Corelight-update

Updated Jan 10, 2025

CONTENTS

The **primary purpose** of the Corelight-update utility is to automate and simplify the workflow of collecting data from disparate sources of dynamic content for Corelight Sensors by integrating into your existing CI/CD process.

This data includes threat intel, Suricata rulesets, vulnerability data, Zeek packages and other Input Framework data. The data can come from pre-formatted local sources, pre-formatted remote sources, or third-part integrations.

There's no need for additional tools if you integrate Corelight-update with your CI/CD or change control process to manage Suricata rulesets, Intel files, Input files or Zeek package bundles.

In addition to collecting and formatting data sources, Corelight-update can optionally apply Corelight best practices to Suricata rulesets, extracting indicators from atomic Suricata rules and creating Zeek Intel files. The corresponding Suricata rules are then disabled, reducing the workload of the Suricata process.

Corelight-update natively supports the concept of hierarchical processing with a single global configuration and multiple policy configurations.

The output of each policy is a single Intel file, a single Suricata ruleset, a single package bundle, and multiple Input files ready to be consumed by a Corelight Sensor.

A **secondary function** of Corelight-update is to push content to Corelight Sensors. It supports ALL types of sensors, both Fleet-managed and stand-alone.

CHAPTER

ONE

QUICKSTART - NEW INSTALL

The Corelight-update utility can run as a service at a scheduled intervals, if enabled, to check for updates to the configured data sources, and distribute updated content to the Fleet Manager policies and sensors.

1.1 System requirements

The minimum system requirements are:

- An x86_64 or ARM64 processor.
- 4 GB memory.
- A host running a Linux OS.
- Network connectivity to the Internet, or to an internal-facing threat intelligence data repository.
- To push content to your sensors, or to Fleet Manager, network connectivity to the management interface is required.

1.2 Installation overview

Select a host to install the Corelight-update utility. If you have a Corelight Fleet Manager installation, Corelight-update can be run on the same host.

1. *Set up the Corelight stable package repository*
2. *Install Corelight-update*
3. *Add the corelight-update group to existing users (optional)*
4. *Configure Corelight-update*
5. *Add proxy configuration (optional)*
6. *Run Corelight-update*

1.2.1 1. Set up the Corelight stable package repository

Bash script - deb Installation

1. Run the script using:

   (Optional) To download the script before running it:

Bash script - rpm Installation

1. Run the script using:

   (Optional) To download the script before running it:

Manual deb Installation

1. Refresh the package cache:

2. If you are running Debian, install `debian-archive-keyring` so that official Debian repositories are verified. Ubuntu installations can skip this step.

3. Ensure the required tools (curl, gpg, apt-transport-https) are installed before proceeding:

4. To install a deb repo, you need to install the GPG key that is used to sign repository metadata. Use a utility called apt-key.

5. Verify the file named `/etc/apt/sources.list.d/corelight_stable.list` contains the repository configuration below.

In the example below, check that the strings **ubuntu** and **trusty** represent your Linux distribution and version:

Valid options for os and dist parameters can be found in Packagecloud's supported OS list.

6. Update the local APT cache:

Manual rpm Installation

1. Install `pygpgme`, a package that allows yum to handle gpg signatures, and a package called `yum-utils` that contains the tools you need for installing source RPMs.

   You might need to install the EPEL repository for your system to install these packages. If you do not install `pygpgme`, GPG verification will not work.

2. **Create a file named `/etc/yum.repos.d/corelight_stable.repo` that contains the repository configuration below.**
   Replace **el** and **6** in the `baseurl=` path with your Linux distribution and version. Valid options for os and dist parameters can be found in the supported OS list in the docs.

3. Update the local yum cache:

1.2.2 2. Install Corelight-update

Debian

RHEL

1.2.3 3. Add the corelight-update group to existing users (optional)

As part of the installation, a system user and group are added to the OS to manage the Corelight-update service. All files and directories that are created for Corelight-update will belong to the user `corelight-update`, and the group `corelight-update`.

To avoid using `sudo` when running `corelight-update` commands, you can add your user to the `corelight-update` group. For example, use this command to append the `corelight-update` group to the users assigned groups:

Tip: Changes made with the `usermod` command do not take effect in the current session. Logging out and in again will apply the changes.

1.2.4 4. Configure Corelight-update

The Corelight-update utility uses a configuration database to track and maintain the sensor inventory, the global service configuration and interval, the policy assignments, and the integrations.

To configure Corelight-update, start with the sample file as a template, and fill-in the various configuration options. Once you've completed filling in your configuration file, use the file to update the policy defined in the Corelight-update configuration database.

A default policy named **defaultPolicy** is created automatically as part of the installation process. The default policy is functional, but also optional, and can be replaced with custom named policies.

There is a configuration sample file provided with the default policy:

Attention: To use Corelight-update for Fleet-managed sensors, you must associate the Fleet Manager sensor policy or suricata policy name with the Corelight-update policy name. See **4.1. Change the policy name (optional)** below for the options available.

1.2.4.1 4.1 Change the policy name (optional)

For Fleet-managed sensors, Corelight-update will use the Fleet Manager sensor policy name to collect a group of sensors that it should deploy "Intel" and "Input" content to. Suricata rulesets and package bundles are uploaded to Fleet Manager policies directly.

As of **Fleet Manager v27.8**, Fleet Manager uses separate policies for sensors and Suricata rulesets.

Corelight-update provides a couple options to associate the Corelight-update policy to a Fleet Manager policies:

1. Use the same name for the Corelight-update policy, the Fleet Manager sensor policy and Fleet Manager suricata policy.
2. When adding your Fleet Manager details to the Corelight-update policy inventory, provide the Fleet Manager sensor policy name in the Corelight-update `sensor_policy:` field, and the Fleet Manager suricata policy name in the Corelight-update `suricata_policy:` field. See **4.2 Customize a policy (optional)** below.
3. Change the Corelight-update policy name to match the existing Fleet Manager sensor and suricata policy name (assuming they have the same name).

   For example, to remove the default Corelight-update policy, and create a new policy named "myFleetPolicy":

   1. Remove the default policy:

   2. Add a new policy named "myFleetPolicy":

   3. Verify the policy is defined:

   Once the new policy is created, a configuration sample file is created, and placed into a folder with the policy name. For example: `/etc/corelight-update/configs/myFleetPolicy/db-config.yaml`.

Note: If the `sensor_policy` field is left blank or the `suricata_policy` field is left blank, it will use the name of the Corelight-update policy for the empty field.

1.2.4.2 4.2 Customize a policy (optional)

Use a configuration example file to create a customized configuration for Corelight-update that defines the Fleet Manager details, sensor inventory, the Corelight-update service configuration, the content assignments, and integrations.

To view a sample policy configuration yaml that enables and pushes the default ETOpen and Corelight suricata rulesets to a single, unmanaged sensor, see *Default policy sources*.

1. Add your sensors to the Corelight-update policy inventory. For information on configuring the sensor inventory, and the use of encrypted passwords, see *Policy inventory settings*.
2. Configure the content you want to deploy to your sensors in the Corelight-update policy configuration. This content can include threat intel, Zeek input files and package bundles, and Suricata rulesets. See *Policy sources*.
3. Configure a custom third-party integration that provides support for a vendor-specific threat source, including source-based customizations and authentication. See *Third-party integrations settings*.

1.2.4.3 4.3 Customize global settings (optional)

1. Customize the default interval settings for data downloading and processing, enabling global integrations, modifying the web service, or deploying global Suricata configuration files. See *Configuration settings*.
2. Customize the default network communication between Corelight-update and Fleet Manager or sensors, including how much time to wait for a status of a file upload. See *Network settings*.

1.2.5 5. Add proxy configuration (optional)

See *Using a proxy with Corelight-update* for instructions.

1.2.6 6. Run Corelight-update

Run `corelight-update` using the CLI commands, or enable the service. See *Corelight-update Service*.

For additional `corelight-update` command options, see *CLI commands*.

CHAPTER

TWO

QUICKSTART - UPGRADE

The Corelight-update utility can run as a service at a scheduled intervals, if enabled, to check for updates to the configured data sources, and distribute updated content to the Fleet Manager policies and sensors.

Attention: On completion of the upgrade, if you have pre-1.0 release policy files, they must be manually imported into the configuration database. See *CLI commands* for details on the `import` command.

2.1 System requirements

For the latest system requirements, see *System requirements* in the references.

2.2 Upgrade overview

Set up the Corelight package repository on the host OS if required. See *QuickStart - new install* for instructions.

1. *Upgrade Corelight-update*
2. *(Optional) Configure new Corelight-update features.*
3. *(Optional) For customers upgrading from a version prior to 1.0 or haven't done so previously, add the "corelight-update" group to existing users.*
4. *(Optional) For customers upgrading from a version prior to 1.0, manually import your existing configurations.*

2.2.1 Upgrade corelight-update

Debian

RHEL

2.2.2 Configure new Corelight-update features (optional)

When Corelight-update gets upgraded, any existing database will automatically be upgraded.

Use the CLI command `corelight-update show -policy <policy name>` to identify new configuration options or `corelight-update show -policy <policy name> -file /etc/corelight-update/config/<policy name>/db-config.yaml` to replace the existing file with the new format. Then modify as necessary.

Once you've completed filling in your configuration file, use the file to update the policy defined in the Corelight-update configuration database.

When updating policies, you can either supply an entire policy configuration or only the sections you want to update.

Warning: When updating from a full or partial configuration, any config section provided must have all none-zero fields provided. Any missing fields will be automatically configured to their **zero** value.

2.2.3 Import policy files from versions prior to v1.0 (optional)

The Corelight-update utility uses a configuration database to track and maintain the sensor inventory, the global service configuration and interval, the policy assignments, and any integrations.

If you have pre-1.0 release policy files, they must be manually imported into the configuration database.

You can import your pre-1.0 policies using `corelight-update import` with the `-v0.23` flag to indicate you are importing from a version 0.23 policy. After importing a pre-v1.0 policy, use the `update` command to add the inventory details to the policy. For example, `corelight-update update -policy defaultPolicy -file /etc/corelight-update/configs/defaultPolicy/inventory.yaml`

Once the pre-v1.0 policy is imported, review the imported configuration using the `corelight-update show` command. For example: `corelight-update show -policy defaultPolicy -yaml`

Note: The `-v0.23` flag can be used with policies from older versions of corelight-update, but you should always review the imported configuration using the `show` command.

Once a policy has been imported, you will switch to using the new policy configuration to update those policies. The pre-1.0 policy files cannot be used to update a policy, they can only be used as an import.

Attention: When Corelight-update gets installed for the first time, it will automatically create the database, a default Global configuration, and a default policy named "defaultPolicy".

If the `corelight-update.db` is deleted, a new `corelight-update.db` will be created the next time the service runs, with a default Global configuration. However, no default policies are created.

CHAPTER

THREE

GLOBAL CONFIGURATION

Corelight-update uses a database store global-level and policy-level configurations and settings. Global settings include:

3.1 Corelight-update service settings

3.1.1 Web service

The web service provides local web access to the documentation, and all of the content created and managed by Corelight-update. The web service is enabled by default, and is optional.

Note: Updating the default certificate is recommended.

3.1.2 Service interval

In some cases it is useful to disable the processing feeds and only have the web service enabled, or modify the default interval for processing data feeds.

When this interval is triggered,

- All caches are updated.
- All local data sources are copied to their respective working folders.
- All remote data sources are copied to their respective working folders.

Additionally, the individual state history for each enabled integration is checked each interval.

- If the integration interval time has lapsed, it processes the integration.
- If the interval has not lapsed, the integration is skipped until the next cycle.

- If the interval is set to `0`, the integrations will be processed each cycle.

For more details, see *Order of operations*

For details on updating the web service or service interval, see *Updating the Global configuration*

Attention: The web service and service interval only apply when running Corelight-update as a service. Any changes to these settings require a service restart to take affect.

3.2 Configuration settings

3.2.1 General settings

3.2.1.1 Additional logging options

If additional logging detail is needed, enable verbose logging. This setting is in addition to the CLI debugging option.

3.2.1.2 Experimental features

There are currently no experimental features available in Corelight-update.

3.2.1.3 Auto-updating policy settings

You can configure Corelight-update to automatically update a policy using a pre-selected configuration file name, and directory path. When `auto_updating_policies` is enabled, Corelight-update monitors the directory path `/etc/corelight-update/configs/<policy_name>/` for a file as defined in `filename`.

On each service interval, Corelight-update checks each path for a policy configuration file, and applies that configuration to the policy. This setting is enabled by default.

Note: When `auto_updating_policies` is enabled, a configuration file matching the filename setting is **required** in each policy directory: `/etc/corelight-update/configs/<policy_name>/`

3.2.1.4 Pushing content to sensors in parallel

By default, Corelight-update will deploy content updates to the sensors concurrently. Corelight-update will open a connection to multiple sensors in a policy, push updated content, and cycle to the next sensor, up to the `parallel_push_limit` setting.

Content updates are performed in a specified order. To review the order of operations, see *Push content for policies*. The default for `parallel_push_limit` is 10 sensors.

3.2.2 Global-level data sources

Corelight-update supports applying a limited selection of data sources at the Global level.

3.2.2.1 GeoIP database

Enables downloading of the Maxmind GeoIP database. The default interval is 1 week.

For additional details, see *Maxmind GeoIP*.

3.2.2.2 Remote data sources

Remote sources are required to be added to each policy configuration. However, any source that's cached globally, will only be downloaded once. See *Remote source settings* for details.

3.2.2.3 Locally managed data sources

In addition to downloading content from external sources for your sensors, Corelight-update will also accept locally-sourced content and configurations that can be applied at a Global-level, or at a Policy-level.

Corelight-update provides folders for input, threat intel, and suricata data at the Global-level and Policy-level, where you can place pre-formatted content to be processed. The following is a list of folder locations files can be placed for automatic processing.

For example, if an intel file is placed in the `global-intel` folder, the contents are added to the published intel file for all policies. If an intel file is placed in a policy `local-intel` folder, the contents are automatically added to the published intel file only for that policy.

The following functions do not require any additional configuration:

Local Intel folders

- All Zeek compatible formatted files in the `global-intel` folder are added to all policies as an intel file.
- Any Zeek compatible formatted files placed in a `local-intel` folder is added to that policy as an intel file.
- Any intel files in the `global-intel`, `local-intel`, or generated by an enabled integration are automatically merged into a single intel.dat file.

Local Suricata folders

- Any Suricata formatted ".rules" or ".rules.tar.gz" ruleset files placed in the `global-suricata` folder are available to all policies.
- Any Suricata formatted ".rules" or ".rules.tar.gz" ruleset placed in a `local-suricata` folder are available to that policy.
- Any ruleset file in the `global-suricata`, `local-suricata`, or generated by an enabled integration are automatically processed and merged into a single suricata.rules file.

Local Input folders

- Any Zeek compatible formatted files placed in the `global-input` folder are available to all policies.
- Any Zeek compatible formatted files placed in a `local-input` folder are available to that policy.
- Any input files in the `global-input`, `local-input`, or generated by an enabled integration, (with the same name) will automatically get merged into a single input file with that name.

To review the order that the configurations are processed in, see *Order of operations*.

3.2.3 Global-level Suricata settings

If you maintain a centralized set of Suricata configuration files for ruleset tuning and management, you can configure Corelight-update to automatically download your Suricata configuration files from a remote source, and apply them to the Corelight-update connected sensors.

The Suricata configuration files `disable.conf`, `enable.conf` and `modify.conf` can be applied at a global level, and at a policy level. If a `disable.conf`, `enable.conf` or `modify.conf` exist in the Global config directory, they will be processed for each policy automatically.

- To learn about the processing order, see *Order of operations*.
- For information about applying Suricata configuration files at the policy level, see *Suricata policy settings*.

Each time the Corelight-update service runs, the Suricata rulesets can be processed up to three times for each policy:

1. Process any enabled Corelight recommended configs,
2. Process any enabled global-level configs,
3. Process the Suricata policy-level configs.

For example, to pull a `modify.conf` file from GitHub and apply it as part of your Global policy:

The supported authentication types are no auth, `basic`, or `token`. When using the no auth option, leave the `auth_type` field empty.

See *Using a proxy with Corelight-update* for details about using a proxy to download remote sources.

3.2.4 Updating the Global configuration

Changes can be made to the global policy using either:

- A config file.
- The Corelight-update CLI command by using the `--global-settings` switch.

Updating via `--global-settings`

The Corelight-update CLI command supports updating the Global Configuration directly using the `--global-settings` switch.

- Multiple settings can be updated using a single command.
- Update nested settings by using a "." , for example, `webserver.enable=true`.
- Other than `remote_global_conf_files`, any setting can be updated using a key=value pair.

For example:

Note: Making changes to a policy using the CLI bypasses the configuration files. To maintain a copy of the current Global Configuration as a config file, export it to a file. See "Show Options" in the *CLI commands*.

See the Complete Global Settings below for a list of fields that can be updated directly.

Updating with a config file

When using a config file, make additions or changes to a policy in a configuration file before loading the file into Corelight-update.

To update the global configuration:

1. Output the current global configuration as a file. For example, to create a global config file in yaml format:
2. Change the settings in the config file.
3. Update the global configuration. For example:

Warning: When making changes to a policy, the configuration file section being modified must also include any previously defined, non-zero fields. Any fields left undefined will be automatically configured to their **zero** value.

After updating a configuration, we recommended verifying the global configuration on the console. For example:

3.2.5 Complete global settings

3.3 Network settings

Use the global network configuration to modify the connection timeout variables between Corelight-update and the sensors, a Fleet Manager instance, or a data source to be downloaded. The `sensor_timeout_settings` are used to manage communication between Corelight-update and a sensor or Fleet Manager instance. The `download_timeout_settings` are used to manage communication between Corelight-update and a data source to download.

3.3.1 Updating the Global network config

Changes can be made to the global network configuration using the Corelight-update CLI command with the `--network-settings` flag.

Updating via `--network-settings`

The Corelight-update CLI command supports updating the Global Network Configuration directly using the `--network-settings` flag.

- Multiple settings can be updated using a single command.
- Update nested settings by using a ".". For example, `sensor_timeout_settings.tls_handshake_seconds=10`.
- Any setting can be updated using a key=value pair.

For example:

See the next section for a list of fields that can be updated directly.

After updating a configuration, verify the global network configuration using the console. For example:

3.3.2 Complete global network settings

(continues on next page)

(continued from previous page)

CHAPTER

FOUR

POLICY CONFIGURATION

The Corelight-update policies controls what content is collected, processed, and pushed to Fleet Manager policies and or sensors. This includes third-party integrations, Suricata rules management, Intel file management, Input file management, historical file retention and Zeek package management.

4.1 Policy sources

Policy sources represent collections of local and remote pre-formatted data. This includes Suricata rulesets, Intelligence Threat feeds, and other relevant data that can be use with the Input framework.

Corelight-update collects data from these sources, along with data from *third-party integrations*, to be processed according to the respective management settings. For more details, see:

Suricata configuration

Intel management

Input management

A policy data source differs from third-party integration, as policy data sources must be pre-formatted content you can download using an unauthenticated, basic-authenticated, or token-authenticated URL.

The URL for a remote policy source must be accessible via HTTPS or HTTP. No other protocols are supported.

See *Using a proxy with Corelight-update* for details about using a proxy to download remote sources.

4.1.1 Locally managed sources

In addition to downloading content from external sources for your sensors, Corelight-update will also accept locally-sourced content and configurations that can be applied at a Global-level, or at a Policy-level.

Corelight-update provides folders for input, threat intel, and suricata data at the Global-level and Policy-level, where you can place pre-formatted content to be processed. The following is a list of folder locations files can be placed for automatic processing.

For example, if an intel file is placed in the `global-intel` folder, the contents are added to the published intel file for all policies. If an intel file is placed in a policy `local-intel` folder, the contents are automatically added to the published intel file only for that policy.

The following functions do not require any additional configuration:

4.1.1.1 Local Intel folders

- All Zeek compatible formatted files in the `global-intel` folder are added to all policies as an intel file.
- Any Zeek compatible formatted files placed in a `local-intel` folder is added to that policy as an intel file.
- Any intel files in the `global-intel`, `local-intel`, or generated by an enabled integration are automatically merged into a single intel.dat file.

4.1.1.2 Local Suricata folders

- Any Suricata formatted ".rules" or ".rules.tar.gz" ruleset files placed in the `global-suricata` folder are available to all policies.
- Any Suricata formatted ".rules" or ".rules.tar.gz" ruleset placed in a `local-suricata` folder are available to that policy.
- Any ruleset file in the `global-suricata`, `local-suricata`, or generated by an enabled integration are automatically processed and merged into a single suricata.rules file.

4.1.1.3 Local Input folders

- Any Zeek compatible formatted files placed in the `global-input` folder are available to all policies.
- Any Zeek compatible formatted files placed in a `local-input` folder are available to that policy.
- Any input files in the `global-input`, `local-input`, or generated by an enabled integration, (with the same name) will automatically get merged into a single input file with that name.

To review the order that the configurations are processed in, see *Order of operations*.

4.1.2 Remote source settings

The following fields are available for configuring a remote policy source:

- The policy source `source_type` field can be set to either `suricata`, `intel`, or `input`. When using the `intel` or `input` source type, the URL must provide the data in a Zeek compatible format. For `suricata`, the URL must provide the data in the Suricata rule format.
- The `global_cache` is disabled (false) by default for all sources. If `global_cache` is disabled, that source will be download once for each policy that uses it.
- The `auth_type` field can be set to `basic`, `token`, or left empty for `no auth`.
- The `filename` field is optional. If it's not specified, it will use the base of the URL as the filename.

4.1.3 Overview of adding policy sources

1. Determine the access url and authentication required for the policy data source.
2. For basic-authenticated sources, use the CLI command `corelight-update encrypt <password>` to generate an encrypted password to store in the policy configuration.
3. Configure the policy data source settings under the `sources:` section of the Corelight-update `db-config` file.

4.1.4 Processing a policy source

When Corelight-update processes a policy source, it:

1. Checks the global cache for the target filename.
    1. If the file is present, use the file to process the source.
    2. If the file is not present in the global cache:
        1. Check for a policy level cache of the file and generate an `If-Modified-Since` HTTP header.
        2. Attempt to download the file using the `If-Modified-Since` HTTP header.
            - If a new file is downloaded, create or update the policy-level cache.
        3. Use the policy-level cache to process the source.

Caution: Matches are made in the global cache using only the filename, not the full URL.

4.1.5 Default policy sources

The default policy provided for Corelight-update includes the pre-configured Suricata rulesets:

- Corelight Labs Suricata Rules
- ET/Open ruleset:

You will find the following pre-configured policy source settings in the `db-config` example file:

(continues on next page)

(continued from previous page)

For more source ideas, see *Commonly used Suricata rulesets*

4.1.5.1 Threat intelligence source example

The threat intel sources managed with Corelight-update must provide their data in a Zeek compatible format.

The following example includes settings for the ThreatQ and MISP threat intel sources:

Note: For more details on these examples, see *Third-party integrations settings*.

4.1.6 Third-party integrations

A third-party integration is a data source that might require custom formatting or parsing of the data for use with a sensor, or has unique authentication requirements.

For more details, and a complete list of the current integrations and their respective configurations, see *Third-party integrations settings*

4.2 Policy inventory settings

4.2.1 Push content settings

You can use Corelight-update to push content to Corelight Sensors. It supports both Fleet-managed and stand-alone sensors. To push content to sensors, it must be enabled in a policy. Pushing content is disabled by default.

Once pushing content is enabled at the policy level, it can be overridden for non-Fleet-managed at the individual sensor level in the inventory for that policy. See *Inventory settings* below for details.

The policy settings for pushing content are:

Tip: Force Pushing Content

By default, Corelight-update will only push new content to sensors. If you add a sensor to the policy, no content is pushed to it until new content is generated. You can use the CLI to force push existing content to sensors. See *CLI commands* for details.

The policy inventory can include Fleet Manager details and/or a list of the Corelight sensors to deploy content to using Corelight-update. The sensors can be a combination of appliances, such as the hardware, virtual, and software sensors.

Sensors that are Fleet managed do not need to be listed individually in the Corelight-update inventory. Corelight-update will utilize Fleet Manager to deploy content to those sensors. For sensors that aren't Fleet managed, you can push content directly to them by listing their details in the inventory.

If you have version 1.x software sensors, you can use Corelight-update to either push content to the software sensor, or publish threat intel content using Corelight-update's web interface for the software sensor to fetch.

4.2.2 Overview of adding Fleet Manager and sensor details to the inventory

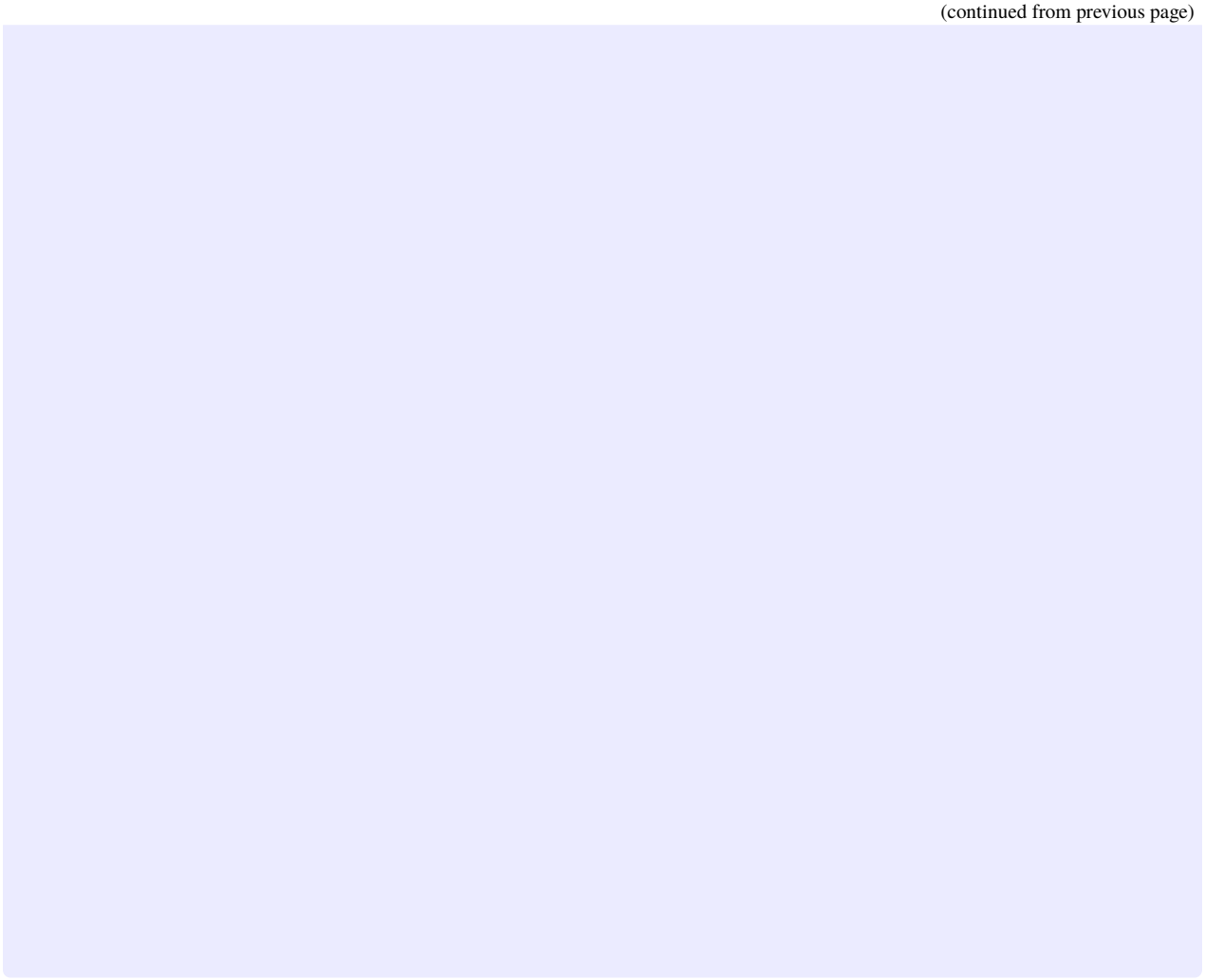
1. Prepare a list of the sensors that Corelight-update will deploy to.
   - For Fleet-managed sensors, the sensor inventory will be collected from Fleet Manager.
   - For all standalone appliance sensors: collect the IP address or FQDN, and the sensor username and password.
   - For all version 1.x software sensors: collect the IP address or FQDN, and the host ssh key, or the sensor username and password.
2. If you have Fleet-managed sensors, configure the connection to your Fleet Manager instance under the `fleet:` section of the Corelight-update `db-config` file.
3. Configure the inventory settings under the `sensors:` portion of the Corelight-update configuration file, adding a new `-name` inventory section and associated fields for each non-Fleet managed sensor type in your inventory.
4. Use the configuration file to update the policy in Corelight-update.

4.2.3 Inventory settings

The following fields are available for configuring the inventory:

(continues on next page)

(continued from previous page)



Warning: The **suricata_config_path** `/etc/corelight/suricata/` does not exist by default on Microsensors and must be created before Suricata config files can be pushed. The folder must be writable for the username listed for that sensor.

Encrypted Passwords

Fleet and individual sensor passwords should be encrypted before they are stored in inventory. Using the `encrypted_pass` field allows you to replace the use of plain text passwords in your Corelight-update configuration file. See **Administering encrypted passwords** later in this topic.

Add Fleet-managed sensors

Corelight-update can use your Fleet Manager instance to collect an inventory of connected sensors, and deploy content to those sensors.

When Corelight-update is deploying content to Fleet-managed sensors, it uses the Fleet Manager API to authenticate and proxy intel files and input files to those sensors through the Fleet Manager instance. If a Fleet-managed sensor is disconnected from Fleet Manager during the content push, that sensor will not receive files until the next content push (assuming it is connected during the push).

Suricata rulesets, Intel files and package bundles are uploaded directly to Fleet Manager and then,

- **For Fleet Manager versions prior to v27.8**:
  - The **sensor policy** in Fleet Manager is updated to use the new content.
- **Starting with Fleet Manager version v27.8**:
  - The **suricata policy** in Fleet Manager is updated to use the new Suricata ruleset
  - The **sensor policy** is updated to use the new package bundle
- **Starting with Fleet Manager version v28.0**: * The **intel policy** in Fleet Manager is available and is updated to use the new Intel ruleset.

Once updated, Fleet Manager will handle pushing the new Suricata ruleset, Intel files and package bundles to the connected sensors. If Fleet Manager details are configured in the Corelight-update policy, new Suricata rulesets, Intel files and package bundles will be uploaded even if no sensors are connected to that policy in Fleet Manager.

To configure Corelight-update to deploy to Fleet-managed sensors, you'll require:

- Network connectivity from the Corelight-update host to the Fleet Manager. No additional network configuration is required other than the default sensor-to-Fleet communications.
- If the sensor policy name used in Fleet Manager and the policy name in Corelight-update do not match, the sensor policy name must be specified.
- The IP address or FQDN of the Fleet Manager.
- The Fleet username and password.

To enable Corelight-update to communicate with the Fleet instance, configure the `fleet:` section of the configuration file.

Corelight-update will collect a list of sensors for each Fleet Manager policy automatically. If you have Fleet managed sensors manually configured in the Corelight-update inventory, they can be removed from the inventory, or remain if set to `fleet: true` in the sensor details. This will cause Corelight-update to skip the sensor while it processes the rest of the policy inventory.

Add standalone appliance sensors

When Corelight-update is deploying content to appliance sensors, such as the hardware and virtual sensors that are not Fleet-managed, it uses the sensor API to authenticate and deploy content to those sensors.

To configure a standalone appliance sensor in Corelight-update, you'll require:

- Network connectivity from the Corelight-update host to the sensor.
- The IP address or FQDN of the sensor.
- The sensor username and password.

The sensor inventory requires one entry for each sensor. You can remove any setting that's not required for a specific sensor's configuration.

Fleet Managed Sensors

If a stanalone appliance sensor is later connected to Fleet Manager, it can be removed from the Corelight-update inventory, or remain if set to `fleet: true` in the sensor details. This will cause Corelight-update to skip the sensor while it processes the rest of the policy inventory.

Add software sensors

When Corelight-update is deploying content to software sensors, it uses SCP to push updates to a specific folder path on the sensor.

To configure a software sensor in Corelight-update, you'll require:

- Network connectivity from the Corelight-update host to the sensor.
- The IP address or FQDN of the sensor.
- The sensor username, and the password or host ssh key.
- The sensor user needs read/write access to the content folders.

Note: The command used to reload the Suricata rules requires sudo access. If you're deploying Suricata rulesets to a software sensor, the host username will also require passwordless sudo access to apply new rulesets.

The sensor inventory requires one entry for each sensor. You can remove any setting that's not required for a specific sensor's configuration.

Using Corelight-update to update a sensor on the same host

If Corelight-update is installed on the same host as a software sensor, no connectivity information is required. The only requirement is to include the path on the sensor to place files. Any package bundles will not be moved, they will just get installed.

4.2.3.1 Administering encrypted passwords

Fleet and individual sensor passwords should be encrypted before they are stored in inventory. Using the `encrypted_pass` field allows you to replace the use of plain text passwords in your Corelight-update configuration file.

To use encrypted passwords:

1. Use the Corelight-update CLI command with the in `encrypt` switch to encrypt the password string. When using special characters in your password string, wrap it in quotes. See *CLI commands* for more details.
2. Copy the encrypted password output from the console, and use it to update the `encrypted_pass:` field of the sensor inventory record, or Fleet configuration in the policy configuration file.
3. Verify the `password:` field of the sensor inventory record, or Fleet configuration is empty.
4. Save the changes, and update the Corelight-update policy.

Note: A Fleet Manager configuration or sensor inventory record should not have both the `password` and `encrypted_pass` fields populated. Make sure to leave the `password` field blank when using the `encrypted_pass` field. If both fields are populated, the `password` field will be used.

Using the Corelight-update CLI command with the in `encrypt` switch encrypts the password string using AES256 encryption. The encryption master key is randomly generated, and stored in the file `/var/corelight-update/.corelight-update`.

If the master key is removed and regenerated, all encrypted passwords will also have to be regenerated. A password must be encrypted with the current key to be decrypted successfully.

To generate a new master key, delete the existing key, and a new one will automatically be created when needed.

4.3 Suricata configuration

In addition to downloading Suricata rulesets from multiple sources, Corelight-update can manage the ruleset. It works by optionally applying Corelight recommended changes to the rulesets, and extracting content from Suricata rules and creating Zeek Intel rules with that content.

Content is only extracted from enabled rules and the "do_notice" flag can individually be set based on rule type. This means you can use the typical enable.conf and disable.conf rules to control what data is extracted. See *Suricata policy settings* for details.

Tip: No configuration is required to include local Suricata rulesets. See *Locally managed sources* for details.

- Any ".rules" or ".rules.tar.gz" ruleset placed in the global-suricata folder is automatically available to all policies.
- Any ".rules" or ".rules.tar.gz" ruleset placed in a local-suricata folder is automatically available to that policy.

4.3.1 Suricata configuration files

Suricata uses four configuration files when processing traffic and/or testing rules.

- suricata.yaml
- classification.config
- reference.config
- threshold.config

These configuration files can be manually placed in the policy configs folder (`/etc/corelight-update/configs/<policy>/`), or the policy can be configured to pull Suricata configuration files from remote sources if desired. See *Remote config files*.

See *Using a proxy with Corelight-update* for details about using a proxy to download remote sources.

Optionally, these configuration files can be pushed to the policy in Fleet Manager or directly to a sensor. See *Push content settings*.

Warning: Suricata configuration files are not pushed to Microsensor.

4.3.2 Disabled rules

By default, disabled rules are not written back to the final Suricata ruleset. If desired, disabled rules can be included in the ruleset file by enabling `write_disabled_rules: true` in the *Suricata policy settings*.

4.3.3 Ruleset testing

By default, Corelight-update attempts to test the ruleset using Suricata, if it's available on the host running Corelight-update. If Suricata is not available, Corelight-update logs that it did not test the ruleset and continues.

If the rulesets is tested, and one or more rules fail the test, the details of the failed rules are logged and processing continues. Optionally, Corelight-update can be configured to discard a failed ruleset, after the failed rules have been logged, by setting `fail_on_ruleset_error: true` in the *Suricata policy settings*.

If any of the Suricata configuration files are placed in the policy configuration folder, or pulled from a remote location, they are automatically used when testing the Suricata ruleset.

Tip: It is recommended to use the same version of Suricata for testing that will be used in production. Testing with the Corelight version of Suricata can be accomplished by installing the Corelight Softsensor (without a license) on the same host running Corelight-update.

For debian based installation, Software Sensor is automatically installed as a "recommended" package. This can be disabled by adding the `--no-install-recommends` when installing Corelight-update.

Corelight-update and Software Sensor use the same package repository so the installation only requires a single command. See Software Sensor Online Installation for details.

See the following sections for more details:

4.3.3.1 Suricata policy settings

The configuration options mentioned in *Suricata configuration* can be changed with the following settings:

(continues on next page)

(continued from previous page)

Atomic rule extraction

Currently, only IP and JA3 based rules can be extracted. For IP based rules, the rule has to have a subnet or IP address in the rule. If it only uses a address group, it will not get extracted.

Remote config files

If you maintain a centralized set of Suricata configuration files for ruleset tuning and management, you can configure Corelight-update to automatically download the files from a remote source, and apply them to the Corelight-update connected sensors.

The Suricata configuration files `disable.conf`, `enable.conf` and `modify.conf` can be applied at a global, and at a policy level.

- To learn about the processing order, see *Order of operations*.
- For information about setting Suricata configuration files at the Global level, see *Configuration settings*.

For example, to pull a `modify.conf` file from GitHub:

The supported authentication types are no auth, `basic`, or `token`. When using the no auth option, leave the `auth_type` field empty.

Supported Suricata configuration files include:

- disable.conf
- enable.conf
- modify.conf
- suricata.yaml
- classification.config
- reference.config
- threshold.config

4.3.3.2 Suricata rules management

Corelight-update uses the familiar disable, enable, and modify.conf files to process and manage Suricata rules. However, Corelight-update offers significant performance and functionality improvements compared to other solutions.

Once all the rules from all the sources are downloaded and merged, Corelight-update makes up to three passes processing the rules:

1. The first pass will process Corelight recommended modifications (if enabled).
2. The second pass will process global modifications.
3. The third pass will process the individual policy modifications.

For each pass, any disable rule filters (disable.conf entries) are processed, then the enable rule filters (enable.conf entries), followed by rule modifiers (modify.conf entries).

File filters

In addition to disabling individual rules, the `disable.conf` entries can be used to ignore entire rulesets by file name. Filters are used to identify and ignore ruleset files as they are copied to the working directory for processing. After the files are downloaded and uncompressed (as necessary), if a ruleset filename matches an entry in `disable.conf`, it is ignored.

Important: The `Filename` filter matches all file names that begin with the entry.

Rule filters

To disable a rule that is enabled by default, add the rule to the disable.conf file. To enable a rule that is disabled by default, add the rule to the enable.conf file.

There are multiple methods to identify rules to be disable or enabled. One method, rule filters can be added by listing the Signature ID <SID> or Generator ID:Signature ID combination <GID>:<SID>.

Another method is to use regex. Rule filters that use a regex pattern will be applied to rules that match that pattern.

Note: Regex patterns must be wrapped in double quotes or have any white space removed. Use a `\s` to represent white space.

Special characters also have to be escaped, for example, use `\$` for `$`.

A method unique to Corelight-update, rule filters can also be added individually or in groups with Field:Value pairs. Use any of these fields to identify the rule:

When using the Metadata field to identify a rule, if there are any white spaces in the string to look for, it must be wrapped in double quotes.

Rule modifiers

To modify a Suricata rule, identifying the rule is the same as rule filters, with the exception that multiple rules can also be identified with GID:SID pairs. Multiple GID:SID entries on the same line need to be comma separated.

Rules can be identified and modified one of four ways:

- The legacy format: `<gid:sid> "<from regex>" "<to string>"` (The `gid` is optional.)
- The legacy regex format: `re:<rule regex> "<from regex>" "<to string>"`
- The new Corelight-update regex format: `re:<rule regex> <field>:<value>`
- The new Corelight-update format: `<rule> <field>:<value>`

Tip: See the Suricata documentation for more information about Suricata rules format.

Legacy format and Legacy regex format

The legacy and legacy regex formats require the <from regex> and <to string> statements to be enclosed in double quotes, and separated with a space `"<from regex>" "<to string>"`. The `" "` between the expressions delineates the two.

With the legacy format, the rule identifier is a combination of one or more `GID:SID` combinations. With the legacy regex format, the rule identifier is a regex pattern `re:<rule regex>`. For example,

(continues on next page)

(continued from previous page)

Caution: Regex patterns used to identify the rule must be wrapped in double quotes or have any white space removed. Use a `\s` to represent white space.

Corelight-update regex format

The Corelight-update regex format can use a regex pattern to identify the rule or rules to be modified, and then use the new Field:Value method to modify the rule.

In the `Field:Value` pair, values can be enclosed in double quotes but are not required (unless double quotes are required in that signature field i.e. "Msg")

Once the rule is identified, it can be modified by listing the field and the value it should be set to. (See *Modify examples*)

New Modify Options for v1.3.0

New in Corelight-update v1.3.0, content can now be appended to the "Metadata" and "Other" fields with `MetadataAppend` and `OtherAppend` respectively. If the "Append" field name is used, any content in the "Value" section will be appended with a space between the existing content and the new content.

(continues on next page)

(continued from previous page)

Corelight-update format

The same as above, in the `Field:Value` pair, values can be enclosed in double quotes but are not required (unless double quotes are required in that signature field i.e. “Msg”)

Any of these fields can be used to identify the rule:

Metadata contains

If the metadata contains value includes white spaces, it must be wrapped in double quotes.

Once the rule is identified, the same fields listed for Corelight-update regex format (listed above) can be modified by listing the field and the value.

4.3.3.3 Modify examples

It is common to modify the source and/or destination of a rule. Multiple addresses or ranges of addresses can be assigned to the same rule. See the Suricata documentation for examples of source and destination operators.

This example modifies a rule so that it only matchs on traffic coming from all `$HOME_NET` sources except 192.168.0.1.

Tip: The unedited rule was added as a comment just to document the original rule.

This example will modifies the rule so it matches any source except 192.168.0.1, and any destination except 192.168.0.2.

This example modifies the rule so it matches all customer networks except customer "B".

The following example modifies the priority of all rules with a classtype of "attempted-user" to 1.

The following example modifies all rules with a specific classtype to another classtype.

4.4 Intel management

You can leverage the Zeek Intelligence framework to match a list of IOC's against live network traffic on the sensor. Use Corelight-update to validate and merge one or more threat intel files, and publish a single, integrated threat intel file.

4.4.1 Intel management settings

Every time a new intel file is generated, a copy of the file with the current timestamp is also created. The `intel_file_cleanup` and `max_intel_file_age` (in hours) control the retention of the timestamped copies.

4.4.2 Disable Threat Intel indicators

If provided, Corelight-update will also use an intel disable file `disable_filename` to remove unwanted indicators from the published intel file, allowing you to effectively "disable" specific threat intel indicators.

The disable.intel file is a text file with a single column of indicators to remove.

4.4.3 Add Threat Intel sources

Threat intel sources are collections of IOC's in Zeek compatible formatted files. These files can be provided by a variety of sources, including security vendors, and as open source IOC collections.

Corelight-update can pull threat intel sources hosted in local and remote repositories.

To add threat intel sources, you'll configure them as Corelight-update *Policy sources*.

For an example of a third-party Threat Intel policy source configuration, see *Threat intelligence source example*

To review the order that the configurations are processed in, see *Order of operations*.

4.5 Input management

You can leverage the Zeek Input framework to provide contextual data for use with enabled Zeek packages. Depending on the Zeek packages, this data can be used to generate logs (alerts), prevent the generation of logs (alerts), and/or enrich logs with additional data from external sources.

Corelight-update can collect input files from local or remote sources and/or generate input files with enabled *third-party integrations*. Once collected, any input files with the same name will automatically get merged into a single input file with that name and published.

4.5.1 Input management settings

Corelight sensors contain a number of Zeek packages that can take advantage of input files. However, none of those files are included out of the box. If `default_input` is enabled, Corelight-update will automatically generate templates for those files and place them in the local-input folder. See *Locally managed sources* for the path.

4.6 Third-party integrations settings

Third-party integrations provide support for a vendor-specific threat source, including source-based customizations and authentication.

Third-party integrations differ from Corelight-update *Policy sources*, in that a Policy source must be pre-formatted content you can download using an unauthenticated, basic-authenticated, or token-authenticated URL.

4.6.1 Axonius

The Axonius integration will download data about all entities known to Axonius that have a current IP address.

Axonius relies on connections to other vendor platforms, and polls for data every 12 hours by default. Once data has been ingested, a discovery process will correlate data from multiple connectors. The discovery process can take up to four hours to complete.

The Axonius integration will automatically check the status of the discovery process each time the service runs, as querying the Axonius API can have unpredictable results if the discovery process is still processing during the query. In scenarios where the discovery process has not completed, the data cached from the previous successful run will be used, and the API will be queried again on the next service interval.

The `interval_hours` setting should not be set lower than the Axonius polling frequency. If it's set to 0, the integration will attempt to download additional content each time the Corelight-update service runs. See *Configuration settings*

Once downloaded, the data can then be used to create two separate Input Framework files. One file includes CVE information, and the other contains Host information. These files can be used by Zeek scripts to generate new logs, or enrich existing logs, such as the **known_hosts.log** or **suricata_corelight.log**.

The input file will be published along with any other input files from other configured integrations, if any. If Corelight-update is configured to push input files, the file will automatically get pushed to the Fleet Manager policy and/or all sensors in the policy. See *Push content settings* for more details.

4.6.1.1 Settings

4.6.1.2 CVE Input file

The input file contains the following information (if it's available):

- IP address (required)
- Hostname
- Host Unique ID: Provided by the device data `'internal_axon_id'` field.
- OS version
- Machine domain
- Endpoint information source (required)
- Known CVE list (required)

The following is a sample input file created by this integration, using tab-separated values.

4.6.1.3 Hosts Input file

The input file contains the following information (if it's available):

- IP address (required)
- MAC address
- Hostname
- Host Unique ID: Provided by the device data `'internal_axon_id'` field.
- OS version
- Endpoint status

- Machine domain
- Endpoint information source (required)

The following is a sample input file created by this integration, using tab-separated values.

Attention: The integration only creates the Input Framework files to be loaded on sensors. Additional Zeek scripts are required to be loaded on the sensors to use this data. If you enable these integrations, Corelight-update will upload the input files to the sensor. But if the desired script isn't available on the sensor, the input data won't be used.

See *Zeek package management* for information about using Corelight-update to manage Zeek package bundles.

See *Zeek-Endpoint-Enrichment* for an example of a Zeek package that can use this data.

4.6.2 CrowdStrike

CrowdStrike integration collects Hosts and Vulnerability data of systems, networks and applications using Falcon Exposure Management. Suricata Rulesets and Indicators are downloaded from CrowdStrike's Falcon Threat Intelligence.

4.6.2.1 Falcon Exposure Management - Hosts & CVEs

The CrowdStrike Falcon Exposure Management integration will download data about all hosts with CVE's that match the provided criteria. If no "entity_type" is specified, all known entities (that have a current IP address) will be listed. If no CVE "status" or "severity" is specified, all CVE's who's status is *NOT* "closed" will be downloaded.

CrowdStrike Falcon Exposure Management relies on endpoint agents and (if configured) performs "network scans" to identify network entities and vulnerabilities. As a result, frequently downloading data from Falcon Exposure Management can provide near-realtime updates. If the 'interval_hours' is set to 0, the integration will attempt to download additional content each time the Corelight-update service runs. See *Configuration settings*

Once downloaded, the data will be used to create an Input Framework file that can be used by a Zeek script to generate new logs, or enrich existing logs, such as the **known_hosts.log**, **suricata_corelight.log** or **notice.log**.

The input file will be published with any other input files from other integrations (if there are any). If "input" in enabled in the "push_content" settings, the file will automatically get pushed to the Fleet Manager policy and/or all sensors in the policy. See *Push content settings* for more details.

CrowdStrike configuration settings:

Hosts Input file

The input file contains the following information (if it's available):

- IP address (required)
- MAC address
- Hostname
- Host Unique ID: Provided by the `'aid'` field of vulnerability data.
- OS version
- Endpoint status
- Machine domain
- Additional description
- Customer ID
- Endpoint information source (required)

The following is a sample input file created by this integration, using tab-separated values.

(continues on next page)

(continued from previous page)

CVE Input file

The input file contains the following information (if it's available):

- IP address (required)
- Hostname
- Host Unique ID: Provided by the `'aid'` field of vulnerability data.
- Machine domain
- OS version
- Endpoint information source (required)
- Customer ID
- CVE list

The following is a sample input file created by this integration, using tab-separated values.

(continues on next page)

(continued from previous page)

Attention: The CrowdStrike Exposure Management integrations only create Input Framework files to be loaded on sensors. Additional Zeek scripts are required to be loaded on the sensors to use this data. If you enable these integrations, Corelight-update will upload the input files to the sensor. But if the desired script isn't available on the sensor, the input data won't be used.

See *Zeek package management* for information about using Corelight-update to manage Zeek package bundles.

See *Zeek package references* for examples of Zeek packages that can use this data.

4.6.2.2 Suricata Ruleset

The CrowdStrike Falcon Suricata ruleset file will only be downloaded if it has changed since the last interval.

If the 'interval_hours' is set to 0, the integration will attempt to download additional content each time the Corelight-update service runs. See *Configuration settings*

Once downloaded, the ruleset will be processed with the rulesets from all other sources.

Attention: Downloading Suricata rules from CrowdStrike requires a Falcon Intelligence Premium subscription. The Client ID and Client Secret need access to the following API: *https://api.crowdstrike.com/intel/entities/rules-latest-files/v1*

Settings

4.6.2.3 Falcon Threat Intelligence

The CrowdStrike Falcon Indicators integration will download all requested indicators at each interval.

There are several configurable options for CrowdStrike indicators. Select the malicious confidence level, how many days worth of history, and which indicators to collect.

Note: Due to the high number of hash indicators available, the length of history is configured separate from other types of indicators.

Intel Malicious confidence options are: "high", "medium", "low", or "unverified". The following definitions apply to *malicious_confidence*:

- high: If indicator is an IP or domain, it has been associated with malicious activity within the last 60 days.
- medium: If indicator is an IP or domain, it has been associated with malicious activity within the last 60-120 days.
- low: If indicator is an IP or domain, it has been associated with malicious activity exceeding 120 days.
- unverified: This indicator has not been verified by a CrowdStrike Intelligence analyst or an automated system.

Once downloaded, the data will be merged with all other intel files (if there are any), and published. If "intel" in enabled in the "push_content" settings, the file will automatically get pushed to the Fleet Manager policy and/or all sensors in the policy. See *Push content settings* for more details.

Attention: Downloading intel indicators from CrowdStrike requires a Falcon Intelligence subscription or better. The Client ID and Client Secret need access to the following API: *https://api.crowdstrike.com/intel/combined/indicators/v1*

Settings

Error: The default request limit is set to 50,000, which works for most customers. However, for some customer subscriptions the request limit cannot be more than 10,000 or an error is returned.

In addition to configuring which indicators to collect, you can also filter the indicators based on the type of target or the threat type.

- To list a single Target or Threat Type, enter the string with both double quotes and single quotes.
- To list multiple Targets or Threat Types, enter the string with both double quotes and square brackets around the entire string, and single quotes around each item.

Examples:

Intel log

This integration will enrich the intel.log with content like the following:

4.6.3 FireEye iSIGHT Threat Intelligence

Configure the FireEye iSIGHT Threat Intelligence integration to set how frequently the integration runs, how much history to initially download, how much history to use in an Intel file, and how much history to maintain in the SQLite DB. This integration uses the Mandiant Threat Intelligence v2 API.

---

do_notice

The `do_notice` flag can be set based on the indicator type. It is set in the DB based on the settings when the indicator is downloaded, and is updated in the intel file each time it is written.

Tip: By default, only MD5 hash support is enabled on a Corelight Sensor. It is recommended that you use only one hash type. If you plan on using another hash type, update the configuration and enable the appropriate package on the sensor.

If the 'interval_hours' is set to 0, the integration will attempt to download additional content each time the Corelight-update service runs. See *Configuration settings*

4.6.3.1 Settings

4.6.3.2 Intel log

This integration will enrich the intel.log with content like the following:

(continues on next page)

(continued from previous page)

4.6.4 Maxmind GeoIP

Corelight physical and virtual sensors include a GeoIP database and are not updated with Corelight-update. This section only applies to Software Sensors.

You can sign up for free and get a license key from https://www.maxmind.com/en/geolite2/signup. Once you have an AccountID and LicenseKey, enter them in the `geoip` configuration below. You can also edit the GeoIP advanced configuration if you want to change additional settings. The GeoIP advanced configuration is in the Global Configuration and Policy settings file located here: `/etc/corelight-update/global/config.yaml`

4.6.4.1 GeoIP settings

Tip: If you are running Corelight-update on the same host as a Corelight Software Sensor, the default location the sensor looks for the GeoIP database is `/usr/share/GeoIP/`

The GeoIP settings:

4.6.4.2 Maxmind configuration settings

If you need to change more settings than listed above, you can edit the Maxmind configuration file as needed.

Tip: The Maxmind configuration file is located here: `/etc/corelight-update/global/GeoIP.conf`

4.6.5 icannTLD Zeek script

icannTLD is a Zeek script that uses the official ICANN Top-Level Domain (TLD) list to extract the relevant information from a DNS query and enrich the DNS log with that information. It can also mark whether it's trusted or not. The source of the ICANN TLDs can be found here: https://publicsuffix.org/list/effective_tld_names.dat.

Today, anyone can create a TLD and ICANN updates the list several times a day, as changes are made.

TLDs are generally split into two categories:

- ccTLDs are Country Code TLDs, such as .us, .jp and .uk
- gTLDs are Generic TLDs and include the traditional names .com, .net, and .org. Generic TLDs also include the new TLDs such as .info, .city, .microsoft, etc.

As of December 2022, there are 6887 Top-Level Domains that can include up to 4 parts.

- 19.2% (1,322) TLDs only contain one part (i.e. .com)
- 52.2% (3,597) TLDs contain two parts (i.e. mo.us)
- 28.5% (1,964) TLDs contain three parts (i.e. k12.mo.us)
- 0.1% (4) TLDs contain four parts (i.e. pvt.k12.ma.us)

As a result, any method of identifying TLDs without using the ICANN TLD database, i.e. regex, will miss identify over 80% of them.

Tip: The Trusted Domains list is a custom list, created by the user, to filter domains during searches.

4.6.5.1 Script functions

icannTLD parses every DNS query and adds the following fields to the DNS Log.

Table 1: New DNS Log Fields

Field	Value	Description
icann_tld		This is the Top-Level Domain based on the official list of TLDs from ICANN.
icann_domain		This is the Domain based on the official list of TLDs from ICANN.
icann_host_ subdomain		This is the remaining nodes of the query after the domain has been removed. In some cases this is the subdomain, in other cases it's the host name, and in others it's host name and subdomain.
is_trusted_domain	true/false	Each query is marked true or false based on the icann_domain and a custom *trusted_domains.dat* file.

Corelight-update can generate the required Input files needed for the icannTLD Zeek script. However, the optional trusted domain list is not generated. See https://github.com/corelight/icannTLD for more details.

If the 'interval_hours' is set to 0, the integration will attempt to download additional content each time the Corelight-update service runs. See *Configuration settings*

The icannTLD settings:

(continues on next page)

(continued from previous page)

4.6.6 Mandiant Threat Intelligence

Configure the Mandiant Threat Intelligence integration to set how frequently the integration runs, how much history to initially download, how much history to use in an Intel file, and how much history to maintain in the SQLite DB. This integration uses the Mandiant Threat Intelligence API v4. To use the v2 API, see *FireEye iSIGHT Threat Intelligence*.

do_notice

The `do_notice` flag can be set based on the individual indicator type, and an overall minimum Confidence Score. For example, setting the `min_confidence_score_doNotice: 95`, would only set the `do_notice` flag to `T`, if the Mandiant Confidence score was 95% or better. It is not set in the database; only when the intel file is created.

Tip: By default, only MD5 hash support is enabled on a Corelight Sensor. It is recommended that you use only one hash type. If you plan on using another hash type, update the configuration and enable the appropriate package on the sensor.

If the 'interval_hours' is set to 0, the integration will attempt to download additional content each time the Corelight-update service runs. See *Configuration settings*

4.6.6.1 Settings

(continues on next page)

(continued from previous page)

- `download_history` defines how many days of indicators to initially download. Once the initial download is complete, the integration will run at the next interval and only pull changes back to the last successful download. If a download fails, or the `download_history` setting is changed, the next download will pull all indicators as defined by the `download_history`.
- `max_history` defines how many days of indicators to store in the local database.
- `use_history` defines how many days of indicators to use in the intel file.
- `min_confidence_score_use` defines the minimum confidence score an indicator must have to be included in the intel file.
- `min_confidence_score_download` defines the minimum confidence score an indicator must have to be downloaded from Mandiant. Note that Mandiant frequently updates its confidence scores for indicators, so configure this setting well below the `min_confidence_score_use`. If an indicator's confidence score is changed and downgraded below this setting, the latest indicator will not be pulled from Madiant, and the indicator record in the local database will retain the last downloaded confidence score until the `max_history` is met and it's aged out.
- `exclude_os_indicators` allows the download of open source indicators. This setting only applies to downloading new indicators. Once the indicator is downloaded, it will remain in the local database and in use until it no longer meets the `use_history` setting. It will remain in the local database until the `max_history` is met and it's aged out.

The following is a sample input file created by this integration, using tab-separated values.

4.6.6.2 Intel log

This integration will enrich the intel.log with content like the following:

If the ExtendIntel Zeek package is loaded, the intel.log will be enriched with additional content like the following: (all indicators will not have all fields)

(continues on next page)

(continued from previous page)

4.6.7 MS Defender

The MS Defender integration will download data about hosts, and any CVE data. It collects data for known and unknown hosts using the Machines API. For CVE data collection, the Vulnerabilities by Machine and Software API is used.

Once downloaded, the data is used to create two separate Input Framework files. One file includes CVE information, and the other contains Host information. These files can be used by Zeek scripts to generate new logs, or enrich existing logs, such as the **known_hosts.log** or **suricata_corelight.log**.

The input file will be published along with any other input files from other configured integrations, if any. If Corelight-update is configured to push input files, the file will automatically get pushed to the Fleet Manager policy and/or all sensors in the policy. See *Push content settings* for more details.

If the 'interval_hours' is set to 0, the integration will attempt to download additional content each time the Corelight-update service runs. See *Configuration settings*

4.6.7.1 Settings

4.6.7.2 CVE Input file

The input file contains the following information (if it's available):

- IP address (required)
- Hostname
- Host Unique ID: Provided by the machine data `'id'` field.
- OS version
- Machine domain
- Endpoint information source (required)
- Known CVE list (required)

The following is a sample input file created by this integration, using tab-separated values.

4.6.7.3 Hosts Input file

The input file contains the following information (if it's available):

- IP address (required)
- MAC address
- Hostname
- Host Unique ID: Provided by the machine data `'id'` field.
- OS version
- Endpoint status

- Machine domain
- Endpoint information source (required)

The following is a sample input file created by this integration, using tab-separated values.

Attention: The integration only creates the Input Framework files to be loaded on sensors. Additional Zeek scripts are required to be loaded on the sensors to use this data. If you enable these integrations, Corelight-update will upload the input files to the sensor. But if the desired script isn't available on the sensor, the input data won't be used.

See *Zeek package management* for information about using Corelight-update to manage Zeek package bundles.

See *Zeek-Endpoint-Enrichment* for an example of a Zeek package that can use this data.

4.6.8 MISP - Zeek export

An export of all attributes of a specific bro type to a formatted plain text file. By default only published and IDS flagged attributes are exported.

You can configure your tools to automatically download a file one of the Bro types.

To restrict the results by tags, use the usual syntax. Please be aware the colons (:) cannot be used in the tag search. Use semicolons instead (the search will automatically search for colons instead). To get ip values from events tagged tag1 but not tag2 use:

Alternatively, it is also possible to pass the filters via the parameters in the URL. The format is as described below:

Zeek Type	MISP Type
all:	All types listed below.
ip:	*ip-src, ip-dst, ip-src|port, ip-dst|port, domain|ip*
url:	url
domain:	hostname, domain, domain|ip
ja3-fingerprint-md5:	ja3-fingerprint-md5
email:	email, email-src, email-dst, target-email
filename:	filename, email-attachment, attachment, filename|md5, filename|sha1, filename|sha256, malware-sample, pdb
filehash:	md5, sha1, sha256, authentihash, ssdeep, imphash, pehash, impfuzzy, sha224, sha384, sha512, sha512/224, sha512/256, tlsh, filename|md5, filename|sha1, filename|sha256, filename|authentihash, filename|ssdeep, filename|imphash, filename|pehash, filename|impfuzzy, filename|sha224, filename|sha384, filename|sha512, filename|sha512/224, filename|sha512/256, filename|tlsh, malware-sample
certhash:	x509-fingerprint-sha1
software:	user-agent

The keywords false or null should be used for optional empty parameters in the URL.

For example, to retrieve all attributes for event #5, including non IDS marked attributes too, use the following line:

4.6.9 AlienVault Open Threat Exchange

The main settings for the AlienVault OTX integration determines how frequently the integration runs, how much history to initially download, how much history to use in an Intel file, and how much history to keep in the SQLite DB.

The initial download will retrieve OTX threat intel "pulses" back to the configured days set in the 'download_history' setting. Each consecutive download will only contain new pulses since the last successful download. If you change the 'download_history' setting, the integration resets, and on the next run it will retrieve all pulses back to the new setting.

---

do_notice

The `do_notice` flag can be set based on the indicator type. It is set in the DB based on the settings when the indicator is downloaded and is updated in the intel file each time it is written.

Tip: By default, only MD5 hash support is enabled on a Corelight Sensor. It is recommended that you use only one hash type. If you plan on using another hash type, update the configuration and enable the appropriate package on the sensor.

If the 'interval_hours' is set to 0, the integration will attempt to download additional content each time the Corelight-update service runs. See *Configuration settings*

4.6.9.1 Settings

(continues on next page)

(continued from previous page)

4.6.9.2 Intel log

This integration will enrich the intel.log with content like the following:

4.6.10 Zeek package management

Corelight-update implements some basic package management functions, similar to the Zeek Package Manager (ZKG). https://docs.zeek.org/projects/package-manager/en/stable/

Corelight-update Zeek Package Management can:

- Build package bundles from a manifest file by downloading packages from the Internet.
- Build package bundles from a manifest file in offline mode.
- Push package bundles, built by Corelight-update, to Fleet Manager policies and/or sensors.
- Push package bundles, built off-box, to Fleet Manager policies and/or sensors.
- Push Corelight-signed package bundles to all sensors except Microsensor.

Corelight-update only generates package bundles from a manifest file. While Corelight-update can push package bundles that are created by other sources, it does not install packages locally or edit existing bundles.

> **Warning:** Enabling "offline_mode" only prevents downloading the Zeek Package Index. If a URL is provided to a package repo in the manifest file, it still attempts to clone it.

The policy settings for Zeek Package Management are:

The inventory settings for pushing Zeek Packages are:

ZKG and Microsensor

Pushing a package bundle to a Microsensor uses SCP and requires a path to place the bundle. After Corelight-update pushes a package bundle, it uses ZKG on the sensor to install the packages.

For details on how to install and setup ZKG on a Microsensor, see *Zeek Package Manager (ZKG)*

4.6.10.1 Create and push a package bundle

To create and push a package bundle:

1. **Enable** `package_management` in the policy configuration.
2. Set the name of the manifest file. For example, `manifest_file: bundle.manifest`
3. Place a manifest file in the policy configuration folder.
4. Set `push_package_bundle: true` in the policy.
5. Ensure `bundle: true` in the inventory file for the desired sensors.
6. If the manifest file changes, a new bundle will automatically be created and pushed each time the Corelight-update service runs.
    - Optionally, force create and push a bundle with the CLI command `corelight-update -b <policy name>`

4.6.10.2 Push external package bundles

To push a package bundle created outside of Corelight-update:

1. **Disable** `package_management` in the policy configuration
2. Set the name of the bundle. For example, `bundle_name: corelight.bundle`
3. Place the package bundle in the `global-bundle` or `local-bundle` folder
   - A package bundle in `local-bundle` takes precedence
4. Set `push_content: package_bundle: true` in the policy
5. Ensure `bundle: true` in the inventory file for the desired sensor
6. External bundles are not automatically pushed to sensors. They must be pushed with the CLI command `corelight-update -b <policy name>`

4.6.11 SentinelOne

The SentinelOne integration will download data about hosts, and any CVE data. It fetches hosts known to SentinelOne, using the Agents and Network Discovery API, or Rouges API. The Agents API will collect host data from its network interfaces for the "secured" hosts. The Network Discovery (Rangers) API will be utilized for the "unsupported", "unknown", and "unsecured" hosts. If Network Discovery is not enabled, the integration will utilize the Rogues API. For CVE data collection, the Application Management Risks endpoint will be utilized.

Once downloaded, the data will be used to create two separate Input Framework files. One file includes CVE information, and the other contains Host information. These files can be used by Zeek scripts to generate new logs, or enrich existing logs, such as the **known_hosts.log** or **suricata_corelight.log**.

The input file will be published with any other input files from other integrations (if there are any). If "input" in enabled in the "push_content" settings, the file will automatically get pushed to the Fleet Manager policy and/or all sensors in the policy. See *Push content settings* for more details.

If the 'interval_hours' is set to 0, the integration will attempt to download additional content each time the Corelight-update service runs. See *Configuration settings*

4.6.11.1 Settings

4.6.11.2 Hosts Input file

The input file contains the following information (if it's available):

- IP address (required)
- MAC address
- Hostname
- Host Unique ID
- OS version
- Endpoint status
- Machine domain
- Description
- Endpoint information source (required)

If the data source is SentinelOne Agents data, the **Host Unique ID** comes from Agent data's `'uuid'` field. When using SentinelOne Rouges data, the `host_uid` comes from Rouges data's `'id'` field. When using SentinelOne Network Discovery data, the `host_uid` comes from Network Discovery data's `'id'` field.

The following is a sample input file created by this integration, using tab-separated values.

4.6.11.3 CVE Input file

The input file contains the following information (if it's available):

- IP address (required)
- Hostname
- Host Unique ID
- Machine domain
- OS version
- Endpoint information source (required)
- CVE list

If the data source is SentinelOne Agents data, the **Host Unique ID** comes from Agent data's `'uuid'` field. When using SentinelOne Rouges data, the `host_uid` comes from Rouges data's `'id'` field. When using SentinelOne Network Discovery data, the `host_uid` comes from Network Discovery data's `'id'` field.

The following is a sample input file created by this integration, using tab-separated values.

Attention: The integration only creates the Input Framework files to be loaded on sensors. Additional Zeek scripts are required to be loaded on the sensors to use this data. If you enable these integrations, Corelight-update

will upload the input files to the sensor. But if the desired script isn't available on the sensor, the input data won't be used.

See *Zeek package management* for information about using Corelight-update to manage Zeek package bundles.

See *Zeek-Endpoint-Enrichment* for an example of a Zeek package that can use this data.

4.6.12 Tenable

Tenable integration collects vulnerability data of systems, networks and applications using Tenable's Vulnerability Management solutions, Tenable.SC and Tenable.IO.

Tenable.SC (On-Prem) and Tenable.IO (Cloud based) integration helps collect data about all hosts, including their associated CVEs. The collected data is used to track changes in the network and Vulnerability data of assets over time and will used by Zeek scripts to enrich logs.

4.6.12.1 Tenable.sc

The configuration required for Tenable Security Center is minimal.

- Each severity and pluginType must be listed.
- Provide the host address and port of the local TenableSC instance.

There is no need to set the integration interval more frequently than the frequency Tenable.SC is scanning the network.

If the `interval_hours` is set to 0, the integration will attempt to download additional content each time the Corelight-update service runs. See *Configuration settings*

Once downloaded, this data will be used to create an Input Framework file that can be used by a Zeek script to generate new logs, or enrich existing logs, such as the **suricata_corelight.log**.

The input file will be published with any other input files from other integrations (if there are any). If "input" is enabled in the "push_content" settings, the file will automatically get pushed to the Fleet Manager policy and/or all sensors in the policy. See *Push content settings* for more details.

Attention: The Nessus (Tenable Security Center) user you're using to provide an `access_key` and `secret_key` must have Security Management rights. It is not recommended to use an admin user.

Settings

Input file

The input file contains the following information (if it's available):

- IP address (required)
- Hostname
- Host Unique ID: Provided by the vulnerability details response data's `'uuid'` field.
- Machine domain
- OS version
- Endpoint information source (required)
- Customer ID
- CVE list

The following is a sample input file created by this integration, using tab-separated values.

Attention: The Tenable.SC integrations only create Input Framework files to be loaded on sensors. Additional Zeek scripts are required to be loaded on the sensors to use this data. If you enable these integrations, Corelight-update will upload the input files to the sensor. But if the desired script isn't available on the sensor, the input data won't be used.

See *Zeek package management* for information about using Corelight-update to manage Zeek package bundles.

See *Zeek-CVE-Enrichment* for an example of a Zeek package that can use this data.

4.6.12.2 Tenable.io

The TenableIO CVE integration will download data about all hosts with CVE's that match the provided criteria. It fetches CVEs known to Tenable, using its Vulnerability Management API. If the 'interval_hours' is set to 0, the integration will attempt to download additional content each time the Corelight-update service runs. See *Configuration settings*

Once downloaded, this data will be used to create an Input Framework file that can be used by Zeek scripts to enrich logs, such as the **notice** or **suricata_corelight.log**.

The input file will be published with any other input files from other integrations (if there are any). If "input" is enabled in the "push_content" settings, the file will automatically get pushed to the Fleet Manager policy and/or all sensors in the policy. See *Push content settings* for more details.

Settings

Input file

The input file contains the following information (if it's available):

- IP address (required)
- Hostname
- Host Unique ID: Provided by the export vulnerabilities Chunk data's `'uuid'` field.
- Customer ID
- Criticality
- Machine domain
- OS version
- Endpoint information source (required)
- CVE list

The following is a sample input file created by this integration, using tab-separated values.

(continues on next page)

(continued from previous page)

Attention: The Tenable.IO integrations only create Input Framework files to be loaded on sensors. Additional Zeek scripts are required to be loaded on the sensors to use this data. If you enable these integrations, Corelight-update will upload the input files to the sensor. But if the desired script isn't available on the sensor, the input data won't be used.

See *Zeek package management* for information about using Corelight-update to manage Zeek package bundles.

See *Zeek-CVE-Enrichment* for an example of a Zeek package that can use this data.

4.6.13 ThreatQ - Zeek export

These steps explain how to export Zeek indicators for use with an external threat detection system. Follow these instructions to export your data.

1. Select the **Settings icon > Exports**.

   The Exports page appears with a table listing all exports in alphabetical order.

2. Click **Add New Export**.

   The Connection Settings dialog box appears.

3. Enter an **Export Name**.

4. Click **Next Step**.

   The Output Format dialog box appears.

5. Provide the following information:

FIELD	VALUE
Which type of information would you like to export?	Indicators
Output Type	text/plain
Special Parameters	indicator.status=Active&indicator.deleted=N

Note: You can edit the output format. This includes the ability to remove unwanted indicator types.

6. Under Output Format Template, enter:

7. Click **Save Settings**.
8. Under **On/Off**, toggle the switch to enable the export.

When finished, use the URL to download the intel data in Zeek format.

Attention: Some integrations, such as Tenable.sc, CrowdStrike Exposure Management, and icannTLD require an additional Zeek script to be loaded on the sensors. See *Zeek package management*. If you enable the integration, Corelight-update will upload the input file to the sensor. But if the required script isn't available on the sensor, the input data won't be used.

CHAPTER

FIVE

REFERENCES

5.1 Internal References

5.1.1 CLI commands

Warning: When updating from a full or partial configuration, any config section provided must have all none-zero fields provided. Any missing fields will be updated to their zero value.

5.1.1.1 CLI help output

To view the available CLI Commands, use `corelight-update -h`

(continues on next page)

(continued from previous page)

(continues on next page)

(continued from previous page)

5.1.2 Corelight-update service

When Corelight-update is installed, in addition to a `corelight-update.service`, a system user and group are automatically created. The service runs as the system user `corelight-update`. However, it's disabled by default. To run Corelight-update as a service, enable the service and start it.

1. Enable the service:

2. Start the service:

3. To view the status of the service:

4. To monitor the logs from the service: The `-f` option makes the command follow the log until it's canceled.

5.1.3 Using a proxy with Corelight-update

If the Corelight-update host requires use of a network proxy to access and download content, you can configure the Corelight-update host service or user session to provide the proxy location using the `HTTPS_PROXY` or `https_proxy` environment variables.

5.1.3.1 Update the service definition

When running Corelight-update as a service, it will automatically use the `HTTPS_PROXY` or `https_proxy` environment variables when set. Corelight-update will not use any HTTP proxy variables. The proxy location can be added to the service definition.

Update the service definition using override.conf.

1. Use `systemctl` to create an override.conf.

2. Create a `[Service]` section in the `override.conf`, and set the `HTTPS_PROXY` environment variable. For example:

3. Save the changes. You can review the `override.conf` in the path `/etc/systemd/system/corelight-update.service.d`

4. Reload systemd.

5. Restart the Corelight-update service.

5.1.3.2 Update the user environment

The options to set a proxy can vary based on the OS distribution being used. Corelight-update will automatically use the `HTTPS_PROXY` or `https_proxy` environment variables when set. Corelight-update will not use any HTTP proxy variables.

In general, you can set the proxy environment variables at the host and user level.

- In Red Hat, update `/etc/profile` to set the proxy at the host level for users.
- In Ubuntu, update `/etc/environment` to set the proxy at the host level for users.
- For user accounts, update the user's shell profile to set the proxy.

For example:

5.1.4 Order of operations

The order of operations for every interval starts with:

1. Read the global policy configuration and each individual policy configuration.
2. Process the global tasks.
3. Process each policy, and push content for that policy.

5.1.4.1 Process global tasks

See *Configuration settings* for configuration options.

1. Process enabled integrations.
2. Download remote Suricata config files and store the in `/etc/corelight-update/global/`.
3. Download new content and update the Global Source Cache.
4. Remove content from the global cache for sources that are no longer configured.

5.1.4.2 Process policy tasks

See *Policy configuration* for configuration options

1. Copy local suricata rulesets from `/etc/corelight-update/configs/<policy>/local-suricata/` to the working directory.
2. Copy global suricata rulesets from `/etc/corelight-update/global/global-suricata/` to the working directory.
3. Copy local intel files from `/etc/corelight-update/configs/<policy>/local-intel/` to the working directory.
4. Copy global intel files from `/etc/corelight-update/global/global-intel/` to the working directory.
5. Remove content from the policy cache for sources that are no longer configured.
6. Download new content from policy sources.
7. Add default Input files to `/etc/corelight-update/configs/<policy>/local-input/` (if enabled - only runs once)
8. Process enabled integrations based on their intervals. See *Third-party integrations settings*
9. Process Input files and update the statefile.
10. Process Suricata rulesets.
    1. Collect ruleset files
        1. Collect new source content and copy it to the suricata working directory.
            - Check the global cache first.
            - If not in the global cache, download new content directly and update the policy level cache.
        2. Check for global `.rules` or `.rules.tar.gz` files in `/etc/corelight-update/global/global-suricata/` and extract/copy them to the suricata working directory.
        3. Check for local `.rules` or `.rules.tar.gz` files in `/etc/corelight-update/configs/<policy>/local-suricata/` and extract/copy them to the suricata working directory.
    2. Merge all of the rulesets into a single ruleset, ignoring any ruleset file identified with *File filters* in the following:
        - Corelight recommended disable.conf (if enabled)
        - global disable.conf (if it exists)
        - policy disable.conf (if it exists)
    3. If enabled, process Corelight recommended **disable.conf**, **enable.conf** and **modify.conf** files in that order.
    4. If they exist, process global **disable.conf**, **enable.conf** and **modify.conf** files in that order.
    5. If they exist, process policy **disable.conf**, **enable.conf** and **modify.conf** files in that order.
    6. If enabled, extract selected atomic rules from the Suricata ruleset and generate a Zeek Intel file.
    7. If enabled and Suricata is installed on the same host, test the new ruleset with Suricata in test mode (see *Suricata configuration* for details).
    8. Publish the new Suricata ruleset - **suricata.rules**.
11. Process Intel files
    1. Check for global intel files in `/etc/corelight-update/global/global-intel/`, and copy them to the intel working directory.

2. Check for local intel files in `/etc/corelight-update/configs/<policy>/local-intel`, and copy them to the intel working directory.
3. Evaluate all of the global, local, and integration intel files from all sources. If `disable.intel` is available, evaluate and remove indicators. Merge and dedupe results into a single file.
4. Publish the new intel file - **intel.dat**

5.1.4.3 Push content for policies

Corelight-update deploys content updates in a specific order:

1. Push new Intel files.
2. Push new Suricata ruleset.
3. Push new Zeek Package bundle.
4. Push new Input files

By default, Corelight-update will push updates to the sensors concurrently. Corelight-update will open a connection to multiple sensors in a policy, push updated content, and cycle to the next sensor, up to the global configuration setting `parallel_push_limit`. See *Configuration settings*.

Tip: Corelight-update only attempts to push new content to sensors. You can manually force a push of all existing content to a group of sensors using the *CLI commands*.

5.1.5 Build test process

Corelight-update is currently tested with Docker to ensure it successfully installs on the following operating systems:

- image: centos:7
- image: rockylinux:8
- image: rockylinux:9
- image: registry.access.redhat.com/ubi8/ubi
- image: debian:10
- image: debian:11
- image: ubuntu:18.04
- image: ubuntu:20.04
- image: ubuntu:22.04
- image: ubuntu:22.04 (arm64)
- image: amazonlinux:2

5.1.6 System requirements

The minimum system requirements are:

- An x86_64 or ARM64 processor.
- 4 GB memory.
- A host running a Linux OS.
- Network connectivity to the Internet, or to an internal-facing threat intelligence data repository.
- To push content to your sensors, or to Fleet Manager, network connectivity to the management interface is required.

5.1.7 Commonly used Suricata rulesets

Any source that can be downloaded in the standard Suricata ruleset format, and does not require authentication, can be added to the list of sources. Here is a list of common Suricata ruleset sources. Just verify the URL, modify as needed, and add it to your list of sources.

- **Corelight Labs Suricata Rules:** `https://feed.corelight.com/rules/corelight.rules`
- **ET/Open:** `https://rules.emergingthreats.net/open/suricata-6.0/emerging.rules.tar.gz`
- **ET/Pro:** `https://rules.emergingthreatspro.com/<insert-et-pro-key-here>/suricata-7.0.3/etpro.rules.tar.gz`

  This ruleset applies to Suricata 7.0.3, which was added in Corelight Sensor v27.11.
- **oisf/trafficid:** `https://openinfosecfoundation.org/rules/trafficid/trafficid.rules`
- **ptresearch/attackdetection:** `https://raw.githubusercontent.com/ptresearch/AttackDetection/master/pt.rules.tar.gz`
- **scwx/enhanced:** `https://ws.secureworks.com/ti/ruleset/<insert-secret-code-here>/Suricata_suricata-enhanced_latest.tgz`
- **scwx/malware:** `https://ws.secureworks.com/ti/ruleset/<insert-secret-code-here>/Suricata_suricata-malware_latest.tgz`
- **scwx/security:** `https://ws.secureworks.com/ti/ruleset/<insert-secret-code-here>/Suricata_suricata-security_latest.tgz`
- **sslbl/ssl-fp-blacklist:** `https://sslbl.abuse.ch/blacklist/sslblacklist.rules`
- **sslbl/js3-fingerprints:** `https://sslbl.abuse.ch/blacklist/ja3_fingerprints.rules`
- **etnetera/aggressive:** `https://security.etnetera.cz/feeds/etn_aggressive.rules`
- **tgreen/hunting:** `https://raw.githubusercontent.com/travisbgreen/hunting-rules/master/hunting.rules`
- **malsilo:** `https://malsilo.gitlab.io/feeds/dumps/malsilo.rules.tar.gz`

5.2 Zeek package references

5.2.1 ExtendIntel

The ExtendIntel Zeek package enriches the intel.log with additional data (if available).

If the intel file contains the following fields, the data will automatically be added to the intel.log.

- threat_score
- verdict
- verdict_source
- confidence
- desc
- lastseen
- firstseen
- url
- reports
- campaigns
- associated
- category

5.2.1.1 Intel log

This is an example of an intel.log without any additional data:

(continues on next page)

(continued from previous page)

If the ExtendIntel Zeek package is loaded, the intel.log will be enriched with additional content like the following: (all indicators will not have all fields)

5.2.2 Zeek-CVE-Enrichment

The Zeek-CVE-Enrichment Zeek package uses an input file named "cve_data.tsv" that contains known CVE information about hosts within an environment, to enrich the suricata_corelight.log and/or the notice.log. The information can come from multiple sources, including a manually created file.

The package works by monitoring every suricata_corelight and notice log entry for CVE alerts.

1. When a suricata_corelight event is triggered, the script will search the suricata alert metadata for a CVE ID.
    1. If no CVE ID is found in the metadata, it will then search the Suricata signature name for the CVE ID.
2. When a notice event is triggered, the script will search the message ("msg") part of the Notice log for a CVE ID.
    1. If no CVE ID is found in the "msg", the script will search the "note" section of the Notice log event.
3. If a CVE ID is found in any location, for either log, the script will look up the host IP address in the "cve_data" table.
    1. If the host is found, the CVE ID found in the log is compared to the list of known CVE's for that host.
        1. If a match is found, the relevant log is enriched with additional information from the table.

As long as the input file is named "cve_data.tsv", and has a match to a CVE alert in a suricata_corelight.log or notice.log, the log will be enriched with additional data.

5.2.2.1 Input file (cve_data.tsv)

The input file should contain the following information (if it's available):

- IP address (required)
- Hostname
- Endpoint information source (required)
- Endpoint criticality
- Endpoint Unique ID
- Customer ID
- Machine domain
- OS version
- CVE list

The following is a sample input file created by this integration, using tab-separated values.

5.2.2.2 suricata_corelight log

A typical suricata_corelight.log provides content similar to this example:

If the Zeek package Zeek-CVE-Enrichment is loaded, the suricata_corelight.log and/or the notice.log will be enriched with additional content provided by the integration, similar to this example:

Note: Field names begin with “orig” or “resp” to identify which host is referenced.

5.2.3 Zeek-Endpoint-Enrichment

The Zeek-Endpoint-Enrichment Zeek package uses the input file "hosts_data.tsv" to enrich multiple logs with relevant data. Depending on the data provided in the "hosts_data.tsv" file and the options enabled, this package can enrich the following logs:

- known_devices
- known_domains
- known_hosts
- known_names
- conn (optional)
- all logs (optional)

Note: Additional fields will only be created if the relevant data is available.

5.2.3.1 Input file (hosts_data.tsv)

The input file contains the following information (if available):

- IP address (required)
- MAC address
- Hostname
- Endpoint information source (required)
- Endpoint criticality
- Endpoint status
- Endpoint Unique ID
- Customer ID
- OS version
- Machine domain
- Description

For example, this is a sample input file created by this integration, formatted using tab-separated values.

(continues on next page)

(continued from previous page)

5.2.3.2 known_hosts log

The known_hosts log will always be enriched (with available data) for local hosts.

For example, a known_hosts.log can contain “endpoint” data similar to the sample below:

5.2.3.3 known_devices log

The known_devices entry will only be created if the MAC is available. For example, a known_devices.log can contain content similar to the sample below:

5.2.3.4 known_domains log

The known_domains entry will only be created if the "Machine Domain" is available. For example, a known_domains.log can contain content similar to the sample below:

5.2.3.5 known_names log

The known_names entry will only be created if the hostname is available. For example, a known_names.log can contain content similar to the sample below:

5.2.3.6 conn log

If enabled, a typical conn.log can contain content similar to the sample below:

Note: Information related to “orig” or “resp” could come from different sources.

(continues on next page)

(continued from previous page)

5.2.3.7 all logs

If enabled, any log with an "id.xxx" field can contain content similar to the sample below:

Note: Information related to "orig" or "resp" could come from different sources.

(continues on next page)

(continued from previous page)

5.2.4 Zeek-Endpoint-Enrichment-conn

The Zeek-Endpoint-Enrichment-conn Zeek package is an options package used with the *Zeek-Endpoint-Enrichment* to enable enrichment of the conn.log. It is a single line that enables the "EndpointEnrichment::extra_logging_conn" option.

Attention: This package requires *Zeek-Endpoint-Enrichment*

5.2.5 Zeek-Endpoint-Enrichment-all

The Zeek-Endpoint-Enrichment-all Zeek package is an options package used with the *Zeek-Endpoint-Enrichment* to enable enrichment of all logs. It is a single line that enables the "EndpointEnrichment::extra_logging_all" option.

Attention: This package requires *Zeek-Endpoint-Enrichment*

5.3 Third-party configuration guides

5.3.1 Zeek Package Manager (ZKG)

5.3.1.1 Quickstart guide

These instructions are intended for installations of ZKG on the same host as a Software Sensor.

5.3.1.2 Dependencies

- Python 3.6+
- git: https://git-scm.com
- GitPython: https://pypi.python.org/pypi/GitPython
- semantic_version: https://pypi.python.org/pypi/semantic_version
- btest: https://pypi.python.org/pypi/btest

Note that following the ZKG installation process via pip3 will automatically install its dependencies for you.

5.3.1.3 Installation

It is recommended to use the latest version of pip3:

```
```

To install the latest release of ZKG on PyPi:

```
```

5.3.1.4 Basic setup

ZKG supports four broad approaches for managing Zeek packages. These details represent one of those approaches and are specific for a Corelight Software Sensor running as root.

1. Create the directory for the ZKG configurations.

   ```
   ```

2. Create/Edit the file `/root/.zkg/config` and add the following contents:

   ```
   ```

3. Run the following command to refresh the Zeek index and create the `/etc/corelight/packages` directory.

   ```
   ```

4. Edit `/etc/corelight/local.zeek` and add the following line:

   ```
   ```


5.3.1.5 Usage

Corelight-update will use ZKG to manage package bundles on a Software Sensor.

Check the output of zkg –help for an explanation of all available functionality of the command-line tool.