity is set to "LOW".
extensions.vulns.vulnerabilities.severity_details	severity	
extensions.vulns.vulnerabilities.about.labels["status"]	status	
extensions.vulns.vulnerabilities.about.labels["threat_tags"]	threatTags	`threatTags` is of type array. The values of an array will be converted into comma-separated strings and then mapped with the UDM field.
about.labels["user_interaction"]	userInteraction	

6 Limitations

- CBN parser will only be able to parse the log mentioned in Supported log Types/Events.
- We suggest using the second generation of Cloud Function. The first generation of Cloud Function has a maximum execution time of 9 minutes and the second generation of Cloud Function has a maximum execution time of 60 minutes. If the execution time of the Cloud Function exceeds timeout then there are chances that the complete data (Alerts, Activities, Devices, Vulnerabilities) is not ingested in the Chronicle.
- Only the HTTPS protocol is supported for Armis API calls.
- Chronicle does not support duplicate batch entries, so if the whole batch is duplicated then it will not ingest into the Chronicle.
- Cloud function has a limitation in that each instance of a function handles only one concurrent request at a time. This will result in data loss if the data fetching process takes time more than the scheduled time. For Example: If the Poll interval and Scheduler time are set to 10 minutes so the scheduler will invoke the cloud function every 10 minutes but suppose the scheduler invokes the function for the first time and it takes 15 minutes to fetch the data then, in this case, the second invocation will be skipped and data loss will occur as a result.
- Poll interval should not be greater than 99 days to avoid few edge cases as Armis API supports a maximum timeframe of 100 days.
- At a time there would be only one valid value of poll interval for all the different event data types.

7 Steps to fetch the historical data

Steps to fetch the historical data all at once and then continue with the real-time data collection:

- Configure the `POLL_INTERVAL` environment variable in minutes for which the historical data needs to be fetched.
- As the cloud function is configured, the function can be triggered using a scheduler or manually by executing the command in Google Cloud CLI.

8 Troubleshooting

- GCloud logs can be used for troubleshooting.

Steps on how we can get GCloud logs:

- Log in to the "https://console.cloud.google.com/ using valid credentials.
- Navigate to 'Cloud functions' and click on the deployed function where you can find the logs module.
- Logs can be filtered using severity.
- Sometimes GCloud default logs are not visible in the logs module of Cloud Function after testing the function manually or when the scheduler job invokes the function. To resolve this issue a quick page refresh is required on the GCloud.
- If you test the cloud function immediately after deploying it on gcloud, It might be possible that the cloud function will not work as expected. To resolve this, wait for a few seconds and then test it.
- If in case, the cloud function stops its execution because memory exceeds the limit, configure the cloud function's memory configuration and increase the memory limit. For more information click here.
- Unauthenticated invocation should be set as true while deploying cloud function.
- If output generated from parser is not as per the expectation, make sure to disable the parser extension and to double check the mappings mentioned in the mapping document.
- If the issue still persists, please contact support@armis.com

9 References

- Install the gcloud CLI
- Deploying cloud functions from local machine