Bringing Darktrace into your Managed SOC Services

Contents

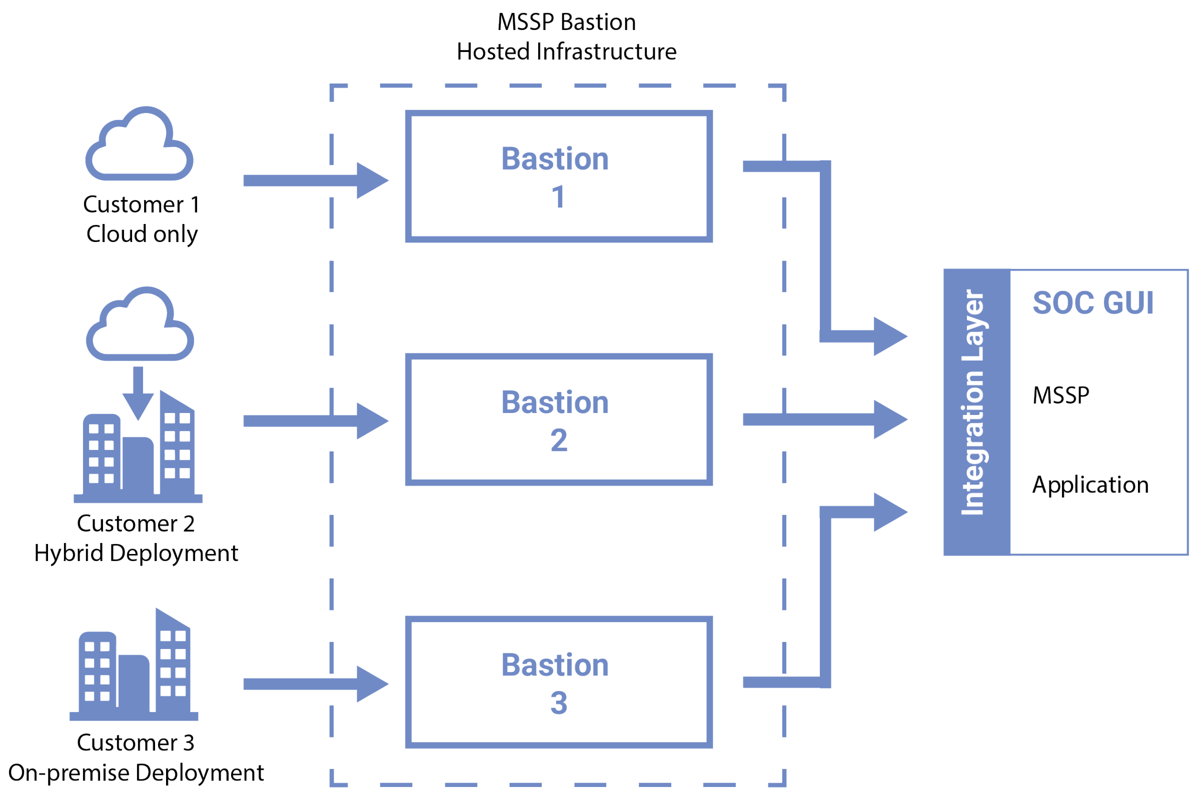
Introduction

Darktrace's Managed Security Service Provider (MSSP) Partner Program allows organizations seeking to bolster the cyber defense services they offer with Darktrace's Enterprise Immune System.

MSSP partners may choose to manage customer support services on Darktrace's behalf in their own data center, in the end user's data center, or in a third-party cloud environment. After licensing Darktrace's solution, MSSP partners have the opportunity to bundle it with other services at whatever rate they deem appropriate.

Once installed in a customer's environment, Darktrace's self-learning technology works to understand the 'pattern of life' of the environment it is in. As such, data from each customer environment needs to be kept isolated from others, so it is not possible to multitenant a single master appliance across multiple customers.

In order to scale service offerings across multiple customers, MSSPs can leverage Darktrace's API to access all the data available within the Darktrace platform, and present it through a single custom-built interface. This allows for rapid customized exporting, integration and orchestration of the Darktrace data directly within you Security Operations Center (SOC).



This document will cover how to authenticate and retrieve information (including alerts in a JSON format) from a Darktrace environment to integrate into your SOC workflow and service offering.

Call Home & Connectivity to your SOC Environment

Darktrace appliances have the ability to connect back to Darktrace central management over a secure and encrypted channel known as Call Home. The Call Home connection enables Darktrace to check the status of the appliance, provide updates to the platform, and perform Health checks and other scheduled maintenance.

The Call Home connection remains within the customer's complete control at all times: it is initiated by your appliance, and you can start, terminate or audit it at any time.

Call Home is fundamental to enable the software communication across the platforms into the SOC. Please note that this communication will not work across dedicated or vendor specific VPN links.

Dual Call Home

A dual Call Home configuration is the recommended set up as this will abstract the MSSP from the regular service and maintenance that is provided by Darktrace to all Darktrace customers, while allowing the MSSP to focus on the security monitoring and operations.

Single Call Home

If the customer only allows a single external connection to the MSSP, the MSSP will need to provide for all of services usually managed directly by Darktrace including:

- Health checks
- Software management and updates
- Configurations

Pushing Darktrace Alerts to your SOC Environment

SOC integration goals will typically require working with Darktrace Alerts outside of the Threat Visualizer, such as:

- Allowing SOC analysts to triage alerts within a 'one pane of glass' environment without having to support separate access to individual alerting tools.
- Combining Darktrace alerts with the output of other tools for the implementation of multi-vendor use cases.
- Correlating Darktrace alerts with additional log sources for detection and efficacy gains.
- Tracking and recording metrics around alerts and investigations.
- Providing context around Darktrace alerts from other data sources, e.g. inventory databases.

Two standard methods of achieving these goals are to implement an API query solution following the steps below or integrating Darktrace's alerting functionality to output a feed to one or more central logging servers in use within the environment.

Alerting Formats and Outputted Information

Multiple alerting formats are available for exporting to a SOC environment. When selecting a format to use, the main consideration will be that of ease of integration vs. richness of information:

- JSON based alerting formats provide the richest information. Typically, consuming JSON-based alerts within a SOC environment will require writing a custom alert parser, although Darktrace does support a JSON Syslog-based alert output.
- Standard Syslog-based alerting formats (CEF, LEEF) will provide sparser information, constrained by the limitations of these standards. However, SOC solutions will typically support these alert formats 'out of the box' without having to write a custom parser.

Typically, a Darktrace Alert will contain the following information:

- The name of the Model that breached
- A unique ID for the Model that breached
- The breach score
- The device that breached the model
- (Optionally) destination endpoint information
- Additional breach information
- Darktrace breach URL

Each model can be modified to issue alerts only in specific contexts. Darktrace provides the following options for defining which breaches to alert on:

- Alert selectively across all Models based on the score of the individual breach
- Alert selectively based on the priority of the Model that breached
- Alert selectively based on the name of the Model that breached
- Alert only on breaches of Models which have the 'Alert' property set

These features can be used to determine which breaches will be sent to the SOC solution, providing precision and prioritization at the alerting level. Typically, this level of integration is performed after experience, mapping or threat-modelling exercises have defined use cases utilizing the Darktrace Models, or identified particular models of interest.

API Query Solutions and Outputted Information

An alternative solution for SOC integration is an API implementation, where the Darktrace Master Appliance is polled regularly on the `/modelbreaches` endpoint to provide comprehensive alert information. Breach data returned in this manner provides contextual information, giving the status of the model at the point of breach and the current state. The data returned can also be finely tuned by a number of query parameters.

This method is the most comprehensive, provides the greatest control, and can be implemented by external SOC teams to monitor and triage breaches without locking up alerting streams for the Appliance operator. Appendix A: Querying the Darktrace API for SOC Integration covers the process of API authentication and suggests a possible architecture for integration.

Typically, a Darktrace model breach event returned from the API endpoint will contain the following information:

- Number of comments on the breach
- Time of the breach
- The status of the Model at the time of breach
    - Unique identifiers, additional breach information and status indicators at that time
- The status of the model now
    - Unique identifiers, additional breach information and status indicators now
- The model components that triggered
- Information regarding the device that breached
- Breach score

Data returned from the Appliance `/modelbreaches` endpoint is a JSON formatted array of breaches. For fuller details of this JSON format, please see **Appendix B: API Model Breaches JSON Schema**.

Alerts in your SOC

While it is possible to alert on all Model Breaches, it is generally advisable to selectively investigate a subset of them. Broadly two options exist for determining which Model Breaches to alert and take action upon:

- Pushing all alerts to a SOC solution and performing filtering and alerting decisions within the SOC platform based on the information provided.
- Pushing relevant alerts to a SOC solution and performing filtering and alerting decisions within Darktrace, or within a staging server between Darktrace and the SOC solution.

Making alerting determinations at either location has advantages and disadvantages. While sending all alerts to the SOC can allow for greater flexibility in leveraging Darktrace detections, it can generate a larger overhead in terms of configuration, performance impact, and/or clash with existing analyst investigation workflows where automatic ticketing is in place for SOC events.

Dynamic Threat Dashboard

The most flexible method of approaching filtering is offered by the Dynamic Threat Dashboard in the Darktrace Threat Visualizer. Here, the threat slider, commenting system and point of view pivot between Model and Device allow for dynamic filtering and adjustment. The Dynamic Threat Dashboard is designed for a SOC environment and provides one click investigation and pinpoint analysis for efficiency and ease when pursuing network threats.

Filtering Alerts

Typically, a full integration of Darktrace alerts into a SOC environment will involve filtering both within Darktrace and the SOC. Filtering capabilities provided within the SOC will vary, but alerting decisions can typically be made based on:

- Darktrace Model Breach score
- Darktrace Model
- Devices involved in the breach
- Time-based frequency of alerts correlated on a common factor

With an API Query solution, triaging can be performed at the interim layer between the SOC and the Appliance itself. It is recommended that all alerts are pulled from the Appliance initially, and then any prioritization is performed on the returned data, rather than excluding any breaches at the query stage. Blanket exclusions of this kind, such as only returning breaches with a score above a specific minimum, risk omitting breaches which may be a particular concern to a specific customer but have been observed before (resulting in a lower score).

Instead, a workflow for triaging returned breaches which is responsive to the business processes and concerns of each customer should be developed. The following recommendations cover classification and prioritization.

Tags

The simplest way to approach filtering the returned breaches is to examine the tags applied to each breach. The tag “Enhanced Monitoring” is applied by Darktrace to any alert which merits further investigation. When looking for breaches to prioritize, this is a good place for a SOC team to start.

Minimum Score

Filtering by minimum score returns only the most anomalous breaches, which can be useful when seeking out exceptional, malicious incidents. Model breaches with a very high score tend to require further investigation.

This kind of filtering should be performed only with awareness that it may exclude lower scoring breaches; a particular customer may be very concerned by a model with a score modulation which tends towards a lower score over time, so would be excluded by this filtering quickly. Therefore, when filtering in this way, take individual business concerns into account.

Example SOC Integration Workflow

The following workflow tracks the integration of Darktrace into a SOC environment, whether existing or newly created, from initial implementation to a filtered, tailored approach.

A way to quickly leverage Darktrace's detections within a SOC environment can involve initially sending a subset of the more 'serious' breaches directly, without having to make decisions on a Model-by-Model basis, with the aim of revising this configuration and narrowing the scope as implementation progresses.

Optionally, alert filtering can be performed exclusively at the SOC layer, once a proper system of prioritization and triaging has been established with granularity and business-specific prioritization. Having all Model Breaches within the SOC provides greater flexibility for rule creation and correlation benefits.

Stage 1: Initial alerting configuration

- All Darktrace Model Breaches with score greater than 50%, sent to SOC for investigation.

- All alerts sent to SOC to be investigated by analysts. Darktrace breach score threshold to be reviewed based on number of alerts being outputted and analyst workload.

As an initial configuration, this allows for quick integration and immediate utilization of Darktrace alerts. This configuration will typically generate unnecessary analyst workload in terms of triggering investigation of breaches that may not need immediate attention and reduce detection efficacy by 'missing' alerts scored under 40% which should have been investigated.

Stage 2: Improved alerting configuration.

- Darktrace Model Breaches with score greater than 50% and those with the tag 'Enhanced Monitoring', sent to SOC.

- Analysts prioritize breach events with the specified tags. Score threshold to be reviewed based on number of alerts being outputted and analyst workload.

The Darktrace tag - 'Enhanced Monitoring' – allows SOC teams to prioritize particularly meaningful or malicious breaches. It is only applied to Models which warrant extra investigation or follow-up and can be an excellent place to start when moving towards prioritization.

Reviewing the Model Breaches being sent to the SOC can identify Models which are not a priority for the business needs of the particular client. These models may be tagged with a user-created tag, 'lower priority', in order to filter out breach events.

As alerting becomes increasingly targeted to specific models and use case detection goals, the Model Breach score and priority thresholds can be reduced to improve coverage and avoid 'missing' alerts for relevant models.

Stage 3: Model specific configuration

- All Darktrace Alerts for models tagged with 'Enhanced Monitoring' sent to SOC.

- SOC configured to trigger alert by default for all Darktrace breaches with score greater than 40%, with Model specific thresholds set on a Model-by-Model basis, based on observed breach rates and Use Case detection.

- Breaches with scores below triggering threshold used in global correlation rules with other alert sources. Darktrace models being sent to the SOC continually reviewed based on on-going Use Case development.

Models with the 'Enhanced Monitoring' tag are always pushed to the SOC for investigation as standard.

After identifying relevant Darktrace models and performing Use Case mapping, all breaches for these models can now be sent to the SOC.

The filtering and alerting capabilities of the SOC can be used to provide more granular alerting decisions on a Model-by-Model basis, as well as to begin improving additional global correlation rules for all Darktrace breaches ingested.

Workflow Recommendations

Defining research workflows for based around Darktrace Model Breaches requires an understanding of the ways in which Darktrace Models can differ from more traditional alerts. A SOC playbook may often describe and mandate a research and investigation workflow that defines validation steps that an analyst should go through when investigating an incident. It is a common goal to increase SOC efficiency to consolidate as much of that investigation workflow as possible within the SOC environment, but this consolidation is limited by the nature of the alerts and functionality of the tools available.

The Enterprise Immune System and Darktrace Threat Visualizer are not limited in the same way; eventes contain a wealth of information, context and content which allow for detailed investigation within a single tool. Darktrace leverages its unique pattern-of-life awareness to identify and contextualize anomalous activity, meaning each alert has greater depth and requires a different approach to traditional intrusion tools.

Two important differences between Model Breaches and traditional alerts, such as from a rule-based Intrusion Detection System, are:

- Darktrace Model beaches are not necessarily, or even primarily, associated with a specific flow or connection. Model breaches can be the result of behavior that extends over long periods of time, and involves many devices. As such it is not always possible to relate a Model Breach to a specific connection, rather, a pattern of observed connections or behavior.

- Darktrace Model breaches are primarily anomaly based. As such, they are inherently related to the behavior of other devices, the past behavior of the device in question, and similar activities that occur on the network. They are highly contextual and leverage rich historical information learned from the network. This means that validating, or investigating, Model Breaches typically requires a substantially different workflow from that used for rule-based alerts.

Both of these factors mean that Darktrace alerts can often not be effectively triaged and investigated within a SOC environment, both because the contextual information that would be used to validate the alert is not available within the SOC, and because SOC interfaces are not typically designed to present anomaly-based information. The unique nature of Darktrace's alerts has driven the development of the Threat Visualizer such that Model breaches are highly coupled to their manner of presentation within the Threat Visualizer.

As a result, an effective investigation workflow will typically require SOC analysts to transition to the Threat Visualizer, fairly early during research. To facilitate this, all Darktrace alerts include a breach URL which will load the Threat Visualizer in a state which presents relevant information for investigating the breach in question.

Example SOC Workflow

Triage the Alert Data

Filter and triage the returned alert data from the appliance. It is important to establish a combination of criteria for this filtering that allows your SOC to prioritize the key breaches; some recommendations for this filtering are made above.

Perform preliminary investigation on the returned breach data

Many models provide enough contextual information in the breach JSON to make an initial decision on whether to escalate investigating the breach on the appliance. Take for example, a Model Breach for a Downloaded Executable File; the information returned enables the operator to verify whether the .exe is benign or not using third party tools, before needing to access the appliance.

Appliance Investigation

Please refer to the investigation workflow below.

Deciding Whether to Notify

When performing external SOC services for a client, deciding whether a breach event requires immediate notification and action or can be reserved for regular reporting is a delicate task. Malicious behavior does not respect working hours and time zones; it is likely you will be or waking a client up, or at least notifying them out of hours. Policy will depend on your relationship with a customer and understanding of their particular business concerns.

Issuing a Notification

When notifying a client for whom you provide external SOC services, we recommend a concise but comprehensive notification which covers all the necessary information for the Customer to take action, but can be produced quickly and in a standardized format. Dependent on your relationship with the client, this may also include recommendations on remediation.

Investigating the Breach on an Appliance

When investigating an alert, a typical workflow will involve starting with summary information For integration solutions which rely on Darktrace Alerts, a breach URL will be included in the returned data to quickly pivot to the Threat Visualizer. For API Query implementations, take the pbid value from the alert you wish to investigate further.

This specific breach can then be accessed using the following syntax:

```
https://<appliance-ip>/#modelbreach/<pbid>
```

Using the example appliance above, this would take the form:

```
https:// 198.51.100.1/#modelbreach/12345
```

From this breach:

1. Identify what type of device this is and how it normally behaves.

2. Look at the events which contributed to the breach in the model breach event log.

3. Look at whether the device has related anomalies at (or prior to) the breach time using the device event log.

4. Plot this activity against related activities to spot any other anomalies / display how important Darktrace considers that particular anomaly (overlay additional metrics including importance metrics in the graph).

5. Consider replaying the events to better understand the context using the main Threat Visualizer user interface.

6. Check whether similar devices are behaving in a similar way (via similar devices in device summary).

7. Investigate the connection or behavior events using Advanced Search.

8. Consider using a third-party resource for context regarding a suspicious domain, IP address or file.

Strategies for Adopting Darktrace Within a SOC Environment

Strategy 1: Traditional SIEM-oriented, multi-vendor, continuous monitoring environment

This strategy is suited to integrating Darktrace into an existing, traditional SOC model characterized by:

- Multiple signature-based systems feeding a SIEM solution to provide a 'one pane of glass' environment.
- Continuous monitoring function based around well- defined use cases and playbook.
- Alert triage process utilizing multi-tier analyst workflow and ticketing-based tracking systems.

This is a well-established model allows for quick investigation and response to alert triggered security events. Within such an existing SOC environment, integration goals might involve:

- Moving away from reliance upon signature-based alerting and log analysis to incorporate the use of advanced machine learning for threat detection.
- Increasing coverage of monitored threats and mitigated risks.
- Improving analyst efficiency through the use of the Threat Visualizer for incident investigation.

To achieve these goals, a deployment strategy might consist of:

- Mapping Darktrace behavioral detection and compliance capabilities (models) to the existing threat taxonomy / playbook in use within the SOC.
- Incorporating a range of models within the existing continuous monitoring playbook or creating new plays.
- Developing and training analysts on a Darktrace research workflow using the Threat Visualizer.
- Tracking metrics of the number of Darktrace alerts (model breaches) being investigated, time to investigation, and outcomes to revise and refine the Darktrace plays.

Strengths
Detection improvements utilizing Darktrace's advanced machine learning alongside existing traditional signature-based systems
Consistency with existing SOC processes
Increased efficiency by combining Darktrace capabilities with existing tools
Improved investigation efficiency through the use of the Threat Visualizer

In this operational model, most of the alert processing workflow will exist outside of Darktrace.

However, the required workflow for investigating Darktrace model breaches is sufficiently unique that it will typically still be required to develop an analyst research workflow using the Threat Visualizer rather than attempting to perform all triage and research steps within the SIEM environment.

Strategy 2: Dedicated, single vendor, threat hunting environment

This strategy is suited to the creation of a dedicated threat hunting function designed around Darktrace.

A dedicated threat hunting function would typically be characterized by:

- Implementation of regular, time bound, 'hunts' using available data sources to identify in-progress or potential threats.
- An open-ended, exploratory workflow without predefined plays or a rigid threat taxonomy.
- The primary goal of threat hunting will typically be to identify 'unknown unknowns' that would otherwise be missed through the use of a traditional playbook system and SIEM-oriented environment.

A strategy for developing a threat hunting function might consist of:

- Creating a specialized threat hunting team of analyst(s) trained on using the Threat Visualizer to identify and investigate potential threats.
- Identifying ways to increase hunting efficiency and align models with business risks.
- Allocating analyst hours for regular 'hunts' using the Threat Visualizer and implementing an escalation and remediation workflow for identified threats.
- Tracking metrics around number and severity of threats identified to review the number of allocated hours on an on-going basis.

Strengths
Allows for detection of 'unknown unknowns' which would otherwise by missed through a playbook system
Increased analyst efficiency by utilizing the Threat Visualizer
Large increase in analyst understanding of network architecture and business processes through exposure to the Threat Visualizer during regular hunting
Exploratory nature of threat hunting and proactive research workflow encourages collaboration with different security teams and business functions

In this operational model, the Darktrace workflow takes the role of the continuous monitoring either by the implementation of the Dynamic Threat Dashboard as the central investigative hub, or through the creation of an API Query solution as described below.

This method allows the SOC to identify anomalous, previously undetectable threats whether or not they fall within the scope of the SOC's predefined threat playbook.

Strategy 3: Mixed traditional & hunting environment

This strategy is suited to utilizing Darktrace within a highly mature, integrated SOC environment with both continuous monitoring and threat hunting functions. Such an environment might include:

- A continuous monitoring function based around a SIEM solution and a threat playbook.
- Supplemental threat hunting designed to extend coverage and detection beyond the playbook in an 80- 20 coverage model (80% of threats detected within the playbook, with 20% from threat hunting)
- Feedback from threat hunting function to continually refine and expand the continuous monitoring playbook in an adaptive 'sec-ops' style environment.

This will typically represent an aspirational model for large organizations at the high end of the maturity model and aligns with industry recommendations for developing an intelligence-driven, adaptive security architecture.

Within this environment, a deployment strategy will typically aim to leverage Darktrace within both the continuous monitoring and the threat hunting functions of the SOC. Such a strategy might consist of:

- Incorporating a subset of Darktrace models within the existing continuous monitoring playbook and SOC solution.
- Training the threat hunting team on the use of the Threat Visualizer and incorporating its use into the regular threat hunting workflow.
- Correlating Darktrace model breaches with additional intel/ alert sources to increase hunting efficiency.
- Reviewing threats detected by Darktrace during threat hunting to identify opportunities for the creation of new continuous monitoring plays based on custom or refined Darktrace models.

Strengths
Detection improvements utilizing Darktrace's advanced machine learning alongside existing traditional signature- based systems
Allows for detection of 'unknown unknowns' which would otherwise by missed with a pure playbook system
Large increase in analyst understanding of network architecture and business processes through exposure to the Threat Visualizer during regular hunting
Encourages continuous cycle of improvement in monitoring efficacy through the development of custom models

Within such a combined environment, the model creation and customization capabilities can be leveraged as the primary mechanism of feedback and refinement between the threat hunting and operational functions of the SOC. This allows for the ongoing expansion of continuous monitoring coverage.

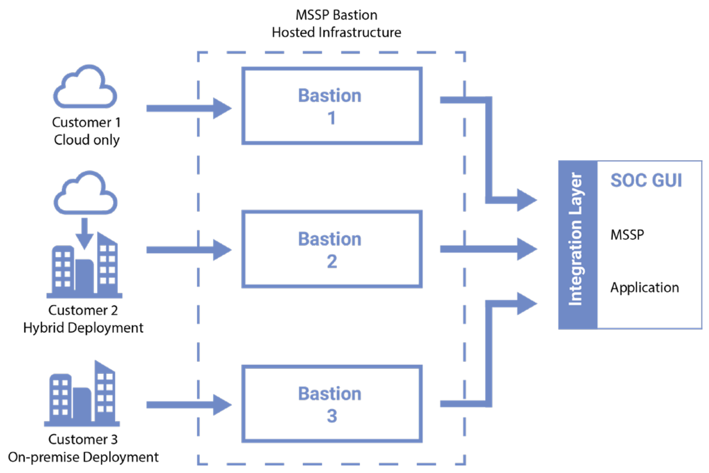
Appendix A: Querying the Darktrace API for SOC Integration

The most flexible method of SOC integration is an API Query implementation, where the Darktrace Master Appliance is polled regularly on the `/modelbreaches` endpoint to provide comprehensive alert information. This kind of implementation returns far more detailed breach data and offers granular control through the range of defined query parameters.

An API query implementation is also scalable, the central service polling a Master appliance can expand to poll a number of Masters if supplied the authentication details. An API implementation can therefore be established by external SOC teams to monitor and triage breaches without locking up alerting streams for the Appliance operator, and curate breaches for Darktrace deployments with a number of Masters.

Simplified Architecture for SOC Integration

Implementing the API Query method for SOC integration requires a structure for polling the Appliance and storing the returned data for filtering. This architecture can be very simple; using segregated hosts for the returned data can simultaneously handle a large number of separate Master appliances for situations where an external SOC team monitors several clients through 'one pane of glass'.



The structure of this architecture has two main elements:

- A central service which polls the Appliance, authenticating using the steps below at a frequent interval and pulling the returned data, and a location for storing this returned data. It is recommended that this location is separate to the SOC environment and stores all unfiltered breach events. The configuration of this location may be more complex depending on your network architecture or Darktrace Appliance distribution.

- A SOC Environment pulling select breaches from the storage location based on a set of use-case filters and prioritization for further analysis by the SOC team.

Acquiring the API Token Pair

Before any data can be queried, an API token pair is needed for each Master Appliance. Creating the API token requires access to the Darktrace Threat Visualizer interface and a User Account with appropriate permissions to access and modify the System Config page.

1. Navigate to the System Config Page on the Threat Visualizer of the Appliance you wish to poll for data.

2. Scroll to 'API Token' and then 'New'.

3. Two values will be displayed, a Public and Private token, the Private token will not be displayed again.

Both Tokens are required to generate the DT-API Signature value, which must be passed with every API call made to the Appliance, so make sure you record them securely.

Building an API request

When querying Model Breach data, the request must be a GET request composed of the following parameters:

- Format: JSON
- The endpoint for each appliance: https://<appliance-ip>/modelbreaches

Possible Parameters:

`GET parameters`	Type	Description
`starttime`	Number	Data start as UNIX timestamp (milliseconds) Example: 151480800000
`endtime`	Number	Data start as UNIX timestamp (milliseconds) Example: 151480806000
`from`	String	Data start as string (YYYY-MM-DDTHH:ii:ss) Example: 2018-01-01T00:00:00
`to`	String	Data end as string (YYYY-MM-DDTHH:ii:ss) Example: 2018-01-01T00:00:00
`minimal`	Boolean	Returns data with more model/component details if false Example: true
`includeacknowledged`	Boolean	Overrides default exclusion of acknowledged breaches. Example: true
`includesuppressed`	Boolean	Overrides default exclusion of suppressed breaches Example: true
`fulldevicedetails`	Boolean	Include full device objects (default false) Example: true
`uuid`	String	Filter by unique ID generated on Model creation Example: 720fb7ae-e00d-4ea4-8bb5-568f236f753c
`pid`	String	Filter by ID of Model breached Example: 869
`pbid`	String	Filter by breach PBID (returns a specific breach) Example: 2632968
`did`	Number	Filter by Device ID of the device causing the breach. Example: 20376
`deviceattop`	Boolean	By default, the device object is included in each component. Setting this to true overrides this, displaying the device object once per breach JSON. Example: true
`expandenums`	Boolean	Replaces enumerated data types with descriptive strings. Example: true

The following recommendations represent a good starting point when initially approaching the query parameters. These parameters may change over time or between different Appliances in response to the business logic and concerns of each customer.

`starttime & endtime` or `from & to`

The goal of a SOC is a quick and reliable response, and organizations may produce a large volume of alerts over a short space of time. It is recommended, therefore, that queries are made at more frequent intervals and cover a shorter duration of time.

`includeacknowledged=true`

An Appliance operator may acknowledge a model breach unintentionally, or without recognizing the implications of the breach itself. It is best to take all breaches, including acknowledged, and make the decision internally on whether the breach needs to be investigated or discussed further.

`includesuppressed=true`

A model may have been configured to fire silently due to overbreaching or by an Appliance operator. That does not necessarily mean it should be excluded altogether, as it may sometimes be indicative of a wider issue worth investigating.

`minimal=false`

Gives full model component and device information in the returned data, allowing for more investigation to be carried within the SOC environment before accessing the Appliance is required.

Making an API call

The following GET request is used to pull breaches from a single appliance, ensuring suppressed and acknowledged breaches are included.

Ensure that minimal=false is passed for the output to include full model/component details.

```
https://<appliance-ip>/modelbreaches?includeacknowledged=true&includesuppressed=true&s
tarttime=${START_TIMESTAMP}&endtime=${END_TIMESTAMP}&minimal=false
```

Where,

- o `$START_TIMESTAMP`= UNIX timestamp in milliseconds (verify 13 digits)
- o `$END_TIMESTAMP`= UNIX timestamp in milliseconds (verify 13 digits)

Once the API query has been formed, it is used to generate the authentication value `<signature>.`

API Authentication

Every API query requires three header values for authentication:

a) DTAPI-Token: <api-token>

`<api-token>` is the public token obtained when creating the API token pair.

b) DTAPI-Date: <date>

`<date>` is the current date and time, which must be within 30 minutes of the Darktrace system time. Any of the following formats are acceptable.

- o YYYYMMDDTHHIISS, i.e. 20180101T120000
- o YYYY-MM-DDTHH:ii:ss, i.e. 2018-01-01T12:00:00
- o YYYY-MM-DD HH:ii:ss, i.e. 2018-01-01 12:00:00
- o Mon, 01 Jan 2018 12:00:00
- o Mon, 01 Jan 2018 12:00:00 [GMT/UTC]
- o Mon Jan 1 12:00:00 2018

c) DTAPI-Signature: <signature>

`<signature>` is determined by computing the HMAC-SHA1 construct of a specific string. This string is composed of the API query string created above, your private API token, the Appliance public API token and the current date in any of the formats above, each separated by a newline character.

Example API Call

Take the following worked example:

- o API Query string:

```
/modelbreaches?includeacknowledged=true&includesuppressed=true&starttime=starttime=1
514808000000&endtime=1514808060000
```

Alternatively, for POST requests, add each post parameter into the query string as follows:

```
POST /postendpoint?param1=value
```

where param1 is a POST field

- o DTAPI-Token:

```
2485df5aaa4278ed80a133e91ae58c5ccccb7862
```

- o Private API token:

```
417f88cedddd679e76cdf4eb0a3ef3d99a99f203
```

- o DTAPI-Date:

2018-01-01 12:00:00

With these values, the <signature> value is calculated using an implementation of the following method.

- o Note that only the /modelbreaches endpoint is used here.
- o Note the '\n' newline characters between the request, API token and timestamp in the 2nd parameter passed to the function:

```
hmac-sha1("417f88cedddd679e76cdf4eb0a3ef3d99a99f203","/modelbreaches?includeacknowledg
ed=true&includesuppressed=true&starttime=1514808000000&endtime=1514808060000\n2485df5a
aa4278ed80a133e91ae58c5ccccb7862\n2018-01-01 12:00:00");
```

pseudocode example

The above function outputs "`e25f42e5b8e77889555766ce22f20de8a63500b1`", which is the `<signature>` value.

Below are two examples of the <signature> generation in Python3 and Bash using the sample parameters we have used thus far. More code examples in other languages, along with full authentication and connection scripts where available, may be requested from Darktrace support.

Python3

```
import hmac

import hashlib

sig = hmac.new('417f88cedddd679e76cdf4eb0a3ef3d99a99f203'.encode('ASCII'),('/modelbreac
hes?includeacknowledged=true&includesuppressed=true&starttime=1514808000000&endti
me=1514808060000' +'\n'+ '2485df5aaa4278ed80a133e91ae58c5ccccb7862' +'\n'+ '2018-01-01
12:00:00').encode('ASCII'), hashlib.sha1).hexdigest()

print(sig)
```

Bash

```
# Be sure to use `printf %s` to prevent a trailing \n from being added to the data.

# Be sure to use a multi-line, double quoted string that doesn't end in \n as

# input for the SHA-1 HMAC.

authSig=$(printf '%s' "/modelbreaches?includeacknowledged=true&includesuppressed=true&s
tarttime=1514808000000&endtime=1514808060000

2485df5aaa4278ed80a133e91ae58c5ccccb7862

2018-01-01 12:00:00" )

hmac="$(echo -n "$authSig" | openssl dgst -sha1 -hex -hmac "417f88cedddd679e76cdf4eb0a3
ef3d99a99f203" -binary | xxd -p )"

echo $hmac
```

Making the API Query

Once the `<signature>` value is generated, we have the three headers values needed for authentication:

```
Headers: {

“DTAPI-Token: 2485df5aaa4278ed80a133e91ae58c5ccccb7862”,

“DTAPI-Date: 2018-01-01 12:00:00”,

“DTAPI-Signature: e25f42e5b8e77889555766ce22f20de8a63500b1”

}
```

Pseudocode example

The API call can now be made in the following format:

```
GET  https://<appliance-ip>/modelbreaches?includeacknowledged=true&includesuppressed=tr
ue&starttime=${START_TIMESTAMP}&endtime=${END_TIMESTAMP}&minimal=false -H "DTAPI-Token:
<public-token>" -H "DTAPI-Date: <date>" -H "DTAPI-Signature:<signature>"
```

Pseudocode example

Appendix B: API Model Breaches JSON Schema

Element	Type	Description
acknowledged	boolean	Whether the breach has been acknowledged by an operator in the Darktrace Threat Visualizer. Example: false
creationTime	number	The timestamp at which record of the breach was created. This is distinct from the "time" field. Example: 1532342039000
suppressed	boolean	Whether the model has been configured in the Threat Visualizer to fire silently and not produce a breach event in the Threat Tray. Example: true
commentCount	number	The number of comments made against this breach. Example: 0
pbid	number	The ID of the model breach. Example: 2632968
time	number	The timestamp of the last record that triggered the full breach in milliseconds since epoch. Example: 1532341962000
{model}	object	
{model}/{then}	object	The model object as observed at the time of breach.
{model}/{then}/name	string	Name of the model that was breached. Example: "Model A"
{model}/{then}/pid	number	The ID of the model that was breached. Example: 869
{model}/{then}/phid	number	The model history ID. Example: 25399
{model}/{then}/{logic}	object	A data structure that describes the conditions to bring about a breach.
{model}/{then}/{logic}/[data]	array<number>	If the model is a checklist type this will be a list of component ID numbers. If this model is a weighted type this will be a list of component ID, weight object pairs. Example: 44702
{model}/{then}/{logic }/type	string	The type of model. Example: "componentList"
{model}/{then}/{logic }/version	number	A number representing the version of model logic. Example: 1
{model}/{then}/[tags]	array<string>	A list of tags that have been applied to this model in the Threat Visualizer model editor. Example: “Active Threat”, “Admin”
{model}/{then}/throttle	number	For an individual device, this id value in seconds for which this model will not fire again. Example: 3600

{model}/{then}/sharedEndpoints	boolean	For models that contain multiple components that reference an endpoint, this value indicates whether all endpoints should be identical for the model to fire. Example: false
{model}/{then}/action	string	The action performed as a result of matching this model firing.
{model}/{then}/interval	number	Where a model contains multiple components, this interval represents the time window in seconds in which all the components should fire for this model to be breached. Example: 0
{model}/{then}/sequenced	boolean	Whether the components are required to fire in the specified order for the model breach to occur. Example: true
{model}/{then}/active	boolean	Whether the model is enabled or disabled. Example: true
{model}/{then}/modified	string	The time in UTC at which the model was last modified. Example:"2018-07-23 10:33:49"
{model}/{then}/{activeTimes}	object	An object describing device whitelisting or blacklisting configured for this model.
{model}/{then}/{activeTimes}/{devices}	object	
{model}/{then}/{activeTimes}/{tags}	object	
{model}/{then}/{activeTimes}/type	string	Example: "exclusions"
{model}/{then}/{activeTimes}/version	number	Example: 2
{model}/{then}/priority	number	The model's priority affects the strength with which it breaches (0-5 scale). Please see the Threat Visualizer user manual for further details. Example: 0
{model}/{then}/description	string	The optional description of the model. Example: "This will breach often"
{model}/{then}/{created}	object	
{model}/{then}/{created}/by	string	Username that created the model. Example: "darktrace"
{model}/{then}/{edited}	object	
{model}/{then}/{edited}/by	string	Username that last edited the model. Example: "darktrace"
{model}/{then}/uuid	string	A unique ID that is generated on creation of the model. Example: "720fb7ae-e00d-4ea4-8bb5-568f236f753c"
{model}/{then}/version	number	The version of the model. Increments on each edit. Example: 7
{model}/{now}	object	The model object in its current state.
{model}/{now}/name	string	Name of the model that was breached. Example: "Model A"
{model}/{now}/pid	number	The ID of the model that was breached. Example: 869
{model}/{now}/phid	number	The model history ID. Example: 25399

{model}/{now}/{logic}	object	A data structure that describes the conditions to bring about a breach.
{model}/{now}/{logic}/[data]	array<number>	If the model is a checklist type this will be a list of component ID numbers. If this model is a weighted type this will be a list of component ID, weight object pairs. Example: 44702
{model}/{now}/{logic }/type	string	The type of model. Example: "componentList"
{model}/{now}/{logic }/version	number	A number representing the version of model logic. Example: 1
{model}/{now}/[tags]	array<string>	A list of tags that have been applied to this model in the Threat Visualizer model editor. Example: “Active Threat”, “Admin”
{model}/{now}/throttle	number	For an individual device this id value in seconds for which this model will not fire again. Example: 3600
{model}/{now}/sharedEndpoints	boolean	For models that contain multiple components that reference an endpoint, this value indicates whether all endpoints should be identical for the model to fire. Example: false
{model}/{now}/action	string	The action performed as a result of this model firing.
{model}/{now}/interval	number	Where a model contains multiple components, this interval represents the time window in seconds in which all the components should fire for this model to be breached. Example: 0
{model}/{now}/sequenced	boolean	Whether the components are required to fire in the specified order for the model breach to occur. Example: true
{model}/{now}/active	boolean	Whether the model is enabled or disabled. Example: true
{model}/{now}/modified	string	The time in UTC at which the model was last modified. Example:"2018-07-23 10:33:49"
{model}/{now}/{activeTimes}	object	An object describing device whitelisting or blacklisting configured for this model.
{model}/{now}/{activeTimes}/{devices}	object	
{model}/{now}/{activeTimes}/{tags}	object	
{model}/{now}/{activeTimes}/type	string	Example: "exclusions"
{model}/{now}/{activeTimes}/version	number	Example: 2
{model}/{now}/priority	number	The model’s priority affects the strength with which it breaches (0-5 scale). Please see the Threat Visualizer user manual for further details. Example: 0
{model}/{now}/description	string	The optional description of the model. Example: "This will breach often"
{model}/{now}/{created}	object	
{model}/{now}/{created}/by	string	Username that created the model. Example: "darktrace"

{model}/{now}/{edited}	object	
{model}/{now}/{edited}/by	string	Username that last edited the model. Example: "darktrace"
{model}/{now}/uuid	string	A unique ID that is generated on creation of the model. Example: "720fb7ae-e00d-4ea4-8bb5-568f236f753c"
{model}/{now}/version	number	The version of the model. Increments on each edit. Example: 7
[triggeredComponents]	array<object>	Details of the components that were triggered during the breach
[trig...]/time	number	The time the component was triggered in milliseconds since epoch. Example: 1532341961000
[trig...]/cbid	number	The component breach ID. Example: 7823385
[trig...]/cid	number	The component ID. Example: 44702
[trig...]/chid	number	The component history ID. Example: 86639
[trig...]/size	number	The size/strength of the model where applicable, otherwise 1. Example: 1
[trig...]/threshold	number	The threshold reached by the size field. Example: 0
[trig...]/interval	number	The interval in seconds as set on the model. Example: 3600
[trig...]/{logic}	object	An object describing the component logic as configured in the model editor. For further information please see the Threat Visualizer user manual.
[trig...]/{logic}/{data}	object	
[trig...]/{logic}/{data}/left	string	Example: "A"
[trig...]/{logic}/{data}/operator	string	Example: "AND"
[trig...]/{logic}/{data}/{right}	string or object	Depending on depth of model logic, this may be a string containing a single value or an object containing further criteria.
[trig...]/{logic}/{data}/{right}/left	string	Example: "B"
[trig...]/{logic}/{data}/{right}/operator	string	Example: "AND"
[trig...]/{logic}/{data}/{right}/right	string or object	Depending on depth of model logic, this may be a string containing a single value or an object containing further criteria. Example: "C"
[trig...]/{logic}/version	string	Example: "v0.1"
[trig...]/metric	object	An object describing the metric used in this component.
[trig...]/{metric}/mlid	number	Example: 16
[trig...]/{metric}/name	string	Example: "connections"
[trig...]/{metric}/label	string	Example: "Connections"
[trig...]/{device}	object	

[trig...]/{device}/did	number	Device ID of the device causing the breach. Example: 20376
[trig...]/{device}/hostname	string	Hostname of the device if available, otherwise undefined. Example: "btables-laptop"
[trig...]/{device}/ip	string	Current IP of this device. This may be an IPv6 address if the setting "IPv6 in Device Ip fields" is set to true, otherwise it may only be an IPv4 address. If there are no IPv4 addresses associated with a device, and "IPv6 in Device Ip fields" is set false, this field will be undefined. Example: "10.40.7.28"
[trig...]/{device}/macaddress	string	MAC address of this device if known, otherwise undefined. Example: "ab:a1:23:45:c6:78"
[trig...]/{device}/typelabel	string	Device type label. Example: "Laptop"
[trig...]/{device}/typename	string	Device type name. Example: "laptop"
[trig...]/[triggeredFilters]	array<object>	A list of filters that applied to this triggered component. For further information on filters please see the Threat Visualizer user manual. Please note that this list may include multiple filters.
[trig...]/[trig...Fil...]/cfid	number	Example: 16
[trig...]/[trig...Fil...]/id	string	Example: "connections"
[trig...]/[trig...Fil...]/filterType	string	Example: "Connections"
[trig...]/[trig...Fil...]/comparatorType	string	Example: "is longer than"
[trig...]/[trig...Fil...]/{arguments}	object	Example: "User agent"
[trig...]/[trig...Fil...]/{arguments}/value	string	Example: "0"
[trig...]/[trig...Fil...]/{trigger}	object	
[trig...]/[trig...Fil...]/{trigger}/value	string	Example:"trustd unknown version CFNetwork/902.1 Darwin/17.7.0 x86_64"
score	number	The intensity of the model breach, represented by a value between 0 and 1. Example: 0.3